赤勇玄心行天道

致力于计算机软件开发。

Speex详解（2019年09月25日更新）

Speex详解

整理者：赤勇玄心行天道

QQ号：280604597

微信号：qq280604597

QQ群：511046632

博客：www.cnblogs.com/gaoyaguo

大家有什么不明白的地方，或者想要详细了解的地方可以联系我，我会认真回复的！

你可以随意转载，无需注明出处！

写文档实属不易，我希望大家能支持我、捐助我，金额随意，1块也是支持，我会继续帮助大家解决问题！

1. 简介

Speex is an Open Source/Free Software patent-free audio compression format designed for speech. The Speex Project aims to lower the barrier of entry for voice applications by providing a free alternative to expensive proprietary speech codecs. Moreover, Speex is well-adapted to Internet applications and provides useful features that are not present in most other codecs. Finally, Speex is part of the GNU Project and is available under the revised BSD license.

Speex（音标[spi:ks]）是一套开源免费的、无专利保护的、针对语音设计的音频压缩格式。Speex项目通过以提供昂贵的专用语音编解码器的免费替代方案为目标，来降低语音应用程序的进入门槛。此外，Speex非常适用于互联网应用程序，并提供了其他大多数编解码器中不存在的有用特性。最后，Speex是GNU项目的一部分，可以在修订后的BSD许可证下使用。

1. Speex详解（2019年09月25日更新）(40765)
2. 音频数字信号详解（2022年04月09日更新）(15076)
3. PortAudio详解（2015年12月1日更新）(13872)
4. 固态硬盘逻辑坏道简单修复方法（2015-04-02更新）(12348)
5. Oracle 11g数据库详解（2017-01-23更新）(8794)

评论排行榜

1. Speex详解（2019年09月25日更新）(2)
2. C、C++函数和类库详解（VC++版）（2016-06-26更新）(2)
3. PortAudio详解（2015年12月1日更新）(1)
4. Windows控制台光标控制(1)

推荐排行榜

1. 音频数字信号详解（2022年04月09日更新）(2)
2. Speex详解（2019年09月25日更新）(2)
3. 必须要做的事(1)
4. PortAudio详解（2015年12月1日更新）(1)
5. Windows控制台光标控制(1)

最新评论

1. Re:Speex详解（2019年02月28日更新）
@ 蓝芝玉林STM32官网的Speex库是阉割版的，把16k和32k采样都阉割掉了，质量等级也是阉割了的，你只能设置到这么高了，阉割的原因就是内存不够用。...
--赤勇玄心行天道
2. Re:Speex详解（2019年02月28日更新）
您好，我想请教您一下关于带宽模式和超带宽模式下的参数设置，我使用的是STM32官方的例程，但是却发现只支持8Khz，8Kbps这一种模式，其余的开启编码后就会出现内存错误，找了很久也找不到原因，今天发...
--蓝芝玉林
3. Re:Windows控制台光标控制
看完之后，非常详细，比较有用，我是初学者，多多包涵
--Thor_one
4. Re:PortAudio编程入门使用手册详解（2015-12-1更新）
不错
--biij5698
5. Re:C、C++函数和类库详解（VC++版）（2014-8-10更新）
@ Jingle Guo必须的...
--高压锅的程序员

Speex库官方网站：http://www.speex.org/

Speex库API官方英文详解：http://www.speex.org/docs/api/speex-api-reference/index.html

NSpeex库（用于.Net和Silverlight的Speex库）官方网站：http://nspeex.codeplex.com/

Speex库目前最新的版本是Speex 1.2.0和SpeexDSP 1.2.0。

注意：Speex编解码器已经被Opus编解码器淘汰，Speex还是可以继续使用，由于Opus比Speex在各方面都更好，所以鼓励大家切换到Opus，但是Opus只支持编码和解码，不支持噪音抑制、声学回音消除等其他处理功能。

1. 历史

1. SpeexDSP 1.2.0 out

June 7, 2019

This is the latest stable release of the SpeexDSP library.

1. Speex 1.2.0 out

December 7, 2016

This is the latest stable release of the Speex 1.2.0 codec library.

1. SpeexDSP 1.2rc3 is out

January 3, 2015

This brown-paper-bag release adds two headers that should have been included with SpeexDSP 1.2rc2. These are needed to build the resampler with NEON optimizations and to build SpeexDSP without the Speex codec library.

1. Speex 1.2rc2 and SpeexDSP 1.2rc2 are out

December 6, 2014

This release splits the speex codec library and the speex DSP library into separate source trees. Both projects received build-system improvements, bugfixes, and cleanup. The speex codec's VBR tuning was improved, while the speexdsp resampler got some NEON optimizations.

1. Speex 1.2rc1 is out

July 23, 2008

This release adds support for acoustic echo cancellation with multiple microphones and multiple loudspeakers. It also adds an API to decorrelate loudspeaker signals to improve multi-channel performance. In the bugfix department, there are fixes for a few bugs in the echo canceller, jitter buffer and preprocessor. At this point, the API for 1.2 should be stable and only a few very minor additions are planned.

1. Speex 1.2beta3 is out

December 11, 2007

The most obvious change in this release is that all the non-codec components (preprocessor, echo cancellation, jitter buffer) have been moved to a new libspeexdsp library. Other changes include a new

jitter buffer algorithm and resampler improvements/fixes. This is also the first release where libspeex can be built without any floating point support. To do this, the float compatibility API must be disabled (--disable-float-api or DISABLE_FLOAT_API) and the VBR feature must be disabled (--disable-vbr or DISABLE_VBR).

1. **Speex 1.2beta2: Fixed-point improvements and more**

May 24, 2007

Again, this new releases brings many improvements. The RAM requirement for wideband has gone down drastically (i.e. more than 2x). A new resampler module has been added, providing arbitrary sampling rate conversion -- fast. The echo canceller has also been improved. A bug in 1.2beta1 that made the echo canceller unstable has been fixed. The echo canceller should now converge faster, be robust and tolerant of incorrect capture-playback synchronisation. The preprocessor has also been greatly improved. Not only should the quality be better, but it is now fully converted to fixed-point. At last, early TriMedia support (incomplete) has been merged.

1. **Speex 1.2beta1: Better, smaller, faster and more**

September 4, 2006

This new release brings many significant improvements. The quality has been improved, both at the encoder level and the decoder level. These include enhancer improvements (now on by default), input/output high-pass filters, as well as fixing minor regressions in previous 1.1.x releases. A strange and rare instability problem with pure sinusoids has also been fixed. On top of that, memory use has been greatly reduced, especially for fixed-point and narrowband. The fixed-point narrowband encoder+decoder memory use has been cut by more than half, making it possible to fit both in less than 6 kB of RAM. In general, CPU requirement had gone down, especially for the fixed-point port. The Blackfin port has been speeded up significantly, thanks to David Rowe. There are also a few fixes for the TI C5X DSPs, as well as better support for C++ compilers and crappy MS compilers. Oh, and before anyone starts worrying, the format (bit-stream) itself has not changed, so Speex is still compatible with version 1.0 and will continue to be in the future.

Non-codec improvements include a extension (easier to use) to the echo canceller API and a Speex-independent version of the jitter buffer. The echo canceller should also be more robust to saturation in the capture path. Last, but not least, the documentation has been updated.

1. **How the Echo Canceller works**

August 7, 2006

Always wanted to know how the echo canceller works? On Adjusting the Learning Rate in Frequency Domain Echo Cancellation With Double-Talk has just been accepted for IEEE Trans. on Audio, Speech and Language Processing.

1. **New website design**

August 6, 2006

We have a new website design. There may be a few problems, so please report them and be patient.

1. **On How Speex Works**

March 19, 2006

Interested in how Speex works? Have a look at "Improved Noise Weighting in CELP Coding of Speech - Applying the Vorbis Psychoacoustic Model To Speex", authored by Jean-Marc Valin and Christopher Montgomery.

1. **Speex 1.1.12 Released**

February 19, 2006

New things:

echo canceller converted to fixed-point (sponsored by Analog Devices)

Improvements to the experimental Vorbis-based masking model (use --enable-vorbis-psy as an argument to the configure script)

several bug fixes

1. **Speex 1.1.11.1 Released**

December 2, 2005

This is a brown-paper-bag release fixing a pretty bad bug that affected the fixed-point port in 1.1.11. Architectures that use float were not affected at all. Architectures that use fixed-point had a big drop in audio quality. Only version 1.1.11 is affected. Sorry about the inconvenience.

1. **Speex 1.1.11 Released**

November 20, 2005

This release includes lots of bug fixes. These include SSE, fixed-point and Blackfin. Also, the echo canceller and packet loss concealment have been improved.

1. **Speex 1.1.10 Released**

June 11, 2005

The main improvement in this release is a Blackfin port funded by Analog Devices. This includes Blackfin assembly optimizations that reduce cpu time by a factor of two. Also, the packet loss concealment code has now been converted to fixed-point and some of bugs for 16-bit architectures were fixed.

1. **Speex 1.1.9 Released**

June 1, 2005

The main improvement in this release is that the acoustic echo canceller is finally usable. This work has been sponsored by Tipic Inc. Also, several bugs have been fixed for the TI C5x port.

1. **Speex 1.1.8 Released**

May 7, 2005

Lots of changes in this release. Initial TI C5x port, some fixed-point improvements and fixes, better temporary memory allocation (smaller), and the size of integer types are now detected automatically.

There is also a new SPEEX_PLC_TUNING option.

1. **Speex 1.0.5 Released**

May 6, 2005

The main change with this release is that it includes API additions from the 1.1.x branch (while being backward compatible), so that transition from 1.0.x to 1.1.x can be made easier.

1. **Speex 1.1.7 Released**

March 2, 2005

The changes for this release are very broad and include generic optimizations in the encoder, ARM-specific optimizations (gcc inline assembly), optional shortcuts in the encoder sacrificing quality for speed, fixed-point improvements (perceptual enhancement converted), reduction in memory usage, the Symbian code now uses the same API, and several bug fixes.

1. **New Specialized libs Released**

November 18, 2004

libspeex_emce.lib is an x86 emulator build (with debug information)

ibspeex_armce.lib is an ARMV4 release build

1. **Speex 1.1.6 Released**

July 28, 2004

There are seven changes in this release.

Improved jitter buffer (now actually works)

Denoiser tuning

Improved echo canceller (please send feedback)

Support for Symbian OS

Gapless playback for speexenc/speexdec

Run-time identification of Speex version with a new speex_lib_ctl() call

Moved the includes to /usr/include/speex/

1. **Speex 1.0.4 Released**

July 21, 2004

There are three changes in this release.

Headers are now in /usr/include/speex/ (but a copy is still in /usr/include for compat reasons).

Pseudo-gapless playback (i.e. playback has the same number of samples)

Fixed a potential bug (unconfirmed) that might cause a segfault in special circumstances.

1. **Speex 1.1.5 Released**

April 21, 2004

The main change in this release is that the 1.1.5 API and ABI are now compatible with 1.0.x. The versions of the functions taking a short* now have an "_int" suffix, as in speex_encode_int().

1. **Speex 1.1.4 Released**

January 20, 2004

Happy Belated New Year. This release has minor fixed-point improvements and a code cleanup. The SSE code has been converted from inline assembly to SSE intrinsics, so it should now work on win32. More functions have been written to use SSE.

1. **Speex 1.1.3 Released**

December 2, 2003

This unstable release brings more improvements to the fixed-point port. Many new functions have been converted and most modes now work in real-time. I encourage everyone to test this code by compiling with --enable-fixed-point and --enable-fixed-point-debug and report any error messages and send the (smallest possible) file which reproduces the problem.

1. **Speex 1.0.3 Released**

November 19, 2003

In this bugfix release: a fix for a multithreading bug and a correction for an underflow problem that could slow decoding dramatically on x86 processors.

1. **Speex 1.1.2 Released**

November 11, 2003

This new unstable release improves on the fixed-point port started in 1.1.1. The port is not yet complete, but many modes are now usable in real-time on ARM processors. The fixed-point version is enabled with --enable-fixed-point and ARM-specific optimizations can be enabled with --enable-arm-asm.

1. **Speex 1.1.1 Released**

November 1, 2003

This release adds a partial fixed-point port which can be enabled using the --enable-fixed-point option at configure time. Not all floating-point operations have been converted yet, but all the code should work.

1. **Speex 1.0.2 Released**

September 24, 2003

Just a bugfix release. This update adds soundcard support for Solaris and the BSDs as well as minor bugfixes and a documentation update.

1. Features 特性

Speex is based on CELP and is designed to compress voice at bitrates ranging from 2 to 44 kbps. Some of Speex's features include:

Speex编解码器是基于CELP（Code Excited Linear Prediction）激励线性预测编码的，并且专门为2至44kbps的语音压缩而设计的。Speex的一些特性包括：

- 只支持8000Hz窄带（Narrow Band）、16000Hz宽带（Wide Band）、32000Hz超宽带（Ultra Wide Band）的三种带模式进行编解码，不支持其他采样频率。
- 只支持单声道，不支持多声道。
- 只能对音频数据进行处理，不支持音频数据的输入输出，也就是不支持录音和播放。
- 支持强化立体声编码（Intensity Stereo Encoding）。
- 支持数据包丢失隐藏（Packet Loss Concealment、PLC）。
- 支持固定比特率（Constant Bit Rate、CBR）。
- 支持可变比特率（Variable Bit Rate、VBR）。
- 支持平均比特率（Average Bit Rate、ABR）。
- 支持非连续传输（Discontinuous transmission、DTX）。
- 支持定点执行（Fixed-point implementation）。
- 支持浮点执行（Floating-point implementation）。
- 支持声学回音消除（Acoustic Echo Canceller、AEC）。
- 支持残余回音消除（Residual Echo Canceller、REC）。
- 支持噪音抑制（Noise Suppression、NS）。
- 支持混响音消除（Dereverb）。

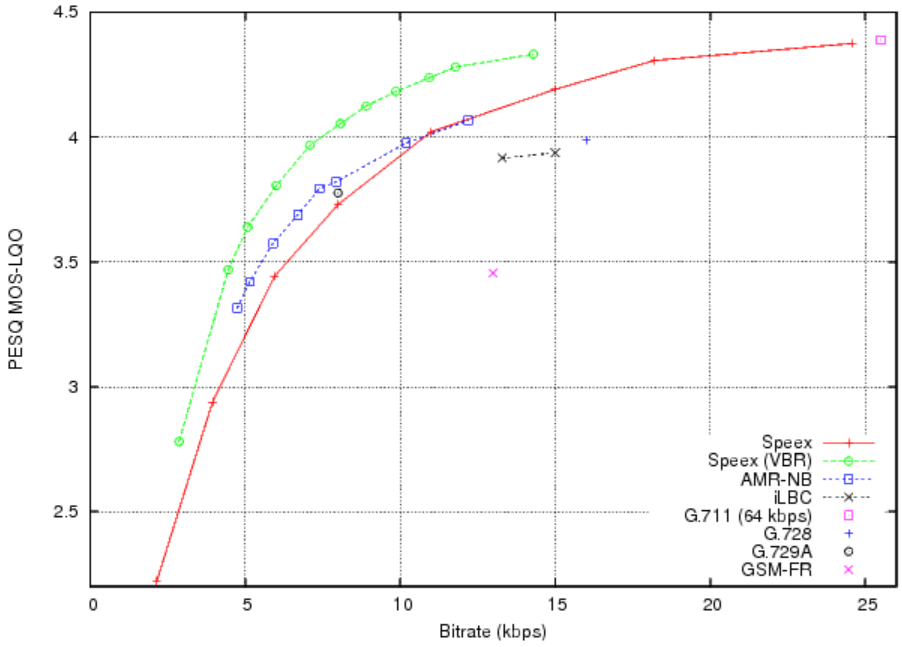
- 支持自动增益控制（Automatic Gain Control、AGC）。
- 支持语音活动检测（Voice Activity Detection、VAD）。
- 支持多速率（multi-rate）。
- 支持嵌入式（Embedded）。
- 支持重采样（Resample）。

Codec Quality Comparison:

编解码器的质量比较:

Warning: these are machine-generated results (not from real listeners) and hence should be taken with a grain of salt.

注意: 这是机器自动生成的结果（不是来源于真正的听众），因此不可全信。



Codec Feature Comparison:

编解码器的特性比较:

Codec 编解码器	Rate 频率 (kHz)	bitrate 比特率 (kbps)	delay 延时 frame+lookahead (ms)	multi-rate 多速率	embedded 嵌入式	VBR 可变比特率	PLC 数据包丢失隐藏	bit-robust	license 授权
Speex	8,16,32	2.15-24.6 (NB) 4-44.2 (WB)	20+10 (NB) 20+14 (WB)	yes	yes	yes	yes		open-source/ free software
iLBC	8	15.2 or 13.3	20+5 or 30+10				yes		no charge, but not open-source

AMR-NB	8	4.75-12.2	20+5?	yes			yes	yes	proprietary
AMR-WB (G.722.2)	16	6.6-23.85	20+5?	yes			yes	yes	proprietary
G.722.1 (Siren 7)	16	(16) 24, 32	20+20	yes			yes	yes	no charge, but not open-source
G.729	8	8	10+5				yes	yes	proprietary
GSM-FR	8	13	20+?				?	?	patented?
GSM-EFR	8	12.2	20+?				yes	yes	proprietary
G.723.1	8	5.3 6.3	37.5				yes	?	proprietary
G.728	8	16	0.625						proprietary
G.722	16	48 56 64	?		yes			?	?

Definitions

定义

multi-rate

多速率

Allows the codec to change bitrate dynamically, at any moment

允许编解码器可以在任何时候动态改变比特率。

embedded

嵌入式

A codec that embeds narrowband bitstreams in wideband bitstreams

编解码器会将窄带比特流嵌入到宽带比特流中。

VBR

可变比特率

Variable bitrate

可变比特率

PLC

数据包丢失隐藏

Packet loss concealment

数据包丢失隐藏

bit-robust

Robust to corruption at the bit level, as found on wireless networks

Special Features

重要特性

Speex

Speex

Speex supports intensity stereo encoding and 32 kHz sampling

Speex支持强化立体声编码和32kHz采样。

iLBC

iLBC

iLBC frames are encoded completely independently; while this provides better quality when 10% (or more) of the packets are being dropped, this makes the codec suboptimal for clean line conditions.

iLBC的帧编码是完全独立的；当丢包率达到10%（或更大）时，它能提供更好的质量，这使编解码器适合不太干净的线路环境。

1. 下载

1. speex-1.2.0

speex-1.2.0包含了以下几个项目：

1. libspeex：libspeex静态库，库里面包含了编码和解码相关的函数。
2. speexenc：speex编码器控制台程序，输入是ogg格式封装的speex编码的spx文件，输出是格式为raw PCM或者WAVE文件，有控制台参数提示信息。依赖libogg库。
3. speexdec：speex解码器控制台程序，输入是格式为raw PCM或者WAVE文件，输出是ogg格式封装的speex编码的spx文件，有控制台参数提示信息。依赖libogg库。
4. testenc：测试窄带编码。
5. testenc_wb：测试宽带编码。
6. testenc_uwb：测试超宽带编码。

1. speexdsp-1.2.0

speexdsp-1.2.0包含了以下几个项目：

1. libspeexdsp：libspeexdsp静态库，库里面包含了预处理器、声学回音消除器、重采样器、自适应抖动缓冲器等相关的函数。
2. testdenoise：测试噪音抑制。
3. testecho：测试声学回音消除。
4. testresample：测试重采样。

1. 编译

1. Visual Studio编译libspeex

1. Visual Studio + speex-1.2.0
2. 编译speex-1.2.0就可以了。

2. Visual Studio编译speexenc、speexdec

1. libogg 下 载 地 址 ： http://www.linuxfromscratch.org/blfs/view/svn/multimedia/libogg.html
2. 打开libogg-1.3.2\win32\VS2010\libogg_static.sln。
3. 编译libogg_static项目。
4. 在speex-1.2.0\include文件夹下，新建ogg文件夹。
5. 复制libogg-1.3.2\include\ogg文件夹下的ogg.h和os_types.h到speex-1.2.0\include\ogg文件夹下。
6. 在speex-1.2.0文件夹下，新建lib文件夹。
7. 复制libogg-1.3.2\win32\VS2010\Win32\Debug文件夹下的libogg_static.lib到speex-1.2.0\lib文件夹下。
8. 打开speex-1.2.0\win32\VS2008\libspeex.sln。
9. 将speex-1.2.0\lib\libogg_static.lib添加到speexenc和speexdec中。
10. 编译speexenc和speexdec项目。
11. 编译后的speexenc.exe和speexdec.exe就在speex-1.2.0\win32\VS2008\Debug中。

2. 编码流程

使用Speex的API函数对音频数据进行压缩编码要经过如下步骤：

- 定义一个Speex格式数据流的内存指针变量vSpeexBits和一个Speex编码器的内存指针变量enc。
- 调用speex_bits_init(&vSpeexBits)函数初始化vSpeexBits。
- 调用enc = speex_encoder_init(&speex_nb_mode)函数初始化enc。其中speex_nb_mode是SpeexMode类型的变量，表示的是窄带模式。还有speex_wb_mode表示宽带模式、speex_uwb_mode表示超宽带模式。
- 调用int speex_encoder_ctl(void * state, int request, void * ptr)函数来设定编码器的参数，其中参数state表示编码器的内存指针；参数request表示要定义的参数类型，如SPEEX_GET_FRAME_SIZE表示设置帧大小，SPEEX_SET_QUALITY表示编码的质量等级；参数ptr表示要设定的值。
- 初始化完毕后，对每一帧声音作如下处理：调用函数speex_bits_reset(&vSpeexBits)重置vSpeexBits，然后调用函数speex_encode(enc_state, input_frame, &vSpeexBits)进行编码，参数bits中保存编码后的Speex格式数据帧，最后调用speex_bits_write(SpeexBits * bits, char * bytes, int max_len)函数将参数vSpeexBits中保存编码后的Speex格式数据帧读取到参数bytes中。
- 编码结束后，调用函数speex_bits_destroy(&bits)，speex_encoder_destroy(enc_state)来销毁SpeexBits和编码器。

1. 解码流程

对已经编码过的Speex格式音频数据帧进行解码要经过以下步骤：

- 定义一个SpeexBits类型变量bits和一个Speex解码器的内存指针变量dec。
- 调用speex_bits_init(&bits) 函数初始化bits。
- 调用dec = speex_decoder_init(&speex_nb_mode) 函数初始化dec。
- 调用函数speex_decoder_ctl(void * state, int request, void * ptr)来设定解码器的参数。
- 调用函数 speex_decode(void * state, SpeexBits * bits, float * out)对参数bits中的Speex格式音频数据帧进行解码，参数out中存放解码后的音频数据帧。
- 调用函数speex_bits_destroy(&bits), speex_decoder_destroy(void * state)来销毁SpeexBits和解码器。

下面是一段实例代码：

```
#include <speex.h>

#include <stdio.h>

/*帧的大小在这个例程中是一个固定的值,但它并不是必须这样*/

#define FRAME_SIZE 160


int main(int argc, char **argv)
```

```
{
        char * inFile;
        FILE * fin;
        short in[FRAME_SIZE];
        float input[FRAME_SIZE];
        char cbits[200];
        int nbBytes;
        void * state; /*保存编码的状态*/
        SpeexBits bits; /*保存字节因此他们可以被speex常规读写*/
        int i, tmp;
        //新建一个新的编码状态在窄宽(narrowband)模式下
        state = speex_encoder_init(&speex_nb_mode);
        //设置质量为8(15kbps)
        tmp=8;
        speex_encoder_ctl(state, SPEEX_SET_QUALITY, &tmp);
        inFile = argv[1];

        fin = fopen(inFile, "r");
        //初始化结构使他们保存数据
        speex_bits_init(&bits);
        while( 1 )
        {
                //读入一帧16bits的声音
                fread(in, sizeof(short), FRAME_SIZE, fin);
                if( feof(fin) )
                        break;
                //把16bits的值转化为float,以便speex库可以在上面工作
                for (i=0;i<FRAME_SIZE;i++)
                        input[i]=in[i];

                //清空这个结构体里所有的字节,以便我们可以编码一个新的帧
                speex_bits_reset(&bits);
                //对帧进行编码
                speex_encode(state, input, &bits);
                //把bits拷贝到一个利用写出的char型数组
                nbBytes = speex_bits_write(&bits, cbits, 200);
                //首先写出帧的大小,这是sampledec文件需要的一个值,但是你的应用程序中可能不一
样
                fwrite(&nbBytes, sizeof(int), 1, stdout);
                //写出压缩后的数组
                fwrite(cbits, 1, nbBytes, stdout);
        }
```

```
        //释放编码器状态量
        speex_encoder_destroy(state);
        //释放bit_packing结构
        speex_bits_destroy(&bits);
        fclose(fin);
        return 0;
}
```

1. 利用speex实现语音流压缩（转载）

原文地址：https://blog.csdn.net/u011473714/article/details/47010445

最近需要做一个基于udp的实时语音聊天的应用，语音流的压缩方面，我选择了使用speex。

Speex是一套主要针对语音的开源免费，无专利保护的音频压缩格式。Speex工程着力于通过提供一个可以替代高性能语音编解码来降低语音应用输入门槛 。另外，相对于其它编解码器，Speex也很适合网络应用，在网络应用上有着自己独特的优势。同时，Speex还是GNU工程的一部分，在改版的BSD协议中得到了很好的支持。

然后，看了一下speex手册和speex的api文档，写了一个简单的例程。

一、speex api的简单介绍

1. 编码：

a) 定义一个SpeexBits类型变量ebits和一个Speex编码器状态变量enc_state。

b) 调用speex_bits_init(&ebits)初始化。

c) 调用speex_encoder_init(&speex_nb_mode)来初始化enc_state。其中speex_nb_mode是SpeexMode类型的变量，表示的是窄带模式。还有speex_wb_mode表示宽带模式、speex_uwb_mode表示超宽带模式。

d) 调用函数int speex_encoder_ ctl(void *state, int request, void *ptr)来设定编码器的参数，其中参数state表示编码器的状态；参数request表示要定义的参数类型，如SPEEX_ GET_ FRAME_SIZE表示设置帧大小，SPEEX_ SET_QUALITY表示量化大小，这决定了编码的质量；参数ptr表示要设定的值。

e) 初始化完毕后，对每一帧声音作如下处理：调用函数speex_bits_reset(&ebits)再次设定SpeexBits，然后调用函数speex_encode_int(enc_state, input_frame, &ebits)，参数ebits中保存编码后的数据流。

f) 编码结束后，调用函数speex_bits_destroy (&ebits)，speex_encoder_destroy (enc_state)来销毁编码器

2. 解码

接口与编码类似，这里就不多说了~ ~

二、配置安装

在使用speex之前，首先当然要配置一下speex的环境，到官网下载speex源码，我使用的是1.2rc1版本。

tar zxvf speex-1.2rc1.tar.gz

cd speex-1.2rc1

./configure --prefix=/home/yzf/lib/speex （路径改成自己喜欢的)

make && make install

编译安装后，把/home/yzf/lib/speex/include 下的文件拷贝到 /usr/include下

把/home/yzf/lib/speex/lib/libspeex.so.1.5.0 拷贝到 /usr/lib下

并重命名为libspeex.so

并建立该文件的软链接 libspeex.so.1 ： ln -s libspeex.so libspeex.so.1

因为有些系统-lspeex使用的是 libspeex.so.1，比如我用的一个服务器的redhat

三、例程：

下面是我写的一个例程，我用"伪单例模式"封装了一下speex的接口，方便自己使用～～

voice.h

```
#ifndef VOICE_H
#define VOICE_H

/*
* 初始化和销毁
*/
void voice_encode_init();
void voice_encode_release();
void voice_decode_init();
void voice_decode_release();
/*
* 压缩编码
* short lin[] 语音数据
* int size 语音数据长度
* char encoded[] 编码后保存数据的数组
* int max_buffer_size 保存编码数据数组的最大长度
*/
int voice_encode(short in[], int size,
char encoded[], int max_buffer_size);
/*
* 解码
* char encoded[] 编码后的语音数据
* int size 编码后的语音数据的长度
* short output[] 解码后的语音数据
* int max_buffer_size 保存解码后的数据的数组的最大长度
*/
int voice_decode(char encoded[], int size,
short output[], int max_buffer_size);
#endif //define VOICE_H
```

voice.cpp

```
#include <speex/speex.h>
#include <cstring>
#include <cstdio>
#include "voice.h"

static int enc_frame_size;//压缩时的帧大小
static int dec_frame_size;//解压时的帧大小

static void *enc_state;
static SpeexBits ebits;
static bool is_enc_init = false;

static void *dec_state;
static SpeexBits dbits;
static bool is_dec_init = false;
//初始话压缩器
void voice_encode_init() {
printf("enc init\n");
int quality = 8;
speex_bits_init(&ebits);
enc_state = speex_encoder_init(&speex_nb_mode);
speex_encoder_ctl(enc_state, SPEEX_SET_QUALITY, &quality);
speex_encoder_ctl(enc_state, SPEEX_GET_FRAME_SIZE, &enc_frame_size);
is_enc_init = true;
}
//销毁压缩器
void voice_encode_release() {
printf("enc release\n");
```

```
speex_bits_destroy(&ebits);
speex_encoder_destroy(enc_state);
is_enc_init = false;
}
//初始化解压器
void voice_decode_init() {
printf("dec init\n");
int enh = 1;
speex_bits_init(&dbits);
dec_state = speex_decoder_init(&speex_nb_mode);
speex_decoder_ctl(dec_state, SPEEX_GET_FRAME_SIZE, &dec_frame_size);
speex_decoder_ctl(dec_state, SPEEX_SET_ENH, &enh);
is_dec_init = true;
}
//销毁解压器
void voice_decode_release() {
printf("dec release\n");
speex_bits_destroy(&dbits);
speex_decoder_destroy(dec_state);
is_dec_init = false;
}
//压缩语音流
int voice_encode(short in[], int size,
char encoded[], int max_buffer_size) {

if (! is_enc_init) {
voice_encode_init();
}

short buffer[enc_frame_size];
char output_buffer[1024 + 4];
int nsamples = (size - 1) / enc_frame_size + 1;
int tot_bytes = 0;
for (int i = 0; i < nsamples; ++ i)
{
speex_bits_reset(&ebits);
memcpy(buffer, in + i * enc_frame_size, enc_frame_size * sizeof(short));

speex_encode_int(enc_state, buffer, &ebits);
int nbBytes = speex_bits_write(&ebits, output_buffer + 4, 1024 - tot_bytes);
memcpy(output_buffer, &nbBytes, 4);

int len =
max_buffer_size >= tot_bytes + nbBytes + 4 ?
nbBytes + 4 : max_buffer_size - tot_bytes;

memcpy(encoded + tot_bytes, output_buffer, len * sizeof(char));

tot_bytes += nbBytes + 4;
}
return tot_bytes;
}
//解压语音流
int voice_decode(char encoded[], int size,
short output[], int max_buffer_size) {

if (! is_dec_init) {
voice_decode_init();
}

char* buffer = encoded;
```

```
short output_buffer[1024];
int encoded_length = size;
int decoded_length = 0;
int i;

for (i = 0; decoded_length < encoded_length; ++ i)
{
speex_bits_reset(&dbits);
int nbBytes = *(int*)(buffer + decoded_length);
speex_bits_read_from(&dbits, (char *)buffer + decoded_length + 4, nbBytes);
speex_decode_int(dec_state, &dbits, output_buffer);

decoded_length += nbBytes + 4;
int len = (max_buffer_size >= dec_frame_size * (i + 1)) ?
dec_frame_size : max_buffer_size - dec_frame_size * i;
memcpy(output + dec_frame_size * i, output_buffer, len * sizeof(short));
}
return dec_frame_size * i;
}
```

main.cpp 主程序

```
#include "voice.h"
#include <cstdio>

#define FRAME_SIZE 160
#define HEAD_SIZE 44

int main(int argc, char **argv) {

char head[HEAD_SIZE];
short in[FRAME_SIZE];
char encoded[FRAME_SIZE * 2];
short decoded[FRAME_SIZE];

size_t read_count;
size_t encoded_count;
size_t decoded_count;

FILE *fp = fopen("female.wav", "r");
FILE *fp2 = fopen("encoded", "w");
FILE *fp3 = fopen("decoded.wav", "w");
//把wav的头信息写到解压后的文件中去，压缩和解压都是对纯语音数据进行操作的
fread(head, sizeof(char), HEAD_SIZE, fp);
fwrite(head, sizeof(char), HEAD_SIZE, fp3);

voice_encode_init();
voice_decode_init();
while (true) {
read_count = fread(in, sizeof(short), FRAME_SIZE, fp);
if (feof(fp)) {
break;
}
encoded_count = voice_encode(in, read_count, encoded, FRAME_SIZE * 2);
decoded_count = voice_decode(encoded, encoded_count, decoded, FRAME_SIZE);
fwrite(encoded, sizeof(char), encoded_count, fp2);
fwrite(decoded, sizeof(short), decoded_count, fp3);
}
voice_encode_release();
voice_decode_release();

fclose(fp);
fclose(fp2);
```

```
fclose(fp3);

return 0;
}
```

编译运行：

```
g++ main.cpp voice.cpp -lspeex -lm -Wall -o speex

./speex
```

运行后当前目录生成了encoded（压缩后的数据）和decoded.wav文件，另外female.wav是在speex官网下载的一个语音文件。decoded.wav播放起来，感觉和female.wav没太大的区别，反正我是听不出来。压缩效果的话，按我所设置的参数，是将160个short（320个字节）压缩成38个字节，因为除了加密数据外，解压时还需要用到每块加密数据的字节数，在这里是38（int），所以总共占用42个字节，感觉压缩效果还是挺好的。

1. 关于Speex延迟问题（转载）

原文地址：https://blog.csdn.net/lishaoqi_scau/article/details/7548934

这里说的语音延迟问题不是网络延迟，那个取决于网络状况，基本上是固定的，除非换个传输方法

这里说的语音延迟问题造成的原因是这样：

A发送说了十秒钟的话，网络延迟是3秒

那么正常情况B会在3秒后开始听到这句话，并在13秒的时候听完

但如果这时候在第8秒的时候，B的网络卡了1秒（这种情况出现很正常）

那么A说的后面5秒的内容，B会在9~14秒听到

那么这里问题就出来了，如果多卡几次，B听到的内容延迟就会越来越大，缓冲区里面的数据也会越来越多

但是后面收到的数据又必须等到之前收到的数据被播放完以后再播放

所以结果就是延迟会越来越长

那么解决这个问题有下面这些办法

1、因为我现在的这款软件本来就是采用中转式的传输，本来就延迟很慢，很难满足正常通话要求，干脆换成对讲机的形式，就不会有这种情况出现了，按住一个键说话，松开话就被发送出去了，这样本来就是异步的

JS：我还是觉得对讲机不太友好，争取努力解决延迟问题，实在不行的话作为最后的选择吧

2、丢弃掉那些延迟的包，就比如说刚才的问题，B在9秒同时收到了A在5秒和6秒说的内容，这时候直接把5秒的包丢了，播放6秒的内容，用这种方法来赶上对方的说话速度

JS:这种方法固然能解决延迟越来越长的问题，但问题是某些内容被丢弃了，用户体验会很差，老是莫名其妙少了一句话，会让人抓狂的

3、如果积累了很多过多的包，则不播放那些没声音的包。这个方法就是利用人说话的空隙时间，接收方收到了过多的包，则说明出现了网络延迟的问题，这时候去分析包，如果没有声音，就干脆不播放直接丢弃，去播放后面的包，以此来赶上说话放的速度。

上面的方法中我最终选择了第三种，因为首先不会影响用户体验，只丢弃那些没声音的包来空出时间，利用对方不说话那段时间把速度赶上来

谁会无止境的说上几个小时呢是吧

但也有弊端，就是背景太过嘈杂的话，就不好分辨了，无法得知对方对方是不是在说话，但这个问题暂时不考虑吧

最开始我是想去把包解码然后分析wave数据，求这个包的平均值，如果平均值低于某个零界则认为是无声包，丢弃

要做到这个功能其实还挺简单的，因为wave数据还是很好看懂的

不过后来找到了更好的办法，那就是speex本身提供的 静音检测VAD 这个选项来做

静音检测（VAD）将检测被编码的音频数据是语音还是静音或背景噪声。这个特性在用变比特率（VBR）进行编码是总是开启的，所以选项设置只对非变比特率（VBR）起作用。在这种情况下，Speex检测非语音周期并对用足够的比特数重新生成的背景噪声进行编码。这个叫"舒适噪声生成（CNG）"。

```
int dtx = 1;
speex_encoder_ctl(state, SPEEX_SET_VAD, &dtx);
```

我跟踪看了一下，不开启这个选项的时候，每个包都是固定大小，如果开启的话，有的包会是15字节，有的则只有2字节

所以我想当积累的过多的包时，直接丢弃掉只有2字节的包，当然现在还是在理论阶段，能不能成功还得试验

另外还有两个与此相关的功能 变比特率（VBR）和 非连续传输（DTX）

变比特率是比较重要的功能，默认情况下speex压缩后每个包大小都是固定的，如果采用了变比特率那么会根据每个段内实际的语音内容而压缩出不同长度的内容

不连续传输（DTX）是静音检测（VAD）/变比特率（VBR）操作的额外选项，它能够在背景噪声固定时，完全的停止传输。如果是基于文件的操作，由于我们不能停止对文件的写入，会有5个比特被用到这种帧内（相对于250bps）。

如果这三个选项开启，能够极大的减少编码后的数据长度，我测试了一下，大概减少了一倍左右

不过可惜，因为wince上因为性能原因我把浮点运算禁用掉了，而变比特率完全是基于浮点运算的，因此也得禁用掉

不过只开启静音检测和不连续传输的话也能一定量的减少传输量

pc上测试没有问题后我就去看wince平台上表现怎么样了

结果发现根本就完全没反应，加了VAD和DTX特性后和没加效果一样

后来想起来因为M8上的浮点运算能力有限，所以禁用掉了浮点运算，而VBR是基于浮点运算的，因此得一起禁用

而在到网上找了下资料发现VAD和DTX都是基于VBR的

这下难道又进死胡同啦，难道wince平台上就没法使用VAD特性?

我重新把speex说明书上面关于CPU性能优化那段拿了出来好好看了一下

The single that will affect the CPU usage of Speex the most is whether it is compiled for floating point or fixed-point. If your

CPU/DSP does not have a floating-point unit FPU, then compiling as fixed-point will be orders of magnitudes faster. If there

is an FPU present, then it is important to test which version is faster. On the x86 architecture, floating-point is generally

faster, but not always. To compile Speex as fixed-point, you need to pass –fixed-point to the configure script or define the

FIXED_POINT macro for the compiler. As of 1.2beta3, it is now possible to disable the floating-point compatibility API,

which means that your code can link without a float emulation library. To do that configure with –disable-float-api or define

the DISABLE_FLOAT_API macro. Until the VBR feature is ported to fixed-point, you will also need to configure with

–disable-vbr or define DISABLE_VBR.

这才想起来好像当时在编译speex1.2rc1版时确实看到一个宏定义叫fixed-point，当时也没在意

说不定以M8上这么强大的CPU运行定点数也完全够呢

于是把DISABLE_FLOAT_API和DISABLE_VBR特性删掉以后再次编译了一遍libspeex

然后再次运行wince上的程序，发现速度很快，几乎和禁用掉浮点运算速度差不多

而且因为有了VBR的特性，VAD和DTX都运行得很好

我拿了一个826k 19秒的一个wav文件做测试，因为为了测试静音检测功能，所以19秒中只稍微讲了几句话，其他都是没有声音

统一都采用了speex_uwb_mode的压缩方式

平台 speex版本 不开启任何特性 不开启任何特性 开启VAD和DTX 开启VAD和DTX 开启VAD和DTX和VBR 开启VAD和DTX和VBR

质量8 质量2 质量8 质量2 质量8 质量2

PC 1.0.5 50K 17K 32K 12K 24K 24K

WINCE 1.2rc1 40K 16K 19K 8K 16K 16k

这里测试结果可以看出，不管是高质量还是低质量，VAD和DTX这两个属性都可以减少压缩后的数据量

而VBR这个属性在高品种的时候可以减少数据量，但低品质的时候反而增加了数据量

而且在wince上加入了VBR以后运行速度明显要慢一两秒

最后再汗一个800K压缩到8K，压缩了100倍，而解压后仍然很清晰，技术的力量真强大啊

1. Android下jni实现speex编解码（转载）

原文地址：https://www.cnblogs.com/xyzlmn/p/3168095.html

1、去Speex官网下载最新Speex源码。

2、创建新的android工程，并创建jni文件夹。

3、把speex源码目录下的libspeex和include目录及其子目录文件全部拷贝到$project/jni目录下。

4、在jni目录下新增Android.mk文件，编辑内容如下：

```
LOCAL_PATH := $(call my-dir)


include $(CLEAR_VARS)


LOCAL_MODULE := libspeex
LOCAL_CFLAGS = -DFIXED_POINT -DUSE_KISS_FFT -DEXPORT="" -UHAVE_CONFIG_H
LOCAL_C_INCLUDES := $(LOCAL_PATH)/include


LOCAL_SRC_FILES :=\
libspeex/bits.c \
libspeex/buffer.c \
libspeex/cb_search.c \
libspeex/exc_10_16_table.c \
libspeex/exc_10_32_table.c \
libspeex/exc_20_32_table.c \
libspeex/exc_5_256_table.c \
libspeex/exc_5_64_table.c \
libspeex/exc_8_128_table.c \
libspeex/fftwrap.c \
libspeex/filterbank.c \
```

```
libspeex/filters.c \
libspeex/gain_table.c \
libspeex/gain_table_lbr.c \
libspeex/hexc_10_32_table.c \
libspeex/hexc_table.c \
libspeex/high_lsp_tables.c \
libspeex/jitter.c \
libspeex/kiss_fft.c \
libspeex/kiss_fftr.c \
libspeex/lpc.c \
libspeex/lsp.c \
libspeex/lsp_tables_nb.c \
libspeex/ltp.c \
libspeex/mdf.c \
libspeex/modes.c \
libspeex/modes_wb.c \
libspeex/nb_celp.c \
libspeex/preprocess.c \
libspeex/quant_lsp.c \
libspeex/resample.c \
libspeex/sb_celp.c \
libspeex/scal.c \
libspeex/smallft.c \
libspeex/speex.c \
libspeex/speex_callbacks.c \
libspeex/speex_header.c \
libspeex/stereo.c \
libspeex/vbr.c \
libspeex/vq.c \
libspeex/window.c \
speex_jni.cpp \


include $(BUILD_SHARED_LIBRARY)
```

5.在jni目录下新增Application.mk文件，编辑内容如下

APP_ABI := armeabi armeabi-v7a

6.在$project/jni/include/speex/目录下新增speex_config_types.h文件，编辑内容如下

```
#ifndef __SPEEX_TYPES_H__
#define __SPEEX_TYPES_H__
typedef short spx_int16_t;
```

```
typedef unsigned short spx_uint16_t;

typedef int spx_int32_t;

typedef unsigned int spx_uint32_t;

#endif
```

7.创建JNI包装类speex_jni.cpp，用来调用Speex中的C代码函数，编辑内容如下

```
#include

#include
#include

#include

static int codec_open = 0;

static int dec_frame_size;
static int enc_frame_size;

static SpeexBits ebits, dbits;
void *enc_state;
void *dec_state;

static JavaVM *gJavaVM;

extern "C"
JNIEXPORT jint JNICALL Java_com_trunkbow_speextest_Speex_open
(JNIEnv *env, jobject obj, jint compression) {
int tmp;

if (codec_open++ != 0)
return (jint)0;

speex_bits_init(&ebits);
speex_bits_init(&dbits);

enc_state = speex_encoder_init(&speex_nb_mode);
dec_state = speex_decoder_init(&speex_nb_mode);
tmp = compression;
speex_encoder_ctl(enc_state, SPEEX_SET_QUALITY, &tmp);
speex_encoder_ctl(enc_state, SPEEX_GET_FRAME_SIZE, &enc_frame_size);
speex_decoder_ctl(dec_state, SPEEX_GET_FRAME_SIZE, &dec_frame_size);
```

```
return (jint)0;
}

extern "C"
JNIEXPORT jint Java_com_trunkbow_speextest_Speex_encode
(JNIEnv *env, jobject obj, jshortArray lin, jint offset, jbyteArray encoded, jint size) {

jshort buffer[enc_frame_size];
jbyte output_buffer[enc_frame_size];
int nsamples = (size-1)/enc_frame_size + 1;
int i, tot_bytes = 0;

if (!codec_open)
return 0;

speex_bits_reset(&ebits);

for (i = 0; i < nsamples; i++) {
env->GetShortArrayRegion(lin, offset + i*enc_frame_size, enc_frame_size, buffer);
speex_encode_int(enc_state, buffer, &ebits);
}
//env->GetShortArrayRegion(lin, offset, enc_frame_size, buffer);
//speex_encode_int(enc_state, buffer, &ebits);

tot_bytes = speex_bits_write(&ebits, (char *)output_buffer,
enc_frame_size);
env->SetByteArrayRegion(encoded, 0, tot_bytes,
output_buffer);

return (jint)tot_bytes;
}

extern "C"
JNIEXPORT jint JNICALL Java_com_trunkbow_speextest_Speex_decode
(JNIEnv *env, jobject obj, jbyteArray encoded, jshortArray lin, jint size) {

jbyte buffer[dec_frame_size];
jshort output_buffer[dec_frame_size];
jsize encoded_length = size;

if (!codec_open)
return 0;
```

```
env->GetByteArrayRegion(encoded, 0, encoded_length, buffer);
speex_bits_read_from(&dbits, (char *)buffer, encoded_length);
speex_decode_int(dec_state, &dbits, output_buffer);
env->SetShortArrayRegion(lin, 0, dec_frame_size,
output_buffer);

return (jint)dec_frame_size;
}

extern "C"
JNIEXPORT jint JNICALL Java_com_trunkbow_speextest_Speex_getFrameSize
(JNIEnv *env, jobject obj) {

if (!codec_open)
return 0;
return (jint)enc_frame_size;

}

extern "C"
JNIEXPORT void JNICALL Java_com_trunkbow_speextest_Speex_close
(JNIEnv *env, jobject obj) {

if (--codec_open != 0)
return;

speex_bits_destroy(&ebits);
speex_bits_destroy(&dbits);
speex_decoder_destroy(dec_state);
speex_encoder_destroy(enc_state);
}
```

8.在Java层创建Speex工具类，内容如下：

```
package com.trunkbow.speextest;

public class Speex {

private static final int DEFAULT_COMPRESSION = 8;

Speex() {
}
```

```
public void init() {
load();
open(DEFAULT_COMPRESSION);
}

private void load() {
try {
System.loadLibrary("speex");
} catch (Throwable e) {
e.printStackTrace();
}

}

public native int open(int compression);
public native int getFrameSize();
public native int decode(byte encoded[], short lin[], int size);
public native int encode(short lin[], int offset, byte encoded[], int size);
public native void close();
}
```

9、使用cygwin编译，生成so文件。

1. 微信speex语音开发

https://mp.weixin.qq.com/advanced/wiki?t=t=resource/res_main&id=mp1444738727

1. Opus的FEC前向纠错

前向纠错也叫前向纠错(Forward Error Correction，简称FEC)，是增加数据通讯可信度的方法。在单向通讯信道中，一旦错误被发现，其接收器将无权再请求传输。FEC 是利用数据进行传输冗余信息的方法，当传输中出现错误，将允许接收器再建数据。

FEC通过冗余编码的方式将当前帧数据冗余一些到后一帧数据，因此当发现当前帧丢失，可以通过后一帧数据恢复。

int opus_decode (OpusDecoder * st, const unsigned char * data, opus_int32 len, opus_int16 * pcm, int frame_size, int decode_fec)

opus_decode可以通过放空包或者打开的FEC的情况下尝试恢复数据。

当data为NULL时，len应该为0， 此时opus尝试解一帧pcm数据，猜出这一帧数据；

当decode_fec为1时，使用FEC机制，尝试恢复前一帧数据；否则编码当前帧；

罗列以下三种情况并例举伪代码：

1. 前一帧与当前帧均正常， 前一帧数据正常解码；

opus_decode(decoder, previous_frame, frame_size, pcm, pcm_size, 0);

2. 前一帧丢失，当前帧正常，可以通过打开FEC的方式解码当前帧，尝试恢复前一帧；

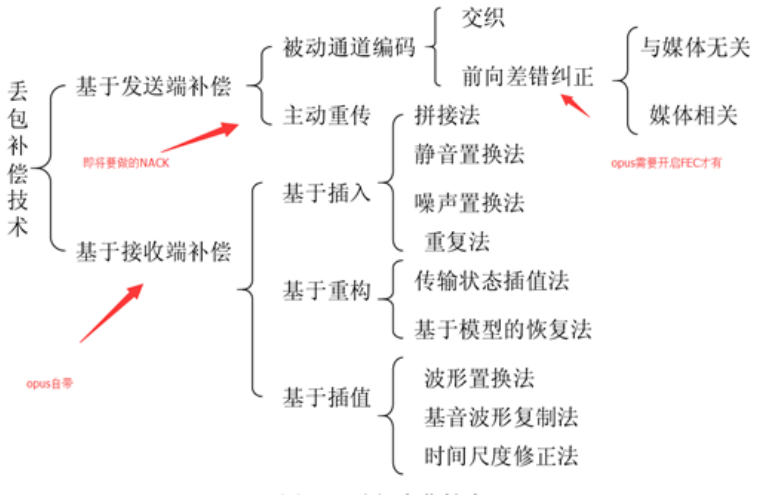
opus_decode(decoder, current_frame, frame_size, pcm, pcm_size, 1);

3. 前一帧与当前帧均丢失，通过放空包的方式，尝试猜出前一帧数据；

opus_decoder(decoder, NULL, 0, pcm, pcm_size, 0);

因此可以通过预先缓存一帧数据的方式进行解码，每次收到一帧数据后，解码前一帧，此时需要考虑上述三种情况决定放入何种数据。

FEC的恢复效果跟预期丢包率设置、还有码率模式设置、还有比特率都有关系。



1. The Speex Codec Manual Speex编解码器手册

The Speex Codec Manual
Version 1.2 Beta 3

Speex编解码器手册
版本1.2 Beta 3

Author: Jean-Marc Valin
翻译：赤勇玄心行天道，AMG

December 8, 2007
2007年12月8日

部分标题。

1. Contents 目录

1. Introduction to Speex Speex介绍

The Speex codec (http://www.speex.org/) exists because there is a need for a speech codec that is open-source and free from software patent royalties. These are essential conditions for being usable in any open-source software. In essence, Speex is to speech what Vorbis is to audio/music. Unlike many other speech codecs, Speex is not designed for mobile phones but rather for packet networks and voice over IP (VoIP) applications. File-based compression is of course also supported.

Speex编解码器（http://www.speex.org/）之所以存在，是因为需要一个开源的免软件专利使用费的语音编解码器。这些都是在任何开源软件中可用的必要条件。从本质上讲，Speex是针对于语音，就像音频压缩格式是针对于音频或音乐一样。与许多其他语音编解码器不同，Speex不是为移动电话而设计的，而是为分组网络和网络电话(VoIP)应用程序而设计的。当然也支持基于文件的压缩。

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).

Speex编解码器设计得非常灵活，并支持很多种语音质量和比特率。对高质量语音的支持也意味着，Speex除了能编码窄带语音（电话质量，8kHz采样频率）外，还能编码宽带语音（16kHz采样频率）。

Designing for VoIP instead of mobile phones means that Speex is robust to lost packets, but not to corrupted ones. This is based on the assumption that in VoIP, packets either arrive unaltered or don't arrive at all. Because Speex is targeted at a wide range of devices, it has modest (adjustable) complexity and a small memory footprint.

为网络电话而不是移动电话而设计，意味着Speex对丢失的数据包鲁棒，但对损坏的数据包不鲁棒。这是基于这样的假设：在网络电话中，数据包要么原封不动地到达，要么根本不到达。由于Speex的目标是在很多种设备上，因此它具有适度（可调）的复杂性和较小的内存占用量。

All the design goals led to the choice of CELP as the encoding technique. One of the main reasons is that CELP has long proved that it could work reliably and scale well to both low bit-rates (e.g. DoD CELP @ 4.8 kbps) and high bit-rates (e.g. G.728 @ 16 kbps).

所有设计目标都导致选择CELP作为编码技术。主要原因之一是CELP在长期以来就证明了它可以可靠地工作，并且可以很好地扩展到低比特率（例如DoD CELP @ 4.8 kbps）和高比特率（例如G.728 @ 16 kbps）。

1. **Getting help 获取帮助**

As for many open source projects, there are many ways to get help with Speex. These include:

对于许多开源项目，有很多途径可以获取到Speex的帮助。它们包括：

- This manual

  本手册

- Other documentation on the Speex website (http://www.speex.org/)

  Speex网站上的其他文档（http://www.speex.org/）

- Mailing list: Discuss any Speex-related topic on speex-dev@xiph.org (not just for developers)

  邮件发送清单：讨论任何有关Speex的话题发送到speex-dev@xiph.org（不仅限于开发人员）

- IRC: The main channel is #speex on irc.freenode.net. Note that due to time differences, it may take a while to get someone, so please be patient.

  IRC：在irc.freenode.net上的主要频道是#speex。请注意，由于时间不同，找人可能需要一段时间，因此请耐心等待。

- Email the author privately at jean-marc.valin@usherbrooke.ca only for private/delicate topics you do not wish to discuss publically.

  给作者私人邮箱jean-marc.valin@usherbrooke.ca发邮件，但仅限于你不想被公开讨论的私人的或精妙的主题。

Before asking for help (mailing list or IRC), **it is important to first read this manual** (OK, so if you made it here it's already a good sign). It is generally considered rude to ask on a mailing list about topics that are clearly detailed in the documentation. On the other hand, it's perfectly OK (and encouraged) to ask for clarifications about something covered in the manual. This manual does not (yet) cover everything about Speex, so everyone is encouraged to ask questions, send comments, feature requests, or just let us know how Speex is being used.

在寻求帮助（邮件列表或IRC）之前，**请务必先阅读本手册**（好的，如果你已经阅读到此处，这已经是一个好兆头）。通常，在邮件列表中询问有关文档中已经明确详细说明的主题的做法是不礼貌的。另一方面，完全可以（并鼓励）询问在手册中没有说清楚的问题。本手册尚未涵盖有关Speex的所有内容，因此鼓励每个人提出问题，发送评论，特性要求，或者只是让我们知道Speex是如何被使用的。

Here are some additional guidelines related to the mailing list. Before reporting bugs in Speex to the list, it is strongly recommended (if possible) to first test whether these bugs can be reproduced using

the speexenc and speexdec (see Section 4) command-line utilities. Bugs reported based on 3rd party code are both harder to find and far too often caused by errors that have nothing to do with Speex.

这里是与邮件列表有关的一些其他准则。在将Speex中的错误报告给邮件列表之前，强烈建议（如果可能）首先测试是否可以使用speexenc和speexdec（请参见第4部分）命令行实用程序来重现这些错误。

1. About this document 关于本文档

This document is divided in the following way. Section 2 describes the different Speex features and defines many basic terms that are used throughoutthis manual. Section 4 documentsthe standard command-line tools providedin the Speex distribution. Section 5 includes detailed instructions about programming using the libspeex API. Section 7 has some information related to Speex and standards.

本文档按以下方式划分。第2章介绍了Speex的不同特性，并定义了本手册中使用的许多基本术语。第4章介绍了Speex发行版中提供的标准命令行工具。第5章包含有关使用libspeex API进行编程的详细说明。第7章提供了一些与Speex和标准有关的信息。

The three last sections describe the algorithms used in Speex. These sections require signal processing knowledge, but are not required for merely using Speex. They are intended for people who want to understand how Speex really works and/or want to do research based on Speex. Section 8 explains the general idea behind CELP, while sections 9 and 10 are specific to Speex.

最后三个章节描述了Speex中使用的算法。这些章节需要信号处理知识，但仅仅是使用Speex则不需要这些知识。它们适用于希望了解Speex的工作原理和/或希望基于Speex进行研究的人员。第8章解释了CELP背后的一般思想，而第9和10章则专门针对Speex。

1. Codec description 编解码器描述

This section describes Speex and its features into more details

本章深入详细介绍Speex和它的特性。

1. Concepts 概念

Before introducing all the Speex features, here are some concepts in speech coding that help better understand the rest of the manual. Although some are general concepts in speech/audio processing, others are specific to Speex.

在介绍所有的Speex特性之前，这里有语音编码的一些概念，可以有助于我们更好地理解本手册的其它部分。虽然一些概念在语音/音频处理过程中是常见的，但是也有一些是Speex特有的。

1. Sampling rate 采样频率

The sampling rate expressed in Hertz (Hz) is the number of samples taken from a signal per second. For a sampling rate of Fs kHz, the highest frequency that can be represented is equal to Fs/2 kHz (Fs/2 is known as the Nyquist frequency). This is a fundamental property in signal processing and is described by the sampling theorem. Speex is mainly designed for three different sampling rates: 8 kHz, 16 kHz, and 32 kHz. These are respectively refered to as narrowband, wideband and ultra-wideband.

采样频率就是每秒钟从信号中采样的样本数量，以赫兹（Hz）为单位。相对于Fs kHz的采样频率而言，其最高的频率可以达到Fs/2 kHz（Fs/2也被称为奈奎斯特频率）。这是在信号处理中的一个基本属性，并通过采样定理说明。Speex主要被设计用于三种不同的采样率：8kHz，16kHz和32kHz。它们分别被称为窄带，宽带和超宽带。

1. Bit-rate 比特率

When encoding a speech signal, the bit-rate is defined as the number of bits per unit of time required to encode the speech. It is measured in bits per second (bps), or generally kilobits per second. It is important to make the distinction between kilobits per second (kbps) and kilobytes per second (kBps).

在对语音信号编码时，比特率被定义为单位时间内所需要的比特数。它是以每秒比特位数（bps）来测量的，或者一般为每秒千比特位数（kbps）。在每秒千比特位数（kbps）和每秒千字节数（kBps）之间进行区分是非常重要的。

1. Quality（variable） 质量（可变的）

Speex is a lossy codec, which means that it archives compression at the expense of fidelity of the input speech signal. Unlike some other speech codecs, it is possible to control the tradeoff made between quality and bit-rate. The Speex encoding process is controlled most of the time by a quality parameter that ranges from 0 to 10. In constant bit-rate (CBR) operation, the quality parameter is an integer, while for variable bit-rate (VBR), the parameter is a float.

Speex是一种有损编解码器，这意味着它的存档压缩是以语音输入信号的保真度为代价的。不像一些其他的语音编解码器，它会尽可能的去控制质量和比特率之间的平衡。在大多数时间，Speex的编码处理是用一个0到10范围内的质量参数来控制的。在固定比特率（CBR）操作中，质量参数是整型，对于可变比特率（VBR），则参数为浮点型。

1. **Complexity（variable） 复杂度（可变的）**

With Speex, it is possible to vary the complexity allowed for the encoder. This is done by controlling how the search is performed with an integer ranging from 1 to 10 in a way that's similar to the -1 to -9 options to gzip and bzip2 compression utilities. For normal use, the noise level at complexity 1 is between 1 and 2 dB higher than at complexity 10, but the CPU requirements for complexity 10 is about 5 times higher than for complexity 1. In practice, the best trade-off is between complexity 2 and 4, though higher settings are often useful when encoding non-speech sounds like DTMF tones.

在Speex中，它可以允许我们改变编码器的复杂度。用1到10的整数来控制如何执行搜索，就像gzip或bzip2压缩工具的-1至-9选项一样。对于正常使用时，复杂度为1时的噪声等级会比复杂度为10时高1至2dB，但是复杂度为10时对CPU需求是复杂度为1时的5倍。实践证明，最佳的权衡是在复杂度为2至4时，然而较高的复杂度则对非语音进行编码时（如DTMF双音多频音调）较为有用。

1. **Variable Bit-Rate（VBR） 可变比特率（VBR）**

Variable bit-rate (VBR) allows a codec to change its bit-rate dynamically to adapt to the "difficulty" of the audio being encoded. In the example of Speex, sounds like vowels and high-energy transients require a higher bit-rate to achieve good quality, while fricatives (e.g. s,f sounds) can be coded adequately with less bits. For this reason, VBR can achive lower bit-rate for the same quality, or a better quality for a certain bit-rate. Despite its advantages, VBR has two main drawbacks: first, by only specifying quality, there's no guaranty about the final average bit-rate. Second, for some real-time applications like voice over IP (VoIP), what counts is the maximum bit-rate, which must be low enough for the communication channel.

可变比特率（VBR）允许编解码器动态改变比特率以适应音频编码的"难度"。拿Speex来说，听起来像元音和瞬间高音的则需较高比特率来达到较好质量，而摩擦音（如S，F音）则适当用较少的比特位数进行编码。出于这种原因，可变比特率（VBR）可以用较低的比特率(bit-rate)达到固定比特率（bit-rate）同样的质量，或比固定比特率（bit-rate）质量更好。尽管它有优势，但可变比特率（VBR）也有两个主要缺点：第一，只能指定质量，不能保证最终的平均比特率（ABR）；第二，在一些实时应用如IP电话（VoIP）中，尽管拥有高的比特率（bit-rate），但为了适应通信信道还是必须要降低。

1. **Average Bit-Rate（ABR） 平均比特率（ABR）**

Average bit-rate solves one of the problems of VBR, as it dynamically adjusts VBR quality in order to meet a specific target bit-rate. Because the quality/bit-rate is adjusted in real-time (open-loop), the global quality will be slightly lower than that obtained by encoding in VBR with exactly the right quality setting to meet the target average bit-rate.

平均比特率（ABR）解决了在可变比特率（VBR）中的一个问题，就是平均比特率（ABR）通过动态调整可变比特率（VBR）的质量来获得一个特定目标的比特率。由于平均比特率（ABR）是实时（开环）调整质量/比特率(bit-rate)的，所以整体质量会略低于通过变比特率（VBR）设置的接近于目标平均比特率进行编码获得的质量。

1. **Voice Activity Detection（VAD） 语音活动检测（VAD）**

When enabled, voice activity detection detects whether the audio being encoded is speech or silence/background noise. VAD is always implicitly activated when encoding in VBR, so the option is only useful in non-VBR operation. In this case, Speex detects non-speech periods and encode them with just enough bits to reproduce the background noise. This is called "comfort noise generation" (CNG).

当启用语音活动检测（VAD）时，它将检测出被编码的音频是语音还是静音/背景噪声。语音活动检测（VAD）在用可变比特率（VBR）进行编码时总是默认开启的，所以这个选项只能用于非变比特率（VBR）。在这种情况下，Speex可以检测到非语音周期，并对它们用足够的比特位数重新编码成背景噪声。这个就叫"舒适噪声生成（CNG）"。

1. **Discontinuous Transmission（DTX） 非连续传输（DTX）**

Discontinuous transmission is an addition to VAD/VBR operation, that allows to stop transmitting completely when the background noise is stationary. In file-based operation, since we cannot just stop writing to the file, only 5 bits are used for such frames (corresponding to 250 bps).

不连续传输（DTX）是对静音检测（VAD）/变比特率（VBR）操作的一个补充，它能够在背景噪声固定的时候完全停止传输。在基于文件的操作中，由于我们不能停止对文件的写入，所以只会有5个比特被用到这种帧内（相对于250bps）。

1. **Perceptual enhancement 知觉增强**

Perceptual enhancement is a part of the decoder which, when turned on, attempts to reduce the perception of the noise/distortion produced by the encoding/decoding process. In most cases, perceptual enhancement brings the sound further from the original objectively (e.g. considering only SNR), but in the end it still sounds better (subjective improvement).

知觉增强中解码器的一部分，当被开启后，将尝试减少在编码/解码过程中产生的噪音/失真的感知。大多数情况下，知觉增强产生的会和最原始的声音会相差较远（如只考虑信噪比（SNR）），但最终仍然听起来更好（主观改善）。

1. **Latency and algorithmic delay 延迟和算法延迟**

Every speech codec introduces a delay in the transmission. For Speex, this delay is equal to the frame size, plus some amount of "look-ahead" required to process each frame. In narrowband operation (8 kHz), the delay is 30 ms, while for wideband (16 kHz), the delay is 34 ms. These values don't account for the CPU time it takes to encode or decode the frames.

每个语音编解码器在传输过程中都会有延迟。就Speex来说，它的延迟就等于每帧大小，再加上每帧需要处理的一些"预先的"操作。在窄带(8kHz)操作中，延迟大概是30ms，宽带操作中，延迟大概是34ms。这些数据是没有将CPU进行编解码帧的时间计算在内的。

1. **Codec 编解码器**

The main characteristics of Speex can be summarized as follows:

Speex的主要特性可以概括如下：

- Free software/open-source, patent and royalty-free

  开源的自由软件，免专利，免版权

- Integration of narrowband and wideband using an embedded bit-stream

  通过嵌入的比特流来集成的窄带和宽带

- Wide range of bit-rates available (from 2.15 kbps to 44 kbps)

  可用比特率的范围广（bit-rate）（从2.15kbps到44kbps）

- Dynamic bit-rate switching (AMR) and Variable Bit-Rate (VBR) operation

  动态比特率交换（AMR）和可变比特率（VBR）操作

- Voice Activity Detection (VAD, integrated with VBR) and discontinuous transmission (DTX)

  语音活动检测（VAD，和变比特率（VBR）集成）和不连续传输（DTX）

- Variable complexity

  可变复杂度

- Embedded wideband structure (scalable sampling rate)

  嵌入的宽带结构（可变的比特率）

- Ultra-wideband sampling rate at 32 kHz

  32kHz的超宽带采样率

- Intensity stereo encoding option

  强化立体声编码选项

- Fixed-point implementation

  定点执行

1. **Preprocessor 预处理器**

This part refers to the preprocessor module introduced in the 1.1.x branch. The preprocessor is designed to be used on the audio before running the encoder. The preprocessor provides three main functionalities:

这部分涉及1.1.x分支介绍的预处理器模块。预处理器被设计在音频被编码前使用。预处理器提供了三个主要功能：

- noise suppression

  噪音抑制

- automatic gain control (AGC)

  自动增益控制（AGC）

- voice activity detection (VAD)

  语音活动检测（VAD）

The denoiser can be used to reduce the amount of background noise present in the input signal. This provides higher quality speech whether or not the denoised signal is encoded with Speex (or at all). However, when using the denoised signal with the codec, there is an additional benefit. Speech codecs in general (Speex included) tend to perform poorly on noisy input, which tends to amplify the noise. The denoiser greatly reduces this effect.

降噪器是用来减少输入信号中的背景噪音的数量。这样可提供更高质量的语音，即使降噪的信号没有经过Speex编码（或其他编码）也一样。然而，当降噪后的信号与编解码器一起使用时，有一个额外的好处。一般的语音编解码器（也包括Speex）往往在噪音输入方面都表现不佳，通常会放大噪音。而降噪器大大降低了这种影响。

Automatic gain control (AGC) is a feature that deals with the fact that the recording volume may vary by a large amount between different setups. The AGC provides a way to adjust a signal to a reference volume. This is useful for voice over IP because it removes the need for manual adjustment of the microphone gain. A secondary advantage is that by setting the microphone gain to a conservative (low) level, it is easier to avoid clipping.

自动增益控制（AGC）是用来处理不同设备录制的音量有很大变化的情况。它提供了一种方法来调整信号到参考音量。这对IP电话（voice over IP）是非常有用的，因为它避免了需要手动去调整麦克风增益。第二个好处是，将麦克风增益设置为保守(低)级别，可有效避免削波。

The voice activity detector (VAD) provided by the preprocessor is more advanced than the one directly provided in the codec.

预处理器提供的语音活动检测（VAD）比直接在编解码器里提供的更高级。

1. **Adaptive Jitter Buffer 自适应抖动缓冲器**

When transmitting voice (or any content for that matter) over UDP or RTP, packet may be lost, arrive with different delay,or even out of order. The purpose of a jitter buffer is to reorder packets and buffer them long enough (but no longer than necessary) so they can be sent to be decoded.

在用UDP或RTP协议传输声音（或其他相关内容）的时候，数据包可能会丢失、到达延迟不同、乱序到达。自适应抖动缓冲器的目的就是将数据包缓冲到足够长（但不超过必要的时间）并对这些包进行重排序，然后才送给解码器进行解码。

1. **Acoustic Echo Canceller 声学回音消除器**

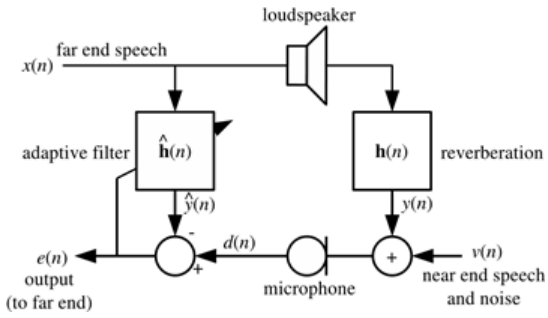


Figure 2.1: Acoustic echo model

图 2.1：声学回音模型

In any hands-free communication system (Fig. 2.1), speech from the remote end is played in the local loudspeaker, propagates in the room and is captured by the microphone. If the audio captured from the microphone is sent directly to the remote end, then the remote user hears an echo of his voice. An acoustic echo canceller is designed to remove the acoustic echo before it is sent to the remote end. It is important to understand that the echo canceller is meant to improve the quality on the remote end.

在任何免提通信系统中（如图2.1），来自远端的语音会在本地扩音器中进行播放，然后在房间中传播，并被麦克风捕获。如果将这些被麦克风捕获的音频被直接发送给远端，然后远端用户就会听到它自己的声音。声学回音消除器就是在发送给远端用户之前将声学回音消除。重要的是要明白，回音消除是用来提高远端用户接收到的语音质量。

1. **Resampler 重新采样器**

In some cases, it may be useful to convert audio from one sampling rate to another. There are many reasons for that. It can be for mixing streams that have different sampling rates, for supporting sampling rates that the soundcard doesn't support, for transcoding, etc. That's why there is now a resampler that is part of the Speex project. This resampler can be used to convert between any two arbitrary rates (the ratio must only be a rational number) and there is control over the quality/complexity tradeoff.

在一些情况下，会用到将音频从一种采样频率转换到另一种。这会有很多原因。例如将拥有不同采样频率的流进行混合、为了支持声卡不支持的采样频率、代码转换等。这就是为什么现在有一个重新采样器会成为Speex项目的一部分。重新采样器可用于在两种任意频率之间转换（频率必须是有理数），它是基于质量/复杂度进行折中的控制。

1. ## Compiling and Porting 编译和移植

Compiling Speex under UNIX/Linux or any other platform supported by autoconf (e.g. Win32/cygwin) is as easy as typing:

在UNIX/Linux、或任何其他支持autoconf的平台（例如Win32/cygwin）上编译Speex，就像打字一样容易：

% ./configure [options选项]

% make

% make install

The options supported by the Speex configure script are:

Speex配置脚本支持的选项有：

-prefix=<path> Specifies the base path for installing Speex (e.g. /usr)

指定Speex的安装路径（比如/usr）

-enable-shared/-disable-shared Whether to compile shared libraries

是否编译成动态库

-enable-static/-disable-static Whether to compile static libraries

是否编译成静态库

-disable-wideband Disable the wideband part of Speex (typically to save space)

禁用Speex的宽带部分（通常为了节约空间）

-enable-valgrind Enable extra hits for valgrind for debugging purposes (do not use by default)

为了调试启用Valgrind（一款用于内存调试、内存泄漏检测以及检查内存其他问题的工具）的额外的匹配记录（默认不使用）

-enable-sse Enable use of SSE instructions (x86/float only)

启用SSE指令(仅限x86/float)

-enable-fixed-point Compile Speex for a processor that does not have a floating point unit (FPU)

在不支持浮点运算单元（FPU）的处理器上编译Speex

-enable-arm4-asm Enable assembly specific to the ARMv4 architecture (gcc only)

启用ARMv4架构的指令集（仅限gcc）

-enable-arm5e-asm Enable assembly specific to the ARMv5E architecture (gcc only)

启用ARMv5E架构的指令集（仅限gcc）

-enable-fixed-point-debug Use only for debugging the fixed-point code (very slow)

仅用于调试定点执行代码（非常慢）

-enable-epic-48k Enable a special (and non-compatible) 4.8 kbps narrowband mode (broken in 1.1.x and 1.2beta)

启用特别的（和不兼容的）4.8kbps窄宽模式（1.1.x和1.2beta中不支持）

-enable-ti-c55x Enable support for the TI C5x family

启用对TI C5x系列的支持

-enable-blackfin-asm Enable assembly specific to the Blackfin DSP architecture (gcc only)

启用Blackfin DSP架构的指令集（仅限gcc）

-enable-vorbis-psycho Make the encoder use the Vorbis psycho-acoustic model. This is very experimental and may be removed in the future.

启用编码器的Vorbis心理声学模型。这是非常实验性的，以后可能会被移除。

1. **Platforms 平台**

Speex is known to compile and work on a large number of architectures, both floating-point and fixed-point. In general, any architecture that can natively compute the multiplication of two signed 16-bit numbers (32-bit result) and runs at a sufficient clock rate (architecture-dependent)is capable of running Speex. Architectures on which Speex is known to work (it probably works on many others) are:

- x86 & x86-64
- Power
- SPARC
- ARM
- Blackfin
- Coldfire (68k family)
- TI C54xx & C55xx
- TI C6xxx
- TriMedia (experimental)

Operating systems on top of which Speex is known to work include (it probably works on many others):

- Linux
- µClinux
- MacOS X
- BSD
- Other UNIX/POSIX variants
- Symbian

The source code directory include additional information for compiling on certain architectures or operating systems in README.xxx files.

1. **Porting and Optimising 移植和优化**

Here are a few things to consider when porting or optimising Speex for a new platform or an existing one.

1. **CPU optimisation CPU优化**

The single that will affect the CPU usage of Speex the most is whether it is compiled for floating point or fixed-point. If your CPU/DSP does not have a floating-point unit FPU, then compiling as fixed-point will be orders of magnitudes faster. If there is an FPU present, then it is important to test which version is faster. On the x86 architecture, floating-point is generally faster, but not always. To compile Speex as fixed-point, you need to pass –fixed-point to the configure script or define the FIXED_POINT macro for the compiler. As of 1.2beta3, it is now possible to disable the floating-point compatibility API, which means that your code can link without a float emulation library. To do that configure with –disable-float-api or define the DISABLE_FLOAT_API macro. Until the VBR feature is ported to fixed-point, you will also need to configure with –disable-vbr or define DISABLE_VBR.

Other important things to check on some DSP architectures are:

- Make sure the cache is set to write-back mode
- If the chip has SRAM instead of cache, make sure as much code and data are in SRAM, rather than in RAM

If you are going to be writing assembly, then the following functions are usually the first ones you should consider optimising:

- filter_mem16()
- iir_mem16()
- vq_nbest()
- pitch_xcorr()
- interp_pitch()

The filtering functions filter_mem16() and iir_mem16() are implemented in the direct form II transposed (DF2T). However, for architectures based on multiply-accumulate (MAC), DF2T requires frequent reload of the accumulator, which can make the code very slow. For these architectures (e.g. Blackfin and Coldfire), a better approach is to implement those functions as direct form I (DF1), which is easier to express in terms of MAC. When doing that however, it is important to make sure that the DF1 implementation still behaves like the original DF2T behaviour when it comes to filter values. This is necessary because the filter is time-varryingand must compute exactly the same value (not counting machine rounding) on any encoder or decoder.

1. **Memory optimisation 内存优化**

Memory optimisation is mainly something that should be considered for small embedded platforms. For PCs, Speex is already so tiny that it's just not worth doing any of the things suggested here. There are several ways to reduce the memory usage of Speex, both in terms of code size and data size. For optimising code size, the trick is to first remove features you do not need. Some examples of things that can easily be disabled if you don't need them are:

- Wideband support (–disable-wideband)
- Support for stereo (removing stereo.c)
- VBR support (–disable-vbr or DISABLE_VBR)
- Static codebooks that are not needed for the bit-rates you are using (*_table.c files)

Speex also has several methodsfor allocatingtemporaryarrays. When using a compilerthat supports C99 properly(as of 2007, Microsoft compilers don't, but gcc does), it is best to define VAR_ARRAYS. That makes use of the variable-size array feature of C99. The next best is to define USE_ALLOCA so that Speex can use alloca() to allocate the temporary arrays. Note that on many systems, alloca() is buggy so it may not work. If none of VAR_ARRAYS and USE_ALLOCA are defined, then Speex falls back to allocating a large "scratch space" and doing its own internal allocation. The main disadvantage of this solution is that it is wasteful. It needs to allocate enough stack for the worst case scenario (worst bit-rate, highest complexity setting, ...) and by default, the memory isn't shared between multiple encoder/decoder states. Still, if the "manual" allocation is the only option left, there are a few things that can be improved. By overriding the speex_alloc_scratch() call in os_support.h, it is possible to always return the same memory area for all states . In addition to that, by redefining the NB_ENC_STACK and NB_DEC_STACK (or similar for wideband), it is possible to only allocate memory for a scenario that is known in advange. In this case, it is important to measure the amount of memory required for the specific sampling rate, bit-rate and complexity level being used.

1. Command-line encoder/decoder 命令行的编码器/解码器

The base Speex distribution includes a command-line encoder (*speexenc*) and decoder (*speexdec*). Those tools produce and read Speex files encapsulated in the Ogg container. Although it is possible to encapsulate Speex in any container, Ogg is the recommended container for files. This section describes how to use the command line tools for Speex files in Ogg.

1. **speexenc**

The *speexenc* utility is used to create Speex files from raw PCM or wave files. It can be used by calling:

```
speexenc [options] input_file output_file
```

The value '-' for input_file or output_file corresponds respectively to stdin and stdout. The valid options are:

–narrowband (-n) Tell Speex to treat the input as narrowband (8 kHz). This is the default

–wideband (-w) Tell Speex to treat the input as wideband (16 kHz)

–ultra-wideband (-u) Tell Speex to treat the input as "ultra-wideband" (32 kHz)

–quality n Set the encoding quality (0-10), default is 8

–bitrate n Encoding bit-rate (use bit-rate n or lower)

–vbr Enable VBR (Variable Bit-Rate), disabled by default

–abr n Enable ABR (Average Bit-Rate) at n kbps, disabled by default

–vad Enable VAD (Voice Activity Detection), disabled by default

–dtx Enable DTX (Discontinuous Transmission), disabled by default

–nframes n Pack n frames in each Ogg packet (this saves space at low bit-rates)

–comp n Set encoding speed/quality tradeoff. The higher the value of n, the slower the encoding (default is 3)

-V Verbose operation, print bit-rate currently in use

–help (-h) Print the help

–version (-v) Print version information

1. **Speex comments**

–comment Add the given string as an extra comment. This may be used multiple times.

–author Author of this track.

–title Title for this track.

1. **Raw input options**

–rate n Sampling rate for raw input

–stereo Consider raw input as stereo

–le Raw input is little-endian

–be Raw input is big-endian

–8bit Raw input is 8-bit unsigned

–16bit Raw input is 16-bit signed

1. **speexdec**

The *speexdec* utility is used to decode Speex files and can be used by calling:

```
speexdec [options] speex_file [output_file]
```

The value '-' for input_file or output_file corresponds respectively to stdin and stdout. Also, when no output_file is specified, the file is played to the soundcard. The valid options are:

–enh enable post-filter (default)

–no-enh disable post-filter

–force-nb Force decoding in narrowband

–force-wb Force decoding in wideband

–force-uwb Force decoding in ultra-wideband

–mono Force decoding in mono

–stereo Force decoding in stereo

–rate n Force decoding at n Hz sampling rate

–packet-loss n Simulate n % random packet loss

-V Verbose operation, print bit-rate currently in use

–help (-h) Print the help

–version (-v) Print version information

1. Using the Speex Codec API (*libspeex*) 使用Speex编解码器API (*libspeex*)

The *libspeex* library contains all the functions for encoding and decoding speech with the Speex codec. When linking on a UNIX system, one must add *-lspeex -lm* to the compiler command line. One

important thing to know is that **libspeex calls are reentrant, but not thread-safe**. That means that it is fine to use calls from many threads, but **calls using the same state from multiple threads must be protected by mutexes**. Examples of code can also be found in Appendix A and the complete API documentation is included in the Documentation section of the Speex website (http://www.speex.org/).

libspeex库包括了所有Speex编解码器的语音编码和解码函数。在Linux系统中链接时，必须在编译器命令行中加入-lspeex和-lm选项。需要知道的是，**虽然libspeex的函数调用是可重入的，但不是线程安全的。**这意味着它可以被多线程调用，但是**多线程使用相同的状态需要用互斥锁保护。**附录A中有代码实例，在Speex站点(http://www.speex.org/)的文档部分能下到完整的API文档。

1. **Encoding 编码**

In order to encode speech using Speex, one first needs to:

为了使用Speex进行语音编码，首先要：

#include <speex/speex.h>

Then in the code, a Speex bit-packing struct must be declared, along with a Speex encoder state:

然后在代码中，必须要声明一个Speex比特包结构体，和一个Speex编码器状态一起声明：

SpeexBits bits;

void *enc_state;

The two are initialized by:

这两个初始化如下：

speex_bits_init(&bits);

enc_state = speex_encoder_init(&speex_nb_mode);

For wideband coding, *speex_nb_mode* will be replaced by *speex_wb_mode*. In most cases, you will need to know the frame size used at the sampling rate you are using. You can get that value in the *frame_size* variable (expressed in **samples**, not bytes) with:

对于宽带编码，将*speex_nb_mode*替换为*speex_wb_mode*。在大多数情况中，你将需要知道你所使用的采样频率的帧大小。你可以用如下方法获取该值到*frame_size*变量（表示为**采样个数**，不是字节个数）中：

speex_encoder_ctl(enc_state,SPEEX_GET_FRAME_SIZE,&frame_size);

In practice, *frame_size* will correspond to 20 ms when using 8, 16, or 32 kHz sampling rate. There are many parameters that can be set for the Speex encoder, but the most useful one is the quality parameter that controls the quality vs bit-rate tradeoff. This is set by:

实际上，当使用8、16或32kHz采样频率的时候，*frame_size*将对应于20ms。Speex编码器有很多参数可以设置，但是其中最有用的一个是质量参数，它控制着质量和比特率的权衡，这个设置如下：

speex_encoder_ctl(enc_state,SPEEX_SET_QUALITY,&quality);

where *quality* is an integer value ranging from 0 to 10 (inclusively). The mapping between quality and bit-rate is described in Table 9.2 for narrowband.

*quality*是一个从0到10（包含10）范围的整数值，窄带（narrowband）的质量和比特率（bit-rate）的对应关系如表9.2所示。

Once the initialization is done, for every input frame:

一旦初始化完成后，对于每个输入帧：

speex_bits_reset(&bits);

    speex_encode_int(enc_state, input_frame, &bits);

    nbBytes = speex_bits_write(&bits, byte_ptr, MAX_NB_BYTES);

where *input_frame* is a *(short *)* pointing to the beginning of a speech frame, *byte_ptr* is a *(char *)* where the encoded frame will be written, *MAX_NB_BYTES* is the maximum number of bytes that can be written to *byte_ptr* without causing an overflow and *nbBytes* is the number of bytes actually written to *byte_ptr* (the encoded size in bytes). Before calling speex_bits_write, it is possible to find the number of bytes that need to be written by calling speex_bits_nbytes(&bits), which returns a number of bytes.

*input_frame*是一个*(short *)*指针，指向一个语音帧的开始，*byte_ptr*是一个*(char *)*指针，已编码帧将写入进去，*MAX_NB_BYTES*是写入到*byte_ptr*不会造成溢出的最大字节数，并且*nbBytes*是实际上写入到*byte_ptr*的字节数（就是已编码的字节长度）。在调用speex_bits_write()之前，可以通过调用speex_bits_nbytes(&bits)来知道需要被写入多少个字节，这个函数将返回一个字节数。

It is still possible to use the *speex_encode()* function, which takes a *(float *)* for the audio. However, this would make an eventual port to an FPU-less platform (like ARM) more complicated. Internally, *speex_encode()* and *speex_encode_int()* are processed in the same way. Whether the encoder uses the fixed-point version is only decided by the compile-time flags, not at the API level.

对于拿到*(float *)*的音频，仍然可以使用*speex_encode()*函数。可是，这将使移植到缺少浮点运算单元（FPU）的平台（如ARM）变得更复杂。本质上，*speex_encode()*和*speex_encode_int()*使用相同的方法处理的。编码器是否使用定点版本仅仅是被编译选项决定的，不是在API级别。

After you're done with the encoding, free all resources with:

在你完成编码之后，用以下方式释放所有的资源：

speex_bits_destroy(&bits);

speex_encoder_destroy(enc_state);

That's about it for the encoder.

以上是关于编码器的内容。

1. **Decoding 解码**

In order to decode speech using Speex, you first need to:

为了使用Speex解码语音，你首先需要：

#include <speex/speex.h>

You also need to declare a Speex bit-packing struct

你也需要声明一个Speex比特包结构体

SpeexBits bits;

and a Speex decoder state

和一个Speex解码器状态

void *dec_state;

The two are initialized by:

这两个初始化如下：

speex_bits_init(&bits);

dec_state = speex_decoder_init(&speex_nb_mode);

For wideband decoding, *speex_nb_mode* will be replaced by *speex_wb_mode*. If you need to obtain the size of the frames that will be used by the decoder, you can get that value in the *frame_size* variable (expressed in **samples**, not bytes) with:

对于宽带解码，将*speex_nb_mode*替换为*speex_wb_mode*。如果你需要获得用于解码器的帧大小，你可以用如下方法获取该值到*frame_size*变量（表示为**采样个数**，不是字节个数）中：

speex_decoder_ctl(dec_state, SPEEX_GET_FRAME_SIZE, &frame_size);

There is also a parameter that can be set for the decoder: whether or not to use a perceptual enhancer. This can be set by:

这里也有一个设置解码器的参数：是否使用知觉增强。这个设置如下：

speex_decoder_ctl(dec_state, SPEEX_SET_ENH, &enh);

where *enh* is an int with value 0 to have the enhancer disabled and 1 to have it enabled. As of 1.2-beta1, the default is now to enable the enhancer.

*enh*是一个整数0就会禁用这个增强，整数1就会启用这个增强。从1.2-beta1开始，默认启用这个增强。

Again, once the decoder initialization is done, for every input frame:

再次，一旦解码器初始化完成后，对于每个输入帧：

speex_bits_read_from(&bits, input_bytes, nbBytes);

speex_decode_int(dec_state, &bits, output_frame);

where input_bytes is a *(char *)* containing the bit-stream data received for a frame, *nbBytes* is the size (in bytes) of that bit-stream, and output_frame is a *(short *)* and points to the area where the decoded speech frame will be written. A NULL value as the second argument indicates that we don't have the bits for the current frame. When a frame is lost, the Speex decoder will do its best to "guess" the correct signal.

input_bytes是一个*(char *)*指针，包含接收到的一帧比特流数据，*nbBytes*是这帧比特流数据的长度，output_frame是一个*(short *)*指针，指向的区域将被写入已解码的语音帧。如果一个NULL值作为第二个参数，则表示我们没有当前这帧的比特流。当一帧已经丢失，Speex解码器将尽可能猜测出正确的信号。

As for the encoder, the *speex_decode()* function can still be used, with a *(float *)* as the output for the audio. After you're done with the decoding, free all resources with:

和编码器类似，仍然可以使用speex_decode()函数，获取一个*(float *)*型的音频输出。在你完成解码之后，用以下方式释放所有的资源：

speex_bits_destroy(&bits);

speex_decoder_destroy(dec_state);

1. **Codec Options (speex_*_ctl) 编解码器选项（speex_*_ctl）**

Entities should not be multiplied beyond necessity – William of Ockham.

实体对象不应该超过所必需的 - William of Ockham。

Just because there's an option for it doesn't mean you have to turn it on – me.

仅仅因为有了一个选项，并不意味着你要打开它 - Speex作者。

The Speex encoder and decoder support many options and requests that can be accessed through the speex_encoder_ctl and speex_decoder_ctl functions. These functions are similar to the ioctl system call and their prototypes are:

Speex编码器和解码器支持很多选项和请求，它们可以通过speex_encoder_ctl和speex_decoder_ctl函数访问。这些函数类似于操作系统的ioctl，它们的原型是：

void speex_encoder_ctl(void *encoder, int request, void *ptr);

void speex_decoder_ctl(void *encoder, int request, void *ptr);

Despite those functions, the defaults are usually good for many applications and optional settings should only be used when one understands them and knows that they are needed. A common error is to attempt to set many unnecessary settings.

虽然有这些函数，默认情况下对于大部分应用程序都是好的，仅当懂得它们并知道它们需要多少才去修改这些设置。通常犯的错误就是尝试去设置无益的设置。

Here is a list of the values allowed for the requests. Some only apply to the encoder or the decoder. Because the last argument is of type void *, the _ctl() functions are not type safe, and shoud thus be used with care. The type spx_int32_t is the same as the C99 int32_t type.

这里列出了所有需求的允许值。某些仅仅适用于编码器或者解码器。因为最后一个参数类型是void *，所以_ctl()函数不是类型安全的，需要小心使用。这个spx_int32_t类型相当于C99标准的int32_t类型。

SPEEX_SET_ENH‡ Set perceptual enhancer to on (1) or off (0) (spx_int32_t, default is on)

SPEEX_SET_ENH‡ 设置知觉增强为打开（1）或者关闭（0） （spx_int32_t，默认为打开）

SPEEX_GET_ENH‡ Get perceptual enhancer status (spx_int32_t)

SPEEX_GET_ENH‡ 获取知觉增强状态 （spx_int32_t）

SPEEX_GET_FRAME_SIZE Get the number of samples per frame for the current mode (spx_int32_t)

SPEEX_GET_FRAME_SIZE 获取当前模式下每帧的采样个数 （spx_int32_t）

SPEEX_SET_QUALITY† Set the encoder speech quality (spx_int32_t from 0 to 10, default is 8)

SPEEX_SET_QUALITY† 设置编码器语音质量 （spx_int32_t从0到10，默认为8）

SPEEX_GET_QUALITY† Get the current encoder speech quality (spx_int32_t from 0 to 10)

SPEEX_GET_QUALITY† 获取当前编码器语音质量 （spx_int32_t从0到10）

SPEEX_SET_MODE† Set the mode number, as specified in the RTP spec (spx_int32_t)

SPEEX_SET_MODE† 设置模式编号，指定在RTP规范中 （spx_int32_t）

SPEEX_GET_MODE† Get the current mode number, as specified in the RTP spec (spx_int32_t)

SPEEX_GET_MODE† 获取当前模式编号，指定在RTP规范中 （spx_int32_t）

SPEEX_SET_VBR† Set variable bit-rate (VBR) to on (1) or off (0) (spx_int32_t, default is off)

SPEEX_SET_VBR† 设置动态比特率（VBR）为打开（1）或者关闭（0） （spx_int32_t，默认为关闭）

SPEEX_GET_VBR† Get variable bit-rate (VBR) status (spx_int32_t)

SPEEX_GET_VBR† 获取动态比特率（VBR）状态 （spx_int32_t）

SPEEX_SET_VBR_QUALITY† Set the encoder VBR speech quality (float 0.0 to 10.0, default is 8.0)

SPEEX_SET_VBR_QUALITY† 设置编码器的动态比特率语音质量 （float从0.0到10.0，默认为8.0）

SPEEX_GET_VBR_QUALITY† Get the current encoder VBR speech quality (float 0 to 10)

SPEEX_GET_VBR_QUALITY† 获取当前编码器的动态比特率语音质量 （float从0.0到10.0）

SPEEX_SET_COMPLEXITY† Set the CPU resources allowed for the encoder (spx_int32_t from 1 to 10, default is 2)

SPEEX_SET_COMPLEXITY† 设置编码器允许使用的CPU资源 （spx_int32_t从0到10，默认为2）

SPEEX_GET_COMPLEXITY† Get the CPU resources allowed for the encoder (spx_int32_t from 1 to 10, default is 2)

SPEEX_GET_COMPLEXITY† 获取编码器允许使用的CPU资源 （spx_int32_t从0到10，默认为2）

SPEEX_SET_BITRATE† Set the bit-rate to use the closest value not exceeding the parameter (spx_int32_t in bits per second)

SPEEX_SET_BITRATE† 设置不超过参数设置的最佳比特率 （spx_int32_t，单位每秒比特）

SPEEX_GET_BITRATE Get the current bit-rate in use (spx_int32_t in bits per second)

SPEEX_GET_BITRATE 获取当前使用的比特率 （spx_int32_t，单位每秒比特）

SPEEX_SET_SAMPLING_RATE Set real sampling rate (spx_int32_t in Hz)

SPEEX_SET_SAMPLING_RATE 设置实时采样频率 （spx_int32_t，单位赫兹）

SPEEX_GET_SAMPLING_RATE Get real sampling rate (spx_int32_t in Hz)

SPEEX_GET_SAMPLING_RATE 获取实时采样频率 （spx_int32_t，单位赫兹）

SPEEX_RESET_STATE Reset the encoder/decoder state to its original state, clearing all memories (no argument)

SPEEX_RESET_STATE 重置编码器或者解码器状态为原始状态，清除所有的记忆 （无参数）

SPEEX_SET_VAD† Set voice activity detection (VAD) to on (1) or off (0) (spx_int32_t, default is off)

SPEEX_SET_VAD† 设置语音活动检测（VAD）为打开（1）或者关闭（0） （spx_int32_t，默认为关闭）

SPEEX_GET_VAD† Get voice activity detection (VAD) status (spx_int32_t)

SPEEX_GET_VAD† 获取语音活动检测（VAD）状态 （spx_int32_t）

SPEEX_SET_DTX† Set discontinuous transmission (DTX) to on (1) or off (0) (spx_int32_t, default is off)

SPEEX_SET_DTX† 设置非持续性传输（DTX）为打开（1）或者关闭（0） （spx_int32_t，默认为关闭）

SPEEX_GET_DTX† Get discontinuous transmission (DTX) status (spx_int32_t)

SPEEX_GET_DTX† 获取非持续性传输（DTX）状态 （spx_int32_t）

SPEEX_SET_ABR† Set average bit-rate (ABR) to a value n in bits per second (spx_int32_t in bits per second)

SPEEX_SET_ABR† 设置平均比特率（ABR）的值，单位每秒比特 （spx_int32_t，单位每秒比特）

SPEEX_GET_ABR† Get average bit-rate (ABR) setting (spx_int32_t in bits per second)

SPEEX_GET_ABR† 获取平均比特率（ABR）的值 （spx_int32_t，单位每秒比特）

SPEEX_SET_PLC_TUNING† Tell the encoder to optimize encoding for a certain percentage of packet loss (spx_int32_t in percent)

SPEEX_SET_PLC_TUNING† 告诉编码器对于已确定的丢包率进行优化编码 （spx_int32_t，单位百分比）

SPEEX_GET_PLC_TUNING† Get the current tuning of the encoder for PLC (spx_int32_t in percent)

SPEEX_GET_PLC_TUNING† 获取编码器的包丢失隐藏的当前调整值 （spx_int32_t，单位百分比）

SPEEX_SET_VBR_MAX_BITRATE† Set the maximum bit-rate allowed in VBR operation (spx_int32_t in bits per second)

SPEEX_SET_VBR_MAX_BITRATE† 设置可变比特率允许的最大比特率 （spx_int32_t，单位每秒比特）

SPEEX_GET_VBR_MAX_BITRATE† Get the current maximum bit-rate allowed in VBR operation (spx_int32_t in bits per second)

SPEEX_GET_VBR_MAX_BITRATE† 获取当前可变比特率允许的最大比特率 （spx_int32_t，单位每秒比特）

SPEEX_SET_HIGHPASS Set the high-pass filter on (1) or off (0) (spx_int32_t, default is on)

SPEEX_SET_HIGHPASS 设置高通滤波器为打开（1）或者关闭（0） （spx_int32_t，默认为打开）

SPEEX_GET_HIGHPASS Get the current high-pass filter status (spx_int32_t)

SPEEX_GET_HIGHPASS 获取当前高通滤波器状态 （spx_int32_t）

† applies only to the encoder

† 仅适用于编码器

‡ applies only to the decoder

‡ 仅适用于解码器

1. Mode queries 模式查询

Speex modes have a query system similar to the speex_encoder_ctl and speex_decoder_ctl calls. Since modes are read-only, it is only possible to get information about a particular mode. The function used to do that is:

Speex的模式有一个查询系统，类似于speex_encoder_ctl和speex_decoder_ctl这样的调用。因为模式是只读的，所以它只能获取模式的详细信息。函数用法如下：

void speex_mode_query(SpeexMode *mode, int request, void *ptr);

The admissible values for request are (unless otherwise note, the values are returned through ptr):

允许请求的值为（除非另有说明，返回值都是放入*prt*）：

SPEEX_MODE_FRAME_SIZE Get the frame size (in samples) for the mode

SPEEX_MODE_FRAME_SIZE 获取模式的帧大小（单位采样个数）

SPEEX_SUBMODE_BITRATE Get the bit-rate for a submode number specified through *ptr* (integer in bps).

SPEEX_SUBMODE_BITRATE 获取子模式数量的比特率放入*ptr*（整数，单位bps）

1. **Packing and in-band signalling 封包和带内信号**

Sometimes it is desirable to pack more than one frame per packet (or other basic unit of storage). The proper way to do it is to call speex_encode N times before writing the stream with speex_bits_write. In cases where the number of frames is not determined by an out-of-band mechanism, it is possible to include a terminator code. That terminator consists of the code 15 (decimal) encoded with 5 bits, as shown in Table 9.2. Note that as of version 1.0.2, calling speex_bits_write automatically inserts the terminator so as to fill the last byte. This doesn't involves any overhead and makes sure Speex can always detect when there is no more frame in a packet.

有时我们希望每个包（或其他基本存储单元）打包超过一帧。正确做法是在调用speex_bits_write写入流之前调用N次speex_encode。这种情况下的帧数不是由带外机制决定的，它会包含一个终结码。如表9.2所示，这个终结码是由用5bits编码的Mode 15组成。如果是1.0.2版本需注意，调用speex_bits_write时，为了填充最后字节，它会自动添加终结码。这不会增加开销，并能确保Speex一直检测到包中没有更多帧为止。

It is also possible to send in-band "messages" to the other side. All these messages are encoded as "pseudo-frames" of mode 14 which contain a 4-bit message type code, followed by the message. Table 5.1 lists the available codes, their meaning and the size of the message that follows. Most of these messages are requests that are sent to the encoder or decoder on the other end, which is free to comply or ignore them. By default, all in-band messages are ignored.

当然也可以通过带内"消息"的方法，所有这些消息是作为Mode14的"伪帧"编码的，Mode14包含4bit的消息类型代码。表5.1列出了可用代码的说明和大小，发送给编/解码器的的消息大部分都可随意的被接受或被忽略。默认情况下，所有带内消息都被忽略掉了。

Code	Size (bits)	Content
0	1	Asks decoder to set perceptual enhancement off (0) or on(1)
1	1	Asks (if 1) the encoder to be less "agressive" due to high packet loss
2	4	Asks encoder to switch to mode N
3	4	Asks encoder to switch to mode N for low-band
4	4	Asks encoder to switch to mode N for high-band
5	4	Asks encoder to switch to quality N for VBR
6	4	Request acknowloedge (0=no, 1=all, 2=only for in-band data)
7	4	Asks encoder to set CBR (0), VAD(1), DTX(3), VBR(5),

		VBR+DTX(7)
8	8	Transmit (8-bit) character to the other end
9	8	Intensity stereo information
10	16	Announce maximum bit-rate acceptable (N in bytes/second)
11	16	reserved
12	32	Acknowledge receiving packet N
13	32	reserved
14	64	reserved
15	64	reserved

Table 5.1: In-band signalling codes

表5.1：带内信号代码

Finally, applications may define custom in-band messages using mode 13. The size of the message in bytes is encoded with 5 bits, so that the decoder can skip it if it doesn't know how to interpret it.

最后，一些应用会使用Mode 13自定义带内消息，消息的字节大小是用5bits编码的，所以如果编码器不知道如何解析它就会跳过。

1. Speech Processing API (*libspeexdsp*) 语音处理 API (*libspeexdsp*)

As of version 1.2beta3, the non-codec parts of the Speex package are now in a separate library called libspeexdsp. This library includes the preprocessor, the acoustic echo canceller, the jitter buffer, and the resampler. In a UNIX environment, it can be linked into a program by adding -lspeexdsp -lm to the compiler command line. Just like for libspeex, **libspeexdsp calls are reentrant, but not thread-safe**. That means that it is fine to use calls from many threads, but **calls using the same state from multiple threads must be protected by mutexes**.

从1.2beta3版本开始，Speex包的非编解码器部分现在在单独的库，叫libspeexdsp。这个库包含了预处理器、声学回音消除器、抖动缓冲器、重采样。在UNIX环境下，程序链接时需要给编译器添加-lspeexdsp -lm命令行选项。像libspeex、libspeexdsp的调用是可重入的，但不是线程安全的。这意味着他可以正常的使用多线程调用，但是**多线程使用相同的状态需要用互斥锁保护。**

1. Preprocessor 预处理器

In order to use the Speex preprocessor, you first need to:

#include <speex/speex_preprocess.h>

Then, a preprocessor state can be created as:

SpeexPreprocessState *preprocess_state = speex_preprocess_state_init(frame_size, sampling_rate);

and it is recommended to use the same value for frame_size as is used by the encoder (20 ms).

For each input frame, you need to call:

speex_preprocess_run(preprocess_state, audio_frame);

where audio_frame is used both as input and output. In cases where the output audio is not useful for a certain frame, it is possible to use instead:

speex_preprocess_estimate_update(preprocess_state, audio_frame);

This call will update all the preprocessor internal state variables without computing the output audio, thus saving some CPU cycles.

The behaviour of the preprocessor can be changed using:

speex_preprocess_ctl(preprocess_state, request, ptr);

which is used in the same way as the encoder and decoder equivalent. Options are listed in Section 6.1.1.

The preprocessor state can be destroyed using:

speex_preprocess_state_destroy(preprocess_state);

1. **Preprocessor options 预处理器选项**

As with the codec, the preprocessor also has options that can be controlled using an ioctl()-like call. The available options are:

SPEEX_PREPROCESS_SET_DENOISE Turns denoising on(1) or off(2) (spx_int32_t)

SPEEX_PREPROCESS_GET_DENOISE Get denoising status (spx_int32_t)

SPEEX_PREPROCESS_SET_AGC Turns automatic gain control (AGC) on(1) or off(2) (spx_int32_t)

SPEEX_PREPROCESS_GET_AGC Get AGC status (spx_int32_t)

SPEEX_PREPROCESS_SET_VAD Turns voice activity detector (VAD) on(1) or off(2) (spx_int32_t)

SPEEX_PREPROCESS_GET_VAD Get VAD status (spx_int32_t)

SPEEX_PREPROCESS_SET_AGC_LEVEL

SPEEX_PREPROCESS_GET_AGC_LEVEL

SPEEX_PREPROCESS_SET_DEREVERB Turns reverberation removal on(1) or off(2) (spx_int32_t)

SPEEX_PREPROCESS_GET_DEREVERB Get reverberation removal status (spx_int32_t)

SPEEX_PREPROCESS_SET_DEREVERB_LEVEL Not working yet, do not use

SPEEX_PREPROCESS_GET_DEREVERB_LEVEL Not working yet, do not use

SPEEX_PREPROCESS_SET_DEREVERB_DECAY Not working yet, do not use

SPEEX_PREPROCESS_GET_DEREVERB_DECAY Not working yet, do not use

SPEEX_PREPROCESS_SET_PROB_START

SPEEX_PREPROCESS_GET_PROB_START

SPEEX_PREPROCESS_SET_PROB_CONTINUE

SPEEX_PREPROCESS_GET_PROB_CONTINUE

SPEEX_PREPROCESS_SET_NOISE_SUPPRESS Set maximumattenuationofthe noise in dB (negativespx_int32_t)

SPEEX_PREPROCESS_GET_NOISE_SUPPRESS Get maximumattenuationofthe noise in dB (negativespx_int32_t)

SPEEX_PREPROCESS_SET_ECHO_SUPPRESS Set maximumattenuationof the residual echoin dB (negativespx_int32_t)

SPEEX_PREPROCESS_GET_ECHO_SUPPRESS Set maximumattenuationof the residual echo in dB (negativespx_int32_t)

SPEEX_PREPROCESS_SET_ECHO_SUPPRESS_ACTIVE Set maximum attenuation of the echo in dB when near end is active (negative spx_int32_t)

SPEEX_PREPROCESS_GET_ECHO_SUPPRESS_ACTIVE Set maximum attenuation of the echo in dB when near end is active (negative spx_int32_t)

SPEEX_PREPROCESS_SET_ECHO_STATE Set the associated echo canceller for residual echo suppression (pointer or NULL for no residual echo suppression)

SPEEX_PREPROCESS_GET_ECHO_STATE Get the associated echo canceller (pointer)

1. **Echo Cancellation 回音消除器**

The Speex library now includes an echo cancellation algorithm suitable for Acoustic Echo Cancellation (AEC). In order to use the echo canceller, you first need to

#include <speex/speex_echo.h>

Then, an echo canceller state can be created by:

SpeexEchoState *echo_state = speex_echo_state_init(frame_size, filter_length);

where frame_size is the amount of data (in samples) you want to process at once and filter_length is the length (in samples) of the echo cancelling filter you want to use (also known as tail length). It is recommended to use a frame size in the order of 20 ms (or equal to the codec frame size) and make sure it is easy to perform an FFT of that size (powers of two are better than prime sizes). The recommendedtail length is approximately the third of the room reverberation time. For example, in a small room, reverberation time is in the order of 300 ms, so a tail length of 100 ms is a good choice (800 samples at 8000 Hz sampling rate).

Once the echo canceller state is created, audio can be processed by:

speex_echo_cancellation(echo_state, input_frame, echo_frame, output_frame);

where input_frame is the audio as captured by the microphone, echo_frame is the signal that was played in the

speaker (and needs to be removed) and output_frame is the signal with echo removed.

One important thing to keep in mind is the relationship between input_frame and echo_frame. It is important that, at any time, any echo that is present in the input has already been sent to the echo canceller as echo_frame. In other words, the echo canceller cannot remove a signal that it hasn't yet received. On the other hand, the delay between the input signal and the echo signal must be small enough because otherwise part of the echo cancellation filter is inefficient. In the ideal case, you code would look like:

write_to_soundcard(echo_frame, frame_size); read_from_soundcard(input_frame, frame_size);

speex_echo_cancellation(echo_state, input_frame, echo_frame, output_frame);

If you wish to further reduce the echo present in the signal, you can do so by associating the echo canceller to the preprocessor (see Section 6.1). This is done by calling:

speex_preprocess_ctl(preprocess_state, SPEEX_PREPROCESS_SET_ECHO_STATE,echo_state);

in the initialisation.

As of version 1.2-beta2, there is an alternative, simpler API that can be used instead of speex_echo_cancellation(). When audio capture and playback are handled asynchronously (e.g. in different threads or using the poll() or select() system call), it can be difficult to keep track of what input_frame comes with what echo_frame. Instead, the playback comtext/thread can simply call:

speex_echo_playback(echo_state, echo_frame);

every time an audio frame is played. Then, the capture context/thread calls:

speex_echo_capture(echo_state, input_frame, output_frame);

for every frame captured. Internally, speex_echo_playback() simply buffers the playback frame so it can be used by speex_echo_capture() to call speex_echo_cancel(). A side effect of using this alternate API is that the playback audio is delayed by two frames, which is the normal delay caused by the soundcard. When capture and playback are already synchronised, speex_echo_cancellation() is preferable since it gives better control on the exact input/echo timing.

The echo cancellation state can be destroyed with:

speex_echo_state_destroy(echo_state);

It is also possible to reset the state of the echo canceller so it can be reused without the need to create another state with:

speex_echo_state_reset(echo_state);

1. **Troubleshooting 发现并修理故障**

There are several things that may prevent the echo canceller from working properly. One of them is a bug (or something suboptimal) in the code, but there are many others you should consider first

- Using a different soundcard to do the capture and plaback will not work, regardless of what you may think. The only exception to that is if the two cards can be made to have their sampling clock "locked" on the same clock source. If not, the clocks will always have a small amount of drift, which will prevent the echo canceller from adapting.

- The delay between the record and playback signals must be minimal. Any signal played has to "appear" on the playback (far end) signal slightly before the echo canceller "sees" it in the near end signal, but excessive delay means that part of the filter length is wasted. In the worst situations, the delay is such that it is longer than the filter length, in which case, no echo can be cancelled.

- When it comes to echo tail length (filter length), longer is *not* better. Actually, the longer the tail length, the longer it takes for the filter to adapt. Of course, a tail length that is too short will not cancel enough echo, but the most common problem seen is that people set a very long tail length and then wonder why no echo is being cancelled.

- Non-linear distortion cannot (by definition) be modeled by the linear adaptive filter used in the echo canceller and thus cannot be cancelled. Use good audio gear and avoid saturation/clipping.

Also useful is reading Echo Cancellation Demystified by Alexey Frunze , which explains the fundamental principles of echo cancellation. The details of the algorithm described in the article are different, but the general ideas of echo cancellation through adaptive filters are the same.

As of version 1.2beta2, a new echo_diagnostic.m tool is included in the source distribution. The first step is to define DUMP_ECHO_CANCEL_DATA during the build. This causes the echo canceller to automatically save the near-end, far-end and output signals to files (aec_rec.sw aec_play.sw and aec_out.sw). These are exactly what the AEC receives and outputs. From there, it is necessary to start Octave and type:

echo_diagnostic('aec_rec.sw', 'aec_play.sw', 'aec_diagnostic.sw', 1024);

The value of 1024 is the filter length and can be changed. There will be some (hopefully) useful messages printed and echo cancelled audio will be saved to aec_diagnostic.sw . If even that output is bad (almost no cancellation) then there is probably problem with the playback or recording process.

1. **Jitter Buffer 抖动缓冲器**

The jitter buffer can be enabled by including:

包含头文件可以启用抖动缓冲器：

#include <speex/speex_jitter.h>

and a new jitter buffer state can be initialised by:

并初始化一个新的抖动缓冲器状态：

JitterBuffer * state = jitter_buffer_init(step);

where the step argument is the default time step (in timestamp units) used for adjusting the delay and doing concealment. A value of 1 is always correct, but higher values may be more convenient sometimes. For example, if you are only able to do concealment on 20ms frames, there is no point in the jitter buffer asking you to do it on one sample. Another example is that for video, it makes no sense to adjust the delay by less than a full frame. The value provided can always be changed at a later time.

其中step参数是用于调整延迟和做隐藏的默认时间步长（以时间戳为单位）。值为1始终是正确的，但是有时更高的值会更方便。例如，如果你能在20ms帧上做隐蔽，则抖动缓冲器中的点无需一个一个做。另一个例子是针对视频，对少于一帧的延时进行调整没有意义，后面可以随时改变这个值。

The jitter buffer API is based on the JitterBufferPacket type, which is defined as:

抖动缓冲器API基于JitterBufferPacket类型，定义如下：

typedef struct {

char *data; /* Data bytes contained in the packet 包含在数据包中的字节数据*/

spx_uint32_t len; /* Length of the packet in bytes 数据包长度，单位字节*/

spx_uint32_t timestamp; /* Timestamp for the packet 数据包的时间戳*/

spx_uint32_t span; } JitterBufferPacket; /* Time covered by the packet (timestamp units) 数据包覆盖的时间（单位时间戳）*/

As an example, for audio the timestamp field would be what is obtained from the RTP timestamp field and the span would be the number of samples that are encoded in the packet. For Speex narrowband, span would be 160 if only one frame is included in the packet.

例如，对于音频，时间戳字段将是从RTP时间戳字段获得的值，span参数将是在数据包中已编码的采样数量。对于Speex窄带，如果这个数据包只包含一帧，则span参数将为160。

When a packet arrives, it need to be inserter into the jitter buffer by:

当一个数据包到达时，它将被插入到抖动缓冲器中：

JitterBufferPacket packet;

/* Fill in each field in the packet struct 填充数据包结构体中每一个字段*/

jitter_buffer_put(state, &packet);

When the decoder is ready to decode a packet the packet to be decoded can be obtained by:

当解码器准备解码一个数据包时，可以通过以下方式获得要解码的数据包：

int start_offset;

err = jitter_buffer_get(state, &packet, desired_span, &start_offset);

If jitter_buffer_put() and jitter_buffer_get() are called from different threads, then you need to protect the jitter buffer state with a mutex.

如果jitter_buffer_put()和jitter_buffer_get()函数被不同的线程调用，那么你需要用互斥锁来保护抖动缓冲器状态。

Because the jitter buffer is designed not to use an explicit timer, it needs to be told about the time explicitly. This is done by calling:

因为抖动缓冲器被设计为不使用显式计时器，所以需要明确地告知它时间，通过如下调用实现：

jitter_buffer_tick(state);

This needs to be done periodically in the playing thread. This will be the last jitter buffer call before going to sleep (until more data is played back). In some cases, it may be preferable to use:

这需要在播放线程中定期调用。这是播放线程在休眠之前的最后一个抖动缓冲器调用（直到更多数据被回放）。在一些情况下，如下调用会更好：

jitter_buffer_remaining_span(state, remaining);

The second argument is used to specify that we are still holding data that has not been written to the playback device. For instance, if 256 samples were needed by the soundcard (specified by desired_span), but jitter_buffer_get() returned 320 samples, we would have remaining=64.

第二个参数用指定还没被写入回放设备的仍保留在抖动缓冲器中的数据。比如，若声卡需要256个采样数据（由deaired_span指定），但jitter_buffer_get()函数返回320个采样数据，则remaining=64。

1. **Resampler 重采样器**

As of version 1.2beta2, Speex includes a resampling modules. To make use of the resampler, it is necessary to include its header file:

#include <speex/speex_resampler.h>

For each stream that is to be resampled, it is necessary to create a resampler state with:

SpeexResamplerState *resampler;

resampler = speex_resampler_init(nb_channels, input_rate, output_rate, quality, & err);

where nb_channels is the number of channels that will be used (either interleaved or non-interleaved), input_rate is the sampling rate of the input stream, output_rate is the sampling rate of the output stream and quality is the requested quality setting (0 to 10). The quality parameter is useful for controlling the quality/complexity/latency tradeoff. Using a higher quality setting means less noise/aliasing, a higher complexity and a higher latency. Usually, a quality of 3 is acceptable for most desktop uses and quality 10 is mostly recommended for pro audio work. Quality 0 usually has a decent sound (certainly better than using linear interpolation resampling), but artifacts may be heard.

The actual resampling is performed using

err = speex_resampler_process_int(resampler, channelID, in, &in_length, out, & out_length);

where channelID is the ID of the channel to be processed. For a mono stream, use 0. The in pointer points to the first sample of the input buffer for the selected channel and out points to the first sample of the output. The size of the input and output buffers are specified by in_length and out_length respectively. Upon completion, these values are replaced by the number of samples read and written by the resampler. Unless an error occurs, either all input samples will be read or all output samples will be written to (or both). For floating-point samples, the function speex_resampler_process_float() behaves similarly.

It is also possible to process multiple channels at once. To be continued...

As of version 1.2beta2, Speex includes a resampling modules. To make use of the resampler, it is necessary to include its header file:

#include <speex/speex_resampler.h>

For each stream that is to be resampled, it is necessary to create a resampler state with:

SpeexResamplerState *resampler;

resampler = speex_resampler_init(nb_channels, input_rate, output_rate, quality, &err);

where nb_channels is the number of channels that will be used (either interleaved or non-interleaved), input_rate is the sampling rate of the input stream, output_rate is the sampling rate of the output stream and quality is the requested quality setting (0 to 10). The quality parameter is useful for controlling the quality/complexity/latency tradeoff. Using a higher quality setting means less noise/aliasing, a higher complexity and a higher latency. Usually, a quality of 3 is acceptable for most desktop uses and quality 10 is mostly recommended for pro audio work. Quality 0 usually has a decent sound (certainly better than using linear interpolation resampling), but artifacts may be heard.

The actual resampling is performed using

err = speex_resampler_process_int(resampler, channelID, in, &in_length, out, &out_length);

where channelID is the ID of the channel to be processed. For a mono stream, use 0. The in pointer points to the first sample of the input buffer for the selected channel and out points to the first sample of the output. The size of the input and output buffers are specified by in_length and out_length respectively. Upon completion, these values are replaced by the number of samples read and written by the resampler. Unless an error occurs, either all input samples will be read or all output samples will be written to (or both). For floating-point samples, the function speex_resampler_process_float() behaves similarly.

It is also possible to process multiple channels at once.

1. **Ring Buffer 环形缓冲器**

Put some stuff there...

1. Formats and standards

Speex can encode speech in both narrowband and wideband and provides different bit-rates. However, not all features need to be supported by a certain implementation or device. In order to be called "Speex compatible" (whatever that means), an implementation must implement at least a basic set of features.

At the minimum, all narrowband modes of operation MUST be supported at the decoder. This includes the decoding of a wideband bit-stream by the narrowband decoder(The wideband bit-stream contains

an embedded narrowband bit-stream which can be decoded alone). If present, a wideband decoder MUST be able to decode a narrowband stream, and MAY either be able to decode all wideband modes or be able to decode the embedded narrowband part of all modes (which includes ignoring the high-band bits).

For encoders, at least one narrowband or wideband mode MUST be supported. The main reason why all encoding modes do not have to be supported is that some platforms may not be able to handle the complexity of encoding in some modes.

1. **RTP Payload Format**

The RTP payloaddraft is includedin appendixC and the latest versionis available at http://www.speex.org/drafts/ latest. This draft has been sent (2003/02/26) to the Internet Engineering Task Force (IETF) and will be discussed at the March 18th meeting in San Francisco.

1. **MIME Type**

For now, you should use the MIME type audio/x-speex for Speex-in-Ogg. We will apply for type audio/speex in the near future.

1. **Ogg file format**

Speex bit-streams can be stored in Ogg files. In this case, the first packet of the Ogg file contains the Speex header described in table 7.1. All integer fields in the headers are stored as little-endian. The speex_string field must contain the "Speex " (with 3 trailing spaces), which identifies the bit-stream. The next field, speex_version contains the version of Speex that encoded the file. For now, refer to speex_header.[ch] for more info. The *beginning of stream* (b_o_s) flag is set to 1 for the header. The header packet has packetno=0 and granulepos=0.

The second packet contains the Speex comment header. The format used is the Vorbis comment format described here: http://www.xiph.org/ogg/vorbis/doc/v-comment.html . This packet has packetno=1 and granulepos=0.

The third and subsequent packets each contain one or more (number found in header) Speex frames. These are identified with packetno starting from 2 and the granulepos is the number of the last sample encoded in that packet. The last of these packets has the *end of stream* (e_o_s) flag is set to 1.

---

7 Formatsandstandards

Field	Type	Size
speex_string	char[]	8
speex_version	char[]	20
speex_version_id	int	4
header_size	int	4
rate	int	4
mode	int	4
mode_bitstream_version	int	4
nb_channels	int	4
bitrate	int	4
frame_size	int	4

vbr	int	4
frames_per_packet	int	4
extra_headers	int	4
reserved1	int	4
reserved2	int	4

Table 7.1: Ogg/Speex header packet

1. Introduction to CELP Coding

Do not meddle *in the affairs of poles, for they are subtle and quick to leave the unit circle.*

Speex is based on CELP, which stands for Code Excited Linear Prediction. This section attempts to introduce the principles behind CELP, so if you are already familiar with CELP, you can safely skip to section 9. The CELP technique is based on three ideas:

1. The use of a linear prediction (LP) model to model the vocal tract
2. The use of (adaptive and fixed) codebook entries as input (excitation) of the LP model
3. The search performed in closed-loop in a "perceptually weighted domain"

This section describes the basic ideas behind CELP. This is still a work in progress.

1. Source-Filter Model of Speech Prediction

The source-filter model of speech productionassumes that the vocal cords are the source of spectrally flat sound (the excitation signal), and that the vocal tract acts as a filter to spectrally shape the various sounds of speech. While still an approximation, the model is widely used in speech coding because of its simplicity.Its use is also the reason why most speech codecs (Speex included) perform badly on music signals. The different phonemes can be distinguished by their excitation (source) and spectral shape (filter). Voiced sounds (e.g. vowels) have an excitation signal that is periodic and that can be approximated by an impulse train in the time domain or by regularly-spaced harmonics in the frequency domain. On the other hand, fricatives (such as the "s", "sh" and "f" sounds) have an excitation signal that is similar to white Gaussian noise. So called voice fricatives (such as "z" and "v") have excitation signal composed of an harmonic part and a noisy part.

The source-filter model is usually tied with the use of Linear prediction. The CELP model is based on source-filter model, as can be seen from the CELP decoder illustrated in Figure 8.1.

1. Linear Prediction (LPC)

Linear prediction is at the base of many speech coding techniques, including CELP. The idea behind it is to predict the signal $x[n]$ using a linear combination of its past samples:

$$y[n] = \sum_{i=1}^{N} a_i x[n-i]$$

where $y[n]$ is the linear prediction of $x[n]$. The prediction error is thus given by:

$$e[n] = x[n] - y[n] = x[n] - \sum_{i=1}^{N} a_i x[n-i]$$

The goal of the LPC analysis is to find the best prediction coefficients a_i which minimize the quadratic error function:

$$E = \sum_{n=0}^{L-1} [e[n]]^2 = \sum_{n=0}^{L-1} \left[x[n] - \sum_{i=1}^{N} a_i x[n-i] \right]^2$$

That can be done by making all derivatives $\frac{\partial E}{\partial a_i}$ equal to zero:

$$\frac{\partial E}{\partial a_i}$$

$$= \frac{\partial}{\partial a_i} \sum_{n=0}^{L-1} \left[x[n] - \sum_{i=1}^{N} a_i x[n-i] \right]^2 = 0 \quad i = 1 \ldots N$$

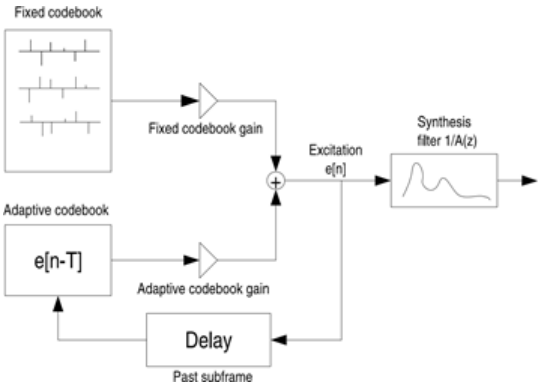


Figure 8.1: The CELP model of speech synthesis (decoder)

For an order N filter, the filter coefficients a_i are found by solving the system $N \times N$ linear system **Ra** = **r**, where

$$\mathbf{R} = \begin{bmatrix} R(0) & R(1) & \cdots & R(N-1) \\ R(1) & R(0) & \cdots & R(N-2) \\ \vdots & \vdots & \ddots & \vdots \\ R(N-1) & R(N-2) & \cdots & R(0) \end{bmatrix}$$

$R(1)$ **r** = □□□ .. □□□□

$R(2)$ □

□ $R(N)$

with $R(m)$, the auto-correlation of the signal $x[n]$, computed as:

$N-1$

$R(m) = \sum x[i]x[i-m]$

$i=0$

Because **R** is Hermitian Toeplitz, the Levinson-Durbin algorithm can be used, making the solution to the problem O N^2 instead of O N^3. Also, it can be proven that all the roots of $A(z)$ are within the unit circle, which means that $1/A(z)$ is always stable. This is in theory; in practice because of finite precision, there are two commonly used techniques to make sure we have a stable filter. First, we multiply $R(0)$ by a number slightly above one (such as 1.0001), which is equivalent to adding noise to the signal. Also, we can apply a window to the auto-correlation, which is equivalent to filtering in the frequency domain, reducing sharp resonances.

1. **Pitch Prediction**

During voiced segments, the speech signal is periodic, so it is possible to take advantage of that property by approximating the excitation signal $e[n]$ by a gain times the past of the excitation:

$e[n] \simeq p[n] = \beta e[n-T]$,

where T is the pitch period,β is the pitch gain. We call that long-term prediction since the excitation is predicted from $e[n-T]$ with $T \gg N$.

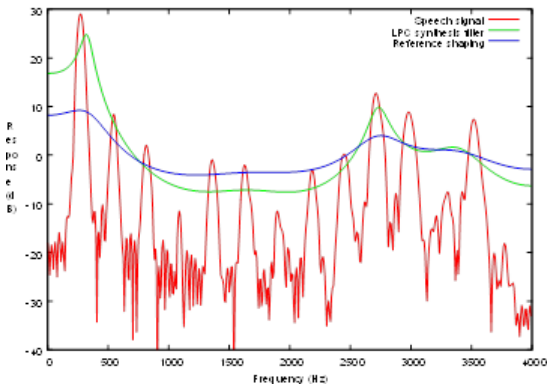


Figure 8.2: Standard noise shaping in CELP. Arbitrary y-axis offset.

1. **Innovation Codebook**

The final excitation $e[n]$ will be the sum of the pitch prediction and an *innovation* signal $c[n]$ taken from a fixed codebook, hence the name *Code* Excited Linear Prediction. The final excitation is given by

$e[n] = p[n] + c[n] = \beta e[n-T] + c[n]$.

The quantization of $c[n]$ is where most of the bits in a CELP codec are allocated. It represents the information that couldn't be obtained either from linear prediction or pitch prediction. In the z-domain we can represent the final signal $X(z)$ as

$C(z)$

$X(z) = A(z)(1-\beta z-T)$

1. **Noise Weighting**

Most (if not all) modern audio codecs attempt to "shape" the noise so that it appears mostly in the frequency regions where the ear cannot detect it. For example, the ear is more tolerant to noise in parts of the spectrum that are louder and *vice versa*. In order to maximize speech quality, CELP codecs minimize the mean square of the error (noise) in the perceptually weighted domain. This means that a perceptual noise weighting filter $W(z)$ is applied to the error signal in the encoder. In most CELP codecs, $W(z)$ is a pole-zero weighting filter derived from the linear prediction coefficients (LPC), generally using bandwidth expansion. Let the spectral envelope be represented by the synthesis filter $1/A(z)$, CELP codecs typically derive the noise weighting filter as

$$W(z) = \frac{A(z/\gamma_1)}{}, \quad (8.1)$$

$A(z/\gamma_2)$

where $\gamma_1 = 0.9$ and $\gamma_2 = 0.6$ in the Speex reference implementation. If a filter $A(z)$ has (complex) poles at p_i in the z-plane, the filter $A(z/\gamma)$ will have its poles at $p'_i = \gamma p_i$, making it a flatter version of $A(z)$.

The weighting filter is applied to the error signal used to optimize the codebook search through analysis-by-synthesis (AbS). This results in a spectral shape of the noise that tends towards $1/W(z)$. While the simplicity of the model has been an important reason for the success of CELP, it remains that $W(z)$ is a very rough approximation for the perceptually optimal noise weighting function. Fig. 8.2 illustrates the noise shaping that results from Eq. 8.1. Throughout this paper, we refer to $W(z)$ as the noise weighting filter and to $1/W(z)$ as the noise shaping filter (or curve).

1. **Analysis-by-Synthesis**

One of the main principles behind CELP is called Analysis-by-Synthesis (AbS), meaning that the encoding (analysis) is performed by perceptually optimising the decoded (synthesis) signal in a closed loop. In theory, the best CELP stream would be produced by trying all possible bit combinations and selecting the one that produces the best-sounding decoded signal. This is obviously not possible in practice for two reasons: the required complexity is beyond any currently available hardware and the "best sounding" selection criterion implies a human listener.

In order to achieve real-time encoding using limited computing resources, the CELP optimisation is broken down into smaller, more manageable, sequential searches using the perceptual weighting function described earlier.

1. # Speex narrowband mode

This section looks at how Speex works for narrowband (8kHz sampling rate) operation. The frame size for this mode is 20ms, corresponding to 160 samples. Each frame is also subdivided into 4 sub-frames of 40 samples each. Also many design decisions were based on the original goals and assumptions:

- Minimizing the amount of information extracted from past frames (for robustness to packet loss)
- Dynamically-selectable codebooks (LSP, pitch and innovation)
- sub-vector fixed (innovation) codebooks

1. **Whole-Frame Analysis**

In narrowband, Speex frames are 20 ms long (160 samples) and are subdivided in 4 sub-frames of 5 ms each (40 samples). For most narrowband bit-rates (8 kbps and above), the only parameters encoded at

the frame level are the Line Spectral Pairs (LSP) and a global excitation gain g_{frame}, as shown in Fig. 9.1. All other parameters are encoded at the sub-frame level.

Linear prediction analysis is performed once per frame using an asymmetric Hamming window centered on the fourth subframe. Because linear prediction coefficients (LPC) are not robust to quantization, they are first are converted to line spectral pairs (LSP). The LSP's are considered to be associated to the 4^{th} sub-frames and the LSP's associated to the first 3 sub-frames are linearly interpolated using the current and previous LSP coefficients. The LSP coefficients and converted back to the LPC filter $A\hat{}(z)$. The non-quantized interpolated filter is denoted $A(z)$ and can be used for the weighting filter $W(z)$ because it does not need to be available to the decoder.

To make Speex more robust to packet loss, no prediction is applied on the LSP coefficients prior to quantization. The LSPs are encoded using vector quantizatin (VQ) with 30 bits for higher quality modes and 18 bits for lower quality.

1. **Sub-Frame Analysis-by-Synthesis**

The analysis-by-synthesis (AbS) encoder loop is described in Fig. 9.2. There are three main aspects where Speex significantly differs from most other CELP codecs. First, while most recent CELP codecs make use of fractional pitch estimation with a single gain, Speex uses an integer to encode the pitch period, but uses a 3-tap predictor (3 gains). The adaptive codebook contribution $e_a[n]$ can thus be expressed as:

$$e_a[n] = g_0 e[n-T-1] + g_1 e[n-T] + g_2 e[n-T+1] \quad (9.1)$$

where g_0, g_1 and g_2 are the jointly quantized pitch gains and $e[n]$ is the codec excitation memory. It is worth noting that when the pitch is smaller than the sub-frame size, we repeat the excitation at a period T. For example, when $n-T+1 \geq 0$, we use $n-2T+1$ instead. In most modes, the pitch period is encoded with 7 bits in the [17,144] range and the β_i coefficients are vector-quantized using 7 bits at higher bit-rates (15 kbps narrowband and above) and 5 bits at lower bit-rates (11 kbps narrowband and below).

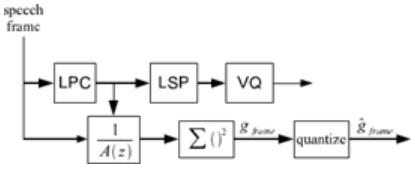


Figure 9.1: Frame open-loop analysis

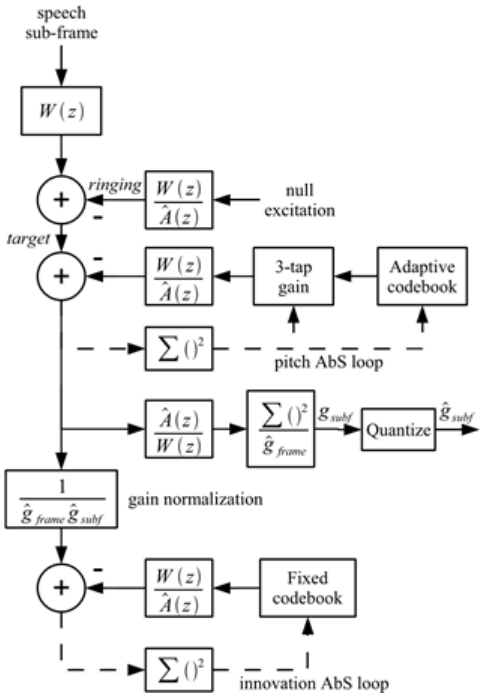


Figure 9.2: Analysis-by-synthesis closed-loop optimization on a sub-frame.

Many current CELP codecs use moving average (MA) prediction to encode the fixed codebook gain. This provides slightly better coding at the expense of introducing a dependency on previously encoded frames. A second difference is that Speex encodes the fixed codebook gain as the product of

the global excitation gain g_{frame} with a sub-frame gain corrections g_{subf}. This increases robustness to packet loss by eliminating the inter-frame dependency. The sub-frame gain correction is encoded before the fixed codebook is searched (not closed-loop optimized) and uses between 0 and 3 bits per sub-frame, depending on the bit-rate.

The third difference is that Speex uses sub-vector quantization of the innovation (fixed codebook) signal instead of an algebraic codebook. Each sub-frame is divided into sub-vectors of lengths ranging between 5 and 20 samples. Each subvector is chosen from a bitrate-dependent codebook and all sub-vectors are concatenated to form a sub-frame. As an example, the 3.95 kbps mode uses a sub-vector size of 20 samples with 32 entries in the codebook (5 bits). This means that the innovation is encoded with 10 bits per sub-frame, or 2000 bps. On the other hand, the 18.2 kbps mode uses a sub-vector size of 5 samples with 256 entries in the codebook (8 bits), so the innovation uses 64 bits per sub-frame, or 12800 bps.

1. **Bit allocation**

There are 7 different narrowband bit-rates defined for Speex, ranging from 250 bps to 24.6 kbps, although the modes below 5.9 kbps should not be used for speech. The bit-allocation for each mode is detailed in table 9.1. Each frame starts with the mode ID encoded with 4 bits which allows a range from 0 to 15, though only the first 7 values are used (the others are reserved). The parameters are listed in the table in the order they are packed in the bit-stream. All frame-based parameters are packed before sub-frame parameters. The parameters for a certain sub-frame are all packed before the following sub-frame is packed. Note that the "OL" in the parameter description means that the parameter is an open loop estimation based on the whole frame.

Parameter	Update rate	0	1	2	3	4	5	6
Wideband bit	frame	1	1	1	1	1	1	1
Mode ID	frame	4	4	4	4	4	4	4
LSP	frame	0	18	18	18	18	30	30
OL pitch	frame	0	7	7	0	0	0	0
OL pitch gain	frame	0	4	0	0	0	0	0
OL Exc gain	frame	0	5	5	5	5	5	5
Fine pitch	sub-frame	0	0	0	7	7	7	7
Pitch gain	sub-frame	0	0	5	5	5	7	7
Innovation gain	sub-frame	0	1	0	1	1	3	3
Innovation VQ	sub-frame	0	0	16	20	35	48	64

Total	frame	5	43	119	160	220	300	364

Table 9.1: Bit allocation for narrowband modes

So far, no MOS (Mean Opinion Score) subjective evaluation has been performed for Speex. In order to give an idea of the quality achievable with it, table 9.2 presents my own subjective opinion on it. It sould be noted that different people will perceive the quality differently and that the person that designed the codec often has a bias (one way or another) when it comes to subjective evaluation. Last thing, it should be noted that for most codecs (including Speex) encoding quality sometimes varies dependingon the input. Note that the complexityis only approximate(within 0.5 mflops and using the lowest complexity setting). Decoding requires approximately 0.5 mflops in most modes (1 mflops with perceptual enhancement).

1. **Perceptual enhancement**

This section was **only valid for version 1.1.12 and earlier. It does not apply to version 1.2-beta1 (and later), for which the new perceptual enhancement is not yet documented.**

This part of the codec only applies to the decoder and can even be changed without affecting inter-operability. For that reason, the implementation provided and described here should only be considered as a reference implementation. The enhancement system is divided into two parts. First, the synthesis filter $S(z) = 1/A(z)$ is replaced by an enhanced filter:

$S'(z) = A(z/a2)A(z/a3)$

$A(z)A(z/a_1)$

Mode	Quality	Bit-rate (bps)	mflops	Quality/description
0	-	250	0	No transmission (DTX)
1	0	2,150	6	Vocoder (mostly for comfort noise)
2	2	5,950	9	Very noticeable artifacts/noise, good intelligibility
3	3-4	8,000	10	Artifacts/noise sometimes noticeable
4	5-6	11,000	14	Artifacts usually noticeable only with headphones
5	7-8	15,000	11	Need good headphones to tell the difference
6	9	18,200	17.5	Hard to tell the difference even with good headphones
7	10	24,600	14.5	Completely transparent for voice, good quality music
8	1	3,950	10.5	Very noticeable artifacts/noise, good intelligibility
9	-	-	-	reserved
10	-	-	-	reserved

11	-	-	-	reserved
12	-	-	-	reserved
13	-	-	-	Application-defined, interpreted by callback or skipped
14	-	-	-	Speex in-band signaling
15	-	-	-	Terminator code

Table 9.2: Quality versus bit-rate

where a_1 and a_2 depend on the mode in use and $a^3 = \frac{1}{r}\left(1 - \frac{1-ra_1}{1-ra_2}\right)$ with r = .9. The second part of the enhancement consists of using a comb filter to enhance the pitch in the excitation domain.

1. Speex wideband mode (sub-band CELP)

For wideband, the Speex approach uses a *q*uadrature *m*irror *f*ilter (QMF) to split the band in two. The 16 kHz signal is thus divided into two 8 kHz signals, one representing the low band (0-4 kHz), the other the high band (4-8 kHz). The low band is encoded with the narrowband mode described in section 9 in such a way that the resulting "embedded narrowband bit-stream" can also be decoded with the narrowband decoder. Since the low band encoding has already been described, only the high band encoding is described in this section.

1. Linear Prediction

The linear prediction part used for the high-band is very similar to what is done for narrowband. The only difference is that we use only 12 bits to encode the high-band LSP's using a multi-stage vector quantizer (MSVQ). The first level quantizes the 10 coefficients with 6 bits and the error is then quantized using 6 bits, too.

1. Pitch Prediction

That part is easy: there's no pitch prediction for the high-band. There are two reasons for that. First, there is usually little harmonic structure in this band (above 4 kHz). Second, it would be very hard to implement since the QMF folds the 4-8 kHz band into 4-0 kHz (reversing the frequency axis), which means that the location of the harmonics is no longer at multiples of the fundamental (pitch).

1. Excitation Quantization

The high-band excitation is coded in the same way as for narrowband.

1. Bit allocation

For the wideband mode, the entire narrowband frame is packed before the high-band is encoded. The narrowband part of the bit-stream is as defined in table 9.1. The high-band follows, as described in table 10.1. For wideband, the mode ID is the same as the Speex quality setting and is defined in table 10.2. This also means that a wideband frame may be correctly decoded by a narrowband decoder with the only caveat that if more than one frame is packed in the same packet, the decoder will need to skip the high-band parts in order to sync with the bit-stream.

Parameter	Update rate	0	1	2	3	4
Wideband bit	frame	1	1	1	1	1
Mode ID	frame	3	3	3	3	3
LSP	frame	0	12	12	12	12

Excitation gain	sub-frame	0	5	4	4	4
Excitation VQ	sub-frame	0	0	20	40	80
Total	frame	4	36	112	192	352

Table 10.1: Bit allocation for high-band in wideband mode

10 Speexwidebandmode(sub-bandCELP)

Mode/Quality	Bit-rate (bps)	Quality/description
0	3,950	Barely intelligible (mostly for comfort noise)
1	5,750	Very noticeable artifacts/noise, poor intelligibility
2	7,750	Very noticeable artifacts/noise, good intelligibility
3	9,800	Artifacts/noise sometimes annoying
4	12,800	Artifacts/noise usually noticeable
5	16,800	Artifacts/noise sometimes noticeable
6	20,600	Need good headphones to tell the difference
7	23,800	Need good headphones to tell the difference
8	27,800	Hard to tell the difference even with good headphones
9	34,200	Hard to tell the difference even with good headphones
10	42,200	Completely transparent for voice, good quality music

Table 10.2: Quality versus bit-rate for the wideband encoder

1. A Sample code

This section shows sample code for encoding and decoding speech using the Speex API. The commands can be used to encode and decode a file by calling:

% sampleenc in_file.sw | sampledec out_file.sw where both files are raw (no header) files encoded at 16 bits per sample (in the machine natural endianness).

1. A.1 sampleenc.c

sampleenc takes a raw 16 bits/sample file, encodes it and outputs a Speex stream to stdout. Note that the packing used is **not** compatible with that of speexenc/speexdec.

Listing A.1: Source code for sampleenc

```
1. #include <speex/speex.h>
2. #include <stdio.h>
3
1. /*The frame size in hardcoded for this sample code but it doesn't have to be*/
2. #define FRAME_SIZE 160
3. int main(int argc, char **argv)
4. {
5. char *inFile;
6. FILE *fin;
7. short in[FRAME_SIZE];
8. float input[FRAME_SIZE];
9. char cbits[200];
10. int nbBytes;
11. /*Holds the state of the encoder*/
12. void *state;
13. /*Holds bits so they can be read and written to by the Speex routines*/
14. SpeexBits bits;
15. int i, tmp;
19
1. /*Create a new encoder state in narrowband mode*/
2. state = speex_encoder_init(&speex_nb_mode);
22
1. /*Set the quality to 8 (15 kbps)*/
2. tmp=8;
3. speex_encoder_ctl(state, SPEEX_SET_QUALITY, &tmp);
26
1. inFile = argv[1];
2. fin = fopen(inFile, "r");
29
1. /*Initialization of the structure that holds the bits*/
2. speex_bits_init(&bits);
3. while (1)
4. {
5. /*Read a 16 bits/sample audio frame*/
6. fread(in, sizeof(short), FRAME_SIZE, fin);
7. if (feof(fin))
8. break;
9. /*Copy the 16 bits values to float so Speex can work on them*/
```

A Samplecode

```
1. for (i=0;i<FRAME_SIZE;i++)
2. input[i]=in[i];
41
1. /*Flush all the bits in the struct so we can encode a new frame*/
2. speex_bits_reset(&bits);
44
1. /*Encode the frame*/
2. speex_encode(state, input, &bits);
3. /*Copy the bits to an array of char that can be written*/
4. nbBytes = speex_bits_write(&bits, cbits, 200);
49
1. /*Write the size of the frame first. This is what sampledec expects but
2. it's likely to be different in your own application*/
```

```
3. fwrite(&nbBytes, sizeof(int), 1, stdout);
4. /*Write the compressed data*/
5. fwrite(cbits, 1, nbBytes, stdout);
```

55

56 }

57

```
1. /*Destroy the encoder state*/
2. speex_encoder_destroy(state);
3. /*Destroy the bit-packing struct*/
4. speex_bits_destroy(&bits);
5. fclose(fin);
6. return 0;
7. }
```

1. A.2 sampledec.c

sampledec reads a Speex stream from stdin, decodes it and outputs it to a raw 16 bits/sample file. Note that the packing used is **not** compatible with that of speexenc/speexdec.

Listing A.2: Source code for sampledec

```
1. #include <speex/speex.h>
2. #include <stdio.h>
```

3

```
1. /*The frame size in hardcoded for this sample code but it doesn't have to be*/
2. #define FRAME_SIZE 160
3. int main(int argc, char **argv)
4. {
5. char *outFile;
6. FILE *fout;
7. /*Holds the audio that will be written to file (16 bits per sample)*/ 11 short out[FRAME_SIZE];
8. /*Speex handle samples as float, so we need an array of floats*/
9. float output[FRAME_SIZE];
10. char cbits[200];
11. int nbBytes;
12. /*Holds the state of the decoder*/
13. void *state;
14. /*Holds bits so they can be read and written to by the Speex routines*/ 19 SpeexBits bits;
```

20 int i, tmp;

21

```
1. /*Create a new decoder state in narrowband mode*/
2. state = speex_decoder_init(&speex_nb_mode);
```

A Samplecode

24

```
1. /*Set the perceptual enhancement on*/
2. tmp=1;
3. speex_decoder_ctl(state, SPEEX_SET_ENH, &tmp);
```

28

```
1. outFile = argv[1];
2. fout = fopen(outFile, "w");
```

31

```
1. /*Initialization of the structure that holds the bits*/
2. speex_bits_init(&bits);
3. while (1)
4. {
5. /*Read the size encoded by sampleenc, this part will likely be
6. different in your application*/
```

```
7. fread(&nbBytes, sizeof(int), 1, stdin);
8. fprintf (stderr, "nbBytes:%d\n", nbBytes);

1. if (feof(stdin))
2. break;

1. /*Read the "packet" encoded by sampleenc*/
2. fread(cbits, 1, nbBytes, stdin);
3. /*Copy the data into the bit-stream struct*/
4. speex_bits_read_from(&bits, cbits, nbBytes);

1. /*Decode the data*/
2. speex_decode(state, &bits, output);

1. /*Copy from float to short (16 bits) for output*/
2. for (i=0;i<FRAME_SIZE;i++)
3. out[i]=output[i];

1. /*Write the decoded audio to file*/
2. fwrite(out, sizeof(short), FRAME_SIZE, fout);
3. }

1. /*Destroy the decoder state*/
2. speex_decoder_destroy(state);
3. /*Destroy the bit-stream truct*/
4. speex_bits_destroy(&bits);
5. fclose(fout);
6. return 0;
7. }
```

1. B Jitter Buffer for Speex

Listing B.1: Example of using the jitter buffer for Speex packets

```
1. #include <speex/speex_jitter.h>
2. #include "speex_jitter_buffer.h"

1. #ifndef NULL
2. #define NULL 0
3. #endif

1. void speex_jitter_init(SpeexJitter *jitter, void *decoder, int sampling_rate)
2. {
3. jitter->dec = decoder;
4. speex_decoder_ctl(decoder, SPEEX_GET_FRAME_SIZE, &jitter->frame_size);

14  jitter->packets = jitter_buffer_init(jitter->frame_size);

1. speex_bits_init(&jitter->current_packet);
2. jitter->valid_bits = 0;

19 }
```

```
20

1. void speex_jitter_destroy(SpeexJitter *jitter)
2. {
3. jitter_buffer_destroy(jitter->packets);
4. speex_bits_destroy(&jitter->current_packet);
5. }

26

1. void speex_jitter_put(SpeexJitter *jitter, char *packet, int len, int timestamp)
2. {
3. JitterBufferPacket p;
4. p.data = packet;
5. p.len = len;
6. p.timestamp = timestamp;
7. p.span = jitter->frame_size;
8. jitter_buffer_put(jitter->packets, &p);
9. }

36

1. void speex_jitter_get(SpeexJitter *jitter, spx_int16_t *out, int *current_timestamp )
2. {
3. int i;
4. int ret;
5. spx_int32_t activity;
6. char data[2048];
7. JitterBufferPacket packet;
8. packet.data = data;

45

                1. if (jitter->valid_bits)
                2. {
                3. /* Try decoding last received packet */
                4. ret = speex_decode_int(jitter->dec, &jitter->current_packet, out);
                5. if (ret == 0)
                6. {
                7. jitter_buffer_tick(jitter->packets);
                8. return;
                9. } else {
               10. jitter->valid_bits = 0;
               11. }
               12. }

58

59  ret = jitter_buffer_get(jitter->packets, &packet, jitter->frame_size, NULL);

60

            1. if (ret != JITTER_BUFFER_OK)
            2. {
            3. /* No packet found */

64

                    1. /*fprintf (stderr, "lost/late frame\n");*/
                    2. /*Packet is late or lost*/
                    3. speex_decode_int(jitter->dec, NULL, out);
                    4. } else {
                    5. speex_bits_read_from(&jitter->current_packet, packet.data, packet.len);
                    6. /* Decode packet */
                    7. ret = speex_decode_int(jitter->dec, &jitter->current_packet, out);
                    8. if (ret == 0)
                    9. {
                   10. jitter->valid_bits = 1;
                   11. } else {
```

```
12. /* Error while decoding */
13. for (i=0;i<jitter->frame_size;i++)
14. out[i]=0;
15. }
16. }
17. speex_decoder_ctl(jitter->dec, SPEEX_GET_ACTIVITY, &activity);
18. if (activity < 30)
19. jitter_buffer_update_delay(jitter->packets, &packet, NULL);
20. jitter_buffer_tick(jitter->packets);
21. }
```

86

```
1. int speex_jitter_get_pointer_timestamp(SpeexJitter *jitter)
2. {
3. return jitter_buffer_get_pointer_timestamp(jitter->packets);
4. }
```

1. ## C IETF RTP Profile

AVT	G. Herlein
Internet-Draft	
Intended status: Standards Track	J. Valin
Expires: October 24, 2007	University of Sherbrooke
	A. Heggestad
	April 22, 2007

RTP Payload Format for the Speex Codec draft-ietf-avt-rtp-speex-01 (non-final)

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as InternetDrafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on October 24, 2007.

Copyright Notice

[Page 1]

April 2007

Abstract

Speex is an open-source voice codec suitable for use in Voice over IP (VoIP) type applications. This document describes the payload format for Speex generated bit streams within an RTP packet. Also included here are the necessary details for the use of Speex with the Session Description Protocol (SDP).

[Page 2]

April 2007

Editors Note

All references to RFC XXXX are to be replaced by references to the RFC number of this memo, when published.

Table of Contents

April 2007

1. Introduction

Speex is based on the CELP [CELP] encoding technique with support for either narrowband (nominal 8kHz), wideband (nominal 16kHz) or ultrawideband (nominal 32kHz). The main characteristics can be summarized as follows:

- Free software/open-source
- Integration of wideband and narrowband in the same bit-stream
- Wide range of bit-rates available
- Dynamic bit-rate switching and variable bit-rate (VBR)
- Voice Activity Detection (VAD, integrated with VBR)
- Variable complexity

To be compliant with this specification, implementations MUST support 8 kHz sampling rate (narrowband)" and SHOULD support 8 kbps bitrate.

The sampling rate MUST be 8, 16 or 32 kHz.

[Page 4]

April 2007

1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC2119 [RFC2119] and indicate requirement levels for compliant RTP implementations.

[Page 5]

April 2007

1. RTP usage for Speex

3.1. RTP Speex Header Fields

The RTP header is defined in the RTP specification [RFC3550]. This section defines how fields in the RTP header are used.

Payload Type (PT): The assignment of an RTP payload type for this packet format is outside the scope of this document; it is specified by the RTP profile under which this payload format is used, or signaled dynamically out-of-band (e.g., using SDP).

Marker (M) bit: The M bit is set to one to indicate that the RTP packet payload contains at least one complete frame Extension (X) bit: Defined by the RTP profile used.

Timestamp: A 32-bit word that corresponds to the sampling instant for the first frame in the RTP packet.

3.2. RTP payload format for Speex

The RTP payload for Speex has the format shown in Figure 1. No additional header fields specific to this payload format are required. For RTP based transportation of Speex encoded audio the standard RTP header [RFC3550] is followed by one or more payload data blocks. An optional padding terminator may also be used.

```
  0   1   2   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |   RTP Header   |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
  |   one or more frames of Speex ....   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |   one or more frames of Speex ....   |   padding   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 1: RTP payload for Speex

3.3. Speex payload

For the purposes of packetizing the bit stream in RTP, it is only necessary to consider the sequence of bits as output by the Speex encoder [speexenc], and present the same sequence to the decoder.

The payload format described here maintains this sequence.

A typical Speex frame, encoded at the maximum bitrate, is approx. 110

octets and the total number of Speex frames SHOULD be kept less than the path MTU to prevent fragmentation. Speex frames MUST NOT be fragmented across multiple RTP packets,

An RTP packet MAY contain Speex frames of the same bit rate or of varying bit rates, since the bit-rate for a frame is conveyed in band with the signal.

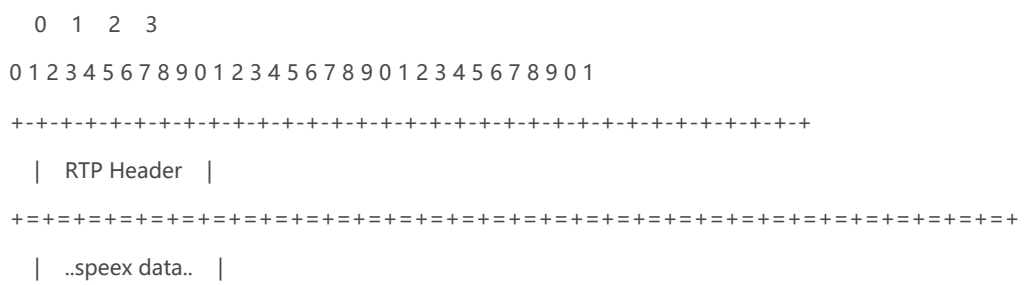
The encoding and decoding algorithm can change the bit rate at any 20 msec frame boundary, with the bit rate change notification provided in-band with the bit stream. Each frame contains both "mode" (narrowband, wideband or ultra-wideband) and "sub-mode" (bit-rate) information in the bit stream. No out-of-band notification is required for the decoder to process changes in the bit rate sent by the encoder.

Sampling rate values of 8000, 16000 or 32000 Hz MUST be used. Any other sampling rates MUST NOT be used.

The RTP payload MUST be padded to provide an integer number of octets as the payload length. These padding bits are LSB aligned in network octet order and consist of a 0 followed by all ones (until the end of the octet). This padding is only required for the last frame in the packet, and only to ensure the packet contents ends on an octet boundary.

3.4. Example Speex packet

In the example below we have a single Speex frame with 5 bits of padding to ensure the packet size falls on an octet boundary.

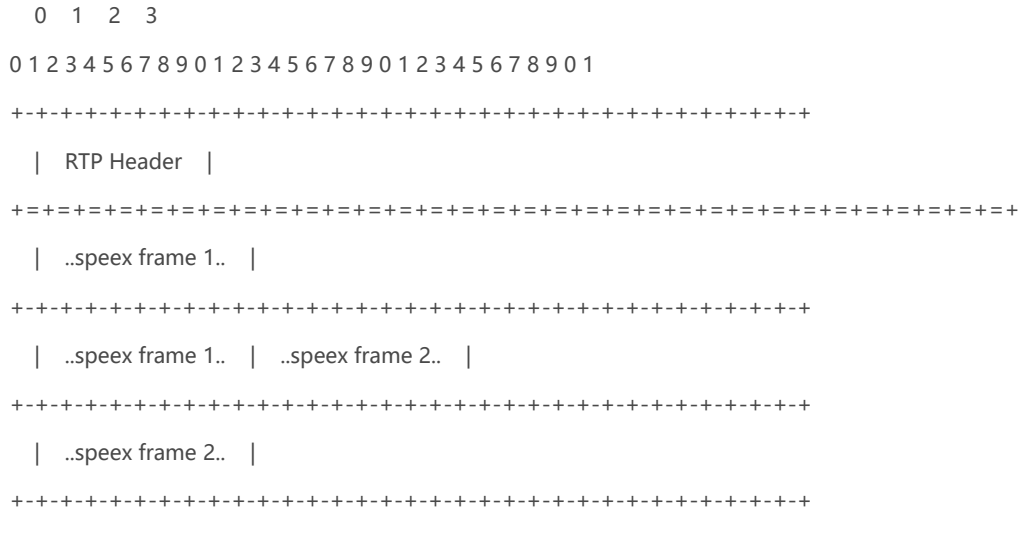


3.5. Multiple Speex frames in a RTP packet

Below is an example of two Speex frames contained within one RTP packet. The Speex frame length in this example fall on an octet boundary so there is no padding.

Speex codecs [speexenc] are able to detect the bitrate from the

payload and are responsible for detecting the 20 msec boundaries between each frame.



1. IANA Considerations

This document defines the Speex media type.

1. Media Type Registration

This section describes the media types and names associated with this payload format. The section registers the media types, as per RFC4288 [RFC4288]

4.1.1. Registration of media type audio/speex

Media type name: audio

Media subtype name: speex Required parameters:

None

Optional parameters:

ptime: see RFC 4566. SHOULD be a multiple of 20 msec.

maxptime: see RFC 4566. SHOULD be a multiple of 20 msec.

Encoding considerations:

This media type is framed and binary, see section 4.8 in [RFC4288].

Security considerations: See Section 6 Interoperability considerations:

None.

Published specification: RFC XXXX [This RFC].

Applications which use this media type:

Audio streaming and conferencing applications.

Additional information: none

Person and email address to contact for further information :

[Page 9]

April 2007

Alfred E. Heggestad: aeh@db.org Intended usage: COMMON Restrictions on usage:

This media type depends on RTP framing, and hence is only defined for transfer via RTP [RFC3550]. Transport within other framing protocols is not defined at this time.

Author: Alfred E. Heggestad Change controller:

IETF Audio/Video Transport working group delegated from the IESG.

[Page 10]

April 2007

5. SDP usage of Speex

When conveying information by SDP [RFC4566], the encoding name MUST be set to "speex". An example of the media representation in SDP for offering a single channel of Speex at 8000 samples per second might be:

m=audio 8088 RTP/AVP 97 a=rtpmap:97 speex/8000

Note that the RTP payload type code of 97 is defined in this media definition to be 'mapped' to the speex codec at an 8kHz sampling frequency using the 'a=rtpmap' line. Any number from 96 to 127 could have been chosen (the allowed range for dynamic types).

The value of the sampling frequency is typically 8000 for narrow band operation, 16000 for wide band operation, and 32000 for ultra-wide band operation.

If for some reason the offerer has bandwidth limitations, the client may use the "b=" header, as explained in SDP [RFC4566]. The following example illustrates the case where the offerer cannot receive more than 10 kbit/s.

m=audio 8088 RTP/AVP 97 b=AS:10 a=rtmap:97 speex/8000

In this case, if the remote part agrees, it should configure its Speex encoder so that it does not use modes that produce more than 10 kbit/s. Note that the "b=" constraint also applies on all payload types that may be proposed in the media line ("m=").

An other way to make recommendations to the remote Speex encoder is to use its specific parameters via the a=fmtp: directive. The following parameters are defined for use in this way: ptime: duration of each packet in milliseconds. sr: actual sample rate in Hz.

ebw: encoding bandwidth - either 'narrow' or 'wide' or 'ultra' (corresponds to nominal 8000, 16000, and 32000 Hz sampling rates). [Page 11]

April 2007

vbr: variable bit rate - either 'on' 'off' or 'vad' (defaults to off). If on, variable bit rate is enabled. If off, disabled. If set to 'vad' then constant bit rate is used but silence will be encoded with special short frames to indicate a lack of voice for that period.

cng: comfort noise generation - either 'on' or 'off'. If off then silence frames will be silent; if 'on' then those frames will be filled with comfort noise.

mode: Speex encoding mode. Can be {1,2,3,4,5,6,any} defaults to 3 in narrowband, 6 in wide and ultra-wide.

Examples:

m=audio 8008 RTP/AVP 97 a=rtpmap:97 speex/8000 a=fmtp:97 mode=4

This examples illustrate an offerer that wishes to receive a Speex stream at 8000Hz, but only using speex mode 4.

Several Speex specific parameters can be given in a single a=fmtp line provided that they are separated by a semi-colon:

a=fmtp:97 mode=any;mode=1

The offerer may indicate that it wishes to send variable bit rate frames with comfort noise:

m=audio 8088 RTP/AVP 97 a=rtmap:97 speex/8000 a=fmtp:97 vbr=on;cng=on

The "ptime" attribute is used to denote the packetization interval (ie, how many milliseconds of audio is encoded in a single RTP packet). Since Speex uses 20 msec frames, ptime values of multiples of 20 denote multiple Speex frames per packet. Values of ptime which are not multiples of 20 MUST be ignored and clients MUST use the default value of 20 instead.

Implementations SHOULD support ptime of 20 msec (i.e. one frame per packet)

In the example below the ptime value is set to 40, indicating that

there are 2 frames in each packet.

m=audio 8008 RTP/AVP 97 a=rtpmap:97 speex/8000 a=ptime:40

Note that the ptime parameter applies to all payloads listed in the media line and is not used as part of an a=fmtp directive.

Values of ptime not multiple of 20 msec are meaningless, so the receiver of such ptime values MUST ignore them. If during the life of an RTP session the ptime value changes, when there are multiple Speex frames for example, the SDP value must also reflect the new value.

Care must be taken when setting the value of ptime so that the RTP packet size does not exceed the path MTU.

[Page 13]

April 2007

1. Security Considerations

RTP packets using the payload format defined in this specification are subject to the security considerations discussed in the RTP specification [RFC3550], and any appropriate RTP profile. This implies that confidentiality of the media streams is achieved by encryption. Because the data compression used with this payload format is applied end-to-end, encryption may be performed after compression so there is no conflict between the two operations.

A potential denial-of-service threat exists for data encodings using compression techniques that have non-uniform receiver-end computational load. The attacker can inject pathological datagrams into the stream which are complex to decode and cause the receiver to be overloaded. However, this encoding does not exhibit any significant non-uniformity.

As with any IP-based protocol, in some circumstances a receiver may be overloaded simply by the receipt of too many packets, either desired or undesired. Network-layer authentication may be used to discard packets from undesired sources, but the processing cost of the authentication itself may be too high.

[Page 14]

April 2007

1. Acknowledgements

The authors would like to thank Equivalence Pty Ltd of Australia for their assistance in attempting to standardize the use of Speex in H.323 applications, and for implementing Speex in their open source

OpenH323 stack. The authors would also like to thank Brian C. Wiles <brian@streamcomm.com> of StreamComm for his assistance in developing the proposed standard for Speex use in H.323 applications.

The authors would also like to thank the following members of the Speex and AVT communities for their input: Ross Finlayson, Federico Montesino Pouzols, Henning Schulzrinne, Magnus Westerlund.

Thanks to former authors of this document; Simon Morlat, Roger

Hardiman, Phil Kerr

[Page 15]

April 2007

1. References

1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

[RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, July 2006.

1. Informative References

[CELP] "CELP, U.S. Federal Standard 1016.", National Technical

Information Service (NTIS) website http://www.ntis.gov/.

[RFC4288] Freed, N. and J. Klensin, "Media Type Specifications and

Registration Procedures", BCP 13, RFC 4288, December 2005.

[speexenc]

Valin, J., "Speexenc/speexdec, reference command-line encoder/decoder", Speex website http://www.speex.org/. [Page 16]

April 2007

Authors' Addresses

Greg Herlein

2034 Filbert Street

San Francisco, California 94123

United States

Email: gherlein@herlein.com

Jean-Marc Valin

University of Sherbrooke

Department of Electrical and Computer Engineering

University of Sherbrooke

2500 blvd Universite

Sherbrooke, Quebec J1K 2R1 Canada

Email: jean-marc.valin@usherbrooke.ca

Alfred E. Heggestad

Biskop J. Nilssonsgt. 20a

Oslo 0659

Norway

Email: aeh@db.org

[Page 17]

April 2007

Full Copyright Statement

Intellectual Property

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).

[Page 18]

1. D Speex License

- Neither the name of the Xiph.org Foundation nor the names of its contributors may be used to en dorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS

''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT

LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR

A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE FOUNDATION OR

CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,

EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,

PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR

PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF

LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING

NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

1. E GNU Free Documentation License

Version 1.1, March 2000

Copyright (C) 2000 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

1. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited

directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LATEX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machinegenerated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

1. **VERBATIM COPYING**

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproducedin all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3. You may also lend copies, under the same conditions stated above, and you may publicly display copies.

1. **COPYING IN QUANTITY**

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machinereadable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computernetwork location containing a complete Transparent copy of the Document, free of added material, which the general networkusing public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

1. **4. MODIFICATIONS**

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and fr om those of previous versions (which should, if there were any, be listed in the History section of t he Document). You may use the same title as a previous version if the original publisher of that ver sion gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of t he modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright noti ces.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum bel ow.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given i n the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no s ection entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as st ated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent c opy of the Document, and likewise the network locations given in the Document for previous versi ons it was based on. These may be placed in the "History" section. You may omit a network locatio n for a work that was published at least four years before the Document itself, or if the original pu blisher of the version it refers to gives permission.
- K. In any section entitled "Acknowledgements"or "Dedications", preserve the section's title, and pr eserve in the section all the substance and tone of each of the contributor acknowledgements an d/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Sec tion numbers or the equivalent are not considered part of the section titles.
- M. Delete any section entitled "Endorsements". Such a section may not be included in the Modifie d Version.
- N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Se ction.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", providedit contains nothing but endorsements of your Modified Version by various parties–forexample, statements of peer review or that the text has been approvedby an organizationas the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

1. **COMBINING DOCUMENTS**

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same

name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination,youmust combineany sections entitled "History" in the various originaldocuments,formingonesection entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

1. **COLLECTIONS OF DOCUMENTS**

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

1. **AGGREGATION WITH INDEPENDENT WORKS**

A compilationof the Documentor its derivativeswith other separate and independentdocumentsor works, in or on a volumeof a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

1. **TRANSLATION**

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

1. **TERMINATION**

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

1. **FUTURE REVISIONS OF THIS LICENSE**

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

1. **Index**

1. config.h文件说明

FLOATING_POINT宏：浮点执行。

FIXED_POINT宏：定点执行。

以上选择其一。

USE_SMALLFT宏：傅里叶算法的一种。

USE_KISS_FFT宏：傅里叶算法的一种。

以上选择其一。

_USE_SSE宏：使用SSE指令集。

_USE_SSE2宏：使用SSE2指令集。

HAVE_CONFIG_H宏：使用config.h头文件。

EXPORT宏：

1. Speex函数库

1. 函数模板(未完成)

函数名称	xxx
头文件	#include "speex/speex.h" #include "speex/speex_preprocess.h" #include "speex/speex_echo.h" #include "speex/speex_resampler.h"
库文件	#pragma comment(lib, "libspeex.lib") #pragma comment(lib, "libspeexdsp.lib")
函数功能	函数主要功能说明。
函数声明	类型 函数名(

	类型 参数1, 类型 参数2, ...);
函数参数	**参数1**，[输入|输出|输入&输出]： 参数说明。
	参数2，[输入|输出|输入&输出]： 参数说明。
	.....
返回值	**返回值1**：返回值说明。 **返回值2**：返回值说明。 .....
错误码	**EXXXX**：错误码说明。 **EXXXX**：错误码说明。 .....
线程安全	是 或 否 或 未知，表示此函数多线程调用是否会产生影响
原子操作	是 或 否 或 未知，表示此函数是否是单一操作，不是多个步骤的组合
执行速度	每毫秒能执行多少次。
其他说明	....

1. Speex encoder Speex编码器

1. speex_encoder_init(未完成)

函数名称	speex_encoder_init
头文件	#include "speex/speex.h"
库文件	#pragma comment(lib, "libspeex.lib")
函数功能	根据Speex格式配置结构体创建并初始化一个Speex编码器。 Speex编码器用于将PCM格式音频数据编码成Speex格式音频数据。

函数声明	void * speex_encoder_init(const SpeexMode * mode);
函数参数	**mode**，[输入]： 存放Speex格式配置结构体的内存指针，用于对PCM格式音频数据编码时使用，参考SpeexMode。 本参数不能手动创建，必须使用Speex库预定义好的，可以为（选一至一个）： **speex_nb_mode静态结构体变量**：Narrow Band窄带模式，音频数据的采样频率为8000Hz。 **speex_wb_mode静态结构体变量**：Wide Band宽带模式，音频数据的采样频率为16000Hz。 **speex_uwb_mode静态结构体变量**：Ultra Wide Band超宽带模式，音频数据的采样频率为32000Hz。
返回值	**非NULL**：成功，返回值就是Speex编码器的句柄。 **NULL**：失败，无法查看错误码，一般是内存不足。
错误码	**无**
线程安全	是
原子操作	否
执行速度	每毫秒能执行多少次。
其他说明	如果是对一条PCM格式音频数据流编码，那么编码从开始到结束都应该用一个Speex编码器，中途不要更换Speex编码器，也不要用一个Speex编码器给多条音频流编码，否则会导致解码后的音频数据和编码前的音频数据相差较大。 当Speex编码器不再使用时，必须调用speex_encoder_destroy()函数销毁Speex编码器，否则会内存泄漏。

1. **speex_encoder_ctl(未完成)**

函数名称	speex_encoder_ctl
头文件	#include "speex/speex.h"
库文件	#pragma comment(lib, "libspeex.lib")
函数功能	设置一个Speex编码器的相关参数。
函数声明	int speex_encoder_ctl(void * state, int request, void * ptr);

函数参数	**state**，[输入]： 存放Speex编码器的句柄。
	request，[输入]： 存放需要设置的参数，可以为（选一至一个）： **SPEEX_SET_ENH宏(0x0000)**：设置是否使用Speex解码器的知觉增强。ptr参数为spx_int32_t型变量的内存指针，非0表示要使用，0表示不使用，默认为1。本参数仅对Speex解码器有效。 **SPEEX_GET_ENH宏(0x0001)**：获取是否使用Speex解码器的知觉增强。ptr参数为spx_int32_t型变量的内存指针，非0表示要使用，0表示不使用，默认为1。本参数仅对Speex解码器有效。 **SPEEX_SET_FRAME_SIZE宏(未定义)**：设置Speex编解码器在编解码时，每个PCM格式音频数据帧的数据长度，单位个采样数据。ptr参数为spx_int32_t型变量的内存指针，窄带默认为160，宽带默认为320，超宽带默认为640。本参数对Speex编解码器均有效。本参数目前无法使用，也就是无法修改每帧的数据长度。 **SPEEX_GET_FRAME_SIZE宏(0x0003)**：获取Speex编解码器在编解码时，每个PCM格式音频数据帧的数据长度，单位个采样数据。ptr参数为spx_int32_t型变量的内存指针，窄带默认为160，宽带默认为320，超宽带默认为640。本参数对Speex编解码器均有效。 **SPEEX_SET_QUALITY宏(0x0004)**：设置Speex编码器在用固定采样频率编码时，音频的质量等级，质量等级越高音质越好、压缩率越低。ptr参数为spx_int32_t型变量的内存指针，取值区间为[0,10]，默认为8。本参数仅对Speex编码器有效。 **SPEEX_GET_QUALITY宏(0x0005)**：获取Speex编码器在用固定采样频率编码时，音频的质量等级，质量等级越高音质越好、压缩率越低。ptr参数为spx_int32_t型变量的内存指针，取值区间为[0,10]，默认为8。本参数仅对Speex编码器有效。本标记目前无法使用。 /** Set sub-mode to use */ #define SPEEX_SET_MODE 6 /** Get current sub-mode in use */ #define SPEEX_GET_MODE 7 /** Set low-band sub-mode to use (wideband only)*/ #define SPEEX_SET_LOW_MODE 8 /** Get current low-band mode in use (wideband only)*/ #define SPEEX_GET_LOW_MODE 9 /** Set high-band sub-mode to use (wideband only)*/ #define SPEEX_SET_HIGH_MODE 10 /** Get current high-band mode in use (wideband only)*/ #define SPEEX_GET_HIGH_MODE 11 **SPEEX_SET_VBR宏(0x000C)**：设置是否使用Speex编码器的动态比特率，使用后可以增加压缩率，且SPEEX_SET_QUALITY参数任然有效。ptr参数为spx_int32_t型变量的内存指针，非0表示要使用，0表示不使用，默认为0。本参数仅对Speex编码器有效。 **SPEEX_GET_VBR宏(0x000D)**：获取是否使用Speex编码器的动态比特率，使用后可以增加压缩率，且SPEEX_SET_QUALITY参数任然有效。ptr参数为spx_int32_t型变量的内存指针，非0表示要使用，0表示不使用，默认为0。本参数仅对Speex编码器有效。

SPEEX_SET_VBR_QUALITY宏(0x000E)：设置Speex编码器在用动态采样频率编码时，音频的质量等级，质量等级越高音质越好、压缩率越低。ptr参数为float型变量的内存指针，取值区间为[0.0,10.0]，默认为10.0。本参数仅对Speex编码器有效。

SPEEX_GET_VBR_QUALITY宏(0x000F)：获取Speex编码器在用动态采样频率编码时，音频的质量等级，质量等级越高音质越好、压缩率越低。ptr参数为float型变量的内存指针，取值区间为[0.0,10.0]，默认为10.0。本参数仅对Speex编码器有效。

SPEEX_SET_COMPLEXITY宏(0x0010)：设置Speex编码器的复杂度，复杂度越高压缩率不变、CPU使用率越高、音质越好。ptr参数为spx_int32_t型变量的内存指针，取值区间为[0,10]，默认为2。本参数仅对Speex编码器有效。

SPEEX_GET_COMPLEXITY宏(0x0011)：获取Speex编码器的复杂度，复杂度越高压缩率不变、CPU使用率越高、音质越好。ptr参数为spx_int32_t型变量的内存指针，取值区间为[0,10]，默认为2。本参数仅对Speex编码器有效。

/** Set bit-rate used by the encoder (or lower) */

#define SPEEX_SET_BITRATE 18

/** Get current bit-rate used by the encoder or decoder */

#define SPEEX_GET_BITRATE 19

/** Define a handler function for in-band Speex request*/

#define SPEEX_SET_HANDLER 20

/** Define a handler function for in-band user-defined request*/

#define SPEEX_SET_USER_HANDLER 22

SPEEX_SET_SAMPLING_RATE宏(0x0018)：设置Speex编解码器在比特率计算时，音频的采样频率。ptr参数为spx_int32_t型变量的内存指针，窄带默认为8000，宽带默认为16000，超宽带默认为32000。本参数对Speex编解码器均有效。

SPEEX_GET_SAMPLING_RATE宏(0x0019)：获取Speex编解码器在比特率计算时，音频的采样频率。ptr参数为spx_int32_t型变量的内存指针，窄带默认为8000，宽带默认为16000，超宽带默认为32000。本参数对Speex编解码器均有效。

SPEEX_RESET_STATE宏(0x001A)：重置Speex编解码器的所有参数为初始状态。ptr参数无意义。本参数对Speex编解码器均有效。

/** Get VBR info (mostly used internally) */

#define SPEEX_GET_RELATIVE_QUALITY 29

SPEEX_SET_VAD宏(0x001E)：设置Speex编码器在用动态采样频率编码时，是否使用语音活动检测，使用后可以提升在无语音活动时动态比特率编码的压缩率。ptr参数为spx_int32_t型变量的内存指针，非0表示要使用，0表示不使用，默认为0。本参数仅对Speex编码器有效。

SPEEX_GET_VAD宏(0x001F)：获取Speex编码器在用动态采样频率编码时，是否使用语音活动检测，使用后可以提升在无语音活动时动态比特率编码的压缩率。ptr参数为spx_int32_t型变量的内存指针，非0表示要使用，0表示不使用，默认为0。本参数仅对Speex编码器有效。

/** Set Average Bit-Rate (ABR) to n bits per seconds */

#define SPEEX_SET_ABR 32

/** Get Average Bit-Rate (ABR) setting (in bps) */

	#define SPEEX_GET_ABR 33 **SPEEX_SET_DTX宏(0x0022)**：设置是否使用Speex编码器的不连续传输，使用后可以降低网络传输的比特率。ptr参数为spx_int32_t型变量的内存指针，非0表示要使用，0表示不使用，默认为0。本参数仅对Speex编码器有效。 **SPEEX_GET_DTX宏(0x0023)**：获取是否使用Speex编码器的不连续传输，使用后可以降低网络传输的比特率。ptr参数为spx_int32_t型变量的内存指针，非0表示要使用，0表示不使用，默认为0。本参数仅对Speex编码器有效。 /** Set submode encoding in each frame (1 for yes, 0 for no, setting to no breaks the standard) */ #define SPEEX_SET_SUBMODE_ENCODING 36 /** Get submode encoding in each frame */ #define SPEEX_GET_SUBMODE_ENCODING 37 /*#define SPEEX_SET_LOOKAHEAD 38*/ /** Returns the lookahead used by Speex separately for an encoder and a decoder. * Sum encoder and decoder lookahead values to get the total codec lookahead. */ #define SPEEX_GET_LOOKAHEAD 39 **SPEEX_SET_PLC_TUNING宏(0x0028)**：设置Speex编码器的数据包丢失隐藏的预计丢失概率，预计丢失概率越高抗网络抖动越强、压缩率越低。ptr参数为spx_int32_t型变量的内存指针，取值区间为[0,100]，默认为2。本参数仅对Speex编码器有效。 **SPEEX_GET_PLC_TUNING宏(0x0029)**：获取Speex编码器的数据包丢失隐藏的预计丢失概率，预计丢失概率越高抗网络抖动越强、压缩率越低。ptr参数为spx_int32_t型变量的内存指针，取值区间为[0,100]，默认为2。本参数仅对Speex编码器有效。 /** Sets the max bit-rate allowed in VBR mode */ #define SPEEX_SET_VBR_MAX_BITRATE 42 /** Gets the max bit-rate allowed in VBR mode */ #define SPEEX_GET_VBR_MAX_BITRATE 43 **SPEEX_SET_HIGHPASS宏(0x002C)**：设置是否使用Speex编码器的**高通滤波器**。ptr参数为spx_int32_t型变量的内存指针，非0表示要使用，0表示不使用，默认为1。本参数对Speex编解码器均有效。 **SPEEX_GET_HIGHPASS宏(0x002D)**：获取是否使用Speex编码器的**高通滤波器**。ptr参数为spx_int32_t型变量的内存指针，非0表示要使用，0表示不使用，默认为1。本参数对Speex编解码器均有效。 /** Get "activity level" of the last decoded frame, i.e. how much damage we cause if we remove the frame */ #define SPEEX_GET_ACTIVITY 47
	ptr，[输入&输出]： 存放设置的参数，本参数是根据request参数来定义的。
返回值	**0**：成功。

	-1：request参数不正确。 **-2**：无效的参数。
错误码	**无**
线程安全	多线程可以同时操作不同的Speex编码器句柄，但不能同时操作相同的Speex编码器句柄。
原子操作	否
执行速度	每毫秒能执行多少次。
其他说明	特别注意：Speex编码器在动态比特率模式下，在极少数情况下会出现读写无效内存错误，从而导致程序崩溃，改用固定比特率可以绕过该问题。

1. **speex_encode(未完成)**

函数名称	speex_encode
头文件	#include "speex/speex.h"
库文件	#pragma comment(lib, "libspeex.lib")
函数功能	用一个Speex编码器将一个单声道16位有符号浮点型20毫秒PCM格式音频数据帧编码成Speex格式。
函数声明	int speex_encode(void * state, float * in, SpeexBits * bits);
函数参数	**state**，[输入]： 存放Speex编码器的句柄。
	in，[输入]： 存放一个单声道16位有符号浮点型20毫秒PCM格式音频数据帧数组的内存指针，音频数据取值区间为[-1.0,1.0]。
	bits，[输出]： 存放SpeexBits结构体变量的内存指针，该变量用于存放编码后的Speex格式音频数据帧。
返回值	**1**：本帧数据编码完毕，如果已使用不连续传输，表示需要传输。 **0**：本帧数据编码完毕，如果已使用不连续传输，表示不要传输。
错误码	**无**

线程安全	多线程可以同时操作不同的Speex编码器句柄，但不能同时操作相同的Speex编码器句柄。
原子操作	否
执行速度	每毫秒能执行多少次。
其他说明	调用本函数前，必须先调用speex_bits_reset()函数清空SpeexBits结构体。 调用本函数后，编码后的Speex格式音频数据帧存放在SpeexBits结构体中，还需要调用speex_bits_write()函数才能取出。 建议一个SpeexBits结构体中存放了一个Speex格式音频数据帧后，就马上取出，不要等到存放多个后再取出，否则取出后我不知道该如何解码。

1. **speex_encode_int(未完成)**

函数名称	speex_encode_int
头文件	#include "speex/speex.h"
库文件	#pragma comment(lib, "libspeex.lib")
函数功能	用一个Speex编码器将一个单声道16位有符号整型20毫秒PCM格式音频数据帧编码成Speex格式。
函数声明	int speex_encode_int(void * state, spx_int16_t * in, SpeexBits * bits);
函数参数	**state**，[输入]： 存放Speex编码器的句柄。
	in，[输入]： 存放一个单声道16位有符号整型20毫秒PCM格式音频数据帧数组的内存指针，音频数据取值区间为[-32768,32767]。
	bits，[输出]： 存放SpeexBits结构体变量的内存指针，该变量用于存放编码后的Speex格式音频数据帧。
返回值	**1**：本帧数据编码完毕，如果已使用不连续传输，表示需要传输。 **0**：本帧数据编码完毕，如果已使用不连续传输，表示不要传输。
错误码	**无**
线程安全	多线程可以同时操作不同的Speex编码器句柄，但不能同时操作相同的Speex编码器句柄。

原子操作	否
执行速度	每毫秒能执行多少次。
其他说明	调用本函数前，必须先调用speex_bits_reset()函数清空SpeexBits结构体。 调用本函数后，编码后的Speex格式音频数据帧存放在SpeexBits结构体中，还需要调用speex_bits_write()函数才能取出。 建议一个SpeexBits结构体中存放了一个Speex格式音频数据帧后，就马上取出，不要等到存放多个后再取出，否则取出后我不知道该如何解码。 编码后Speex格式音频数据帧的数据长度表：

采样频率	编码比特率	质量等级	复杂度	编码前字节数	编码后字节数	压缩率%	比特率bps
8000	固定	0	0~10	320	6	1.875	2400
		1			10	3.125	4000
		2			15	4.6875	6000
		3			20	6.25	8000
		4			20	6.25	8000
		5			28	8.75	11200
		6			28	8.75	11200
		7			38	11.875	15200
		8			38	11.875	15200
		9			46	14.375	18400
		10			62	19.375	24800
	动态	0	0~10		10,6,1		
		1			20,15,10,6,1		
		2			28,20,15,10,6,1		
		3			28,20,15,10,6,1		

				4			38, 28, 20, 15, 10, 6, 1		
				5			38, 28, 20, 15, 10, 6, 1		
				6			46, 38, 28, 20, 15, 10, 6, 1		
				7			46, 38, 28, 20, 15, 10, 6, 1		
				8			46, 38, 28, 20, 15, 10, 6, 1		
				9			62, 46, 38, 28, 20, 15, 10, 6, 1		
				10			62, 46, 38, 28, 20, 15, 10, 6, 1		
			16000	固定	0	0~10	640	10	1.5625	4000
					1			15	2.34375	6000
					2			20	3.125	8000
					3			25	3.90625	10000
					4			32	5	12800
					5			42	6.5625	16800
					6			52	8.125	20800
					7			60	9.375	24000
					8			70	10.9375	28000
					9			86	13.4375	34400

		10			100	15.625	40000
	动态	0	0~10		25, 20, 15, 10, 2		
		1			25, 20, 15, 10, 2		
		2			32, 25, 20, 15, 10, 2		
		3			32, 25, 20, 15, 10, 2		
		4			52, 42, 32, 25, 20, 15, 10, 2		
		5			52, 42, 32, 25, 20, 15, 10, 2		
		6			60, 52, 42, 32, 25, 20, 15, 10, 2		
		7			70, 60, 52, 42, 32, 25, 20, 15, 10, 2		
		8			86, 70, 60, 52, 42, 32, 25, 20, 15, 10, 2		
		9			100, 86, 70, 60, 52, 42, 32, 25, 20, 15, 10, 2		
		10			100, 86, 70, 60, 52, 42, 32, 25, 20, 15, 10, 2		
32000	固定	0	0~10	1280	11	0.859375	4400

						1			19	1.484375	7600	
						2			24	1.875	9600	
						3			29	2.265625	11600	
						4			37	2.890625	14800	
						5			47	3.671875	18800	
						6			56	4.375	22400	
						7			64	5	25600	
						8			74	5.78125	29600	
						9			90	7.03125	36000	
						10			100	7.8125	40000	
					动态	0	0~10		29, 24, 20, 15, 11, 2			
						1			37, 29, 24, 20, 15, 11, 2			
						2			37, 29, 24, 20, 15, 11, 2			
						3			56, 47, 37, 29, 24, 20, 15, 11, 2			
						4			56, 47, 37, 29, 24, 20, 15, 11, 2			
						5			64, 56, 47, 37, 29, 24, 20, 15, 11, 2			
						6			64, 56, 47, 37, 29, 24, 20, 15, 11, 2			

				7			90, 74, 64, 56, 47, 37, 29, 24, 20, 15, 11, 2		
				8			90, 74, 64, 56, 47, 37, 29, 24, 20, 15, 11, 2		
				9			100, 90, 74, 64, 56, 47, 37, 29, 24, 20, 15, 11, 2		
				10			100, 90, 74, 64, 56, 47, 37, 29, 24, 20, 15, 11, 2		

1. **speex_encoder_destroy(未完成)**

函数名称	speex_encoder_destroy
头文件	#include "speex/speex.h"
库文件	#pragma comment(lib, "libspeex.lib")
函数功能	销毁一个Speex编码器。
函数声明	void speex_encoder_destroy(void * state);
函数参数	**state**，[输入]： 存放Speex编码器的句柄。
返回值	**无**
错误码	**无**
线程安全	多线程可以同时操作不同的Speex编码器句柄，但不能同时操作相同的Speex编码器句柄。

原子操作	否
执行速度	每毫秒能执行多少次。
其他说明	Speex编码器句柄在销毁后就不能再使用了。

1. Speex decoder Speex解码器

1. speex_decoder_init(未完成)

函数名称	speex_decoder_init
头文件	#include "speex/speex.h"
库文件	#pragma comment(lib, "libspeex.lib")
函数功能	根据Speex格式配置结构体创建并初始化一个Speex解码器。 Speex解码器用于将Speex格式音频数据解码成PCM格式音频数据。
函数声明	void * speex_decoder_init(const SpeexMode * mode);
函数参数	**mode**，[输入]： 存放Speex格式配置结构体的内存指针，用于对Speex格式音频数据解码时使用，参考SpeexMode。 本参数不能手动创建，必须使用Speex库预定义好的，可以为（选一至一个）： **speex_nb_mode静态结构体变量**：Narrow Band窄带模式，音频数据的采样频率为8000Hz。 **speex_wb_mode静态结构体变量**：Wide Band宽带模式，音频数据的采样频率为16000Hz。 **speex_uwb_mode静态结构体变量**：Ultra Wide Band超宽带模式，音频数据的采样频率为32000Hz。
返回值	**非NULL**：成功，返回值就是Speex解码器的句柄。 **NULL**：失败，无法查看错误码，一般是内存不足。
错误码	**无**
线程安全	是
原子操作	否
执行速度	每毫秒能执行多少次。
其他说明	如果是对一条Speex格式音频数据流解码，那么解码从开始到结束都应该用一个Speex解码器，中途不要更换Speex解码器，也不要用一个Speex解码器给多条音频流解码，否则会导致解码后的音频数据和编码前的音频数据相差较大。

	当Speex解码器不再使用时，必须调用speex_decoder_destroy()函数销毁Speex解码器，否则会内存泄漏。 如果Speex解码器的采样频率和Speex编码器设置不一样，解码出来的音频数据也是可以正常使用的。例如将Speex编码器设置为8000Hz、Speex解码器设置为32000Hz，那么解码出来的音频数据就会是32000Hz的，但音质和8000Hz是一样的，有点类似于将8000Hz重采样成32000Hz，反过来将Speex编码器设置为32000Hz、Speex解码器设置为8000Hz也是可以的。

1. **speex_decoder_ctl(未完成)**

函数名称	speex_encoder_ctl
头文件	#include "speex/speex.h"
库文件	#pragma comment(lib, "libspeex.lib")
函数功能	设置一个Speex解码器的相关参数。
函数声明	int speex_decoder_ctl(void * state, int request, void * ptr);
函数参数	**state**，[输入]： 存放Speex解码器的句柄。
	request，[输入]： 存放需要设置的参数，可以为（选一至一个）： **SPEEX_SET_ENH宏(0x0000)**：设置是否使用Speex解码器的知觉增强。ptr参数为spx_int32_t型变量的内存指针，非0表示要使用，0表示不使用，默认为1。本参数仅对Speex解码器有效。 **SPEEX_GET_ENH宏(0x0001)**：获取是否使用Speex解码器的知觉增强。ptr参数为spx_int32_t型变量的内存指针，非0表示要使用，0表示不使用，默认为1。本参数仅对Speex解码器有效。 **SPEEX_SET_FRAME_SIZE宏(未定义)**：设置Speex编解码器在编解码时，每个PCM格式音频数据帧的数据长度，单位个采样数据。ptr参数为spx_int32_t型变量的内存指针，窄带默认为160，宽带默认为320，超宽带默认为640。本参数对Speex编解码器均有效。本参数目前无法使用，也就是无法修改每帧的数据长度。 **SPEEX_GET_FRAME_SIZE宏(0x0003)**：获取Speex编解码器在编解码时，每个PCM格式音频数据帧的数据长度，单位个采样数据。ptr参数为spx_int32_t型变量的内存指针，窄带默认为160，宽带默认为320，超宽带默认为640。本参数对Speex编解码器均有效。 **SPEEX_SET_QUALITY宏(0x0004)**：设置Speex编码器在用固定采样频率编码时，音频的质量等级，质量等级越高音质越好、压缩率越低。ptr参数为spx_int32_t型变量的内存指针，取值区间为[0,10]，默认为8。本参数仅对Speex编码器有效。 **SPEEX_GET_QUALITY宏(0x0005)**：获取Speex编码器在用固定采样频率编码时，音频的质量等级，质量等级越高音质越好、压缩率越低。ptr参数为spx_int32_t型变量的内存指针，取值区间为[0,10]，默认为8。本参数仅对Speex编码器有效。本标记目前无法使用。

/** Set sub-mode to use */

#define SPEEX_SET_MODE 6

/** Get current sub-mode in use */

#define SPEEX_GET_MODE 7

/** Set low-band sub-mode to use (wideband only)*/

#define SPEEX_SET_LOW_MODE 8

/** Get current low-band mode in use (wideband only)*/

#define SPEEX_GET_LOW_MODE 9

/** Set high-band sub-mode to use (wideband only)*/

#define SPEEX_SET_HIGH_MODE 10

/** Get current high-band mode in use (wideband only)*/

#define SPEEX_GET_HIGH_MODE 11

SPEEX_SET_VBR宏(0x000C)：设置是否使用Speex编码器的动态比特率，使用后可以增加压缩率，且SPEEX_SET_QUALITY参数任然有效。ptr参数为spx_int32_t型变量的内存指针，非0表示要使用，0表示不使用，默认为0。本参数仅对Speex编码器有效。

SPEEX_GET_VBR宏(0x000D)：获取是否使用Speex编码器的动态比特率，使用后可以增加压缩率，且SPEEX_SET_QUALITY参数任然有效。ptr参数为spx_int32_t型变量的内存指针，非0表示要使用，0表示不使用，默认为0。本参数仅对Speex编码器有效。

SPEEX_SET_VBR_QUALITY宏(0x000E)：设置Speex编码器在用动态采样频率编码时，音频的质量等级，质量等级越高音质越好、压缩率越低。ptr参数为float型变量的内存指针，取值区间为[0.0,10.0]，默认为10.0。本参数仅对Speex编码器有效。

SPEEX_GET_VBR_QUALITY宏(0x000F)：获取Speex编码器在用动态采样频率编码时，音频的质量等级，质量等级越高音质越好、压缩率越低。ptr参数为float型变量的内存指针，取值区间为[0.0,10.0]，默认为10.0。本参数仅对Speex编码器有效。

SPEEX_SET_COMPLEXITY宏(0x0010)：设置Speex编码器的复杂度，复杂度越高压缩率越高、CPU使用率越高、音质略好。ptr参数为spx_int32_t型变量的内存指针，取值区间为[0,10]，默认为2。本参数仅对Speex编码器有效。

SPEEX_GET_COMPLEXITY宏(0x0011)：获取Speex编码器的复杂度，复杂度越高压缩率越高、CPU使用率越高、音质略好。ptr参数为spx_int32_t型变量的内存指针，取值区间为[0,10]，默认为2。本参数仅对Speex编码器有效。

/** Set bit-rate used by the encoder (or lower) */

#define SPEEX_SET_BITRATE 18

/** Get current bit-rate used by the encoder or decoder */

#define SPEEX_GET_BITRATE 19

/** Define a handler function for in-band Speex request*/

#define SPEEX_SET_HANDLER 20

/** Define a handler function for in-band user-defined request*/

#define SPEEX_SET_USER_HANDLER 22

SPEEX_SET_SAMPLING_RATE宏(0x0018)：设置Speex编解码器在比特率计算时，音频的采样频率。ptr参数为spx_int32_t型变量的内存指针，窄带默认为8000，宽带默认为16000，超宽带默认为32000。本参数对Speex编解码器均有效。

SPEEX_GET_SAMPLING_RATE宏(0x0019)：获取Speex编解码器在比特率计算时，音频的采样频率。ptr参数为spx_int32_t型变量的内存指针，窄带默认为8000，宽带默认为16000，超宽带默认为32000。本参数对Speex编解码器均有效。

SPEEX_RESET_STATE宏(0x001A)：重置Speex编解码器的所有参数为初始状态。ptr参数无意义。本参数对Speex编解码器均有效。

/** Get VBR info (mostly used internally) */

#define SPEEX_GET_RELATIVE_QUALITY 29

SPEEX_SET_VAD宏(0x001E)：设置Speex编码器在用动态采样频率编码时，是否使用语音活动检测，使用后可以提升在无语音活动时动态比特率编码的压缩率。ptr参数为spx_int32_t型变量的内存指针，非0表示要使用，0表示不使用，默认为0。本参数仅对Speex编码器有效。

SPEEX_GET_VAD宏(0x001F)：获取Speex编码器在用动态采样频率编码时，是否使用语音活动检测，使用后可以提升在无语音活动时动态比特率编码的压缩率。ptr参数为spx_int32_t型变量的内存指针，非0表示要使用，0表示不使用，默认为0。本参数仅对Speex编码器有效。

/** Set Average Bit-Rate (ABR) to n bits per seconds */

#define SPEEX_SET_ABR 32

/** Get Average Bit-Rate (ABR) setting (in bps) */

#define SPEEX_GET_ABR 33

SPEEX_SET_DTX宏(0x0022)：设置是否使用Speex编码器的不连续传输，使用后可以降低网络传输的比特率。ptr参数为spx_int32_t型变量的内存指针，非0表示要使用，0表示不使用，默认为0。本参数仅对Speex编码器有效。

SPEEX_GET_DTX宏(0x0023)：获取是否使用Speex编码器的不连续传输，使用后可以降低网络传输的比特率。ptr参数为spx_int32_t型变量的内存指针，非0表示要使用，0表示不使用，默认为0。本参数仅对Speex编码器有效。

/** Set submode encoding in each frame (1 for yes, 0 for no, setting to no breaks the standard) */

#define SPEEX_SET_SUBMODE_ENCODING 36

/** Get submode encoding in each frame */

#define SPEEX_GET_SUBMODE_ENCODING 37

/*#define SPEEX_SET_LOOKAHEAD 38*/

/** Returns the lookahead used by Speex separately for an encoder and a decoder.

* Sum encoder and decoder lookahead values to get the total codec lookahead. */

#define SPEEX_GET_LOOKAHEAD 39

SPEEX_SET_PLC_TUNING宏(0x0028)：设置Speex编码器的数据包丢失隐藏的预计丢失概率，预计丢失概率越高抗网络抖动越强、压缩率越低。ptr参数为spx_int32_t型变量的内存指针，取值区间为[0,100]，默认为2。本参数仅对Speex编码器有效。

SPEEX_GET_PLC_TUNING宏(0x0029)：获取Speex编码器的数据包丢失隐藏的预计丢失概率，预计丢失概率越高抗网络抖动越强、压缩率越低。ptr参数为spx_int32_t型变量的内存指针，取值区间为[0,100]，默认为2。本参数仅对Speex编码器有效。

	/** Sets the max bit-rate allowed in VBR mode */ #define SPEEX_SET_VBR_MAX_BITRATE 42 /** Gets the max bit-rate allowed in VBR mode */ #define SPEEX_GET_VBR_MAX_BITRATE 43 **SPEEX_SET_HIGHPASS宏(0x002C)**：设置是否使用Speex编码器的**高通滤波器**。ptr参数为spx_int32_t型变量的内存指针，非0表示要使用，0表示不使用，默认为1。本参数对Speex编解码器均有效。 **SPEEX_GET_HIGHPASS宏(0x002D)**：获取是否使用Speex编码器的**高通滤波器**。ptr参数为spx_int32_t型变量的内存指针，非0表示要使用，0表示不使用，默认为1。本参数对Speex编解码器均有效。 /** Get "activity level" of the last decoded frame, i.e. how much damage we cause if we remove the frame */ #define SPEEX_GET_ACTIVITY 47
	ptr，[输入&输出]： 存放设置的参数，本参数是根据request参数来定义的。
返回值	**0**：成功。 **-1**：request参数不正确。 **-2**：无效的参数。
错误码	**无**
线程安全	多线程可以同时操作不同的Speex解码器句柄，但不能同时操作相同的Speex解码器句柄。
原子操作	否
执行速度	每毫秒能执行多少次。
其他说明	

1. **speex_decode(未完成)**

函数名称	speex_decode
头文件	#include "speex/speex.h"
库文件	#pragma comment(lib, "libspeex.lib")
函数功能	用一个Speex解码器将一个单声道16位有符号浮点型20毫秒Speex格式音频数据帧解码成PCM格式。
函数声明	int speex_decode(

	void * state, SpeexBits * bits, float * out);
函数参数	**state**，[输入]： 存放Speex解码器的句柄。
	bits，[输入]： 存放SpeexBits结构体变量的内存指针，该变量已经存放了一个单声道16位有符号浮点型20毫秒Speex格式音频数据帧，音频数据取值区间为[-1.0,1.0]。 如果本参数为NULL，则表示该音频数据帧丢失了，Speex解码器会使用包丢失隐藏算法进行猜测。
	out，[输出]： 存放解码后PCM格式音频数据帧数组的内存指针。
返回值	**0**：成功。 **-1**：音频流已经结束。 **-2**：Speex格式音频数据帧不正确。
错误码	**无**
线程安全	多线程可以同时操作不同的Speex解码器句柄，但不能同时操作相同的Speex解码器句柄。
原子操作	否
执行速度	每毫秒能执行多少次。
其他说明	调用本函数前，必须先调用speex_bits_reset()函数清空SpeexBits结构体，然后再调用speex_bits_read_from()函数将Speex格式音频数据存放到SpeexBits结构体中，最后再调用本函数进行解码。 解码后的PCM格式音频数据帧和编码前的PCM格式音频数据帧会有很大区别，只有靠听才能判断音频数据是否正确，是否符合要求。

1. **speex_decode_int(未完成)**

函数名称	speex_decode_int
头文件	#include "speex/speex.h"
库文件	#pragma comment(lib, "libspeex.lib")
函数功能	用一个Speex解码器对一个单声道16位有符号整型20毫秒Speex格式音频数据帧解码成PCM格式。

函数声明	int speex_decode_int(void * state, SpeexBits * bits, spx_int16_t * out);
函数参数	**state**，[输入]： 存放Speex解码器的句柄。
	bits，[输入]： 存放SpeexBits结构体变量的内存指针，该变量已经存放了一个单声道16位有符号整型20毫秒Speex格式音频数据帧，音频数据取值范围最高为+32767，最低为-32768。 如果本参数为NULL，则表示该音频数据帧丢失了，Speex解码器会使用包丢失隐藏算法进行猜测。
	out，[输出]： 存放解码后PCM格式音频数据帧数组的内存指针。
返回值	**0**：成功。 **-1**：音频流已经结束。 **-2**：Speex格式音频数据帧不正确。
错误码	**无**
线程安全	多线程可以同时操作不同的Speex解码器句柄，但不能同时操作相同的Speex解码器句柄。
原子操作	否
执行速度	每毫秒能执行多少次。
其他说明	调用本函数前，必须先调用speex_bits_reset()函数清空SpeexBits结构体，然后再调用speex_bits_read_from()函数将Speex格式音频数据存放到SpeexBits结构体中，最后再调用本函数进行解码。 解码后的PCM格式音频数据帧和编码前的PCM格式音频数据帧会有很大区别，只有靠听才能判断音频数据是否正确，是否符合要求。

1. **speex_decoder_destroy(未完成)**

函数名称	speex_decoder_destroy
头文件	#include "speex/speex.h"
库文件	#pragma comment(lib, "libspeex.lib")
函数功能	销毁一个Speex解码器。

函数声明	void speex_decoder_destroy(void * state);
函数参数	**state**，[输入]： 存放Speex解码器的句柄。
返回值	**无**
错误码	**无**
线程安全	多线程可以同时操作不同的Speex解码器句柄，但不能同时操作相同的Speex解码器句柄。
原子操作	否
执行速度	每毫秒能执行多少次。
其他说明	Speex解码器句柄在销毁后就不能再使用了。

1. ## Speex bits Speex格式数据流

    1. **speex_bits_advance**

void speex_bits_advance (SpeexBits * bits,

int n

)

Advances the position of the "bit cursor" in the stream

Parameters:

bits Bit-stream to operate on

n Number of bits to advance

    1. **speex_bits_destroy**

void speex_bits_destroy (SpeexBits * bits)

Frees all resources associated to a SpeexBits struct. Right now this does nothing since no resources are allocated, but this could change in the future.

    1. **speex_bits_init**

void speex_bits_init (SpeexBits * bits)

Initializes and allocates resources for a SpeexBits struct

    1. **speex_bits_init_buffer**

void speex_bits_init_buffer (SpeexBits * bits,

void * buff,

int buf_size

)

Initializes SpeexBits struct using a pre-allocated buffer

    1. **speex_bits_insert_terminator**

void speex_bits_insert_terminator (SpeexBits * bits)

Insert a terminator so that the data can be sent as a packet while auto-detecting the number of frames in each packet

Parameters:

bits Bit-stream to operate on

1. **speex_bits_nbytes**

int speex_bits_nbytes (SpeexBits * bits)

Returns the number of bytes in the bit-stream, including the last one even if it is not "full"

Parameters:

bits Bit-stream to operate on

Returns:

Number of bytes in the stream

1. **speex_bits_pack**

void speex_bits_pack (SpeexBits * bits,

int data,

int nbBits

)

Append bits to the bit-stream

Parameters:

bits Bit-stream to operate on

data Value to append as integer

nbBits number of bits to consider in "data"

1. **speex_bits_peek**

int speex_bits_peek (SpeexBits * bits)

Get the value of the next bit in the stream, without modifying the "cursor" position

Parameters:

bits Bit-stream to operate on

Returns:

Value of the bit peeked (one bit only)

1. **speex_bits_peek_unsigned**

unsigned int speex_bits_peek_unsigned (SpeexBits * bits,

int nbBits

)

Same as speex_bits_unpack_unsigned, but without modifying the cursor position

Parameters:

bits Bit-stream to operate on

nbBits Number of bits to look for

Returns:

Value of the bits peeked, interpreted as unsigned

1. **speex_bits_read_from**

void speex_bits_read_from (SpeexBits * bits,

char * bytes,

int len

)

Initializes the bit-stream from the data in an area of memory

1. **speex_bits_read_whole_bytes**

void speex_bits_read_whole_bytes (SpeexBits * bits,

char * bytes,

int len

)

Append bytes to the bit-stream

Parameters:

bits Bit-stream to operate on

bytes pointer to the bytes what will be appended

len Number of bytes of append

1. **speex_bits_remaining**

int speex_bits_remaining (SpeexBits * bits)

Returns the number of bits remaining to be read in a stream

Parameters:

bits Bit-stream to operate on

Returns:

Number of bits that can still be read from the stream

1. **speex_bits_reset**

void speex_bits_reset (SpeexBits * bits)

Resets bits to initial value (just after initialization, erasing content)

1. **speex_bits_rewind**

void speex_bits_rewind (SpeexBits * bits)

Rewind the bit-stream to the beginning (ready for read) without erasing the content

1. **speex_bits_set_bit_buffer**

void speex_bits_set_bit_buffer (SpeexBits * bits,

void * buff,

int buf_size

)

Sets the bits in a SpeexBits struct to use data from an existing buffer (for decoding without copying data)

1. **speex_bits_unpack_signed**

int speex_bits_unpack_signed (SpeexBits * bits,

int nbBits

)

Interpret the next bits in the bit-stream as a signed integer

Parameters:

bits Bit-stream to operate on

nbBits Number of bits to interpret

Returns:

A signed integer represented by the bits read

1. **speex_bits_unpack_unsigned**

unsigned int speex_bits_unpack_unsigned (SpeexBits * bits,

int nbBits

)

Interpret the next bits in the bit-stream as an unsigned integer

Parameters:

bits Bit-stream to operate on

nbBits Number of bits to interpret

Returns:

An unsigned integer represented by the bits read

1. **speex_bits_write**

int speex_bits_write (SpeexBits * bits,

char * bytes,

int max_len

)

Write the content of a bit-stream to an area of memory

Parameters:

bits Bit-stream to operate on

bytes Memory location where to write the bits

max_len Maximum number of bytes to write (i.e. size of the "bytes" buffer)

Returns:

Number of bytes written to the "bytes" buffer

1. **speex_bits_write_whole_bytes**

int speex_bits_write_whole_bytes (SpeexBits * bits,

char * bytes,

int max_len

)

Like speex_bits_write, but writes only the complete bytes in the stream. Also removes the written bytes from the stream

1. Speex echo Speex声学回音消除器

1. **speex_echo_state_init(未完成)**

函数名称	speex_echo_state_init

头文件	#include "speex/speex_echo.h"
库文件	#pragma comment(lib, "libspeex.lib")
函数功能	创建并初始化一个Speex声学回音消除器。 Speex声学回音消除器用于对16位有符号整型单声道PCM格式音频数据进行回音消除。
函数声明	SpeexEchoState * speex_echo_state_init(int frame_size, int filter_length);
函数参数	**frame_size**，[输入]： 存放一个16位有符号整型单声道PCM格式音频数据帧的数据长度，单位个采样数据，一般为10毫秒到20毫秒。 例如：8000Hz采样频率20毫秒本参数就是160。
	filter_length，[输入]： 存放声学回音消除器的过滤器长度，单位个采样数据，一般为100毫秒到500毫秒，推荐为500毫秒。 如果采样频率是8000Hz，选择300毫秒，本参数就是8000÷1000×300。 本参数具体大小要慢慢调，调的好不好直接影响声学回音消除的效果。
返回值	Speex声学回音消除器句柄。
错误码	**无**
线程安全	是
原子操作	否
执行速度	每毫秒能执行多少次。
其他说明	对一条音频流进行声学回音消除时，从开始到结束都应该用同一个Speex声学回音消除器，中途不要更换，也不要用一个Speex声学回音消除器给多条音频流进行声学回音消除，否则会导致处理后的音频数据不正确。 当Speex声学回音消除器不再使用时，必须调用speex_echo_state_destroy()函数销毁，否则会内存泄漏。

1. **speex_echo_ctl(未完成)**

函数名称	speex_echo_ctl
头文件	#include "speex/speex_echo.h"

库文件	#pragma comment(lib, "libspeex.lib")
函数功能	设置一个Speex声学回音消除器的相关参数。
函数声明	int speex_echo_ctl (SpeexEchoState * st, int request, void * ptr);
函数参数	**st**，[输入]： 存放Speex声学回音消除器句柄。
	request，[输入]： 存放需要控制的参数，可以为（选一至一个）： **SPEEX_ECHO_GET_FRAME_SIZE宏(0x0003)**：获取Speex声学回音消除在声学回音消除时，每个PCM格式音频数据帧的数据长度，单位个采样数据。ptr参数为int型变量的内存指针。 **SPEEX_ECHO_SET_SAMPLING_RATE宏(0x0018)**：设置Speex声学回音消除器处理一帧音频数据时的采样频率，单位赫兹。ptr参数为int型变量的内存指针，默认为8000。 **SPEEX_ECHO_GET_SAMPLING_RATE宏(0x0019)**：获取Speex声学回音消除器处理一帧音频数据时的采样频率，单位赫兹。ptr参数为int型变量的内存指针。 /* Can't set window sizes */ /** Get size of impulse response (int32) */ #define SPEEX_ECHO_GET_IMPULSE_RESPONSE_SIZE 27 /* Can't set window content */ /** Get impulse response (int32[]) */ #define SPEEX_ECHO_GET_IMPULSE_RESPONSE 29
	ptr，[输入&输出]： 存放控制参数。 本参数是根据request参数来定义的。
返回值	**0**：成功。 **-1**：request参数无法识别。
错误码	**无**
线程安全	多线程可以同时操作不同的Speex声学回音消除器句柄，但不能同时操作相同的Speex声学回音消除器句柄。
原子操作	否

执行速度	每毫秒能执行多少次。
其他说明	

1. **speex_echo_cancel(废弃的)**

本函数已经废弃，使用无效。

1. **speex_echo_cancellation(未完成)**

函数名称	speex_echo_cancellation
头文件	#include "speex/speex_echo.h"
库文件	#pragma comment(lib, "libspeex.lib")
函数功能	使用Speex声学回音消除器对一个单声道16位有符号整型PCM格式音频数据帧进行声学回音消除。
函数声明	void speex_echo_cancellation(SpeexEchoState * st, const spx_int16_t * rec, const spx_int16_t * play, spx_int16_t * out);
函数参数	**st**，[输入]： 存放Speex声学回音消除器句柄。
	rec，[输入]： 存放由音频输入设备录音的一帧16位有符号整型单声道PCM格式音频数据的内存指针，不能为NULL。
	play，[输入]： 存放由音频输出设备播放的一帧16位有符号整型单声道PCM格式音频数据的内存指针，不能为NULL。
	out，[输出]： 存放经过声学回音消除后的一帧16位有符号整型单声道PCM格式音频数据的内存指针，不能为NULL。
返回值	**无**
错误码	**无**
线程安全	多线程可以同时操作不同的Speex声学回音消除器句柄，但不能同时操作相同的Speex声学回音消除器句柄。

原子操作	否
执行速度	每毫秒能执行多少次。
其他说明	声学回音消除的原理就是将音频输入数据中所采集到的音频输出数据清除掉，所以要求音频输入数据和音频输出数据必须是同步的，且音频输入数据中所采集到的回音数据肯定在实际的音频输出数据之后出现。目前还没有找到不同步就可以做回音消除的方法。 在声学回音消除后，最好再使用Speex预处理器对音频输入数据进行噪音抑制、混响消除、自动增益控制、残余回音消除等操作，不要在声学回音消除前做预处理操作，否则声学回音消除效果将降低。 如果感觉回音消除效果不好，就把rec、play、out这些参数打印日志出来看看，然后调整speex_echo_state_init()函数的filter_length参数，再测试。 已知问题： 当录音里的回音的音量大于播放的音量，则本函数认为这个不是回音，就不会消除掉。这种情况主要在开了麦克风增益、或者喇叭离麦克风特别近时才会产生。 当录音里的回音和播放的声音区别较大时，则本函数认为这个不是回音，就不会消除掉。这种情况主要是麦克风或音响的音质不好造成的，大多出现在台式机，笔记本一般不会。 音频流的刚开始几秒钟内产生的声学回音可能消除不掉，因为声学回音消除算法有收敛时间。

1. **speex_echo_state_reset(未完成)**

函数名称	speex_echo_state_reset
头文件	#include "speex/speex_echo.h"
库文件	#pragma comment(lib, "libspeex.lib")
函数功能	将一个Speex声学回音消除器重置为初始状态。
函数声明	void speex_echo_state_reset （ SpeexEchoState * st);
函数参数	**st**，[输入]： 存放Speex声学回音消除器句柄。
返回值	**无**
错误码	**无**
线程安全	多线程可以同时操作不同的Speex声学回音消除器句柄，但不能同时操作相同的Speex声学回音消除器句柄。
原子操作	否

执行速度	每毫秒能执行多少次。
其他说明	

1. **speex_echo_state_destroy(未完成)**

函数名称	speex_echo_state_destroy
头文件	#include "speex/speex_echo.h"
库文件	#pragma comment(lib, "libspeex.lib")
函数功能	销毁一个Speex声学回音消除器。
函数声明	void speex_echo_state_destroy(SpeexEchoState * st);
函数参数	**st**，[输入]： 存放Speex声学回音消除器句柄。
返回值	**无**
错误码	**无**
线程安全	多线程可以同时操作不同的Speex声学回音消除器句柄，但不能同时操作相同的Speex声学回音消除器句柄。
原子操作	否
执行速度	每毫秒能执行多少次。
其他说明	Speex声学回音消除器句柄在销毁后就不能再使用了。

1. Speex preprocess Speex预处理器

1. **speex_preprocess_state_init(未完成)**

函数名称	speex_preprocess_state_init
头文件	#include "speex/speex_preprocess.h"
库文件	#pragma comment(lib, "libspeexdsp.lib")
函数功能	创建并初始化Speex预处理器。

	Speex预处理器用于对PCM格式音频数据进行预处理。
函数声明	SpeexPreprocessState * speex_preprocess_state_init (int frame_size, int sampling_rate);
函数参数	**frame_size**，[输入]： 存放音频数据一帧的大小，单位多少个采样单元。
	sampling_rate，[输入]： 存放音频数据的采样频率，单位赫兹。
返回值	Speex预处理器句柄。
错误码	**无**
线程安全	是
原子操作	否
执行速度	每毫秒能执行多少次。
其他说明	如果是对一条音频流预处理，那么预处理从开始到结束都应该用一个Speex预处理器，中途不要更换Speex预处理器，也不要用一个Speex预处理器给多条音频流预处理，否则会导致预处理后的音频数据不正确。 当Speex预处理器不再使用时，必须调用speex_preprocess_state_destroy()函数销毁预处理器，否则会内存泄漏。

1. **speex_preprocess_ctl(未完成)**

函数名称	speex_preprocess_ctl
头文件	#include "speex/speex_preprocess.h"
库文件	#pragma comment(lib, "libspeexdsp.lib")
函数功能	设置一个Speex预处理器的相关参数。
函数声明	int speex_preprocess_ctl (SpeexPreprocessState * st, int request, void * ptr);

函数参数	**st**，[输入]： 存放Speex预处理器句柄。
	request，[输入]： 存放需要控制的参数，可以为（选一至一个）： **SPEEX_PREPROCESS_SET_DENOISE宏(0x0000)**：设置是否使用Speex预处理器的噪音抑制。ptr参数为spx_int32_t型变量的内存指针，非0表示要使用，0表示不使用，默认为1。 **SPEEX_PREPROCESS_GET_DENOISE宏(0x0001)**：获取是否使用Speex预处理器的噪音抑制，ptr参数为spx_int32_t型变量的内存指针，非0表示要使用，0表示不使用。 **SPEEX_PREPROCESS_SET_AGC宏(0x0002)**：设置是否使用Speex预处理器的自动增益控制。ptr参数为spx_int32_t型变量的内存指针，非0表示要使用，0表示不使用，默认为0。 **SPEEX_PREPROCESS_GET_AGC宏(0x0003)**：获取是否使用Speex预处理器的自动增益控制。ptr参数为spx_int32_t型变量的内存指针，非0表示要使用，0表示不使用。 **SPEEX_PREPROCESS_SET_VAD宏(0x0004)**：设置是否使用Speex预处理器的语音活动检测。ptr参数为spx_int32_t型变量的内存指针，非0表示要使用，0表示不使用，默认为0。 **SPEEX_PREPROCESS_GET_VAD宏(0x0005)**：获取是否使用Speex预处理器的语音活动检测。ptr参数为spx_int32_t型变量的内存指针，非0表示要使用，0表示不使用。 **SPEEX_PREPROCESS_SET_AGC_LEVEL宏(0x0006)**：设置Speex预处理器在自动增益控制时，增益的目标等级，目标等级越大增益越大。ptr参数为float型变量的内存指针，取值区间为[1.0,32768.0]，默认为8000.0。本参数与SPEEX_PREPROCESS_SET_AGC_TARGET宏参数意义相同，区别在于ptr参数的类型。 **SPEEX_PREPROCESS_GET_AGC_LEVEL宏(0x0007)**：获取Speex预处理器在自动增益控制时，增益的目标等级，目标等级越大增益越大。ptr参数为float型变量的内存指针，取值区间为[1.0,32768.0]。本参数与SPEEX_PREPROCESS_GET_AGC_TARGET宏参数意义相同，区别在于ptr参数的类型。 **SPEEX_PREPROCESS_SET_DEREVERB宏(0x0008)**：设置是否使用Speex预处理器的混响音消除。ptr参数为spx_int32_t型变量的内存指针，非0表示要使用，0表示不使用，默认为0。 **SPEEX_PREPROCESS_GET_DEREVERB宏(0x0009)**：设置是否使用Speex预处理器的混响音消除。ptr参数为spx_int32_t型变量的内存指针，非0表示要使用，0表示不使用。 **SPEEX_PREPROCESS_SET_DEREVERB_LEVEL宏(0x000A)**：设置Speex预处理器在混响音消除时，消除的等级。ptr参数为float型变量的内存指针，本参数目前无法使用。 **SPEEX_PREPROCESS_GET_DEREVERB_LEVEL宏(0x000B)**：获取Speex预处理器在混响音消除时，消除的等级。ptr参数为float型变量的内存指针，本参数目前无法使用。 **SPEEX_PREPROCESS_SET_DEREVERB_DECAY宏(0x000C)**：设置Speex预处理器在混响音消除时，衰减的分贝值。ptr参数为float型变量的内存指针，本参数目前无法使用。 **SPEEX_PREPROCESS_GET_DEREVERB_DECAY宏(0x000C)**：获取Speex预处理器在混响音消除时，衰减的分贝值。ptr参数为float型变量的内存指针，本参数目前无法使用。 **SPEEX_PREPROCESS_SET_PROB_START宏(0x000E)**：设置Speex预处理器在语音活动检测时，从无语音活动到有语音活动的判断百分比概率，概率越大越难判断为有语音活动。ptr参数为spx_int32_t型变量的内存指针，取值区间为[0,100]，默认为34。 **SPEEX_PREPROCESS_GET_PROB_START宏(0x000F)**：获取Speex预处理器在语音活动检测时，从无语音活动到有语音活动的判断百分比概率，概率越大越难判断为有语音活动。ptr参数为

spx_int32_t型变量的内存指针，取值区间为[0,100]。

SPEEX_PREPROCESS_SET_PROB_CONTINUE宏(0x0010)：设置Speex预处理器在语音活动检测时，从有语音活动到无语音活动的判断百分比概率，概率越大越容易判断为无语音活动。ptr参数为spx_int32_t型变量的内存指针，取值区间为[0,100]，默认为20。

SPEEX_PREPROCESS_GET_PROB_CONTINUE宏(0x0011)：获取Speex预处理器在语音活动检测时，从有语音活动到无语音活动的判断百分比概率，概率越大越容易判断为无语音活动。ptr参数为spx_int32_t型变量的内存指针，取值区间为[0,100]。

SPEEX_PREPROCESS_SET_NOISE_SUPPRESS宏(0x0012)：设置Speex预处理器在噪音抑制时，噪音最大衰减的分贝值，分贝值越小衰减越大。ptr参数为spx_int32_t型变量的内存指针，取值区间为[-2147483648,0]，默认为-15。

SPEEX_PREPROCESS_GET_NOISE_SUPPRESS宏(0x0013)：获取Speex预处理器在噪音抑制时，噪音最大衰减的分贝值，分贝值越小衰减越大。ptr参数为spx_int32_t型变量的内存指针，取值区间为[-2147483648,0]。

SPEEX_PREPROCESS_SET_ECHO_SUPPRESS宏(0x0014)：设置Speex预处理器在残余回音消除时，残余回音最大衰减的分贝值，分贝值越小衰减越大。ptr参数为spx_int32_t型变量的内存指针，取值区间为[-2147483648,0]，默认为-40。

SPEEX_PREPROCESS_GET_ECHO_SUPPRESS宏(0x0015)：获取Speex预处理器在残余回音消除时，残余回音最大衰减的分贝值，分贝值越小衰减越大。ptr参数为spx_int32_t型变量的内存指针，取值区间为[-2147483648,0]。

SPEEX_PREPROCESS_SET_ECHO_SUPPRESS_ACTIVE宏(0x0016)：设置Speex预处理器在残余回音消除时，有近端语音活动时的残余回音最大衰减的分贝值，分贝值越小衰减越大。ptr参数为spx_int32_t型变量的内存指针，取值区间为[-2147483648,0]，默认为-15。

SPEEX_PREPROCESS_GET_ECHO_SUPPRESS_ACTIVE宏(0x0017)：获取Speex预处理器在残余回音消除时，有近端语音活动时的残余回音最大衰减的分贝值，分贝值越小衰减越大。ptr参数为spx_int32_t型变量的内存指针，取值区间为[-2147483648,0]。

SPEEX_PREPROCESS_SET_ECHO_STATE宏(0x0018)：设置是否使用Speex预处理器的残余回音消除，使用后应在进行Speex回音消除后再进行Speex预处理。ptr参数为SpeexEchoState *类型的Speex声学回音消除器句柄，ptr参数为NULL表示关闭残余回音消除。

SPEEX_PREPROCESS_GET_ECHO_STATE宏(0x0019)：获取是否使用Speex预处理器的残余回音消除。ptr参数为SpeexEchoState *类型的Speex声学回音消除器句柄，ptr参数为NULL表示关闭残余回音消除。

SPEEX_PREPROCESS_SET_AGC_INCREMENT宏(0x001A)：设置Speex预处理器在自动增益控制时，每秒最大增益的分贝值，分贝值越大增益越大。ptr参数为spx_int32_t型变量的内存指针，取值区间为[0,2147483647]，默认为12。

SPEEX_PREPROCESS_GET_AGC_INCREMENT宏(0x001B)：获取Speex预处理器在自动增益控制时，每秒最大增益的分贝值，分贝值越大增益越大。ptr参数为spx_int32_t型变量的内存指针，取值区间为[0,2147483647]。

SPEEX_PREPROCESS_SET_AGC_DECREMENT宏(0x001C)：设置Speex预处理器在自动增益控制时，每秒最大减益的分贝值，分贝值越小减益越大。ptr参数为spx_int32_t型变量的内存指针，取值区间为[-2147483648,0]，默认为-40。

SPEEX_PREPROCESS_GET_AGC_DECREMENT宏(0x001D)：获取Speex预处理器在自动增益控制时，每秒最大减益的分贝值，分贝值越小减益越大。ptr参数为spx_int32_t型变量的内存指针，取值区间为[-2147483648,0]。

SPEEX_PREPROCESS_SET_AGC_MAX_GAIN宏(0x001E)：设置Speex预处理器在自动增益控制时，最大增益的分贝值，分贝值越大增益越大。ptr参数为spx_int32_t型变量的内存指针，取值区间为[0,2147483647]，默认为30。

SPEEX_PREPROCESS_GET_AGC_MAX_GAIN宏(0x001F)：获取Speex预处理器在自动增益控制时，最大增益的分贝值，分贝值越大增益越大。ptr参数为spx_int32_t型变量的内存指针，取值区间为[0,2147483647]。

```
/* Can't set loudness */
/** Get loudness */
#define SPEEX_PREPROCESS_GET_AGC_LOUDNESS 33

/* Can't set gain */
/** Get current gain (int32 percent) */
#define SPEEX_PREPROCESS_GET_AGC_GAIN 35

/* Can't set spectrum size */
/** Get spectrum size for power spectrum (int32) */
#define SPEEX_PREPROCESS_GET_PSD_SIZE 37

/* Can't set power spectrum */
/** Get power spectrum (int32[] of squared values) */
#define SPEEX_PREPROCESS_GET_PSD 39

/* Can't set noise size */
/** Get spectrum size for noise estimate (int32) */
#define SPEEX_PREPROCESS_GET_NOISE_PSD_SIZE 41

/* Can't set noise estimate */
/** Get noise estimate (int32[] of squared values) */
#define SPEEX_PREPROCESS_GET_NOISE_PSD 43

/* Can't set speech probability */
/** Get speech probability in last frame (int32). */
#define SPEEX_PREPROCESS_GET_PROB 45
```

SPEEX_PREPROCESS_SET_AGC_TARGET宏(0x002E)：设置Speex预处理器在自动增益控制时，增益的目标等级，目标等级越大增益越大。ptr参数为spx_int32_t型变量的内存指针，取值区间为[1,32768]，默认为8000。本参数与SPEEX_PREPROCESS_SET_AGC_LEVEL宏参数意义相同，区别在于ptr参数的类型。

SPEEX_PREPROCESS_GET_AGC_TARGET宏(0x002F)：获取Speex预处理器在自动增益控制时，增益的目标等级，目标等级越大增益越大。ptr参数为spx_int32_t型变量的内存指针，取值区间为[1,32768]。本参数与SPEEX_PREPROCESS_SET_AGC_LEVEL宏参数意义相同，区别在于ptr参数的类型。

ptr，[输入&输出]：

存放设置参数。

	本参数是根据request参数来定义的。
返回值	**0**：成功。 **非0**：失败，request参数无效。
错误码	**无**
线程安全	多线程可以同时操作不同的Speex预处理器句柄，但不能同时操作相同的Speex预处理器句柄。
原子操作	否
执行速度	每毫秒能执行多少次。
其他说明	

1. **speex_preprocess(废弃的)**

本函数已经废弃，使用无效。

1. **speex_preprocess_run(未完成)**

函数名称	speex_preprocess_run
头文件	#include "speex/speex_preprocess.h"
库文件	#pragma comment(lib, "libspeexdsp.lib")
函数功能	用一个Speex预处理器对一个单声道16位有符号整型PCM格式音频数据帧进行预处理。
函数声明	int speex_preprocess_run (SpeexPreprocessState * st, spx_int16_t * x);
函数参数	**st**，[输入]： 存放Speex预处理器的内存指针。
	x，[输入]： 存放一个单声道16位有符号整型PCM格式音频数据帧数组的内存指针。
返回值	**1**：预处理成功，如果已使用语音活动检测，表示本音频数据帧为有语音活动。 **0**：预处理成功，如果已使用语音活动检测，表示本音频数据帧为无语音活动。
错误码	**无**
线程安全	多线程可以同时操作不同的Speex预处理器句柄，但不能同时操作相同的Speex预处理器句柄。

原子操作	否
执行速度	每毫秒能执行多少次。
其他说明	

1. **speex_preprocess_estimate_update(未完成)**

函数名称	speex_preprocess_estimate_update
头文件	#include "speex/speex_preprocess.h"
库文件	#pragma comment(lib, "libspeexdsp.lib")
函数功能	用Speex预处理器对一帧16位有符号整型单声道PCM格式音频数据进行预处理，但只对Speex预处理器进行内部状态更新，不修改音频数据。
函数声明	void speex_preprocess_estimate_update (SpeexPreprocessState * st, spx_int16_t * x);
函数参数	**st**，[输入]： 存放Speex预处理器句柄。
	x，[输入]： 存放一帧16位有符号整型单声道PCM格式音频数据的内存指针。
返回值	**无**
错误码	**无**
线程安全	多线程可以同时操作不同的Speex预处理器句柄，但不能同时操作相同的Speex预处理器句柄。
原子操作	否
执行速度	每毫秒能执行多少次。
其他说明	

1. **speex_preprocess_state_destroy(未完成)**

函数名称	speex_preprocess_state_destroy

头文件	#include "speex/speex_preprocess.h"
库文件	#pragma comment(lib, "libspeexdsp.lib")
函数功能	销毁一个Speex预处理器。
函数声明	void speex_preprocess_state_destroy (SpeexPreprocessState * st);
函数参数	**st**，[输入]： 存放Speex预处理器句柄。
返回值	**无**
错误码	**无**
线程安全	多线程可以同时操作不同的Speex预处理器句柄，但不能同时操作相同的Speex预处理器句柄。
原子操作	否
执行速度	每毫秒能执行多少次。
其他说明	Speex预处理器句柄在销毁后就不能再使用了。

1. Speex resampler Speex重采样器

1. speex_resampler_init(未完成)

函数名称	speex_resampler_init
头文件	#include "speex/speex_resampler.h"
库文件	#pragma comment(lib, "libspeexdsp.lib")
函数功能	创建并初始化一个具有整数的输入和输出采样频率的Speex重采样器。
函数声明	SpeexResamplerState * speex_resampler_init(spx_uint32_t nb_channels, spx_uint32_t in_rate, spx_uint32_t out_rate, int quality, int * err);

函数参数	**nb_channels**，[输入]： 存放要处理的声道数。
	in_rate，[输入]： 存放整数的输入采样频率，单位赫兹。
	out_rate，[输入]： 存放整数的输出采样频率，单位赫兹。
	quality，[输入]： 存放重采样后音频的质量等级，取值区间为[0,10]，该值越大音质越好。
	err，[输出]： 存放用于存放本函数错误码的变量的内存指针，可以为NULL。
返回值	**非NULL**：成功，返回值就是Speex重采样器的句柄。 **NULL**：失败，通过err参数查看错误码。
错误码	**RESAMPLER_ERR_SUCCESS枚举(0x0000)**：。 **RESAMPLER_ERR_ALLOC_FAILED枚举(0x0001)**：。 **RESAMPLER_ERR_BAD_STATE枚举(0x0002)**：。 **RESAMPLER_ERR_INVALID_ARG枚举(0x0003)**：。 **RESAMPLER_ERR_PTR_OVERLAP枚举(0x0004)**：。
线程安全	是
原子操作	否
执行速度	每毫秒能执行多少次。
其他说明	

1. **speex_resampler_init_frac(未完成)**

函数名称	speex_resampler_init_frac
头文件	#include "speex/speex_resampler.h"
库文件	#pragma comment(lib, "libspeexdsp.lib")
函数功能	创建并初始化一个具有小数输入和输出采样频率的重采样器。采样频率比是任意有理数，分子和分母都是32位整数。
函数声明	SpeexResamplerState * speex_resampler_init_frac(spx_uint32_t nb_channels,

	spx_uint32_t ratio_num, spx_uint32_t ratio_den, spx_uint32_t in_rate, spx_uint32_t out_rate, int quality, int * err);
函数参数	**nb_channels**，[输入]： 存放要处理的声道数。
	ratio_num，[输入]： 存放采样频率比的32位整数分子。
	ratio_den，[输入]： 存放采样频率比的32位整数分母。
	in_rate，[输入]： 存放输入采样频率四舍五入到最接近的整数，单位赫兹。
	out_rate，[输入]： 存放输出采样频率四舍五入到最接近的整数，单位赫兹。
	quality，[输入]： 存放重采样后的质量等级，取值区间为[0,10]，该值越大音质越好。
	err，[输出]： 存放用于存放本函数错误码的变量的内存指针，可以为NULL。
返回值	**非NULL**：成功，返回值就是Speex重采样器的句柄。 **NULL**：失败，通过err参数查看错误码。
错误码	**RESAMPLER_ERR_SUCCESS枚举(0x0000)**：成功。 **RESAMPLER_ERR_ALLOC_FAILED枚举(0x0001)**：。 **RESAMPLER_ERR_BAD_STATE枚举(0x0002)**：。 **RESAMPLER_ERR_INVALID_ARG枚举(0x0003)**：。 **RESAMPLER_ERR_PTR_OVERLAP枚举(0x0004)**：。
线程安全	是
原子操作	否
执行速度	每毫秒能执行多少次。
其他说明	

1. **speex_resampler_process_float(未完成)**

函数名称	speex_resampler_process_float
头文件	#include "speex/speex_resampler.h"
库文件	#pragma comment(lib, "libspeexdsp.lib")
函数功能	用一个Speex重采样器对一个单或多声道32位有符号浮点型PCM格式音频数据帧进行重采样。
函数声明	int speex_resampler_process_float(SpeexResamplerState * st, spx_uint32_t channel_index, const float * in, spx_uint32_t * in_len, float * out, spx_uint32_t * out_len);
函数参数	**st**，[输入]： 存放Speex重采样器的句柄。
	channel_index，[输入]： 存放声道的索引号，从0开始。 如果是单声道，本参数就填0。
	in，[输入]： 存放重采样前的一个单或多声道32位有符号浮点型PCM格式音频数据帧数组的内存指针。
	in_len，[输入&输出]： 输入时，存放用于存放重采样前的音频数据帧数组长度的变量的内存指针，单位个采样数据。 输出时，存放已处理多少个重采样前的音频采样数据。
	out，[输出]： 存放重采样后的一个单或多声道32位有符号浮点型PCM格式音频数据帧数组的内存指针。
	out_len，[输入&输出]： 输入时，存放用于存放重采样后的音频数据帧数组长度的变量的内存指针，单位个采样数据。 输出时，存放已写入多少个重采样后的音频采样数据。
返回值	**RESAMPLER_ERR_SUCCESS枚举(0x0000)**：成功。 **RESAMPLER_ERR_ALLOC_FAILED枚举(0x0001)**：失败，内存分配失败。 **RESAMPLER_ERR_BAD_STATE枚举(0x0002)**：失败，错误的Speex重采样器句柄。 **RESAMPLER_ERR_INVALID_ARG枚举(0x0003)**：失败，错误的参数。

	RESAMPLER_ERR_PTR_OVERLAP枚举(0x0004)：失败。
错误码	返回值就是错误码
线程安全	多线程可以同时操作不同的Speex重采样器句柄，但不能同时操作相同的Speex重采样器句柄。
原子操作	否
执行速度	每毫秒能执行多少次。
其他说明	重采样前的和重采样后的音频数据帧数组的内存不能重叠。

1. **speex_resampler_process_int(未完成)**

函数名称	speex_resampler_process_int
头文件	#include "speex/speex_resampler.h"
库文件	#pragma comment(lib, "libspeexdsp.lib")
函数功能	用一个Speex重采样器对一个单或多声道16位有符号整型PCM格式音频数据帧进行重采样。
函数声明	int speex_resampler_process_int(SpeexResamplerState * st, spx_uint32_t channel_index, const spx_int16_t * in, spx_uint32_t * in_len, spx_int16_t * out, spx_uint32_t * out_len);
函数参数	**st**，[输入]： 存放Speex重采样器的句柄。
	channel_index，[输入]： 存放声道的索引号，从0开始。 如果是单声道，本参数就填0。
	in，[输入]： 存放重采样前的一个单或多声道16位有符号整型PCM格式音频数据帧数组的内存指针。
	in_len，[输入&输出]： 输入时，存放用于存放重采样前的音频数据帧数组长度的变量的内存指针，单位个采样数据。 输出时，存放已处理多少个重采样前的音频采样数据。

	out，[输出]： 存放重采样后的一个单或多声道16位有符号整型PCM格式音频数据帧数组的内存指针。
	out_len，[输入&输出]： 输入时，存放用于存放重采样后的音频数据帧数组长度的变量的内存指针，单位个采样数据。 输出时，存放已写入多少个重采样后的音频采样数据。
返回值	**RESAMPLER_ERR_SUCCESS枚举(0x0000)**：成功。 **RESAMPLER_ERR_ALLOC_FAILED枚举(0x0001)**：失败。 **RESAMPLER_ERR_BAD_STATE枚举(0x0002)**：失败。 **RESAMPLER_ERR_INVALID_ARG枚举(0x0003)**：失败。 **RESAMPLER_ERR_PTR_OVERLAP枚举(0x0004)**：失败。
错误码	返回值就是错误码
线程安全	多线程可以同时操作不同的Speex重采样器句柄，但不能同时操作相同的Speex重采样器句柄。
原子操作	否
执行速度	每毫秒能执行多少次。
其他说明	重采样前的和重采样后的音频数据帧数组的内存不能重叠。

1. **speex_resampler_process_interleaved_float(未完成)**

函数名称	speex_resampler_process_interleaved_float
头文件	#include "speex/speex_resampler.h"
库文件	#pragma comment(lib, "libspeexdsp.lib")
函数功能	/** Resample an interleaved float array. The input and output buffers must *not* overlap. * @param st Resampler state * @param in Input buffer * @param in_len Number of input samples in the input buffer. Returns the number * of samples processed. This is all per-channel. * @param out Output buffer * @param out_len Size of the output buffer. Returns the number of samples written. * This is all per-channel. */
函数声明	int speex_resampler_process_interleaved_float(SpeexResamplerState *st, const float *in, spx_uint32_t *in_len,

	float *out, spx_uint32_t *out_len); 类型 函数名(类型 参数1, 类型 参数2, ...);
函数参数	**参数1**，[输入|输出|输入&输出]： 参数说明。
	参数2，[输入|输出|输入&输出]： 参数说明。
	.....
返回值	**返回值1**：返回值说明。 **返回值2**：返回值说明。 .....
错误码	**EXXXX**：错误码说明。 **EXXXX**：错误码说明。 .....
线程安全	是 或 否 或 未知，表示此函数多线程调用是否会产生影响
原子操作	是 或 否 或 未知，表示此函数是否是单一操作，不是多个步骤的组合
执行速度	每毫秒能执行多少次。
其他说明	....

1. **speex_resampler_process_interleaved_int(未完成)**

函数名称	speex_resampler_process_interleaved_int
头文件	#include "speex/speex_resampler.h"
库文件	#pragma comment(lib, "libspeexdsp.lib")
函数功能	/** Resample an interleaved int array. The input and output buffers must *not* overlap. * @param st Resampler state * @param in Input buffer * @param in_len Number of input samples in the input buffer. Returns the number

	* of samples processed. This is all per-channel. * @param out Output buffer * @param out_len Size of the output buffer. Returns the number of samples written. * This is all per-channel. */
函数声明	int speex_resampler_process_interleaved_int(SpeexResamplerState *st, const spx_int16_t *in, spx_uint32_t *in_len, spx_int16_t *out, spx_uint32_t *out_len); 类型 函数名(类型 参数1, 类型 参数2, ...);
函数参数	**参数1**，[输入|输出|输入&输出]： 参数说明。
	参数2，[输入|输出|输入&输出]： 参数说明。
	.....
返回值	**返回值1**：返回值说明。 **返回值2**：返回值说明。 .....
错误码	**EXXXX**：错误码说明。 **EXXXX**：错误码说明。 .....
线程安全	是 或 否 或 未知，表示此函数多线程调用是否会产生影响
原子操作	是 或 否 或 未知，表示此函数是否是单一操作，不是多个步骤的组合
执行速度	每毫秒能执行多少次。
其他说明	....

1. **speex_resampler_skip_zeros(未完成)**

函数名称	speex_resampler_skip_zeros
头文件	#include "speex/speex_resampler.h"
库文件	#pragma comment(lib, "libspeexdsp.lib")
函数功能	Make sure that the first samples to go out of the resamplers don't have leading zeros. This is only useful before starting to use a newly created resampler. It is recommended to use that when resampling an audio file, as it will generate a file with the same length. For real-time processing, it is probably easier not to use this call (so that the output duration is the same for the first frame).
函数声明	int speex_resampler_skip_zeros(SpeexResamplerState * st);
函数参数	**st**，[输入]： 存放Speex重采样器的句柄。
返回值	**RESAMPLER_ERR_SUCCESS枚举(0x0000)**：成功。 **RESAMPLER_ERR_ALLOC_FAILED枚举(0x0001)**：失败。 **RESAMPLER_ERR_BAD_STATE枚举(0x0002)**：失败。 **RESAMPLER_ERR_INVALID_ARG枚举(0x0003)**：失败。 **RESAMPLER_ERR_PTR_OVERLAP枚举(0x0004)**：失败。
错误码	返回值就是错误码
线程安全	多线程可以同时操作不同的Speex重采样器句柄，但不能同时操作相同的Speex重采样器句柄。
原子操作	否
执行速度	每毫秒能执行多少次。
其他说明	调用本函数后会导致重采样后的音频数据帧数组数据长度比应该要写入的少，比如重采样前是8000Hz的160个采样数据，重采样后应该为16000Hz的320个采样数据，但实际可能只有64个，如果不调用本函数，则重采样后就是320个采样数据，所以建议不要调用本函数。

1. **speex_resampler_reset_mem(未完成)**

函数名称	speex_resampler_reset_mem
头文件	#include "speex/speex_resampler.h"
库文件	#pragma comment(lib, "libspeexdsp.lib")
函数功能	Reset a resampler so a new (unrelated) stream can be processed.

函数声明	int speex_resampler_reset_mem(SpeexResamplerState * st);
函数参数	**st**，[输入]： 存放Speex重采样器的句柄。
返回值	**RESAMPLER_ERR_SUCCESS枚举(0x0000)**：成功。 **RESAMPLER_ERR_ALLOC_FAILED枚举(0x0001)**：失败。 **RESAMPLER_ERR_BAD_STATE枚举(0x0002)**：失败。 **RESAMPLER_ERR_INVALID_ARG枚举(0x0003)**：失败。 **RESAMPLER_ERR_PTR_OVERLAP枚举(0x0004)**：失败。
错误码	返回值就是错误码
线程安全	多线程可以同时操作不同的Speex重采样器句柄，但不能同时操作相同的Speex重采样器句柄。
原子操作	否
执行速度	每毫秒能执行多少次。
其他说明	

1. **speex_resampler_destroy(未完成)**

函数名称	speex_resampler_destroy
头文件	#include "speex/speex_resampler.h"
库文件	#pragma comment(lib, "libspeexdsp.lib")
函数功能	销毁一个Speex重采样器。
函数声明	void speex_resampler_destroy(SpeexResamplerState * st);
函数参数	**st**，[输入]： 存放Speex重采样器的句柄。
返回值	**无**
错误码	**无**

线程安全	多线程可以同时操作不同的Speex重采样器句柄，但不能同时操作相同的Speex重采样器句柄。
原子操作	否
执行速度	每毫秒能执行多少次。
其他说明	Speex重采样器句柄在销毁后就不能再使用了。

1. 自适应抖动缓冲器

本人认为效果不好，没有再使用了。

1. Speex专用自适应抖动缓冲器

本人认为效果不好，没有再使用了。

1. 结构体库

1. 结构体模板(未完成)

结构体名称	xxx
头文件	#include "speex/speex.h"
结构体称呼	结构体的中文称呼。
结构体说明	结构体主要用途说明。
相关函数	Func1()、Func2()、Func3()...

结构体声明	struct xxx
	{
	类型 成员变量1;
	类型 成员变量2;
	.....
	};

成员变量	**成员变量1**： 成员变量说明。
	成员变量2： 成员变量说明。
	.....

其他说明	…… ……

1. SpeexMode(未完成)

结构体名称	SpeexMode
头文件	#include "speex/speex.h"
结构体称呼	Speex格式配置结构体。
结构体说明	用于指定编解码器在压缩和解压缩时的相关参数。
相关函数	speex_encoder_init()、Func2()、Func3()…
结构体声明	typedef struct SpeexMode { const void * mode; /** Pointer to the low-level mode data */ mode_query_func query; /** Pointer to the mode query function */ const char * modeName; int modeID; int bitstream_version; encoder_init_func enc_init; encoder_destroy_func enc_destroy; encode_func enc; decoder_init_func dec_init; decoder_destroy_func dec_destroy; decode_func dec; encoder_ctl_func enc_ctl; decoder_ctl_func dec_ctl; } SpeexMode;
成员变量	**mode**： 成员变量说明。
	query： 存放Speex格式配置的查询函数的内存指针。 如果是speex_nb_mode静态结构体，本参数为nb_mode_query()函数的内存指针。 如果是speex_wb_mode静态结构体，本参数为wb_mode_query()函数的内存指针。

	如果是speex_uwb_mode静态结构体，本参数为wb_mode_query()函数的内存指针。
	modeName： 存放Speex格式配置的名称字符串。 如果是speex_nb_mode静态结构体，本参数为"narrowband"。 如果是speex_wb_mode静态结构体，本参数为"wideband (sub-band CELP)"。 如果是speex_uwb_mode静态结构体，本参数为"ultra-wideband (sub-band CELP)"。
	modeID： 存放Speex格式配置的ID值。 如果是speex_nb_mode静态结构体，本参数为0。 如果是speex_wb_mode静态结构体，本参数为1。 如果是speex_uwb_mode静态结构体，本参数为2。
	bitstream_version： 存放bitstream的版本号，版本越高数值越大，主要为了提升兼容性而升级。
	enc_init： 存放Speex格式配置的编码器初始化函数的内存指针。
	enc_destroy： 存放Speex格式配置的编码器销毁函数的内存指针。
	enc： 存放Speex格式配置的编码器压缩函数的内存指针。
	dec_init： 存放Speex格式配置的解码器初始化函数的内存指针。
	dec_destroy： 存放Speex格式配置的解码器销毁函数的内存指针。
	dec： 存放Speex格式配置的解码器解压缩函数的内存指针。
	enc_ctl： 存放Speex格式配置的编码器控制函数的内存指针。
	dec_ctl： 存放Speex格式配置的解码器控制函数的内存指针。
其他说明	

赤勇玄心行天道
粉丝 - 15 关注 - 1
+加关注

« 上一篇： 必须要做的事
» 下一篇： LibOciLib使用说明（2017-1-26更新）

posted @ 2015-12-09 15:15 赤勇玄心行天道 阅读(40766) 评论(2) 编辑 收藏 举报