CereProc CereVoice SDK User Guide

Chris Pidcock

Contents

Introduction **1**

Installation **2**

Prerequisites **3**

The CereVoice Engine API **4**
- Recommended Input 4
- Simple API 4
  - Simple API Examples 4
- Engine API 5
  - Adjusting Memory Usage with Engine API Load Configurations 5
  - Engine API Examples 6

Example Programs **12**
- C Examples 12
  - Building the Applications Under Linux, OS X, or Cygwin 12
  - Building the Applications Using Visual Studio 12
- Python Examples 12
- iPhone OS (iOS) Example 13
  - Building With XCode 13

Audio API **14**
- Audio API Examples 14
  - Additional Audio API Calls 14

Software Dependencies **15**
- CereVoice Engine Library Dependencies 15
- CereVoice Audio Library Dependencies 15
- Licenses for 3rd Party Libraries 15

Speech Synthesis Markup Language (SSML) support **16**
- Supported Tags 16

CereProc Tag Set **17**
- Variant Tags 17
- Vocal Gestures 17
- Emotion Tags 17
  - Happy Emotion Tag 17
  - Sad Emotion Tag 17
  - Calm Emotion Tag 18
  - Cross Emotion Tag 18

Obtaining Support **19**
- Support Requests 19
- Direct Email 19

Appendix 1 .... **20**
- List of vocal gesture IDs 20

Contents

Appendix 2....................22
CereVoice English RP (Southern English) Phone Set....................22
CereVoice English SC (Scottish) Phone Set....................23
CereVoice English GA (General American) Phone Set....................24
CereVoice German Phone Set....................25
CereVoice Spanish Phone Set....................26
CereVoice French Phone Set....................27
CereVoice Italian Phone Set....................29

Appendix 3....................31
Change Log....................31

Introduction

The CereVoice SDK is a text-to-speech (TTS) software development kit for Windows, Mac OS X and Linux. It can be used to speech enable applications via a C, C++ or Python interface. The SDK provides the CereVoice Engine API (`cerevoice_eng`) for for use by developers, and a set of example programs. 32-bit and 64-bit applications are supported on all platforms.

Please see the accompanying `LICENSE` file for terms of use.

Installation

The CereVoice SDK does not need to be installed in a specific location. Extract the library package (on Windows, a zip file is provided, on Linux and Mac OS X the package is a gzipped tar file).

The package extracts to a new directory containing the following subdirectories:

- *cerevoice_eng* - CereVoice engine library, allows users to generate speech data via the CereVoice Engine API
- *cerevoice_aud* - CereVoice audio library, allows users to play back audio data (not available on iOS and Android)
- *docs* - full API documentation (HTML) for *cerevoice_eng* and *cerevoice_aud*
- *examples* - example code in C and Python
- *example_data* - example input files
- CereVoice dependencies, for advanced use only:
    - *cerehts* - CereVoice HTS synthesis library
    - *cerevoice* - CereVoice unit selection synthesis library
    - *cerevoice_pmod* - CereVoice pitch/rate modification library

The package contains the files:

- `QUICK_INSTALL` - quick start guide
- `CereVoiceSdkGuide.pdf` - this document
- `LICENSE_3RDPARTY` - licenses for 3rd party tools, this file should be shipped with products based on CereVoice

Prerequisites

Before generating TTS output, ensure that at least one voice file (e.g. *cerevoice_heather_3.0.0_16k.voice*) has been downloaded, and that the license key for the voice has been saved to a file. The example commands in this document refer to generic *speaker.voice* and *license.lic* files, replace these values with the voice and license that CereProc has supplied.

If you do not have access to a voice or a license key, please use the support request link to contact CereProc: http://www.cereproc.com/support/support_request

The CereVoice Engine API

Full API documentation, with descriptions of all available function calls, can be viewed by opening the `docs/cerevoice_eng/index.html` file in a web browser. This document contains descriptions and code snippets for the most commonly used features of the engine.

Recommended Input

CereProc recommends the use of XML input documents, especially the W3C standard SSML (see http://www.w3.org/TR/speech-synthesis). XML markup is required for some functionality, such as markers and emotional synthesis. Text input is supported, note that XML markup within a text document may not be processed correctly.

Simple API

The simple API, defined in `cerevoice_eng/include/cerevoice_eng_simp.h`, can be used to speak text or XML to a file or a memory buffer. For more advanced functionality, such as incremental (low-latency) processing, multiple voices, or parsing events such as markers or phonemes, use the Engine API.

The *engine* (`CPRCEN_engine`) is responsible for managing synthesis voices, and can also be used for simple synthesis requests.

Simple API Examples

Run the Python example from `cerevoice_eng/pylib`, or ensure that the full path to the `cerevoice_eng/pylib` directory is on the Python path. On OS X Snow Leopard, Python version 2.5 must be used (invoke with `python2.5` on the command line).

Synthesise to a Wave File

C:

```
#include <cerevoice_eng_simp.h>
CPRCEN_engine * eng = CPRCEN_engine_load("license.lic", "speaker.voice");
CPRCEN_engine_speak_to_file(eng, "Testing 1 2 3", "out.wav");
CPRCEN_engine_delete(eng);
```

Python:

```
from cerevoice_eng import *
eng = CPRCEN_engine_load("license.lic", "speaker.voice")
CPRCEN_engine_speak_to_file(eng, "Testing 1 2 3", "out.wav")
CPRCEN_engine_delete(eng)
```

Synthesise to a buffer

C (this example is not appropriate for Python):

```
#include <cerevoice_eng_simp.h>
CPRCEN_engine * eng = CPRCEN_engine_load("license.lic", "speaker.voice");
CPRCEN_wav * sound = CPRCEN_engine_speak(eng, "Testing 1 2 3");
CPRCEN_engine_delete(eng);
```

The audio data can then be processed by the user. The `CPRCEN_wav` structure is defined as:

```
struct CPRCEN_wav {
  short * wavdata; /** Linear PCM short data pointer */
  int size; /** Size of the wavdata */
  int sample_rate; /** Sample rate of the output audio */
};
```

Simple Synthesis With Audio API Playback

C (this example is not appropriate for Python due to the use of pointers):

```
#include <cerevoice_eng_simp.h>
#include <cerevoice_aud.h>
CPRCEN_engine * eng = CPRCEN_engine_load("license.lic", "speaker.voice");
CPRCEN_wav * sound = CPRCEN_engine_speak(eng, "Testing 1 2 3");
/* Create an audio player */
CPRC_sc_player * player = CPRC_sc_player_new(sound->sample_rate);
/* Create a buffer containing audio for the player, the buffer
   is deleted after playback finishes. */
CPRC_sc_audio * buf = CPRC_sc_audio_short_disposable(sound->wavdata, sound->size);
/* Cue the audio and start playing */
CPRC_sc_audio_cue(player, buf);
/* Wait for playback to finish before cleaning up */
while (CPRC_sc_audio_busy(player)) CPRC_sc_sleep_msecs(50);
CPRC_sc_player_delete(player);
CPRCEN_engine_delete(eng);
```

Engine API

The Engine API is defined in `cerevoice_eng/include/cerevoice_eng.h`. As with the Simple API, the *engine* (`CPRCEN_engine`) is responsible for managing synthesis voices. In most cases an application should have one engine, with multiple voices of any language loaded into the same engine. The Engine API uses the concept of a synthesis *channel* (`CPRCEN_channel_handle`), which is associated with a particular voice. A *callback* function can be used to process synthesis requests incrementally, allowing large documents to be synthesised with low latency.

Individual channels are not thread safe. If using channels in a multi-threaded environment, always create a new channel for each synthesis thread. Sharing a channel handle between threads will lead to undefined behaviour. Channels can be closed with `CPRCEN_engine_channel_close` (note that `CPRCEN_engine_delete` will also clean up any open channels).

Adjusting Memory Usage with Engine API Load Configurations

Voices can be loaded using three different *load types*. These parameters can be used to modify the amount of RAM that is required to run a voice. They allow high-quality unit selection voices to run on small footprint devices. The load types are passed as the final parameter to the configurable engine load function:

```
CPRCEN_engine * eng = CPRCEN_engine_new(); /* New empty engine */
# Load voice, NULL argument can be replaced with "" in the Python API
CPRCEN_engine_load_voice(eng, "license.lic", NULL, "speaker.voice", CPRC_VOICE_LOAD)
```

Load Type	RAM Usage	Recommended For	Description
CPRC_VOICE_LOAD	~150Mb	Long running server-type applications	All voice data is loaded to RAM, produces the fastest synthesis but is impractical for small footprint devices

CPRC_VOICE_LOAD_EMB_AUDIO	~40Mb	Desktop applications	Audio data is read from disk, lowering the RAM footprint with a minor impact on performance
CPRC_VOICE_LOAD_EMB	~10Mb	Embedded devices	Audio and index data is read from disk, lowering the footprint further with a small impact on performance

Loading a User Lexicon

Custom lexicons containing user-specified pronunciations can be loaded in to the engine (an example lexicon file for British RP English is supplied in the SDK package in the `example_data/additional.lex` file). An integer voice index is required to load the lexicon. If only one voice is loaded, this will always be *0*:

```
CPRCEN_engine_load_user_lexicon(eng, 0, "example_data/additional.lex";
```

If multiple voices are loaded, the user lexicon should be loaded into each voice. Note that each accent has a different phonetic description (see the phone set listings in Appendix 2).

For more information about the voice index, see the Using Multiple Voices section.

Engine API Examples

Run the Python examples from `cerevoice_eng/pylib`, or ensure that the full path to the `cerevoice_eng/pylib` directory is on the Python path. On OS X Snow Leopard, Python version 2.5 must be used (invoke with `python2.5` on the command line).

Synthesise to a Wave File

Synthesise to a Wave File Using a Channel

Use the *channel* to synthesise, and process the text input in chunks.

C:

```
#include <cerevoice_eng.h>
char txt1[] =  "Testing";
char txt2[] =  " 1 ";
char txt3[] =  "2 3";
CPRCEN_engine * eng = CPRCEN_engine_load("license.lic", "speaker.voice"); /* Load */
CPRCEN_channel_handle chan = CPRCEN_engine_open_default_channel(eng); /* Create channel */
CPRCEN_engine_channel_to_file(eng, chan, "out.wav", CPRCEN_RIFF); /* File output on channel */
/* Speak with streaming input */
CPRCEN_engine_channel_speak(eng, chan, txt1, strlen(txt1), 0);
CPRCEN_engine_channel_speak(eng, chan, txt2, strlen(txt2), 0);
/* Speak with flush (final argument is TRUE) */
CPRCEN_engine_channel_speak(eng, chan, txt3, strlen(txt3), 1);
CPRCEN_engine_delete(eng); /* Clean up */
```

Python:

```
from cerevoice_eng import *
txt1 =  "Testing"
txt2 = " 1 "
txt3 = "2 3"
eng = CPRCEN_engine_load("license.lic", "speaker.voice")
```

```
chan = CPRCEN_engine_open_default_channel(eng)
CPRCEN_engine_channel_to_file(eng, chan, "out.wav", CPRCEN_RIFF)
# Speak with streaming input
CPRCEN_engine_channel_speak(eng, chan, txt1, strlen(txt1), 0)
CPRCEN_engine_channel_speak(eng, chan, txt2, strlen(txt2), 0)
# Speak with flush (final argument)
CPRCEN_engine_channel_speak(eng, chan, txt1, len(txt1), 1)
CPRCEN_engine_delete(eng)
```

Synthesise to a Buffer Using a Channel

C:

```
#include <cerevoice_eng.h>
char txt1[] =  "Testing.";
char txt2[] =  "1 2 3";
CPRCEN_engine * eng = CPRCEN_engine_load("license.lic", "speaker.voice"); /* Load */
CPRCEN_channel_handle chan = CPRCEN_engine_open_default_channel(eng); /* Create channel */
/* Speak the first piece of text, with flush */
abuf = CPRCEN_engine_channel_speak(eng, chan, txt1, strlen(txt1), 1);
/* The user can now process the audio buffer as required */
/* Audio is appended to the buffer unless it is cleared.
   This clears the engine's internal callback. */
CPRCEN_engine_clear_callback(eng, chan);
/* Speak the second piece of text, with flush */
abuf = CPRCEN_engine_channel_speak(eng, chan, txt2, strlen(txt2), 1);
/* The user can now process the audio buffer as required */
CPRCEN_engine_delete(eng); /* Clean up */
```

Python:

```
from cerevoice_eng import *
txt1 =  "Testing."
txt2 = "1 2 3"
eng = CPRCEN_engine_load("license.lic", "speaker.voice")
chan = CPRCEN_engine_open_default_channel(eng)
# Speak the first piece of text, with flush
abuf = CPRCEN_engine_channel_speak(eng, chan, txt1, strlen(txt1), 1)
# The user can now process the audio buffer as required
# Audio is appended to the buffer unless it is cleared.
# This clears the engine's internal callback.
CPRCEN_engine_clear_callback(eng, chan)
# Speak the second piece of text, with flush
abuf = CPRCEN_engine_channel_speak(eng, chan, txt2, strlen(txt2), 1)
# The user can now process the audio buffer as required
CPRCEN_engine_delete(eng)
```

Append to a Wave File Using a Callback

The callback function, if set, is fired for every phrase returned by the synthesiser.

C:

```
#include <cerevoice_eng.h>
/* A simple example callback function, appends audio to a file. */
/* Callback functions must accept these arguments. */
void channel_callback(CPRC_abuf * abuf, void * userdata) {
   /* The void pointer should be cast by the user to their chosen
   data type, here it is a string containing the output file name. */
   char * f = (char *) userdata;
   CPRC_riff_append(abuf, f);
}
```

```
char txt[] =  "Testing 1 2 3";
char * outfile = "out.wav";
CPRCEN_engine * eng = CPRCEN_engine_load("license.lic", "speaker.voice");
CPRCEN_channel_handle chan = CPRCEN_engine_open_default_channel(eng);
/* When setting the callback, the user data must be cast to void, the
   user casts it back in the callback function. */
CPRCEN_engine_set_callback(eng, chan, (void *)outfile, channel_callback);
CPRCEN_engine_channel_speak(eng, chan, txt, strlen(txt), 1);
CPRCEN_engine_delete(eng); /* Clean up */
```

Python (note that the Python callback configuration is slightly different to C. A Python-specific function `engine_set_callback` is used, and the user data must be set on a callback class):

```
from cerevoice_eng import *
# A simple callback class that appends audio to a file
class CallbackExample:
    # The user can add their own data to the callback class
    def __init__(self, userdata):
        self.outfile = userdata
    # The callback function must be called 'channel_callback'
    def channel_callback(self, data):
        abuf = data_to_abuf(data)
        CPRC_riff_append(abuf, self.outfile)

txt =  "Testing 1 2 3"
outfile = "out.wav"
eng = CPRCEN_engine_load("license.lic", "speaker.voice")
chan = CPRCEN_engine_open_default_channel(eng)
callback = CallbackExample(outfile)
engine_set_callback(eng, chan, callback)
CPRCEN_engine_channel_speak(eng, chan, txt, len(txt), 1)
CPRCEN_engine_delete(eng)
```

Synthesise and Play with Low Latency using the Audio API

Note that the Audio API is not available on iOS and Android. Native audio playback should be used on these platforms (see the platform-specific example applications for more information).

C:

```
#include <cerevoice_aud.h>
#include <cerevoice_eng.h>
/* A simple example callback function which plays back audio. The
   callback is fired for each phrase, so audio playback can start
   as soon as the first phrase has been synthesised.
*/
/* Callback functions must accept these arguments. */
void channel_callback(CPRC_abuf * abuf, void * userdata) {
   /* The void pointer should be cast by the user to their chosen
      data type, here it is an audio player. */
   CPRC_sc_player * player = (CPRC_sc_player * player) userdata;
   /* Disposable audio buffer, cleaned up automatically after playback. */
   CPRC_sc_audio * buf = CPRC_sc_audio_short_disposable(CPRC_abuf_wav_data(abuf),
                                                        CPRC_abuf_wav_sz(abuf));
   /* Cue up the audio, it will be played when other audio has finished,
      or immediately if no audio is playing. */
   CPRC_sc_audio_cue(player, buf);
}
char txt[] =  "Testing 1 2 3. Using a callback, each phrase is returned separately.";
CPRCEN_engine * eng = CPRCEN_engine_load("license.lic", "speaker.voice");
CPRCEN_channel_handle chan = CPRCEN_engine_open_default_channel(eng);
int freq = atoi(CPRCEN_channel_get_voice_info(eng, chan, "SAMPLE_RATE"));
CPRC_sc_player * player = CPRC_sc_player_new(freq);
```

```
/* When setting the callback, the user data must be cast to void, the
   user casts it back in the callback function. */
CPRCEN_engine_set_callback(eng, chan, (void *)player, channel_callback);
CPRCEN_engine_channel_speak(eng, chan, txt, strlen(txt), 1);
/* Wait for playback to finish before cleaning up */
while (CPRC_sc_audio_busy(player)) CPRC_sc_sleep_msecs(50);
CPRC_sc_player_delete(player); /* Clean up */
CPRCEN_engine_delete(eng);
```

Python:

```
from cerevoice_aud import *
from cerevoice_eng import *
# A simple example callback class which plays back audio. The
# callback is fired for each phrase, so audio playback can start
# as soon as the first phrase has been synthesised.
class CallbackExample:
    # The user can add their own data to the callback class
    def __init__(self, userdata):
        self.player = userdata
    # The callback function must be called 'channel_callback'
    def channel_callback(self, data):
        abuf = data_to_abuf(data)
        # Disposable audio buffer, cleaned up automatically after playback.
        buf = CPRC_sc_audio_short_disposable(CPRC_abuf_wav_data(abuf),
                                             CPRC_abuf_wav_sz(abuf))
        # Cue up the audio, it will be played when other audio has finished,
        # or immediately if no audio is playing.
        CPRC_sc_audio_cue(self.player, buf)

txt =  "Testing 1 2 3. Using a callback, each phrase is returned separately."
eng = CPRCEN_engine_load("license.lic", "speaker.voice")
chan = CPRCEN_engine_open_default_channel(eng)
freq = int(CPRCEN_channel_get_voice_info(eng, chan, "SAMPLE_RATE"))
player = CPRC_sc_player_new(freq)
callback = CallbackExample(player)
engine_set_callback(eng, chan, callback)
CPRCEN_engine_channel_speak(eng, chan, txt, len(txt), 1)
while CPRC_sc_audio_busy(player):
    CPRC_sc_sleep_msecs(50)
CPRCEN_engine_delete(eng)
```

Synthesise and Display Speech Events

C:

```
#include <cerevoice_eng.h>
/* An example callback function, prints information */
void channel_callback(CPRC_abuf * abuf, void * userdata) {
    /* Transcriptions contain markers, phonetic information, a
       list of these items is available for each audio buffer. */
    const CPRC_abuf_trans * trans;
    const char * name;
    float start, end;
    /* Process the transcription buffer items and print information. */
    for(int i = 0; i < CPRC_abuf_trans_sz(abuf); i++) {
        trans = CPRC_abuf_get_trans(abuf, i);
        start = CPRC_abuf_trans_start(trans); /* Start time in seconds */
        end = CPRC_abuf_trans_end(trans); /* End time in seconds */
        name = CPRC_abuf_trans_name(trans); /* Label, type dependent */
        if (CPRC_abuf_trans_type(trans) == CPRC_ABUF_TRANS_PHONE) {
            printf("INFO: phoneme: %.3f %.3f %s\n", start, end, name);
        } else if (CPRC_abuf_trans_type(trans) == CPRC_ABUF_TRANS_WORD) {
            printf("INFO: word: %.3f %.3f %s\n", start, end, name);
        } else if (CPRC_abuf_trans_type(trans) == CPRC_ABUF_TRANS_MARK) {
            printf("INFO: marker: %.3f %.3f %s\n", start, end, name);
        } else if (CPRC_abuf_trans_type(trans) == CPRC_ABUF_TRANS_ERROR) {
            printf("ERROR: could not retrieve transcription at '%d'", i);
        }
    }
}
/* Using XML so a marker can be inserted */
char txt[] =  "<speak>Testing <mark name='test marker'/>1 2 3</speak>";
CPRCEN_engine * eng = CPRCEN_engine_load("license.lic", "speaker.voice");
CPRCEN_channel_handle chan = CPRCEN_engine_open_default_channel(eng);
CPRCEN_engine_set_callback(eng, chan, (void *)NULL, channel_callback);
CPRCEN_engine_channel_speak(eng, chan, txt, strlen(txt), 1);
CPRCEN_engine_delete(eng);
```

Python:

```
from cerevoice_eng import *
# An example callback class, prints information
class CallbackExample:
    # The callback function must be called 'channel_callback'
    def channel_callback(self, data):
        abuf = data_to_abuf(data)
        # Transcriptions contain markers, phonetic information, a
        # list of these items is available for each audio buffer.
        # Process the transcription buffer items and print information. */
        for i in range(CPRC_abuf_trans_sz(abuf)):
            trans = CPRC_abuf_get_trans(abuf, i)
            start = CPRC_abuf_trans_start(trans) # Start time in seconds
            end = CPRC_abuf_trans_end(trans) # End time in seconds
            name = CPRC_abuf_trans_name(trans) # Label, type dependent
            if CPRC_abuf_trans_type(trans) == CPRC_ABUF_TRANS_PHONE:
                print "INFO: phoneme: %.3f %.3f %s" % (start, end, name)
            elif CPRC_abuf_trans_type(trans) == CPRC_ABUF_TRANS_WORD:
                print "INFO: word: %.3f %.3f %s" % (start, end, name)
            elif CPRC_abuf_trans_type(trans) == CPRC_ABUF_TRANS_MARK:
                print "INFO: marker: %.3f %.3f %s" % (start, end, name)
            elif CPRC_abuf_trans_type(trans) == CPRC_ABUF_TRANS_ERROR:
                print "ERROR: could not retrieve transcription at '%d'" % i

txt =  "<speak>Testing <mark name='test marker'/>1 2 3</speak>"
eng = CPRCEN_engine_load("license.lic", "speaker.voice")
chan = CPRCEN_engine_open_default_channel(eng)
callback = CallbackExample()
engine_set_callback(eng, chan, callback)
CPRCEN_engine_channel_speak(eng, chan, txt, len(txt), 1)
CPRCEN_engine_delete(eng)
```

Using Multiple Voices

Multiple voices can be loaded into an engine, for example:

```
CPRCEN_engine * eng = CPRCEN_engine_new(); /* New empty engine */
/* Load two voices (in Python, use "" instead of NULL) */
CPRCEN_engine_load_voice(eng, "heather.lic", NULL, "heather.voice", CPRC_VOICE_LOAD_EMB_AUDIO);
CPRCEN_engine_load_voice(eng, "suzanne.lic", NULL, "suzanne.voice", CPRC_VOICE_LOAD_EMB_AUDIO);
```

When multiple voices are loaded, the channel can be opened based on a variety of parameters:

- ISO language code (for example *en*, *fr*, *es*, *it*)
- ISO region code (for English voices, British *gb* and American *us* can be selected)
- CereProc voice name (for example *Heather*, *Sarah*, *Suzanne*)

- Sample rate (usually *8000*, *16000* or *22050*)

The `CPRCEN_engine_open_channel` function is defined as:

```
CPRCEN_channel_handle CPRCEN_engine_open_channel(CPRCEN_engine * eng,
                                                 const char * iso_language_code,
                                                 const char * iso_region_code,
                                                 const char * voice_name,
                                                 const char * srate);
```

Examples:

```
/* Open an channel to the Heather voice */
CPRCEN_channel_handle chan = CPRCEN_engine_open_channel(eng, "", "", "Heather", "");
/* Open a channel to a French voice */
CPRCEN_channel_handle chan = CPRCEN_engine_open_channel(eng, "fr", "", "", "");
/* Open an American English voice */
CPRCEN_channel_handle chan = CPRCEN_engine_open_channel(eng, "en", "us", "", "");
```

The *best match* voice will be returned, backing off to the default voice if no matches are found.

Getting information for multiple voices (useful keys include *VOICE_NAME*, *SAMPLE_RATE*, *LANGUAGE_CODE_ISO* and *COUNTRY_CODE_ISO*):

```
int num_voices = CPRCEN_engine_get_voice_count(eng);
for (int i=0; i < num_voices; i++) {
     const char * voicename = CPRCEN_engine_get_voice_info(eng, i, "VOICE_NAME");
     printf("Voice name %d is '%s'\n", i, voicename);
}
```

Example Programs

C Examples

Three example applications are provided (pre-compiled on Linux, Windows, and OS X and as source code), demonstrating the Simple and Engine APIs. The binary versions can be used to generate TTS output, and the example code is a good starting point for integrating CereVoice into an application. The examples are found in the `examples/basictts` directory.

Tool name	Description
`basictts`	Basic TTS application using the `cerevoice_eng_simp.h` API, uses the audio library for playback or writes to a wave file
`txt2wav`	Wave file creation application using `cerevoice_eng.h`, uses incremental processing and configurable load
`tts_callback`	TTS application with incremental processing and callback function for low-latency output, logs all available events (phonemes, markers etc)

Building the Applications Under Linux, OS X, or Cygwin

An example Makefile is provided to build the example applications on Linux (`Makefile.linux`), OSX (`Makefile.osx`) and Cygwin (`Makefile.cygwin`). To compile, run `make` with the *-f* flag and supply the correct Makefile, eg:

```
make -f Makefile.linux
```

Each Makefile has a `LIBS` variable, containing the list of link libraries required for TTS. The `AUDIOLIBS` variable contains a list of the link libraries that are required when the `cerevoice_aud` audio playback library is also being used. These parameters, along with the include and lib locations, can be copied into an alternative IDE (such as Xcode, CodeWarrior etc).

The Cygwin build uses the `-mno-cygwin` flag to `gcc`, which allows the creation of DLLs and executables that do not rely on the Cygwin DLL (the Cygwin MinGW package should be installed).

Building the Applications Using Visual Studio

The example applications provided with the SDK are built with a 32-bit compiler. 64-bit libraries are provided, and the applications can be recompiled as 32 or 64-bit.

Example Microsoft Visual Studio solution files are provided:

Solution Name	Description
`basictts.sln`	Solution created with Visual Studio 2003 for backwards-compatibility (32-bit only)
`basictts_v10.sln`	Created with Visual Studio 2010 (supports 32 and 64-bit builds)

The Visual Studio 2010 project can be used to build 32 and 64-bit versions of the example executables.

Python Examples

Two example applications are provided, demonstrating the Simple and Engine APIs. Each can be used to generate TTS output, and the example code is a good starting point for integrating CereVoice into an application. The examples are found in the `examples/python` directory.

Example Programs

Tool name	Description
`txt2wav.py`	Wave file creation application using `cerevoice_eng.py`, uses incremental processing and configurable load
`tts_callback.py`	TTS application with incremental processing, audio playback, and callback function for low-latency output, logs all available events (phonemes, markers etc)

On OS X Snow Leopard, Python version 2.5 must be used (invoke the scripts with `python2.5` on the command line).

iPhone OS (iOS) Example

The iPhone Engine SDK includes an iOS demo application in the `examples/CereProciOSDemo` directory. It includes library builds for the simulator and for iOS itself (armv6 and armv7).

Building With XCode

The XCode project can be used to build a GUI application. The GUI text box can be used to enter text to synthesise.

Prerequisites

The application build takes a voice file from `examples/CereProciOSDemo/staging/tts.voice` and a license file from `examples/CereProciOSDemo/staging/license.lic`, and builds them in to the application. A voice and license file must be copied into the `staging` directory, with the correct file names, for a build to succeed.

Integration Notes

Synthesis initialisation (engine creation and voice loading) is performed in `CereProciOSDemo/Classes/CereProciOSDemoAppDelegate.m`, with the TTS generation function `synthesiseText` in `CereProciOSDemo/TtsViewController.m`. The code is fully commented, and uses the iOS `AVAudioPlayer` to render the audio incrementally, with low latency, using a callback. A non-callback version of the function, `synthesiseTextSimple`, is also available. However, it is only suitable for rendering short sentences, or where latency is not important (for example in a background application).

Note that the `cerevoice_aud` library is not available on iOS. The iOS native audio playback libraries should be used.

Audio API

Full Audio API documentation, with descriptions of all available function calls, can be viewed by opening the `docs/cerevoice_aud/index.html` file in a web browser.

Audio API Examples

See the synthesis and audio playback example code for Simple Synthesis With Audio API Playback and Synthesise and Play with Low Latency using the Audio API.

Additional Audio API Calls

The audio API has playback controls that can be used to manipulate the stream in a user application, for example:

```
/* Pause and resume playback */
CPRC_sc_audio_pauseon(CPRC_sc_player * player);
CPRC_sc_audio_pauseoff(CPRC_sc_player * player);
/* Stop and cancel playback */
CPRC_sc_audio_stop(CPRC_sc_player * player);
```

Software Dependencies

CereVoice Engine Library Dependencies

On most platforms, CereVoice does not have any dependencies beyond standard C and C++ libraries.

The Windows x64 CereVoice libraries require the following DLLs (supplied in the `3rdparty/lib64` directory):

```
libstdc++-6.dll
w64gcc_s_sjlj-1.dll
```

CereVoice Audio Library Dependencies

The `cerevoice_aud` library has different dependencies on each platform, as it relies on different native audio providers:

Platform	**Dependencies**
Linux	libasound
Mac OS X	CoreAudio, AudioToolbox, AudioUnit and Carbon frameworks
Windows	winmm

These libraries do not usually need to be supplied with an application if it is correctly linked.

Licenses for 3rd Party Libraries

The CereVoice SDK relies on several 3rd party software libraries. Some of these libraries require users to supply copyright notices with any release (Speex, Celt, PCRE, PortAudio, Big Digits). The license information for these packages can be found in the *LICENSE_3RDPARTY* file. Any shipping product based on the CereVoice Engine must include this license information with their software.

Speech Synthesis Markup Language (SSML) support

SSML is an open standard for TTS markup. It is a subset of VoiceXML (VXML), all SSML tags are available in a VXML environment. Usage, with examples, is described on the W3C page at http://www.w3.org/TR/speech-synthesis. The SSML tags currently supported in CereVoice can be found in the table below.

Supported Tags

Tag	Supported
break	yes (break strength of "none" is not supported)
emphasis	yes
lexicon	no (use the user lexicon feature)
mark	yes
meta	ignored
metadata	ignored
p	yes
phoneme	yes (CereProc phone set only)
prosody	yes (semitone values are not supported)
say-as	yes (VoiceXML builtins are supported, allowing interaction with the output of a VXML recogniser)
sub	yes
s	yes
voice	yes (if multiple voices are loaded into the engine, a `<voice>` tag can be used to switch between them)
xml:lang	no

CereProc Tag Set

CereProc has implemented additional TTS functionality that is not part of the SSML specification.

Variant Tags

The variant tag allows the user to request a different version of the synthesis for a particular section of speech. This is a very useful tag that can be used to make sections of speech sound more appropriate. The variant number can be increased to produce more and more different versions of the speech. The original version is equivalent to variant 0. For example, to change the version of the word *test* in *This is a test sentence*, use:

```
<s>
  This is a <usel variant='1'>test</usel> sentence.
</s>
```

The variant tag can be used to produce a bespoke rendering of a particular piece of speech. For example, an often-used speech prompt could be tuned to give a different rendering if desired.

Vocal Gestures

Non-speech sounds, such as laughter and coughing, can be inserted into the output speech. The `<spurt>` tag is used with an audio attribute to select a *vocal gesture* to included in the synthesis output, for example:

```
<speak>
  <spurt audio="g0001_004">cough</spurt>, excuse me, <spurt audio="g0001_018">err</spurt>, hello.
</speak>
```

The `<spurt>` tag cannot be empty, however the text content of the tag is not read, it is replaced by the gesture.

See the List of vocal gesture IDs for the full list of available gestures.

Emotion Tags

Available in voices with emotional support (for example *Heather*, *Sarah*, *William*, *Katherine*).

Happy Emotion Tag

For example:

```
<s>
  Today, <voice emotion='happy'>the sun is shining.</voice>
</s>
```

Sad Emotion Tag

```
<s>
  The outbreak<voice emotion='sad'>cast a shadow</voice> over the former
  Victorian holiday resort.
</s>
```

Calm Emotion Tag

```
<s>
  The beautiful gardens have been restored to all their
  <voice emotion='calm'>eccentric Victorian splendour.</voice>
</s>
```

Cross Emotion Tag

```
<s>
  When people leave a tip they want to know it will
  <voice emotion='cross'> not be used</voice> to make up the minimum wage.
</s>
```

Obtaining Support

CereProc offers support via email. There are two methods of contacting CereProc Support:

Support Requests

The fastest way to contact CereProc Support is via a support request. First log in, or create a user, at https://www.cereproc.com/user/login. Registered users can then use the support form at http://www.cereproc.com/support/support_request. Please select the appropriate product from the list and submit the support request.

Direct Email

CereProc support can be emailed at support@cereproc.com. However, queries sent to this address may take longer to reach the appropriate technical support representative than requests sent using the support request form.

Appendix 1

List of vocal gesture IDs

These IDs can be used to insert a 'vocal gesture' (non-speech sound) into synthesis.

Note that gesture *g0001_035* is available in Scottish voices only.

Gesture ID	**Gesture content**
g0001_001	tut
g0001_002	tut tut
g0001_003	cough
g0001_004	cough
g0001_005	cough
g0001_006	clear throat
g0001_007	breath in
g0001_008	sharp intake of breath
g0001_009	breath in through teeth
g0001_010	sigh happy
g0001_011	sigh sad
g0001_012	hmm question
g0001_013	hmm yes
g0001_014	hmm thinking
g0001_015	umm
g0001_016	umm
g0001_017	err
g0001_018	err
g0001_019	giggle
g0001_020	giggle
g0001_021	laugh
g0001_022	laugh
g0001_023	laugh
g0001_024	laugh
g0001_025	ah positive
g0001_026	ah negative
g0001_027	yeah question
g0001_028	yeah positive
g0001_029	yeah resigned
g0001_030	sniff
g0001_031	sniff
g0001_032	argh
g0001_033	argh
g0001_034	ugh
g0001_035	ocht

Appendix 1

g0001_036	yay
g0001_037	oh positive
g0001_038	oh negative
g0001_039	sarcastic noise
g0001_040	yawn
g0001_041	yawn
g0001_042	snore
g0001_043	snore phew
g0001_044	zzz
g0001_045	raspberry
g0001_046	raspberry
g0001_047	brrr cold
g0001_048	snort
g0001_050	ha ha (sarcastic)
g0001_051	doh
g0001_052	gasp

Appendix 2

CereVoice English RP (Southern English) Phone Set

Upper-case phonemes are *archiphonemes*, they can only exist word-finally and are converted to the lower-case equivalent by context rules.

Phoneme	Example Word	Example pronunciation
@	the (reduced)	dh_@
@@	nurse	n_@@_r_s
a	trap	t_r_a_p
aa	palm	p_aa_m
ai	price	p_r_ai_s
au	mouth	m_au_th
b	bee	b_ii
ch	each	ii_ch
d	dye	d_ai
dh	then	dh_e_n
e	dress	d_r_e_s
e@	square	s_k_w_e@_R
ei	face	f_ei_s
f	fan	f_a_n
g	guy	g_ai
h	hat	h_a_t
i	kit	k_i_t
i@	near	n_i@_R
ii	fleece	f_l_ii_s
jh	edge	e_jh
k	key	k_ii
l	lay	l_ei
m	me	m_ii
n	knee	n_ii
ng	song	s_o_ng
o	lot	l_o_t
oi	choice	ch_oi_s
oo	thought	th_oo_t
ou	goat	g_ou_t
p	pea	p_ii
r	ray	r_ei
s	sea	s_ii
sh	she	sh_ii
t	tea	t_ii
th	thin	th_i_n
u	foot	f_u_t

u@	cure	k_y_u@_r
uh	strut	s_t_r_uh_t
uu	goose	g_uu_s
v	van	v_a_n
w	way	w_ei
y	yes	y_e_s
z	zoom	z_uu_m
zh	beige	b_ei_zh
R	for	liaison_/r/

CereVoice English SC (Scottish) Phone Set

Phoneme	Example Word	Example pronunciation
@	the (reduced)	dh_@
@@	nurse	n_@@_s
a	trap	t_r_a_p
aa	palm	p_aa_m
ai	price	p_r_ai_s
au	mouth	m_au_th
b	bee	b_ii
ch	each	ii_ch
d	dye	d_ai
dh	then	dh_e_n
e	dress	d_r_e_s
ei	face	f_ei_s
f	fan	f_a_n
g	guy	g_ai
h	hat	h_a_t
i	kit	k_i_t
ii	fleece	f_l_ii_s
jh	edge	e_jh
k	key	k_ii
l	lay	l_ei
m	me	m_ii
n	knee	n_ii
ng	song	s_o_ng
o	lot	l_o_t
oi	choice	ch_oi_s
oo	thought	th_oo_t
ou	goat	g_ou_t
p	pea	p_ii
r	ray	r_ei
s	sea	s_ii
sh	she	sh_ii
t	tea	t_ii

th	thin	th_i_n
u	foot	f_u_t
uh	strut	s_t_r_uh_t
uu	goose	g_uu_s
v	van	v_a_n
w	way	w_ei
x	loch	l_o_x
y	yes	y_e_s
z	zoom	z_uu_m
zh	beige	b_ei_zh

CereVoice English GA (General American) Phone Set

Upper-case phonemes are *archiphonemes*, they can only exist word-finally and are converted to the lower-case equivalent by context rules.

Phoneme	**Example Word**	**Example pronunciation**
aa	balm	b_aa_m
ae	trap	t_r_ae_p
ah	strut	s_t_r_ah_t
ao	caught	k_ao_t
aw	mouth	m_aw_th
ax	the (reduced)	dh_ax
ay	rice	r_ay_s
b	bee	b_iy
ch	each	iy_ch
d	dye	d_ai
dx	bother	b_ah_dx_er
dh	then	dh_e_n
eh	dress	d_r_eh_s
er	butter	b_ah_dx_er
ey	face	f_ey_s
f	fan	f_ah_n
g	guy	g_ay
hh	hat	h_a_t
ih	hit	h_ih_t
iy	fleece	f_l_iy_s
jh	edge	e_jh
k	key	k_iy
l	lay	l_ey
m	me	m_iy
n	knee	n_iy
ng	song	s_o_ng
ow	goat	g_ow_t
oy	choice	ch_oy_s

p	pea	p_iy
r	ray	r_ey
s	sea	s_iy
sh	she	sh_iy
t	tea	t_iy
th	thin	th_ih_n
uh	foot	f_uh_t
uw	loose	l_uw_s
v	van	v_a_n
w	way	w_ey
y	yes	y_eh_s
z	zoom	z_uw_m
zh	beige	b_ey_zh
R	for	liaison_/r/

CereVoice German Phone Set

Phoneme	Example Word	Example pronunciation
@	machen	m_a_x_@_n
a	Anfang	q_a_n_f_a_ng
ah	Zahn	ts_ah_n
ae	hätte	h_ae_t_e
aeh	spät	sh_p_aeh_t
e	Pedal	p_e_d_a_l
eh	Leder	l_eh_d_er
i	still	sh_t_i_l
ih	Niere	n_ih_r_@
o	Motte	m_o_t_@
oh	Dose	d_oh_z_@
oi	Leute	l_oi_t_@
oe	könnte	k_oe_n_t_@
oeh	schön	sh_oeh_n
u	bunt	b_u_n_t
uh	Mut	m_uh_t
ue	hübsch	h_ue_p_sh
ueh	rügen	r_ueh_g_@_n
p	Prinz	p_r_i_n_ts
b	Abend	q_ah_b_@_n_t
t	Hütte	h_ue_t_@
d	jeder	j_eh_d_er
g	Auge	q_au_g_@
k	Kunst	k_u_n_s_t
f	Affe	q_a_f_@
v	warum	v_ah_r_u_m
s	Tasse	t_a_s_@

z	Hase	h_ah_z_@
zh	Genie	zh_eh_n_ih
sh	Schiff	sh_i_f
th	think	th_i_ng_k
dh	that	dh_ae_t
dzh	Djungle	dzh_u_ng_@_l
tsh	deutsch	d_oi_tsh
m	mein	m_ai_n
n	nein	n_ai_n
ng	Ding	d_i_ng
l	bald	b_a_l_t
w	water	w_o_t_er
j	Jahr	j_ah_rv
r	Rat	r_ah_t
rv	Herde	h_ae_rv_d_@
rl	traffic	t_rl_ae_f_i_k
h	Hafen	h_ah_f_@_n
ch	sicher	z_i_ch_er
x	Buch	b_uh_x
er	Vater	f_ah_t_er
en	Teint	t_en
an	Chance	sh_an_s_@
on	Pardon	p_a_rv_d_on
oen	Parfum	p_a_rv_f_oen
ts	Zahl	ts_ah_l
pf	Pfund	pf_u_n_t
ai	weiß	v_ai_s
au	Auto	au_t_oh
ei	Steak	s_t_ei_k
q	Ast	q_a_s_t

CereVoice Spanish Phone Set

Phoneme	**Example Word**	**Example pronunciation**
@	girona	jh_i0_r_o1_n_@
a	casa	k_a1_s_a0
b	boca	b_o1_k_a0
v	abate	a0_v_a1_t_e0
ch	churro	ch_u1_rr_o0
d	dedo	d_e1_df_o0
df	dedo	d_e1_df_o0
e	este	e1_s_t_e0
ee	aquest	a_k_ee1_s_t
f	fe	f_e1
g	gato	g_a1_t_o0

gf	aboga	a0_v_o1_gf_a0
x	jota	x_o1_t_a0
i	ti	t_i0
j	miedo	m_j_e1_d_o0
jj	peine	p_e1_jj_n_e
y	yate	y_a1_t_e0
jh	girona	jh_i1_r_o0_n_a0
k	que	k_e1
l	libro	l_i1_b_r_o0
ll	lloro	ll_o1_r_o0
m	mano	m_a1_n_o0
n	no	n_o0
ng	kong	k_o1_ng
ny	eñe	e1_ny_e0
o	yo	j_o0
oo	os	oo1_s
p	pan	p_a1_n
r	pera	p_e1_r_a0
rr	perro	p_e1_rr_o0
s	si	s_i0
sh	show	sh_o1_w
t	tu	t_u1
th	cero	th_e1_r_o0
u	tu	t_u1
w	cuatro	k_w_a1_t_r_o0
ww	pausa	p_a1_ww_s_a0
zz	urtzi	u1_r_t_zz_i0
B	boda	vellar_b
D	dedo	vellar_d
G	gato	vellar_g
R	ser	coda_r

CereVoice French Phone Set

Upper-case phonemes are *archiphonemes*, they can only exist word-finally or word-initially and are converted to the lower-case equivalent by context rules. They are mainly used to handle liaisons in French. Many phones in that list (a, ai, ch, dh, e@, ei, h, i@, jh, oi, oo, ou, r, th, u@, uh, un, uu, R, Z) are only relevant for the bilingual French-English voice, and using them in a purely French voice will probably lead to undesired results.

Phoneme	**Example Word**	**Example pronunciation**
aa	pas	p_aa_ZZ
a	palm	p_a_m
ai	price	p_r_ai_s
e	paix	p_e
an	pan	p_an

Appendix 2

au	peau	p_au
b	bas	b_aa_ZZ
ch	each	ii_ch
sh	chat	sh_aa_TT
d	deux	d_ex_ZZ
dh	then	dh_e_n
@	ce	s_@
e@	square	s_k_w_e@_R
ee	les	l_ee_ZZ
ei	face	f_ei_s
ex	deux	d_ex_ZZ
f	faim	f_in
g	gain	g_in
h	hat	h_aa_t
i	kit	k_i_t
i@	near	n_i@_R
ii	pis	p_ii_ZZ
in	pain	p_in
zh	joue	zh_u
jh	edge	e_jh
k	coup	k_u
l	loup	l_u
m	mou	m_u
n	nous	n_u_ZZ
ng	song	s_o_ng
ny	poigne	p_wa_ny
oi	choice	ch_oi_s
@@	peur	p_@@_rr
on	pont	p_on_TT
oo	thought	th_oo_t
ou	goat	g_ou_t
o	port	p_o_rr
u	pou	p_u
p	pas	p_aa_ZZ
r	ray	r_ei
rr	riz	rr_ii_ZZ
s	sa	s_aa
t	ta	t_aa
th	thin	th_i_n
u@	cure	k_y_u@_r
uh	strut	s_t_r_uh_t
un	dumping	d_un_p_ii_n_g
uu	goose	g_uu_s
yy	put	p_yy_TT

uy	lui	l_uy_ii
v	va	v_aa
w	poids	p_w_aa_ZZ
y	nier	n_y_ee_RR
z	aise	e_z
wa	poids	p_wa_ZZ
wi	point	p_wi_TT
yn	pion	p_yn
ye	fiais	f_ye_ZZ
ya	fia	f_ya
yo	rafiot	rr_aa_f_yo
yz	fiez	f_yz
R	for	liaison_r
Z	cats	plural_aphone
ZZ	pas	liaison_z
TT	doit	liaison_t
NN	rien	liaison_n
KK	gag	liaison_k
PP	drap	liaison_p
RR	premier	liaison_r
HH	hache	h_aspire
EE	centre	reduction_@

CereVoice Italian Phone Set

Phoneme	**Example Word**	**Example pronunciation**
a	casa	k_a1_s_a0
b	bocca	b_o1_kd_a0
bd	pubblico	p_u0_bd_l_i1_k_o0
ch	cena	ch_e1_n_a0
chd	braccio	b_r_a1_chd_j_o0
d	dado	d_a1_d_o0
dd	cadde	k_a1_dd_e0
dg	giro	dg_i1_r_o1
dgd	maggio	m_a1_dgd_j_o0
dz	zona	dz_oa1_n_a0
dzd	mezzo	m_ee1_dzd_o0
e	rete	r_e1_t_e0
ee	festa	f_ee1_s_t_a0
f	fuga	f_u1_g_a0
fd	beffa	b_ee1_fd_a0
g	gatto	g_a1_td_o0
gd	fugga	f_u1_gd_a0
gl	gli	gl_i1

Appendix 2

gld	aglio	a1_gld_j_o0
gn	gnomo	gn_oa1_m_o0
gnd	bagno	b_a1_gnd_o0
i	pino	p_i1_n_o0
j	piano	p_j_a1_n_o0
k	caso	k_a1_s_o0
kd	cocco	k_oa1_kd_o0
l	libro	l_i1_b_r_o0
ld	pollo	p_oa1_ld_o0
m	mano	m_a1_n_o0
md	mamma	m_a1_md_a0
n	no	n_o0
nd	nonno	n_oa1_nd_o0
ng	panca	p_a1_ng_a0
o	dove	d_o1_v_e1
oa	foto	f_oa1_t_o0
p	pane	p_a1_n_e0
pd	pappa	p_a1_pd_a0
r	pera	p_e1_r_a0
rr	carro	k_a1_rr_o
s	si	s_i1
sd	cassa	k_a1_sd_a0
sh	scia	sh_i1_a0
shd	lascia	l_a1_shd_j_a0
t	tu	t_u1
td	petto	p_ee1_td_o0
ts	zitto	ts_i1_td_o0
tsd	pazzo	p_a1_tsd_o0
u	scudo	s_k_u1_d_o0
v	valle	v_a1_ld_e0
vd	ovvio	o1_vd_j_o0
w	quadro	k_w_a1_d_r_o0
z	sbaglio	z_b_a1_gl_j_o0
zh	garage	g_a0_r_a1_zh

Appendix 3

Change Log

SDK Version	*Changes*
3.0.1	Added `cerevoice_aud` API cues with integrated clean up
	Updated example application with audio cue change
	Added QUICK_INSTALL guide
3.0.0	Engine API, combining and rationalising the separate 2.x normaliser and synthesis APIs

Last updated: **Thu 23 Jun 2011 11:57:01 BST**