CereProc CereVoice Cloud User Guide

Brendan Austin, Chris Pidcock

Contents

Contents

Introduction

The CereVoice Cloud is a text-to-speech (TTS) web service. It can be used to speech enable internet-connected applications via a C/C++, C#, Objective-C, PHP, Java or Python interface, or any other language or technology that supports SOAP or REST requests. The CereVoice Cloud User Guide describes the CereVoice Cloud API for use by developers, and a set of example programs.

Terms of use can be found in the CereVoice Cloud Terms and Conditions, you must agree to these terms to use the CereVoice Cloud service.

Signing Up for the CereVoice Cloud

To sign up for the CereVoice Cloud:

1. You must first be a registered CereProc website user. To become a CereProc user fill in the â User Registration Form.
2. Once you have been registered as a CereProc user, navigate to the â CereVoice Cloud Registration page.
3. You will then be asked to input an email address which will be associated with your CereVoice Cloud account. A validation email will be sent to this address containing a confirmation link.
4. Click the link sent to the submitted email address. This will activate your CereVoice Cloud account and you will receive another email containing your *accountID* and *password*, which you will need to access the service.

IMPORTANT: always keep your *accountID* and *password* private, and do not publicly expose the credentials in any service or application that uses the CereVoice Cloud. The password can be reset using the instructions below.

Resetting Your CereVoice Cloud Credentials

To reset your CereVoice Cloud credentials, navigate to the â CereVoice Cloud Reset page. There, you will be asked to input the email address used to sign up to the service. A confirmation link will be sent to this email to confirm the request for the credentials to be reset. Once this link is clicked, a new password will be sent to the user via email.

CereVoice Cloud Credits

CereVoice Cloud credit is required to generate TTS output, 1 credit = 1 character of text input. A free tier of credit is provided to all registered CereVoice Cloud users, providing 10,000 characters of text input per month. Additional credit can be purchased from the CereProc Store:

1. CereVoice Cloud 100k Credit - Â£24.99 (100,000 credits)
2. CereVoice Cloud 1M Credit - Â£149.99 (1,000,000 credits)

Euro and dollar pricing can be selected in the store. Cloud credit products are visible to logged in CereVoice Cloud users. For subscriptions or larger packages, please contact â sales@cereproc.com.

The CereVoice Cloud API

Recommended Input

CereProc recommends the use of XML input documents, especially the W3C standard â Speech Synthesis Markup Language (SSML). SSML includes tags for modifying pitch, rate, and pronunciation. It also supports inserting audio, markers, and breaks. XML markup is required for some functionality, such as markers and emotional synthesis. Plain text input is also supported (mixing XML markup within an invalid text document is not recommended). Custom user pronunciations are supported using CereProc phone sets and IPA as part of the â Pronunciation Lexicon Specification (PLS).

Supported SSML Tags

Tag	Supported
audio	yes (sample rate should match text-to-speech output setting)
break	yes (break strength of "none" is not supported)
emphasis	yes
lexicon	yes (IPA with PLS, CereProc format)
mark	yes
meta	ignored
metadata	ignored
p	yes
phoneme	yes (IPA with PLS, CereProc phone set)
prosody	yes (semitone values are not supported)
say-as	yes (â VoiceXML builtins are supported, allowing interaction with the output of a VXML recogniser)
sub	yes
s	yes
voice	yes
xml:lang	no

Simple Speak function

The CereVoice Cloud `speakSimple()` function synthesises input text with the selected voice.

```
function speakSimple(accountID, password, voice, text)
```

Parameters:

- **accountID** - User account ID
- **password** - User account password for validation
- **voice** - Name of the voice to be used for synthesis. This can be an empty string, in which case a default voice will be used. The value is one of the voice names returned by `listVoices()`. The default voice for this function is *Heather*.
- **text** - Text or XML input string to synthesise

Return:

```
Array containing output parameters:
- fileUrl
```

```
   - Output audio file URL for download
- charCount
   - Number of characters used in this request
- resultCode
   - Return code of the function
- resultDescription
   - Human-readable description of the return code
```

Extended Speak Function

The `extendedSpeak()` function allows for more control over the audio output (see Using 3D Audio or Using Metadata for more information these features).

```
function speakExtended(accountID, password, voice, text, audioFormat, sampleRate,
                       audio3D, metadata)
```

Parameters:

- **accountID** - User account ID
- **password** - User account password for validation
- **voice** - Name of the voice to be used for synthesis. This can be an empty string, in which case a default voice will be used. The value is one of the voice names returned by `listVoices()`. The default voice for this function is *Heather*.
- **text** - Text or XML input string to synthesise
- **audioFormat** - The audio format encoding of the returned synthesis. The value is one of the formats returned by `listAudioFormats()`. The default value is *ogg* (*wav* and *mp3* are also supported). Can be an empty string, in which case the default will be used.

- **sampleRate** - The sample rate to be used for the output audio - *8000*, *11025*, *16000*, *22050*, *32000*, *44100*, and *48000* are supported. Can be an empty string, in which case the default *48000* will be used.
- **audio3D** - A boolean value which can be set to activate the 3D Audio Library, defaults to *False*. Can be an empty string, in which case the default will be used.
- **metadata** - A boolean value to which can be set to retrieve the metadata associated with the speech, defaults to *False*. Can be an empty string, in which case the default will be used.

Return:

```
Array containing output parameters:
- fileUrl
   - Output audio file URL for download
- charCount
   - Number of characters used in this request
- resultCode
   - Return code of the function
- resultDescription
   - Human-readable description of the return code
- metadataUrl
   - Output XML file containing speech metadata
```

List Voices Function

The CereVoice Cloud `listVoices()` function outputs information about the voices available on the Cloud.

```
function listVoices(accountID, password)
```

Parameters:

- **accountID** - User account ID
- **password** - User account password for validation

Return:

```
Array of VoiceArray(s), containing output parameters:
 - sampleRate
    - the sample rate of the voice
 - voiceName
    - the name of the voice
 - languageCodeISO
    - the two-letter classification code of the language used by the voice (ISO 639),
      required for uploading user lexicons and abbreviations
 - countryCodeISO
    - the two-letter country code for the voice
 - accentCode
    - the two-letter CereProc accent code for the voice, required for uploading user lexicons
 - sex
    - the gender of the voice
 - languageCodeMicrosoft
    - the corresponding Microsoft code to the country code
 - country
    - the country of the voice
 - region
    - the region of the voice
 - accent
    - the accent of the voice
```

Upload Lexicon Function

The CereVoice Cloud `uploadLexicon()` function uploads and stores a custom lexicon file, to be used with `speakSimple()` and `speakExtended()` during synthesis. A custom lexicon can be used to add or override new pronunciations to the TTS output. Once a user lexicon has been uploaded, it will be used in synthesis with voices in the specified language and accent. The user lexicon is not shared with other accounts. See below for information on the file format.

```
function uploadLexicon(accountID, password, lexiconFile, language, accent)
```

- **accountID** - User account ID
- **password** - User account password for validation
- **lexiconFile** - URL location of the custom user lexicon
- **language** - The two-letter language code (ISO 639) associated with the lexiconFile
- **accent** - The two-letter CereProc accent code associated with the lexiconFile. See CereProc Accent Codes

Return:

```
Array containing the output parameters:
 - resultCode
    - Return code of the function
 - resultDescription
    - Human-readable description of the return code
```

Adding User Lexicons

In order for the CereVoice Cloud to use a custom lexicon, that custom lexicon must be uploaded to the CereVoice Cloud. To do this, the API function `uploadLexicon()` can be used. The user lexicon file *must* be available to download via HTTP. This means that you must provide the function with a URL that the CereVoice Cloud can access (for example, a public location on your web server, in Amazon S3, or Dropbox). Once this function has returned, the CereVoice Cloud will have stored a copy of your custom lexicon.

User Lexicon Format

The user lexicon format consists of a *headword* and *pronunciation*, one per line. Example lexicon line (English RP):

```
mourinho m_@@0_r_ii1_n_y_ou2
```

The headword should be lower case, and consist of alphabetic characters only (to process a string that includes non-alphabetic characters, use an abbreviations file). Vowel phonemes, such as @@, *ii* and *ou* in the example pronunciation, can have stress levels specified. The stress levels are *1* for primary stress, *2* for secondary stress, and *0* for no stress.

Each accent has a different phone set. Different user lexicons are required for British and American voices, for example. However, the same user lexicon could be used between voices with the same accent, for example the *Sarah* and *William* English RP voices. See the â CereVoice Phone Sets document for example pronunciations.

If words with accented characters are added, the encoding must be UTF-8.

CereProc Accent Codes

The CereProc accent codes for the current voices are:

Accent	Language	Accent ID
Received Pronunciation (England)	en	rp
General American	en	ga
Scottish	en	sc
Irish	en	ie
Sexy French	en	sf
Castilian	es	ca
Metropolitan French	fr	fr
Italian	it	it
Hochdeutsch	de	hd
Portuguese	pt	pt
Brazilian Portuguese	pt	br

List Lexicon(s) Function

The CereVoice Cloud `listLexicon()` function lists the custom lexicon files available for synthesis on the CereVoice Cloud.

```
function listLexicons(accountID, password)
```

Parameters:

- **accountID** - User account ID
- **password** - User account password for validation

Return:

```
Array of LexiconArray(s), containing output parameters:
 - url
    - the url of the lexicon file
 - language
    - the two-letter language code of the lexicon file (ISO 639)
 - accent
    - the two-letter !CereProc accent code of lexicon file
 - lastModified
    - the date the file was last modified
 - size
    - the size (in bytes) of the file
```

Upload Abbreviations Function

The CereVoice `uploadAbbreviations()` function uploads and stores a custom abbreviation file, to be used with `speakSimple()` and `speakExtended()` during synthesis. User abbreviations can be used to expand mixed case tokens, or tokens including digits, into words. Once a custom abbreviation file has been uploaded, it will be used in synthesis with voices in the specified language.

```
function uploadAbbreviations(accountID, password, abbreviationFile, language)
```

- **accountID** - User account ID
- **password** - User account password for validation
- **abbreviationFile** - URL location of the custom user abbreviation file.
- **language** - The two-letter language code (ISO 639) associated with the abbreviationFile

Return:

```
Array containing the output parameters:
 - resultCode
    - Return code of the function
 - resultDescription
    - Human-readable description of the return code
```

Adding User Abbreviations

In order for the CereVoice Cloud to use a custom abbreviation file, that custom abbreviation must be uploaded to the CereVoice Cloud. To do this, the API function `uploadAbbreviations()` can be used. The user abbreviation file *must* be available to download via HTTP. This means that you must provide the function with a URL that the CereVoice Cloud can access (for example, a public location on your web server, in Amazon S3, or Dropbox). The user abbreviation file is not shared with other accounts. Once this function has returned, the CereVoice Cloud will have stored a copy of your custom abbreviation file.

User Abbreviations Format

The user lexicon format consists of a *token*, *no break flag*, and *replacement text*, line-by-line. Example entries (taken from the current CereProc abbreviation list):

```
3G      0    three g
7/11    0    seven eleven
Dr      1    doctor
FAQ     0    f a:letter q
```

The first column is a whitespace-delimited *token* in the input text. The second column, the *no break flag*, describes the handling of punctuation following the token. When set to *1*, following punctuation is ignored. In the examples, this would cause "Dr. Johnson" to be read without a pause between the words.

Some languages, such as English, have single letters that can be pronounced differently in an acronym compared to free text. The letter pronunciation can be ensured by adding *:letter* after the letter (see the example for *FAQ*).

List Abbreviation(s) Function

The CereVoice Cloud `listAbbreviations()` function lists the custom abbreviation files that are available for synthesis on the CereVoice Cloud.

```
function listAbbreviations(accountID, password)
```

Parameters:

- **accountID** - User account ID
- **password** - User account password for validation

Return:

```
Array of AbbreviationArray(s), containing output parameters:
 - url
    - the url of the abbreviation file
 - language
    - the two-letter code of the language for these abbreviations (ISO 639)
 - lastModified
    - the date the file was last modified
 - size
    - the size (in bytes) of the file
```

List Audio Format(s)

Use the CereVoice `listAudioFormats()` function to list the available audio encoding formats available for synthesis on the CereVoice Cloud. The current formats are **ogg**, **wav**, and **mp3**.

```
function listAudioFormats(accountID, password)
```

Parameters:

- **accountID** - User account ID
- **password** - User account password for validation

Return:

```
An associative array of strings containing:
 - audioFormat
    - the type of audio encoding available for synthesis
```

Get Credit Function

The CereVoice Cloud `getCredit()` function retrieves the credit information for the given account. 1 credit = 1 character of text input.

```
function getCredit(accountID, password)
```

Parameters:

- **accountID** - User account ID
- **password** - User account password for validation

Return:

```
Array containing output parameters:
 - freeCredit
    - the remaining amount of free credit available on the supplied account.
 - paidCredit
    - the remaining amount of paid credit available on the supplied account.
 - charsAvailable
    - Number of characters that can be synthesised
```

Free credit is deducted first, until their is no free credit remaining, at which point the paid credit will be deducted for subsequent synthesis requests. The free credit amount is reset at the start of each month.

Generating Simple TTS Output

speakSimple with SOAP

SOAP is a protocol for exchanging structured information between Web Services. To see the WSDL description of the CereVoice Cloud, navigate to (â https://cerevoice.com/soap/soap_1_1.php?wsdl). The following code examples show how to make SOAP requests to the CereVoice Cloud.

Python

CereProc recommends the use of the lightweight Python SOAP client â Suds, which can be installed using `pip install suds`. Below is a code example of a `speakSimple()` request using Suds:

```
from suds.client import Client
## SOAP Client
soapclient = Client("https://cerevoice.com/soap/soap_1_1.php?WSDL")
## SOAP Request
request = soapclient.service.speakSimple(accountID, password, 'Stuart',
                                         'Python soap speak simple')
print request
```

By printing the request to the console, we obtain an output similar to:

```
(reply){
   fileUrl = "https://cerevoice.s3.amazonaws.com/Stuart4800018a7a9d9cf14613cb6046ef85.ogg"
   charCount = 24
   resultCode = 1
   resultDescription = "Successful Synthesis"
 }
```

Programmatically, however, we have not ascertained whether the response was successful nor can we make use of the url from which we can obtain our synthesised text. To check the resultCode and access the output URL:

```
if(request.resultCode != 1):
    print "ERROR: request failed -", request.resultDescription
else:
    print "URL:", request.fileUrl
```

This code will now alert you to a failed request, for example:

```
ERROR: request failed -  XML was not well formed. Please check your request.
```

Or print the URL from which you can obtain the synthesis:

```
URL: https://cerevoice.s3.amazonaws.com/Stuart4800018a7a9d9cf14fc9cb6046ef85.ogg
```

PHP

Below is an example of a SOAP request using PHP's SOAP module. When using a SOAP client in PHP, PHP must have been compiled with `--enable-soap`.

```
$client = new SoapClient("https://cerevoice.com/soap/soap_1_1.php?wsdl",
                         array("trace" => 1, "exceptions"  => 0));

$request = array('accountID'     => accountID,
```

```
                'password'      => password,
                'voice'         => 'Sarah',
                'text'          => "PHP soap speak simple");

$response = $client->__soapCall('speakSimple', $request);

if($response['resultCode'] != 1)
   printf("ERROR: request failed - %s", $response['resultDescription']);
else
   printf("URL: %s",$response['fileUrl']);
```

Java

Below is an example of a SOAP request in Java. CereProc recommends the use of the *wsimport* command for *JAX-WS* applications (â wsimport). The *wsimport* tool will automatically generate the Java artefacts needed to communicate with the CereVoice Cloud.

```
import javax.xml.ws.Holder;
import https.cerevoice_com.soap.cloudservice.CereVoiceCloud;
import https.cerevoice_com.soap.cloudservice.CereVoiceCloudPortType;

public class Client {

    public static void main(String [] args) {
        try {

            CereVoiceCloud cere = new CereVoiceCloud();

            Holder<String> url = new Holder<String>();
            Holder<Integer> char_count = new Holder<Integer>();
            Holder<Integer> res_code = new Holder<Integer>();
            Holder<String> res_des = new Holder<String>();

            cere.getCereVoiceCloudPort().speakSimple(accountID, password, "Heather",
                                                     "Java soap speak simple", url,
                                                     char_count, res_code, res_des);

            if(res_code.value != 1)
                System.out.println("ERROR: request failed - " + res_des.value);
            else
                System.out.println("URL: " + url.value);

        } catch (Exception e) {
            System.err.println(e.toString());
        }
    }
}
```

C

Below is an example of a SOAP request in ANSI C. CereProc recommends the use of the *wsdl2h* facility provided by â gSOAP. Like the *wsimport* provided for Java, gSOAP will provide libraries and automatically generate header files needed to communicate with the CereVoice Cloud via C.

```
#include "soapH.h"
#include "CereVoiceCloudBinding.nsmap"

int main(void) {

  struct soap soap;
  const char end_point[] = "https://cerevoice.com/soap/soap_1_1.php?wsdl";
```

```
  const char soap_action[] = "#speakSimple";

  soap_init1(&soap, SOAP_XML_INDENT);

  struct ns4__speakSimpleResponse * response = NULL;
  response = (struct ns4__speakSimpleResponse *)malloc(sizeof
                                                  (struct ns4__speakSimpleResponse*));

  char * acc_id = "accountID";
  char * pswrd = "password";
  char * voice = "William";
  char * text = "C soap speak simple";


  if(soap_call_ns4__speakSimple(&soap, end_point, soap_action, acc_id, pswrd, voice, text,
                                response) == SOAP_OK) {
    if(response->resultCode != 1) {
      printf("ERROR: request failed - %s", response->resultDescription);
      return 0;
    }
    else
      printf("URL: %s\n", response->fileUrl);
  }
  else {
    soap_print_fault(&soap, stderr);
  }
  soap_destroy(&soap);
  soap_end(&soap);
  soap_done(&soap);
  return 0;
}
```

C#

When using a SOAP client in C#, Microsoft provide some WSDL code generation tools (e.g â Service References, and â Web References).

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Text;

namespace SoapClient
{
  class Client
  {
    static void Main(string[] args)
    {
      CereVoice.CereVoiceCloudPortType cere = new CereVoice.CereVoiceCloudPortTypeClient();
      CereVoice.speakSimpleRequest request = new CereVoice.speakSimpleRequest(accountID,
                 password, "Isabella", "c sharp soap speak simple");
      CereVoice.speakSimpleResponse response = new CereVoice.speakSimpleResponse();
      response = cere.speakSimple(request);
      if (response.resultCode != 1)
         Console.WriteLine("ERROR: request failed - " + response.resultDescription);
      else
         Console.WriteLine("URL: " + response.fileUrl);
         return;
      }
   }
}
```

Objective C (iOS)

CereProc recommends the use of the code generation facility provided by â SudzC. Below is an example of a SOAP request made to the CereVoice Cloud using the downloaded code generated from SudzC. The code below will make a `speakSimple()` request and print the response to the debug console of an iPhone.

```
#import "ObjectiveCSoapAppDelegate.h"
#import "CereVoiceCloud.h"
@implementation ObjectiveCSoapAppDelegate

@synthesize window;

- (void)applicationDidFinishLaunching:(UIApplication *)application {

    // Override point for customization after application launch
    CereVoiceCloud* cere = [[CereVoiceCloud alloc] init];
    SoapRequest* request = [cere speakSimple:self action:@selector(handleSpeakSimple:)
                            accountID:@"accountID" password:@"password" voice:@"Adam"
                            text:@"objective c soap speak simple"];
    [request setDeserializeTo:nil];
    [window makeKeyAndVisible];
}

- (void)handleSpeakSimple: (id)value {
    if([value isKindOfClass:[NSError class]] || [value isKindOfClass:[SoapFault class]]) {
        NSLog(@"NSError encountered: %@", value);
        return;
    }
    if(![value isKindOfClass:[NSMutableDictionary class]]) {
        NSLog(@"Not a dictionary");
        return;
    }
    NSString* resultCode = [value objectForKey:@"resultCode"];
    if(![resultCode isEqualToString:@"1"]) {
        NSLog(@"INFO: request failed - %@", [value objectForKey:@"resultDescription"]);
    }
    else {
        NSLog(@"FILE URL: %@", [value objectForKey:@"fileUrl"]);
    }
}

- (void)dealloc {
    [window release];
    [super dealloc];
}

@end
```

JavaScript

CereProc recommends the use of a jQuery Plugin â jQuery Soap. Below is an example of a SOAP request made to the CereVoice Cloud using the jquery.soap plugin. The code below will make a `speakSimple()` request and print the response to the debug console of a web browser.

```
    // Set up some variables
    var cereurl = "https://cerevoice.com/soap/soap_1_1.php?";
    var ceremethod = "speakSimple";
    var accID = "accountID";
    var pword = "password";
    var cerevoice = "Heather";
    var txt = "This is a test of speak simple using jquery.soap";
```

```
    $.soap({
            url: cereurl,
                method: ceremethod,
                data: {
                    accountID: accID,
                    password : pword,
                    voice    : cerevoice,
                    text     : txt
                },

                success: function (soapResponse) {
                // do stuff with soapResponse
                // if you want to have the response as JSON use soapResponse.toJSON();
                // or soapResponse.toString() to get XML string
                // or soapResponse.toXML() to get XML DOM
                console.log(soapResponse.toString());
                var xmlDoc = soapResponse.toXML();

                if ($(xmlDoc).find("resultCode").innerHTML == "0") {
                    console.log(xmlDoc.getElementsByTagName("resultDescription")[0].textContent);
                    return false;
                } else {
                    console.log(xmlDoc.getElementsByTagName("resultDescription")[0].textContent);
                    console.log(xmlDoc.getElementsByTagName("fileUrl")[0].textContent);
                }
            },
                error: function (SOAPResponse) {
                // show error
                console.log(SOAPResponse.toString());
            }
        });
```

speakSimple with REST

The RESTful implementations of the CereVoice Cloud API functions follow the same definitions as the SOAP WSDL (â https://cerevoice.com/soap/soap_1_1.php?wsdl). The main difference however is that the request to be sent to CereVoice Cloud, **must** be in XML.

Below are a few examples of REST requests sent to the CereVoice Cloud via HTTP POST. Unlike the examples until now, we must first create the request, then post it to the Cloud. Using the CereVoice Cloud API definitions as a starting point, we can easily form an XML request which we can then send to the CereVoice Cloud.

Creating a request

First, we'll take the API definition of `speakSimple()`:

```
function speakSimple(accountID, password, voice, text)
```

The 4 arguments this function requires are:

- accountID
- password
- voice
- text

By taking the arguments needed for each individual function and specifying which function it is we'd like to call, we can form the following template XML request:

```
<?xml version='1.0'?>
<speakSimple>
   <accountID></accountID>
   <password></password>
   <voice></voice>
   <text></text>
</speakSimple>
```

Finally we place the values for each argument in their corresponding XML tags and we have a valid request to send to the CereVoice Cloud.

Python

The following code is an example of a `speakSimple()` REST request in Python.

```
import httplib2
from xml.etree import ElementTree as ET

restclient = httplib2.Http()
resturl = "https://cerevoice.com/rest/rest_1_1.php"
headers = {'Content-type': 'text/xml'}

xml = """<?xml version='1.0'?><speakSimple><accountID>accountID</accountID>
       <password>password</password><voice>Caitlin</voice>
       <text>python rest speak simple</text></speakSimple>"""

response, content = restclient.request(resturl, 'POST', headers=headers, body=xml)
print content
```

By printing the content of the request to the console, we would obtain an ouput similar to:

```
<?xml version='1.0'?>
<speakSimpleResponse>
 <fileUrl>https://cerevoice.s3.amazonaws.com/Caitlin480001a07d448e0ef1a0b66a.ogg</fileUrl>
 <charCount>17</charCount>
 <resultCode>1</resultCode>
 <resultDescription>Successful Synthesis</resultDescription>
</speakSimpleResponse>
```

Programmatically however, we have not ascertained whether the response was successful nor can we make use of the url of the synthesised text. Therefore, it is better programming practice to use your favourite XML parser to parse the reply from the CereVoice Cloud, check the resultCode and act accordingly. For example:

```
reply = ET.XML(content)
if(reply.find("resultCode").text == '0'):
     print "ERROR: request failed -", reply.find("resultDescription").text
else:
     print "URL:", reply.find("fileUrl").text
```

This code will now alert you to a failed request:

```
ERROR: request failed -  Requested voice 'blankvoice' is not available on the server
```

Or print the url from which you can obtain the synthesis:

```
URL: https://cerevoice.s3.amazonaws.com/Heather4800018a7a9d9c6e461359cb6046ef85.ogg
```

PHP

Below is a REST request being sent to the CereVoice Cloud via POST, using PHP's curl library.

```
define('URL', 'https://cerevoice.com/rest/rest_1_1.php');

define('XML', "<?xml version='1.0'?><speakSimple><accountID>accountID</accountID>
               <password>password</password><voice>Jack</voice>
               <text>PHP rest speak simple</text></speakSimple>");

$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, URL);
curl_setopt($ch, CURLOPT_HTTPHEADER, array('Content-Type: text/xml'));
curl_setopt($ch, CURLOPT_POST, 1);
curl_setopt($ch, CURLOPT_POSTFIELDS, XML);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

$response_str = curl_exec($ch);
$response = simplexml_load_string($result);

if($response['resultCode'] != 1)
    printf("ERROR: request failed - %s", $response['resultDescription']);
else
    printf("URL: %s", $response['fileUrl']);
```

Java:

Below is a REST request made to the CereVoice Cloud using HTTP POST in Java.

```
import java.net.*;
import java.io.*;

public class SOAPClient {

    public static void main(String[] args) {

        String xml = "<?xml version='1.0'?><speakSimple>"
            +"<accountID>accountID</accountID>"
            +"<password>password</password>"
            +"<voice>Jess</voice>"
            +"<text>Java rest speak simple</text></speakSimple>";

        try {
            URL u = new URL("https://cerevoice.com/rest/rest_1_1.php");
            URLConnection uc = u.openConnection();
            HttpURLConnection connection = (HttpURLConnection) uc;

            connection.setDoOutput(true);
            connection.setDoInput(true);
            connection.setRequestMethod("POST");
            connection.setRequestProperty("Content-Type","text/xml; charset=utf-8");
            connection.setRequestProperty("Content-Length", String.valueOf(xml));

            OutputStream out = connection.getOutputStream();
            Writer wout = new OutputStreamWriter(out);
            wout.write(xml);
            wout.flush();
            wout.close();

            InputStream in = connection.getInputStream();
            int c;
            while ((c = in.read()) != -1) System.out.write(c);
            in.close();
```

```
                    ...

            /*
               Using your favourite Java XML Parser,
               parse the XML output, check that resultCode is
               equal to 1 and act accordingly
            */
                    ...

        }
        catch (Exception e) {
            System.err.println(e);
        }
    }
}
```

C

The following is a REST request made to the CereVoice Cloud via ANSI C using libcurl to send the request via POST.

```
#include <curl/curl.h>
#include <curl/easy.h>
int main(void) {

    CURL *curl;
    CURLcode res;

    curl = curl_easy_init();

    if(curl) {
        curl_easy_setopt(curl,CURLOPT_URL,"https://cerevoice.com/rest/rest_1_1.php");
        curl_easy_setopt(curl,CURLOPT_POSTFIELDS,"<?xml version='1.0'?>
                                                  <speakSimple>
                                                  <accountID>accountID</accountID>
                                                  <password>password</password>
                                                  <voice>Katherine</voice>
                                                  <text>C rest speak simple</text>
                                                  </speakSimple>");
        res = curl_easy_perform(curl);

                ...

        /*
           Using your favourite C XML Parser,
           parse the XML output, check that resultCode is
           equal to 1 and act accordingly
        */
                    ...

        curl_easy_cleanup(curl);
    }
    return 0;
}
```

C#

Example REST request sent to the CereVoice Cloud via C#.

```
using System;
using System.IO;
```

Generating Simple TTS Output

```
using System.Net;
using System.Text;

namespace restclient
{
   class Client
   {

      public static void Main (string[] args)
      {

         WebRequest request = WebRequest.Create("https://cerevoice.com/rest/rest_1_1.php");
         request.Method = "POST";
         string postData = "<?xml version='1.0'?>"
                           + "<speakSimple>"
                           + "<accountID>accountID</accountID>"
                           + "<password>password</password>"
                           + "<voice>Kirsty</voice>"
                           + "<text>C sharp rest speak simple</text></speakSimple>";
         byte[] byteArray = Encoding.UTF8.GetBytes(postData);
         request.ContentType = "text/xml";
         request.ContentLength = byteArray.Length;
         Stream dataStream = request.GetRequestStream();
         dataStream.Write(byteArray, 0, byteArray.Length);
         dataStream.Close();

         WebResponse response = request.GetResponse();
         dataStream = response.GetResponseStream();
         StreamReader reader = new StreamReader(dataStream);
         XmlDocument xml = new XmlDocument();
         xml.Load(XmlReader.Create(new StringReader(reader.ReadToEnd())));
         XmlNodeList resultCode = xml.GetElementsByTagName("resultCode");
         if (String.Compare(resultCode[0].InnerText, "1", true) != 0)
         {
             Console.WriteLine("INFO: request failed - " + xml.GetElementsByTagName(
                           "resultDescription")[0].InnerText);
         }
         else
         {
             Console.WriteLine("FILE URL: " + xml.GetElementsByTagName("fileUrl")[0].
                           InnerText);
         }

      }
   }
}
```

Objective-C

Below is a REST request made to the CereVoice Cloud using Objective-C. (note that this code requires a version of OS X >= 10.6).

```
#import <Foundation/Foundation.h>

int main(void) {

   NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];

   NSURL* url = [NSURL URLWithString:@"https://cerevoice.com/rest/rest_1_1.php"];
   NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:url];

   NSString * str = [NSString stringWithFormat:@"<?xml version=\"1.0\"?>
                   <speakSimple><accountID>accountID</accountID>
                   <password>password</password><voice>Stuart</voice><text>Objective
```

```
                    C rest speak simple</text></speakSimple>"];
   [request setHTTPMethod:@"POST"];
   [request setValue:@"application/xml; charset=utf-8" forHTTPHeaderField:@"Content-Type"];
   [request setHTTPBody:[str dataUsingEncoding:NSUTF8StringEncoding]];

   NSURLResponse * response = [[NSURLResponse alloc] init];
   NSError * error = nil;
   NSData * dataReply = [NSURLConnection sendSynchronousRequest:request
                         returningResponse:&response error:&error];
   NSString * stringReply = [[NSString alloc] initWithData:dataReply
                             encoding:NSASCIIStringEncoding];
   NSLog(@"string reply: %@", stringReply);
   [pool release];

          ...

   /*
     Using your favourite Objective-C XML Parser,
     parse the XML output, check that resultCode is
     equal to 1 and act accordingly
   */
          ...
}
```

JavaScript

The following is a REST request made to the CereVoice Cloud via the XMLHttpRequest object to send the request via POST

```
    // Create the client
    var client = new XMLHttpRequest();

    // Set up some variables
    var cereurl = "https://cerevoice.com/rest/rest_1_1.php";
    var str = "<?xml version='1.0'?><speakSimple>" +
              "<accountID>accountID</accountID>" +
              "<password>password</password>" +
              "<voice>Heather</voice>" +
              "<text>This is a test of speak simple</text></speakSimple>";

    // Send the request
    client.open("POST", cereurl, false);
    try {
        client.send(str);
    } catch(err) {
        console.log("ERROR: ", err);
    }

    var response = client.responseText;
    var parser = new DOMParser();
    var xmlDoc = parser.parseFromString(response, "application/xml");

    if($(xmlDoc).find("resultCode")[0].innerHTML == "0") {
        console.log(xmlDoc.getElementsByTagName("resultDescription")[0].textContent);
        return false;
    } else {
        console.log(xmlDoc.getElementsByTagName("resultDescription")[0].textContent);
        console.log(xmlDoc.getElementsByTagName("fileUrl")[0].textContent);
    }
```

URL

Alternatively, all the API functions can be invoked from the address bar of a browser. The url can be created by including the parameter names and values after the CereVoice REST location. An example address would be:

```
https://cerevoice.com/rest/rest_1_1.php?method=speakExtended&accountID=accountID&password=
password&voice=Heather&text=speak+extended+rest+url&audioFormat=mp3&sampleRate=48000&audio3D
=True&metadata=True
```

Result Codes

```
# SUCCESS/ERROR
CS_FAILURE = 0
CS_SUCCESS = 1
```

Using Metadata

To obtain the metadata associated with the synthesised input, set the *metadata* parameter of the `speakExtended()` function to *TRUE*. Contained within the response from the CereVoice Cloud, will be an item `metadataUrl`. Like `fileUrl`, this will be the URL location of the metadata file. Metadata is in XML format, including start and end times for phonemes, along with stress and visemes (SAPI and Disney sets). Below is an example of the metadata output.

```
<trans>
<phone name="sil" start="0.000" end="0.030" sapi_viseme="0" disney_viseme="0" stress="0"/>
<mark name="cptk_0_0_5_5" start="0.030" end="0.030"/>
<word name="hello" start="0.030" end="0.030"/>
<phone name="hh" start="0.030" end="0.120" sapi_viseme="12" disney_viseme="9" stress="0"/>
<phone name="ax" start="0.120" end="0.160" sapi_viseme="1" disney_viseme="9" stress="0"/>
<phone name="l" start="0.160" end="0.292" sapi_viseme="14" disney_viseme="6" stress="0"/>
<phone name="ow" start="0.292" end="0.408" sapi_viseme="8" disney_viseme="12" stress="1"/>
<mark name="cptk_6_6_5_5" start="0.408" end="0.408"/>
<word name="world" start="0.408" end="0.408"/>
<phone name="w" start="0.408" end="0.535" sapi_viseme="7" disney_viseme="2" stress="0"/>
<phone name="er" start="0.535" end="0.635" sapi_viseme="5" disney_viseme="7" stress="1"/>
<phone name="l" start="0.635" end="0.775" sapi_viseme="14" disney_viseme="6" stress="0"/>
<phone name="d" start="0.775" end="0.845" sapi_viseme="19" disney_viseme="7" stress="0"/>
<phone name="sil" start="0.845" end="0.875" sapi_viseme="0" disney_viseme="0" stress="0"/>
<mark name="cprc_final" start="0.875" end="0.875"/>
<phone name="sil" start="0.875" end="0.905" sapi_viseme="0" disney_viseme="0" stress="0"/>
</trans>
```

Using 3D Audio

Introduction

The 3D Audio library allows generation of stereo output, and audio generated within a 3D space. The audio stream can be located within stereo space from +90 (full left) to -90 degrees (full right) using panning.

Alternatively, a "true 3D" positioning can be performed using HRTF obtained from the â LISTEN database. Any position from -180 to +180 degrees can be used, although it will be rounded to the closest 15 degree step. Elevation can range -45 to +45 degrees with steps of 15 degrees.

Audio Marks

3D audio effects are triggered by standard bookmarks in the XML input.

The marker names are:

- **NZConvolve_<filtername>** - Convolve all following audio items with this filter
- **NZPanning_<panning position =-90->+90>** - Pan the voice from this position
- **NZHrtf_<angle from -180 to 180>[;<elevation from -45 to 45>]** - Perform HRTF with this angle and elevation
- **NZHead_<hrtf id>** - Use HRTF head corresponding to that file; 50 heads from the LISTEN database are available, named listen1002 .. listen1059.

Example input:

```
<doc>
<voice name='Sarah'>
  <mark name='NZPanning_0'/>
  This is an example of zero panning angle, spoken by Sarah
</voice>
<voice name='Heather'>
  <mark name='NZPanning_45'/>This is an example of 45 degrees of Panning spoken with Heather
</voice>
<voice name='Jess'>
  <mark name='NZHrtf_-180'/>This is an example of an HRTF angle of minus 180 with Jess
</voice>
<voice name='Caitlin'>
  <mark name='NZHrtf_+45'/>This is an example of an HRTF elevation of plus 45 degrees with Caitlin
</voice>
</doc>
```

Using Multiple Voices

The CereVoice Cloud allows the use of multiple voices during synthesis. By placing tags around the input text, the user is able to assign specific text to specific voices. For example, if the following call was sent to the CereVoice Cloud:

```
speakSimple(accountID, password, "Stuart", "<doc>Hello. My name is Stuart. \
            This is my CereProc sister, Heather. \
            <voice name='Heather'>Hello, my name is Heather.</voice></doc>")
```

The text - *Hello, my name is Heather.* is spoken by *Heather*, whilst the rest is spoken by *Stuart*.

CereProc Tag Set

CereProc has implemented additional TTS functionality that is not part of the â 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, or to vary otherwise repetitive content. The variant number can be increased to produce 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>
```

Setting *variant="2"* produces another different version, and so on. 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. Please note that the variant tag should mainly be used for creating *static* prompts (i.e. audio files). The effect of the variant number is different between voices, and may also change when a new version of the same voice is produced (this is because the underlying speech engine is being constantly improved, and the default rendering may change).

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 include 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 *Adam*, *Caitlin*, *Heather*, *Isabella*, *Jack*, *Jess*, *Katherine*, *Kirsty*, *Laura*, *Sarah*,*Stuart*,*Suzanne*, *William*).

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 â log in to the CereProc website. Registered users can then access the â support request form. 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

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