CereProc cServer User Guide

Contents

Introduction

The cServer is a multi-channel Text-to-Speech (TTS) server for Windows (64-bit) and Linux environments. The CereProc client library, cSpeech, is supported on Windows (32 and 64-bit), Linux (including ARM variants) and

Mac OS X. Microsoft SAPI 5 is also supported for 32 and 64-bit platforms (Windows clients only with Windows or Linux cServer). Client-side wrappers for Python and Java are also available.

Installation

CereProc provides Microsoft Installer (MSI) files for Windows platforms, and Redhat Package Manager (RPM) files for Linux platforms.

Windows Operating System

CereProc digitally signs release MSI installers to ensure that they are valid. During installation the MSI vendor will identified as *CereProc Ltd*.

cServer and Voice Installation

Install the cServer MSI first. CereVoice 6.0.0 Windows cServer requires a 64-bit version of Windows (32-bit 4.0.x installers are available from support). Each MSI contains the cServer binaries and a compatible voice. The installer file name contains the voice and sample rate that is bundled with the server. For example, the cServer installer for the 48kHz version of the British English voice *Sarah* is called `cServerSarah48000.msi`.

From CereVoice 6.0.0, to ensure that your licenses to use CereProc voices don't time out unexpectedly, a new way to validate licenses was introduced. During the default GUI voice installation a license key validation window will appear. The preferred option is to use CereProc login details (the same you used to purchase the voices on our website). When this option is selected, the login website will be opened in the default Internet browser. After successful login, voice installation can be continued. The files needed for license verification will be downloaded along the voice being installed. There should be 4 files: 1) license file 2) license server certificate file (`root_certificate.pem`) 3) client certificate file (`<VID>_client.crt`) 4) client authentication key file (`<VID>_client.key`), where VID is the Voice ID used by CereProc license server to ensure the license for the specified voice is up-to-date. It is important that these file names are not changed. Alternatively, the license key text can be copied directly if it was sent by email. Enter the license key text and click *Validate*. If you do not have a license key, please raise a support request at http://www.cereproc.com/support. The voice will not load without a valid license.

Multiple voices are supported. The cServer configuration is automatically updated each time an additional voice is installed.

The cServer can be installed in *unattended* or *quiet* mode (for example with a command line install such as `msiexec.exe -i cServerSarah48000.msi -qn`). In this mode files needed for license verification must be installed manually. The 4 files needed for license verification should be saved to the directory containing the voice file. The license file should be called `license.lic`, the license server authentication file should be called `root_certificate.pem`, and the client authentication certificate and key file names should be of the format `<VID>_client.crt` and `<VID>_client.key` respectively where VID is the Voice ID used by CereProc license server to ensure the license for the specified voice is up-to-date. It is important that these file names are not changed.

The default location for the license verification files for the British English *Sarah* voice is `C:\Program Files\CereProc\cServer 6.0.0\voices\Sarah 6.0.0 48000\`. For example, the default license file path is at `C:\Program Files\CereProc\cServer 6.0.0\voices\Sarah 6.0.0 48000\license.lic`, the default license server certificate under `C:\Program Files\CereProc\cServer 6.0.0\voices\Sarah 6.0.0 48000\root_certificate.pem`, the client authentication certificate under `C:\Program Files\CereProc\cServer 6.0.0\voices\Sarah 6.0.0 48000\<VID>_client.crt` and key `C:\Program Files\CereProc\cServer 6.0.0\voices\Sarah 6.0.0 48000\<VID>_client.key` (replace `<VID>` with the actual VID provided).

cSpeech Client Installation

Run the `windows_client_installer.msi` installer. The installer includes the cSpeech client library, header files, Python and Java wrappers, 3rd party library dependencies, and example application (`cspeechclient`) with a Visual Studio Project and Cygwin Makefile. For simple configurations, where all clients use the Microsoft SAPI interface, it is not necessary to install the cSpeech client.

Compiling and running Java example client (Cygwin example) The `cspeech.jar` Java library and dependency `cspeech.dll` are installed in `C:\Program Files (x86)\CereProc\CereVoice cServer cSpeech Client 6.0.0\cSpeech\javalib` (`javalib64` for 64-bit versions). Example Java applications are included in `C:\Program Files (x86)\CereProc\CereVoice cServer cSpeech Client 6.0.0\examples\java`. To compile example Java client (for 32-bit):

```
/cygdrive/c/Java_x86/jdk1.8.0_144/bin/javac.exe \
-cp "c:/Program Files (x86)/CereProc/CereVoice cServer \
cSpeech Client 6.0.0/cSpeech/javalib/cspeech.jar" \
"c:/Program Files (x86)/CereProc/CereVoice \
cServer cSpeech Client 6.0.0/examples/java/cSpeechAudioClient.java" \
"c:/Program Files (x86)/CereProc/CereVoice \
cServer cSpeech Client 6.0.0/examples/java/cSpeechPlayerClient.java"
```

Note that absolute path of JDK compiler has been used here for clarity; your system may have different version and/or have `javac` set up as an alias. If "Access Denied", run Cygwin as administrator.

To run (32-bit), ensure third-party DLLs are present in the java library dir (i.e. copy all DLLs in `3rdparty/lib` into `cSpeech/javalib`) and do:

```
/cygdrive/c/Java_x86/jdk1.8.0_144/bin/java.exe \
-Djava.library.path="c:/Program Files (x86)/CereProc/CereVoice\
cServer cSpeech Client 5.0.0/cSpeech/javalib" \
-cp "c:/Program Files (x86)/CereProc/CereVoice cServer cSpeech \
Client 5.0.0/cSpeech/javalib/cspeech.jar; \
c:/Program Files (x86)/CereProc/CereVoice cServer cSpeech \
Client 5.0.0/examples/java" \
cSpeechAudioClient localhost 8989 in.xml out.raw
```

cSpeech SAPI Client Installation

Run the `sapi_client_installer.msi` installer. The installer adds a SAPI configuration tool in *Start Menu -> All Programs -> CereProc -> CereVoice cServer SAPI Client Voice Setup*. To allow SAPI to access a cServer voice, the voice information must be installed to the client machine registry. To install a SAPI voice on a client machine, run the voice setup application and enter the details of the cServer in the *Automatic setup* panel. The setup application adds the cServer voices to the SAPI registry, making them available to SAPI applications (SAPI applications may need to be restarted to enable the additional cServer voices).

Uninstallation

All cServer packages can be uninstalled using *Control Panel -> Add/Remove Programs* (*Programs And Features* under Windows Vista).

Linux Operating System

The CereProc cServer is supported on Red Hat Enterprise Linux (RHEL) version 5, 6, and 7 (cServer 4.x) and RHEL 6 and 7 (cServer 5.x) and RHEL 7 (cServer 6.x). Compatible distributions such as Centos and Scientific Linux are also supported.

CereProc digitally signs release RPM packages to ensure that they are valid. Before installing any other packages, the CereProc Release GPG key should be installed (as the *root* user) using:

```
rpm --import http://www.cereproc.com/files/RPM-GPG-KEY-cereprocrelease
```

cServer Installation

First install the cServer package (eg `rpm -Uvh cserver-6.0.0-0.i386.rpm` or `yum install cserver-6.0.0-0_el7.x86_64.rpm`. When using `yum` missing software dependencies are installed automatically. By default the `cserver` binary is installed in `/usr/bin/cserver`. Please contact CereProc support if dependencies are not available.

Voice Installation

Voice files and licenses are automatically discovered by the cServer in subdirectories of `/var/lib/cereproc/voices`. To install a voice, first extract the license zip file (found under the *Files* tab of the user's CereProc website account) in the `/var/lib/cereproc/voices` directory. The cServer Sarah 48k Standard license files, for example, would be extracted into `/var/lib/cereproc/voices/Sarah-48000-6.0.0`. Download the voice file to the same location, then restart the cServer.

The voice will not load without a valid license. Multiple voices in separate subdirectories are supported.

cSpeech Client Installation

Install the cSpeech client RPM package (eg `rpm -Uvh cserver-client-6.0.0-0.i386.rpm` or `yum install cserver-client-6.0.0-0.i386.rpm`). The cSpeech libraries are installed in `/usr/lib` or `/usr/lib64`, with `cspeech.h` in `/usr/include`. The cSpeech Python module is also installed. The example client applications, `cspeechclient` and `cspeechclient.py`, are installed to `/usr/bin`, with source code and Makefile in `/usr/src/cereproc`.

Compiling and running Java example client The `cspeech.jar` Java library and JNI wrapper `libcspeech.so` are installed in `/usr/lib/java` or `/usr/lib64/java`. Example Java applications are included in `/usr/src/cereproc`. To compile the example Java clients (use *lib64* on a 64-bit platform):

```
javac -cp /usr/lib/java/cspeech.jar cSpeechAudioClient.java
javac -cp /usr/lib/java/cspeech.jar cSpeechPlayerClient.java cSpeechAudioClient.java
```

Example commands to run the Java applications (use *lib64* on a 64-bit platform):

```
java -Djava.library.path=/usr/lib/java -cp /usr/lib/java/cspeech.jar:. cSpeechAudioClient \
localhost 8989 in.xml out.raw
java -Djava.library.path=/usr/lib/java -cp /usr/lib/java/cspeech.jar:. cSpeechPlayerClient \
localhost 8989 in.xml
```

Compiling and running C# example client The `cspeech.cs` C# interface and `libcspeech.so` wrapper are installed in `/usr/lib/csharp` or `/usr/lib64/csharp`. An example C# application is included in `/usr/src/cereproc`. To compile the example client (use *lib64* on a 64-bit platform):

```
mcs cSpeechClient.cs /usr/lib/csharp/cspeech.cs
```

Example commands to run the C# application (use *lib64* on a 64-bit platform):

```
export LD_LIBRARY_PATH=/usr/lib64/csharp
mono cSpeechClient.exe localhost 8989 in.xml out.raw
```

Uninstallation

All cServer packages can be uninstalled with an `rpm` or `yum` command, for example `rpm -e "cserver-*"` or `yum erase "cserver-*"`.

Additional Client Side Platforms

The cSpeech client supports additional platforms. The cSpeech library is also available on ARM Linux and Mac OS X, please contact CereProc Support to obtain downloads for these clients.

Running the cServer

Windows Operating System

The CereVoice cServer installs as a *Windows Service*. Control over starting and stopping the cServer is carried out using Microsoft's Computer Management Console (to access the Management Console, right-click on *My Computer* or *Computer* in the *Start Menu* and select *Manage*). The *cServer* Windows Service will be visible under *Services and Applications -> Services*.

The cServer will start automatically after cServer and voice installation (for a non-GUI installation, where a license key is not available, the cServer will exit immediately). If multiple voices are being installed, the cServer should be restarted after the final voice install. This allows all installed voices to become available on the cServer.

Windows Log File

By default, a log file is written to the cServer installation directory in `log\cserver.log` (for example `C:\Program Files\CereProc\cServer 6.0.0\log\cserver.log`). In the event of a problem with the cServer the log should be inspected for error information.

On cServer load a previous log file is renamed by appending the current date to the file name.

Command Line Running

The cServer can also be invoked from the command line on Windows, for example:

```
cserver.exe -c ..\config\config.xml
```

By default a Windows Service will be started. To run without starting a Windows Service, use the *-n* flag, for example:

```
cserver.exe -n -c ..\config\config.xml
```

Administrator privileges are required.

Linux Operating System

On platforms that use init scripts, the Linux cServer RPM installs a *cserver* init script to `/etc/init.d/cserver`. This enables the the *root* user to start the cServer with the `service cserver start` command. By default the cServer will start when the system is booted to run level 3, 4 or 5. The cServer can be shut down using `service cserver stop`.

For platforms using systemd (such as RHEL 7 or Centos 7), the cServer installs and enables a *cserver* service . The cServer can be started using `systemctl start cserver` and stopped with `systemctl stop cserver`.

Linux Log File

By default, a log file is written to `/var/log/cserver.log`. In the event of a problem with the cServer the log should be inspected for error information.

On cServer load a previous log file is renamed by appending the current date to the file name.

Command Line Running

The cServer can also be invoked from the command line as the user *root*, for example:

```
/usr/bin/cserver -c /etc/cserver.conf
```

Maximum Incoming Connections

The maximum number of connections, or *sessions*, for each cServer voice is set in the voice license. For example, a license containing the field `"SES": "50"` (or in case of older licenses, with the line `SES=50`) allows 50 concurrent sessions for the voice. If required, the total number of connections available on the cServer can be reduced in the configuration file (see the cServer Configuration Options section).

Exceeding the Incoming Connection Limit

If the cServer concurrent session limit is reached, clients remain able to **open** a connection to the cServer. This indicates that the cServer itself is still running. However if the maximum number of connections has been exceeded, the client will receive an error when a request is sent over the connection, and the connection is closed by the cServer.

Voice License Expiry

The voice license may contain an expiry date in the *EXP* field. For example, a license containing the field `"EXP": "20201231"` (or in case of older licenses, with the line `EXP=20201231`) would expire on December 31, 2020. The voice cannot be loaded after this date, however additional voices will continue to function if they have not expired.

cServer Configuration Options

cServer Configuration File

By default the cServer XML configuration file is installed to `C:\Program Files\CereProc\cServer 6.0.0\conf\config.xml` (Windows) or `/etc/cserver.conf` (Linux). On Windows, configuration files for a voice are typically downloaded during installation. For installation without network access the voice, license and certificates must be manually added to the configuration.

Example Linux configuration with automatic voice discovery:

```
<?xml version="1.0"?>
<server_config>
  <server port="8989" max_connections="500" log_file="/var/log/cserver.log" log_level="WARNING"/>
  <voices dir="/var/lib/cereproc/voices"/>
</server_config>
```

Example configuration file with two manually configured voices (paths are the Linux defaults):

```
<?xml version="1.0"?>
<server_config>
  <server port="8989" max_connections="500" log_file="/var/log/cserver.log" log_level="WARNING"/>
  <voices>
    <voice name="Sarah"
           file="/var/lib/cereproc/voices/Sarah-48000-6.0.0/cerevoice_6.0.0_sarah_48k.voice"
           license="/var/lib/cereproc/voices/Sarah-48000-6.0.0/Sarah123_license.lic"
           rootcertificate="/var/lib/cereproc/voices/Sarah-48000-6.0.0/root_certificate.pem"
           certificate="/var/lib/cereproc/voices/Sarah-48000-6.0.0/Sarah123_client.crt"
           key="/var/lib/cereproc/voices/Sarah-48000-6.0.0/Sarah123_client.key"
           default="false">
   </voices>
   <voice name="Heather"
           file="/var/lib/cereproc/voices/Heather-8000-6.0.0/cerevoice_6.0.0_heather_8k.voice"
           license="/var/lib/cereproc/voices/Heather-8000-6.0.0/Heather123_license.lic"
           rootcertificate="/var/lib/cereproc/voices/Heather-8000-6.0.0/root_certificate.pem"
           certificate="/var/lib/cereproc/voices/Heather-8000-6.0.0/Heather123_client.crt"
           key="/var/lib/cereproc/voices/Heather-8000-6.0.0/Heather123_client.key"
           lexicon="/var/lib/cereproc/en_sc_additional.lex"
           abbrev="/var/lib/cereproc/en.abb">
   </voices>
</server_config>
```

User-configurable Options

Server Options

The `<server>` element contains global cServer configuration information, and has several user-configurable attributes. The server listen port can be modified by changing the *port* attribute (all clients would need to access the cServer on the new port). The *max_connections* attribute defines a theoretical maximum number of channels for the server. However, the actual number of channels will depend on the *SES* value in the voice license. The *max_connections* attribute can be used to throttle down the number of connections to a number lower than the total specifed in the license *SES* value. The *log_level* attribute can be *ERROR*, *WARNING*, *INFO* or *DEBUG*, in order of increasing verbosity.

There are extra parameters that may be used to further customise the behaviour of the server, although they should only be used in special circumstances. The *send_timeout* and *receive_timeout* attributes can be used to modify the default timeouts (in ms) for connections. The default timeouts are respectively 5000 (i.e. 5 seconds) for sending data to a client, 600000 (60 seconds) for receiving data. In case a setup requires persistent connections (even if there is no

data being transferred), the *receive_timeout* should be modified. It can either be set to a larger value, or to "-1" for no timeout. For a heavily loaded server on a high latency network, if some "send_timeout" events are detected, the *send_timeout* attribute may be modified to a larger value.

Voice Options and User Lexicons

The `<voice>` elements contain the configuration information for individual voices. The *default* attribute can be used to set the default voice on a cServer with multiple voices. If all voices are set to `default="false"`, the first voice in the configuration file is used as the default.

User Lexicons The `<voice>` element can take an optional attribute, *lexicon*, which points to a lexicon file specified by the user.

Example lexicon line (English RP):

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

The basic lexicon format is a *headword* followed by a *pronunciation*. The *headword* must **not** contain numbers or punctuation characters (to process a string that includes non-alphabetic characters, use an abbreviations file).

Vowel phonemes, such as *@@*, *ii* and *ou* in the example pronunciation, should have stress levels. These are *1* for *primary stress*, *2* for *secondary stress*, and *0* for *no stress*.

Each accent has a different phone set. Different user lexicons would be required for British and American voices, for example the *Megan* English GA voice and the *Jess* English RP voice. 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 CereVoice Phone Sets document for example pronunciations.

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

User lexicons can be reloaded dynamically, see the section on *Controlling the cServer with Queries* for more information.

User Abbreviations The `<voice>` element can take an optional attribute, *abbrev*, which points to an abbreviations file specified by the user.

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*).

SSML and User Lexicons An SSML URI can be used to specify a user lexicon for a particular document, for example:

```
<speak>
<lexicon uri="http://www.example.com/user.lex"/>
Hello Mr Mourinho.
</speak>
```

SSML and User Abbreviations An SSML URI can be used to specify user abbreviations for a particular document, for example:

```
<speak>
<lexicon uri="http://www.example.com/user.abb" type="cprc.abbrev"/>
At the 7/11.
</speak>
```

Voice and License Validation File Paths

CereProc do not recommend changing the file paths unless absolutely necessary. If problems are encountered after changing paths the log file should be checked for messages.

Monitoring options

Starting with version 3.0.0 of the cServer, automatic tools for monitoring the load of the server have been introduced. These tools are disabled by default. They can be activated by adding the following line to the server configuration:

```
<usage rootname="/var/log/cserver_usage" sampling_period="600" genere_period="86400"/>
```

The `<usage>` has three attribute to control the behaviour of the monitoring functions. The *rootname* attribute allows the user to specify the location of the log files generated. The *sampling_period* (in seconds) controls how often the load should be polled. The *genere_period* (in seconds) controls how often a new log file should be generated.

cSpeech Client

The cSpeech client library can be used for integration into a C, C++, Python, or Java client application.

To perform the simplest possible synthesis request, client applications must:

- Create a callback to handle the cServer response
- Open a connection to a running cServer
- Send text or XML to the cServer (*must* be null-terminated if C/C++)

cSpeech Client Integration

Minimal cSpeech Client Code Examples

Note that these examples do not test that API calls succeed. In a deployment system return codes should be checked (for example, if a cServer is unavailable *CS_SUCCESS* will not be returned when opening a connection).

C++ cSpeech Client Example

```
#include <stdio.h>
#include "cspeech.h"

FILE * fp;
int callback(void * user, int event_id, void * data, long data_length);

// Minimal cSpeech callback function to write audio to a file
int callback(void * user, int event_id, void * data, long data_length){
    switch(event_id) {
    case PACKET_AUDIO:
    fwrite(data, data_length, 1, fp);
    break;
    }
    // Callback must return success or future calls will be skipped
    return CS_SUCCESS;
}

int main(int argc, char **argv) {
    // Speak 'Hello world' to a file using cSpeech
    fp = fopen("out.raw", "wb");
    CS_CONNECTION connection = csCreateConnection();
    csOpenConnection(connection, "localhost", 8989);
    csSetCallBack(connection, callback);
    // The input string must be null terminated
    csRenderSpeech(connection, "Hello world");
    csDestroyConnection(connection);
}
```

Python cSpeech Client Example

```
import cspeech
import array

# Minimal cSpeech callback class to write audio to a file
class callback:
    def __init__(self, outfp):
        self.outfp = outfp
```

```
    # This function is called by cSpeech when data is returned
    def callback(self, event_id, data_handle, data_length):
        if event_id == cspeech.PACKET_AUDIO:
            # Convert audio to an array of shorts and write data
            audiodata = cspeech.data_to_audio(data_handle)
            shortarray = array.array("h")
            for i in range(data_length/2):
                shortarray.append(cspeech.audio_val(audiodata, i))
            self.outfp.write(shortarray.tostring())
        return cspeech.CS_SUCCESS

# Speak 'Hello world' to a file using cSpeech
outfp = open('out.raw', 'w')
mycallback = callback(outfp)
connection = cspeech.csCreateConnection()
cspeech.csOpenConnection(connection, "localhost", 8989)
cspeech.set_callback(connection, mycallback)
cspeech.csRenderSpeech(connection, "Hello world")
cspeech.csDestroyConnection(connection)
```

Alternative Synthesis Calls

cSpeech offers six synthesis calls, depending on whether the user requires XML parsing, asynchronous processing, or input data streaming. In XML mode it is *essential* that the synthesis request forms an XML document with valid opening and closing tags. If broken XML is synthesised the server may begin reading XML content inappropriately.

Synthesis call	Input type	Blocking	Streaming
csRenderSpeech	Text	Yes	No
csRenderSpeechXML	XML	Yes	No
csRenderSpeechAsync	Text	No	No
csRenderSpeechXMLAsync	XML	No	No
csPushSpeechStream	Text	Yes	Yes
csPushSpeechStreamXML	XML	Yes	Yes

Using the Asynchronous Calls Where the asynchronous calls are being used, the calling program may need to wait for synthesis to finish. To achieve this, the callback function should check for a *PACKET_THREAD_FINISHED* event and inform the main program that synthesis is complete on the callback thread (see the `cspeechclient.cc` code for an example).

Using the Streaming Calls The streaming calls allow data to be streamed over the connection in chunks. For example:

```
csPushSpeechStreamXML(connection, "<speak>\n");
csPushSpeechStreamXML(connection, "Hello ");
```

```
csPushSpeechStreamXML(connection, "world");
csPushSpeechStreamXML(connection, "\n</speak>\n");
csFlushSpeechStreamXML(connection);
```

Is equivalent to:

```
csRenderSpeechXML(connection, "<speak>\nHello world\n</speak>\n")
```

Strings *must* be null terminated. If the output contains extra data that was not in the input, it may be a null termination problem.

The streaming mode may be preferred if a large amount of data is being processed and the application splits the input data into pieces. See the `cspeechclient.cc` code for another example.

Example Applications in C++, Python, and C#

The example applications `cspeechclient.cc`, `cspeechclient.py`, and `cSpeechClient.cs` contain examples of using the cSpeech client in a variety of configurations. They include error-checking when making cSpeech library calls, and contain examples of using the various text, XML, synchronous, asynchronous, and streaming modes.

cSpeech Header File

The header file `cspeech.h` contains full documentation on all of the cSpeech API calls.

Controlling the cServer with Queries

The cServer implements an XML query API, along with some helper functions, allowing various types of non-TTS request to be sent to the cServer.

Obtaining Voice Information

Information about the available cServer voices is obtained using the *csEnumerateConnectionVoices* function. The function returns an iterator, and requires an open connection to a cServer to be supplied as its argument. For example (Python):

```
voice_enumerator = cspeech.csEnumerateConnectionVoices(connection)
voice_name = cspeech.csNextVoice(voice_enumerator)
while(voice_name):
    print "Voice name is", voice_name
    print "Sample rate is", \
      cspeech.csGetVoiceIntegerValue(voice_enumerator, cspeech.CS_VOICE_INTEGER_SAMPLE_RATE)
    print "Gender is", \
      cspeech.csGetVoiceStringValue(voice_enumerator, cspeech.CS_VOICE_STRING_GENDER)
    voice_name = cspeech.csNextVoice(voice_enumerator)
cspeech.csCloseEnumerator(voice_enumerator)
```

See *cspeechclient.cc* or *cspeechclient.py* for a full example. The *cspeech.h* header also includes details of the additional information that can be read from the enumerator. Information on language, country, accent, compression codec, and bits per sample is available.

Selecting a Voice

The cServer voice can be selected with an XML query, for example:

```
csQueryServer(connection, "<voice name='Heather' select='1'/>");
```

When the named voice is unavailable, the request is ignored and the default voice continues to be used.

If the cSpeech client callback is set up to capture *PACKET_XML_COMMAND* responses, the client will receive an XML string containing information about the voice that has been selected. If the voice is not available the XML document body returned is `<xml_reply><voice name="Unknown"/></xml_reply>`.

As an alternative to using the voice enumerator function, the client can use `csQueryServer(connection, "<voice name='all'/>")` to retrieve an XML string containing information about all cServer voices.

Reloading a User Lexicon

The cServer is able to dynamically reload a user lexicon into a running voice. The reload must be triggered via an XML query, for example:

```
csQueryServer(connection, "<reloadlex voice='Heather'/>");
```

To reload all user lexicons, use *reloadlex* without any attributes:

```
csQueryServer(connection, "<reloadlex/>");
```

Note: the reload function requires the lexicon to be specified in the cServer configuration at the time the cServer was loaded. When adding a user lexicon to a voice for the first time the cServer must be restarted to enable user lexicon functionality for the voice.

Obtaining cServer Load Information

The *getserverinfo* query can be used to retrieve the total number of available connections and the number currently in use, for example:

```
csQueryServer(connection, "<getserverinfo/>");
```

The response is sent to the callback as a *PACKET_XML_COMMAND* event containing an XML document. A 20-port server with 5 ports in use would return:

```
<xml_reply>
  <server connections="5" max_connections="20"/>
</xml_reply>
```

Note that the XML query itself is sent over a connection, so the minimum value for *connections* is *1*.

Shutting Down a cServer Remotely

The cServer can be shut down by a remote cSpeech client using an XML query:

```
csQueryServer(connection, "<shutdown/>")
```

SAPI Client (Windows clients only)

The SAPI client supports phoneme, viseme, word, sentence, and marker events. 32 and 64-bit versions of Windows are supported.

SAPI rate, pitch, emphasis, volume, and break settings are supported.

SAPI XML mode must be used if the client sends SSML or CereProc XML markup. If SAPI XML mode is not used XML characters are read out.

Note that the cSpeech SAPI client does not allow XML queries to be sent to the cServer, for example to reload a user lexicon. To access these API functions, install the cSpeech client library and send *csQueryServer* requests via the cSpeech client.

CereProc cServer MRCP Connector

The CereProc cServer MRCP Connector allows MRCP version 1 and 2 clients to access a cServer for TTS generation. The integration is implemented as a plugin for the open-source, open-standard UniMRCP Server. The connector runs as a service, converting MRCP requests into cSpeech API calls, sending them to a cServer, and returning the resulting audio to an MRCP client.

Installation

The default installation assumes that the cServer is running on the install machine on port 8989. If the cServer is running on a different machine and/or port, see the advanced configuration section.

Windows Operating System

1. Ensure the cServer is installed and running correctly, as above.
2. Download and install the UniMRCP `unimrcp-1.0.0.exe` package from http://code.google.com/p/unimrcp/downloads/detail 1.0.0.exe (the MRCP cServer Connector is compatible with the 32-bit version of UniMRCP), during installation uncheck *Demo synthesizer plugin* on the *Select components* screen.
3. Install the cServer MRCP Connector (`mrcp_connector_installer.msi`).

Note that if the *Demo synthesizer plugin* is installed the *Demo-Synth-1* plugin must be set to *enable="false"* in the `unimrcpserver.xml` configuration file.

Linux Operating System

1. Ensure the cServer is installed and running correctly, as above.
2. Install the cpunimrcpserver RPM (for example `cpunimrcpserver-6.0.0-0_el7.x86_64.rpm` in the case of the RHEL 7 version).

Running the cServer MRCP Connector

Windows Operating System

The cServer MRCP Connector installs as a *Windows Service*. Control over starting and stopping the connector is carried out using Microsoft's Computer Management Console (to access the Management Console, right-click on *My Computer* or *Computer* in the *Start Menu* and select *Manage*). The *UniMRCP Server* Windows Service will be visible under *Services and Applications -> Services*. Right click the service name and select *Properties* to enable automatic startup of the service.

Windows Log File

By default, a `unimrcpserver-0.log` file is written to the `log` directory in the UniMRCP installation directory. In the event of a problem with the MRCP connector, the log should be inspected for error information. The *UniMRCP Server* service *Properties* pane can also be used to change the logging level by adding the `-l` flag, `-l 7` enables debug logging, `-l 0` emergency logging.

Linux Operating System

The Linux cServer MRCP Connector RPM installs a *cpunimrcpserver* init script to `/etc/init.d/cpunimrcpserver`. This enables the the *root* user to start the cServer MRCP Connector with the `service cpunimrcpserver start` command. By default the connector will start when the system is booted to run level 3, 4 or 5. The connector can be shut down using `service cpunimrcpserver stop`.

Linux Log File

By default, a log file is written to `/var/lib/cereproc/mrcp/log/unimrcpserver-0.log`. In the event of a problem with the connector, the log should be inspected for error information.

Command Line Running

The cServer MRCP Connector can also be invoked from the command line as the user *root*, for example:

```
/usr/bin/cpunimrcpserver -r /var/lib/cereproc/mrcp
```

In this mode, log messages are printed to the terminal. The log level can be changed by adding the `-l` flag, `-l 7` enables debug logging, `-l 0` emergency logging.

Advanced cServer MRCP Connector Configuration

The MRCP connector can access a cServer running on a remote host or a different port to the default 8989. Configuration information is stored in the `unimrcpserver.xml` configuration file. On Windows, this file is found in the `conf` directory of the UniMRCP installation. On Linux, the file is in `/var/lib/cereproc/mrcp/conf/unimrcpserver.xml`.

To change the cServer or port that the MRCP connector uses to generate TTS, edit the cSpeech information in the section. For example, to change the port to *9999* and the host name to *host.example.com*, the cSpeech *<engine>* section would be changed to:

```
<engine id="CPRC-Synth-1" name="libcspeech_synth_engine_shared" enable="true">
  <!-- Host name and port of the cServer to connect to -->
  <param name="host" value="host.example.com"/>
  <param name="port" value="9999"/>
</engine>
```

High Definition Calls

The cServer MRCP connector supports high definition 16kHz speech output. The installed cServer voices 16kHz to support this mode. To enable high definition support, edit the *<settings>* section of the `unimrcpserver.xml` configuration file to support high definition codecs:

```
<codecs own-preference="false">PCMU PCMA PCMU/97/16000 PCMA/98/16000 L16/99/16000</codecs>
```

Remove the existing line containing 8kHz codecs only. MRCP clients can then request and receive 16kHz output.

MRCP Clients

Note that when running the connector, cServer, and an MRCP client on the same machine, it might be necessary to use the machine's host name (not *localhost* or *127.0.0.1*) when running the client.

Compatible clients include UniMRCP (MRCP v1 and v2), Aculab (MRCP v1), Voxeo (MRCP v1), Voxpilot (MRCP v1) and Holly Connnects (MRCP v1, reported by the UniMRCP project).

Aculab Configuration

The Aculab MRCP server compatiblity test is found in the AIT in `example_code/mrcp/server_compatibility_test`. It can be run with:

```
./mrcp_sct -s host.example.com -t -p 1554 -y speechsynthesizer -r speechrecognizer
```

Voxpilot Configuration

First edit the Voxpilot SSML configuration (by default `C:\Voxpilot\ssmlprocessor\conf\ssmlprocessor.conf`), and set *NumChannelsMRCP*. For the free 2-port verision use *NumChannelsMRCP=2*. Then edit the MRCP

configuration (by default `C:\Voxpilot\ssmlprocessor\conf\mrcp.xml`), adding an entry for the cServer MRCP Connector:

```
<!-- Note that when running the UniMRCP and Voxpilot servers on the same machine, it might be necessary
     to use the machine name (not ''localhost'' or ''127.0.0.1'')
-->
<resource lang="en-GB" resourcetype="speechsynth" vendor="CereProc" persist="true">
    <server uri="rtsp://host.example.com:1554/speechsynthesizer" priority="0"/>
</resource>
```

Restart the Voxpilot SSML Processor service and the cServer voices will be available on the Voxpilot server.

Voxeo Prophecy 10 Configuration

Prophecy 10 is the first version of Prophecy to support MRCP. Earlier versions of Prophecy are not supported.

If Prophecy and the cServer MRCP Connector are going to be on the same server, the connector configuration file must be edited so that `<ip>127.0.0.1</ip>` is set under `<properties>` (see the Advanced configuration section for information on editing the connector configuration). If the value is not set, the connector binds to the host name, but Prophecy assumes `127.0.0.1`.

In a web browser, connect to the Prophecy Commander web interface (eg http://host.example.com:9996). Click the *Servers* button on the top bar, and select the server name (for example *host.example.com*). An *Edit Server* box appears at the bottom of the page. Click the *Configuration* tab, then *Add* (under *Services*). Select *TTS* then *Loquendo Speech Server MRCPv1 (v7)*, choose a voice, and confirm. Back in the *Edit Server* window, expand the new entry for the MRCP voice and change the port to *1554*. Then click *Save*. The MRCP resource is then added to the server.

Next, change the virtual platform to use the new resource. Under the *Servers* tab, click *Virtual Platform List*, select *Default*, and the *TTS* role line, select *Default*, and change the *Service* to the new *Loquendo Speech Server MRCPv1 (v7)* voice. After confirming the choices, click *Save* under *Edit Virtual Platform* and the changes will be applied.

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. SSML includes tags for modifying pitch, rate, and pronunciation. It also supports inserting audio, markers, and breaks. Usage, with examples, is described on the W3C page. The SSML tags currently supported in CereVoice can be found in the table below. Custom user pronunciations are supported using CereProc phone sets and IPA as part of the Pronunciation Lexicon Specification (PLS).

Supported SSML Tags

Tag	Supported
audio	yes (HTTP and file URLs, sample rate should match text-to-speech output)

Tag	Supported
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 (VoiceXML builtins are supported, see below
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

VoiceXML Built-in List

CereProc supports VXML built-in grammar types as *say-as* tags, allowing interaction with the output of a VXML recogniser. For example, `<say-as interpret-as="vxml:time">1230p</say-as>` would be read as *twelve thirty P M* by an English voice. The VXML built-in tags are the only open standard for *interpret-as* values. Normally, the input is produced by a speech recogniser running in a VXML environment. However, the input formats can also be used directly to ensure consistent text processing.

Available *interpret-as* values for *say-as* tags:

interpret-as value	Input
vxml:boolean	*1 = true*, *2 = false*
vxml:date	Format is *yyyymmdd*, unspecified values can be replaced with *????* or *??*
vxml:digits	String of digits
vxml:currency	Format is *UUUmm.nn*, where *UUU* is the ISO4217 currency code, and *mm.nn* the currency amount
vxml:number	String of digits, optionally including a decimal point and/or a plus/minus sign
vxml:phone	String of digits, optionally including *x* for extension
vxml:time	Format is *hhmmx*, where *x* is *a*, *p* or *h* for AM, PM, or 24 hour clock respectively

See the VoiceXML 2 documentation Appendix P for more information.

CereProc Tag Set

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

Variant Tags

Please note that the variant tag is not currently supported by CereWave AI DNN voices

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. 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 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>
```

Troubleshooting

The majority of cServer problems can be solved by checking the cServer log file. The log file will contain warning or error messages in most cases where problems occur. Errors are logged in the following cases, for example:

- Voice file is invalid or missing
- License file is invalid, expired or missing
- User lexicon is invalid or missing
- Listen port is unavailable
- Maximum number of connections exceeded

Obtaining Support

CereProc offers support for the cServer 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/suppo
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

Gesture ID	Gesture content
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