云知声语音云开发指南

C 语言接口

北京云知声信息技术有限公司

Beijing Unisound Information Technology Co., Ltd

重要声明

版权声明

商标声明

北京云知声信息技术有限公司的产品是北京云知声信息技术有限公司专有。在提及其他公司及其产品时将使用各自公司所拥有的商标，这种使用的目的仅限于引用。本文档可能涉及北京云知声信息技术有限公司的专利（或正在申请的专利）、商标、版权或其他知识产权，除非得到北京云知声信息技术有限公司的明确书面许可协议，本文档不授予使用这些专利（或正在申请的专利）、商标、版权或其他知识产权的任何许可协议。

不作保证声明

北京云知声信息技术有限公司不对此文档中的任何内容作任何明示或暗示的陈述或保证，而且不对特定目的的适销性及适用性或者任何间接、特殊或连带的损失承担任何责任。本手册内容若有变动，恕不另行通知。本手册例子中所用的公司、人名和数据若非特别声明，均属虚构。未得到北京云知声信息技术有限公司明确的书面许可，不得为任何目的、以任何形式或手段（电子的或机械的）复制或传播手册的任何部分。

保密声明

本文档（包括任何附件）包含的信息是保密信息。接收人了解其获得的本文档是保密的，除用于规定的目的外不得用于任何目的，也不得将本文档泄露给任何第三方。

本软件产品受最终用户许可协议（EULA）中所述条款和条件的约束，该协议位于产品文档和/或软件产品的联机文档中，使用本产品，表明您已阅读并接受了 EULA 的条款。

1. 概述

Linux、Windows 语音平台版旨在使第三方应用便利的集成和使用云知声的语音识别及语音合成服务。

1.1. 目的

本文档对 Linux、Windows 语音平台接口定义进行说明。

文档读者为使用 Linux、Windows 语音平台接口进行开发的产品设计师、软件工程师。

1.2. 范围

本文档定义语音识别、语音合成的使用说明、体系结构、API 接口。

不包含核心引擎的性能定义，也不包含其它配套或附赠产品的使用说明。

2. 使用说明

2.1. 开发说明

在开发应用程序时，仅需关注文档中所提供的接口函数而不用了解具体实现。

2.2. 开发前准备

对于个人开发者，使用语音服务，需要经过我们的授权，请到“http://dev.hivoice.cn”注册成为我们的开发者，并为所开发的软件申请 App Key。

2.3. 支持的平台

支持 Linux、Windows 操作系统。

3. 环境搭建

用户需使用 Linux、Windows 操作系统，安装 Linux、Windows 开发工具，使用公有云需要能够连接网络。

Linux 系统运行 demo 步骤：进入到/sample 目录下，执行./run_sample.sh 命令即可。

windows 系统运行 demo 步骤：进入到/test 目录下，双击执行 run_demo.bat 文件即可。

4. 语音识别函数接口

4.1. 简述

开发者只需简单调用几个方法就可以使用云知声语音转写服务，接口包括创建识别 usc_create_service，设置参数 usc_set_option，登录语音识别服务 usc_usc_login_service，启动识别 usc_start_recognizer，上传语音数据识别并收取数据 usc_feed_buffer，从缓冲获得识别结果 usc_get_result，停止识别 usc_stop_recognizer，释放识别接口实例 usc_release_service。

4.2. 语音识别接口说明

4.2.1. 创建识别接口

创建识别接口实例。

```
int usc_create_service(USC_HANDLE* handle)
```

参数

handle [out]

识别实例引用句柄指针。

返回

如果调用成功返回USC_OK，否则返回错误代码参见附1错误代码表。

示例

```
// 创建识别实例
USC_HANDLE handle = 0;
int ret = usc_create_service(&handle);
```

```
    if(ret != USC_OK) {
        fprintf(stderr,"usc_create_service error %d\n", ret);
    }
```

说明

创建语音识别接口实例，handle返回接口实例引用，完成识别后必须调用函数usc_release_service释放接口实例，否则会产生内存泄露。

4.2.2. 创建识别接口（私有云）

创建识别接口实例。

```
int usc_create_service_ext(USC_HANDLE* handle, const char*  host, unsigned short port)
```

参数

handle [out]
识别实例引用句柄指针。

host
识别服务器地址

port
识别服务器端口号

返回

如果调用成功返回USC_OK，否则返回错误代码参见附1错误代码表。

示例

```
    // 创建识别实例
    USC_HANDLE handle = 0;
    int ret = usc_create_service_ext(&handle, "192.168.1.2", 80);
    if(ret != USC_OK) {
        fprintf(stderr,"usc_create_service error %d\n", ret);
    }
```

说明

创建语音识别接口实例，handle返回接口实例引用，完成识别后必须调用函数usc_release_service释放接口实例，否则会产生内存泄露。

4.2.3. 设置识别参数

设置识别所需的参数。

```
int usc_set_option(HANDLE handle,int  option_id, const  char* value)
```

参数

handle

识别接口实例句柄。

option_id

设置参数的id。

value

设置参数option_id所对应的值。

参数 ID	参数值
USC_OPT_SERVICE_KEY （必设）	应用程序授权 key，根据此参数关联应用程序服务后台信息。
USC _OPT_USER_SECRET（必设）	设置用户 secret。
USC_OPT_INPUT_AUDIO_FORMAT	指定输入语音数据格式，目前支持16000/8000Hz 16bit 单声道 PCM，对应值为;AUDIO_FORMAT_PCM_16K、AUDIO_FORMAT_PCM_8K。 16k 采样率的语音音质会明显优于 8k，这使得语音识别的准确度得到更好的保证，建议选用16k 采样率。
USC_OPT_RECOGNITION_FIELD	根据语音应用场景选择对应的识别领域，可以改善识别效果。目前支持领域如下;RECOGNITION_FIELD_GENERAL(通用)、RECOGNITION_FIELD_POI(POI)、RECOGNITION_FIELD_SONG (音乐)、RECOGNITION_FIELD_MEDICAL (医药)、RECOGNITION_FIELD_MOVIETV (影视)。
USC_OPT_NLU_PARAMETER	设置语义参数，详情见语义参数文档。返回格式为 json 样例：usc_set_option(handle, USC_OPT_NLU_PARAMETER, "filterName=search;returnType=json;");
USC_OPT_PUNCTUATION_ENABLED	设置是否使用标点符号。

返回

调用成功返回USC_OK，否则返回错误代码参见附1错误代码表。

说明

识别开始前进行参数配置，USC_OPT_SERVICE_KEY 与 USC _OPT_USER_SECRET 为必须参数，输入语音数据格式如不配置默认为 AUDIO_FORMAT_PCM_16K。识别领域如果不设置默认使用通用领域。

注：目前 appkey 与 SDK 包已绑定，不同 SDK 间 appkey 混用会出现授权错误(-91725)。

示例

```
// 设置appkey
int ret = usc_set_option(handle, USC_OPT_APP_KEY, appKey);
if(ret != USC_OK) {
    fprintf(stderr,"usc_set_option error %d\n", ret);
}
```

4.2.4. 登录识别服务

连接识别服务器开始识别。

```
int  usc_login_service (HANDLE handle)
```

参数

handle

识别接口实例句柄。

返回

调用成功返回USC_OK，否则返回错误代码参见附1错误代码表。

说明

连接识别服务器启动识别。一次调用本函数 login 成功之后，可以多次调用 usc_start_recognizer、usc_feed_buffer、usc_stop_recognizer 函数进行多次识别，详情参照示例。

示例

```
// 连接服务器启动识别
int ret = usc_login_service (handle);
if(ret != USC_OK ) {
    fprintf(stderr,"usc_login_service error %d\n", ret);
}
```

4.2.5. 启动识别

连接识别服务器开始识别。

```
int  usc_start_recognizer(HANDLE handle)
```

参数

handle

识别接口实例句柄。

返回

调用成功返回USC_OK，否则返回错误代码参见附1错误代码表。

说明

连接识别服务器启动识别。

示例

```
// 连接服务器启动识别
int ret = usc_start_recognizer(handle);
```

```
    if(ret != USC_OK ) {
        fprintf(stderr,"usc_start_recognizer error %d\n", ret);
    }
```

4.2.6. 识别语音数据

上传语音数据进行识别并检测识别结果。

```
int usc_feed_buffer(HANDLE handle,const unsigned char* buffer,int lenght)
```

参数

`handle`

识别接口实例句柄。

`buffer`

语音数据起始地址。

`lenght`

有效语音数据的长度。

返回

返回识别状态，数值说明见下表;

返回值	说明
USC_OK = 0	识别正常
USC_RECOGNIZER_PARTIAL_RESULT=2	有识别结果需要取出
USC_RECOGNIZER_SPEAK_END=101	检测到语音中有说话停顿
小于0	小于零为错误，错误代码参见附1错误代码表

说明

向服务器发送语音数据，并实时接收识别结果，如果识别返回状态为 USC_RECOGNIZER_PARTIAL_RESULT、USC_RECOGNIZER_SPEAK_END 需要及时获取识别结果，否则会被下一次识别结果覆盖。如果返回小于零的错误代码需要终止当前识别。

示例

```
    std::string result = "";
    char buff[1024];
    int ret = 0;
    // 循环上传语音数据进行识别
    while(true){
        int nRead = fread(buff,sizeof(char), sizeof(buff),fpcm);
        if(nRead <= 0) {
            break;
```

```
        }
        ret = usc_feed_buffer(handle, buff, nRead);
        if(ret == USC_RECOGNIZER_PARTIAL_RESULT ||
           ret == USC_RECOGNIZER_SPEAK_END ) {
            // 获取中间部分识别结果
            result.append(usc_get_result(handle));
        }
        else if( ret < 0) {
            // 网络出现错误退出
            fprintf(stderr,"usc_feed_buffer error %d\n", ret);
            break;
        }
    }
```

4.2.7. 获得识别结果

获取缓存的识别结果。

```
const char*  usc_get_result(HANDLE handle)
```

参数

handle

识别接口实例句柄。

返回

返回识别结果。

说明

根据 usc_feed_buffer 方法返回状态及时取出识别结果，否则会被后续的结果覆盖。usc_stop_recognizer 方法执行后需要获取识别结果。

示例

参见 usc_feed_buffer

4.2.8. 停止识别

停止识别。

```
int  usc_stop_recognizer(HANDLE handle)
```

参数

handle

识别接口实例句柄。

返回

成功返回 USC_OK，失败返回小于零错误代码参见附 1 错误代码表。

说明

向服务器发送请求结束识别，获取结果关闭服务器网络连接。执行后需要调用 usc_get_result 获取本地缓存识别结果。

示例

```
// 停止语音输入
int ret = usc_stop_recognizer(handle);
if(ret != USC_OK) {
    // 网络出现错误退出
    fprintf(stderr,"usc_stop_recognizer error %d\n", ret);
}
else {
    // 获取剩余识别结果
    printf(usc_get_result(handle));
}
```

4.2.9. 释放识别接口

释放识别接口实例。

```
void  usc_release_service(HANDLE handle)
```

参数

handle

识别接口实例句柄。

说明

释放识别接口实例回收占用资源。调用些方法后 handle 接口实例句柄不能再用于识别，否则会产生异常。

示例

```
// 释放识别接口实例
usc_release_service(handle);
handle = 0;
```

4.3. 语义理解接口说明

4.3.1. 设置参数

设置识别所需的参数。备注：在请在识别创建之后中调用

```
int usc_set_option(HANDLE handle,int option_id, const char* value)
```

参数

handle

识别接口实例句柄。

option_id

语义id固定值为：USC_OPT_NLU_PARAMETER

value

设置参数option_id所对应的值。为“key=value;”格式。

事例

usc_set_option(handle, USC_OPT_NLU_PARAMETER,

"filterName=search;returnType=json;");

具体参数见语义文档。

5. 语音识别扩展功能

5.1.1. 语音时间信息

5.1.2. 设置 VAD 参数

设置长语音 VAD 断句参数

```
void usc_vad_set_timeout(USC_HANDLE handle, int frontSil,int backSil)
```

参数

handle

识别接口实例句柄。

frontSil

语音中不说话超时断句时间,单位为毫秒(ms)。

backSil

语音中停止说话断句时间，单位为毫秒(ms),范围为 200~3000ms。

说明

长语音通过 VAD 断句，通过调节 backSil 参数改变 VAD 断句灵敏度。方法 usc_feed_buffer 检测到说话结束返回标志 USC_RECOGNIZER_SPEAK_END 代表一句话结束。需要及时获取识别结果，否则会被下一次识别结果覆盖。同时可通过方法 usc_get_result_begin_time、usc_get_result_end_time 获取当前句子时间信息。

对于背景噪音大的语音,VAD 断句可能出现不准确现象。

示例

```
// 设置VAD断句参数改变断句灵敏度
usc_vad_set_timeout(handle, 3000,1000);
```

5.1.3. 语音开始位置

获取当前识别结果对应的开始时间

```
int usc_get_result_begin_time(USC_HANDLE handle)
```

参数

handle

识别接口实例句柄。

返回

当前语音说话开始位置，单位为毫秒。

说明

具体使用参考示例代码。

5.1.4. 语音结束位置

获取当前识别结果对应的结束时间

```
int usc_get_result_end_time(USC_HANDLE handle)
```

参数

handle

识别接口实例句柄。

返回

当前语音说话停止位置，单位为毫秒。

说明

具体使用参考示例代码。

示例代码

```
// 打开语音文件(16K/8K 16Bit pcm)
FILE* fpcm = fopen("16k.pcm", "rb");
if( fpcm == NULL) {
    fprintf(stderr,"Cann't Open File\n");
    return -1;
}
std::string result = "";
char buff[640];
int ret = 0;

USC_HANDLE handle = 0;
// 创建识别实例
ret = usc_create_service(&handle);
if(ret != USC_OK) {
    fprintf(stderr,"usc_start_recognizer error %d\n", ret);
    goto close_files;
}

// 设置识别AppKey
ret = usc_set_option(handle, USC_OPT_APP_KEY, USC_ASR_SDK_APP_KEY);
// 设置不说话停止判断值的灵敏度
usc_vad_set_timeout(handle, 3000,600);

// 启动识别
ret = usc_start_recognizer(handle);
if(ret != USC_OK ) {
    fprintf(stderr,"usc_start_recognizer error %d\n", ret);
    goto usc_release;
}

while( true){
    int nRead = fread(buff,sizeof(char), sizeof(buff),fpcm);
    if(nRead <= 0) {
        break;
```

```
        }

        ret = usc_feed_buffer(handle, buff, nRead);
        if(ret == USC_RECOGNIZER_PARTIAL_RESULT) {
            // 获取中间部分识别结果
            result.append(usc_get_result(handle));
        }
        else if(ret == USC_RECOGNIZER_SPEAK_END){
            // 获取中间部分识别结果
            result.append(usc_get_result(handle));
            if(result.size() > 0) {
                // 获取当前语音时间轴信息
                printf("[%06dms--%06dms]\t%s\n",
                    usc_get_result_begin_time(handle),
                    usc_get_result_end_time(handle),
                    result.c_str());
                result.clear();
            }
        }
        else if( ret < 0 ) {
            // 网络出现错误退出
            fprintf(stderr,"usc_feed_buffer error %d\n", ret);
            goto usc_release;
        }
    }

    //  停止语音输入
    ret = usc_stop_recognizer(handle);
    if(ret == 0) {
        //  获取剩余识别结果
        result.append(usc_get_result(handle));
        if(result.size() > 0) {
            // 获取当前语音时间轴信息
            printf("[%06dms--%06dms]\t%s\n",
                usc_get_result_begin_time(handle),
                usc_get_result_end_time(handle),
                result.c_str());
        }
    }
    else {
        fprintf(stderr,"usc_stop_recognizer error %d\n", ret);
    }

usc_release:
```

```
    // 释放识别接口
    usc_release_service(handle);
    if( ret < 0) {
        // 识别出现错误
        fprintf(fresult,"!!!!!!!!-------> Error %d <--------!!!!!!!!!!!\n", ret);
    }
close_files:

    fclose(fpcm);
    return ret;
}
```

6. 语音合成函数接口

6.1. 简述

开发者只需简单调用几个方法就可以使用云知声语音合成服务，接口包括创建合成接口实例 usc_tts_create_service，设置参数 usc_tts_set_option，启动合成 usc_tts_start_synthesis，上传合成文本 usc_tts_text_put，从缓冲获得合成结果 usc_tts_get_result，停止合成 usc_tts_stop_synthesis，释放合成接口实例 usc_tts_release_service。

6.2. 接口说明

6.2.1. 创建合成接口

创建合成接口实例。

```
int usc_tts_create_service(TTSHANDLE* handle)
```

参数

handle [out]

合成实例引用句柄指针。

返回

如果调用成功返回USC_SUCCESS，否则返回错误代码参见附1错误代码表。

示例

```
// 创建合成实例
TTSHANDLE handle = 0;
// 设置appkey
int ret = usc_tts_create_service(&handle);
if(ret != USC_SUCCESS) {
    fprintf(stderr,"usc_create_service error %d\n", ret);
}
```

说明

创建语音合成接口实例，handle返回接口实例引用，完成合成后必须调用函数usc_tts_release_service释放接口实例，否则会产生内存泄露。

6.2.2. 创建合成接口（私有云）

创建合成接口实例。

```
int usc_tts_create_service_ext(TTSHANDLE* handle, const char* host,const char* port)
```

参数

handle [out]
合成实例引用句柄指针。

host
合成服务器地址

port
合成服务器端口号

返回

如果调用成功返回USC_SUCCESS，否则返回错误代码参见附1错误代码表。

示例

```
// 创建合成实例
TTSHANDLE handle = 0;
int ret = usc_tts_create_service_ext(&handle, "192.168.1.2", "80");
if(ret != USC_SUCCESS) {
    fprintf(stderr,"usc_create_service error %d\n", ret);
}
```

说明

创建语音合成接口实例，handle返回接口实例引用，完成合成后必须调用函数usc_tts_release_service释放接口实例，否则会产生内存泄露。

6.2.3. 设置合成参数

设置合成所需的参数。

```
int usc_tts_set_option(TTSHANDLE handle, const char* key ,const char* value)
```

参数

handle

识别接口实例句柄。

key

设置参数的key。

value

参数值

事例

设置appkey(必设)	usc_tts_set_option(handle, "appkey", USC_ASR_SDK_APP_KEY);
设置secret(必设)	usc_tts_set_option(handle, "secret", USC_ASR_SDK_SECRET_KEY);
设置合成pcm语音(默认)	usc_tts_set_option(handle, "audioFormat", "audio/x-wav"); usc_tts_set_option(handle, "audioCodec", "opus");
设置合成mp3语音	usc_tts_set_option(handle, "audioFormat", "audio/mpeg"); usc_tts_set_option(handle, "audioCodec", "raw");
设置中文发音人(默认)	usc_tts_set_option(handle, "vcn", "xiaoli");
设置英文发音人	usc_tts_set_option(handle, "vcn", "Joe");
设置合成文本text(默认)	usc_tts_set_option(handle, "ttp", "text");
设置合成文本ssml格式	usc_tts_set_option(handle, "ttp", "ssml");
设置语速(默认50)	usc_tts_set_option(handle, "spd", "50");
设置音调(默认50)	usc_tts_set_option(handle, "pit", "50");
设置音量(默认50)	usc_tts_set_option(handle, "vol", "50");
设置开始静音时间(单位ms，默认100)	usc_tts_set_option(handle, "smt", "100");
设置结束静音时间(单位ms，默认100)	usc_tts_set_option(handle, "emt", "100");

返回

调用成功返回USC_SUCCESS，否则返回错误代码参见附1错误代码表。

6.2.4. 启动合成

连接合成服务器开始合成。

```
int usc_tts_start_synthesis(TTSHANDLE handle)
```

参数

handle

合成接口实例句柄。

返回

调用成功返回USC_SUCCESS，否则返回错误代码参见附1错误代码表。

说明

连接合成服务器启动合成，一次调用本函数 start 成功之后，可以反复调用 usc_tts_text_put、usc_tts_get_result、usc_tts_stop_synthesis 来进行每次的语音合成，一直到合成结束，调用 release 为止。

示例

```
// 连接服务器启动合成
int ret = usc_tts_start_ synthesis(handle);
if(ret != USC_SUCCESS ) {
    fprintf(stderr,"usc_start_recognizer error %d\n", ret);
}
```

6.2.5. 上传合成文本

上传文本数据进行合成。

```
int usc_tts_text_put(TTSHANDLE handle,const unsigned char* buffer,int length)
```

参数

handle

合成接口实例句柄。

buffer

文本数据起始地址。

length

有效文本数据的长度。

返回

调用成功返回USC_SUCCESS，否则返回错误代码参见附1错误代码表。

说明

示例

```
    //示例在确定好接口后再添加
    //传入文本
    int ret = usc_tts_text_put(handle, src_text, text_len);
    if ( ret != USC_SUCCESS )
    {
        fprintf(stderr,"usc_tts_text_put: UscTts put text failed Error
code %d.\n",ret);
        usc_tts_stop_synthesis(handle, "TextPutError");
        return ret;
    }
    while (1)
    {
        //获取合成结果
        const void *data = usc_tts_get_result(handle, &audio_len, &synth_status,
&ret);
        if (NULL != data)
        {
            output_fd.write((const char *)data, audio_len);
        }
        if (synth_status == AUDIO_DATA_RECV_DONE || ret != 0)
        {
            break;
        }
    }//合成状态synth_status取值可参考开发文档
    output_fd.close();
```

6.2.6. 获得合成结果

获取合成结果。

```
const void* usc_tts_get_result(TTSHANDLE handle,unsigned int*
audioLen,int* synthStatus,int* errorCode)
```

参数

handle

合成接口实例句柄。

`audioLen [out]`

语音长度

`synthStatus [out]`

合成状态参见附 2 语音状态表。

`errorCode [out]`

错误码

返回

返回合成结果。

示例

参见 usc_tts_text_put

6.2.7. 停止合成

停止合成。

```
int  usc_tts_stop_synthesis(TTSHANDLE handle)
```

参数

`handle`

合成接口实例句柄。

返回

成功返回 USC_SUCCESS，失败返回小于零错误代码参见附 1 错误代码表。

说明

向服务器发送请求结束合成，获取结果关闭服务器网络连接。

示例

```
// 停止语音输入
int ret = usc_tts_stop_synthesis(handle);
if(ret != USC_SUCCESS) {
    // 网络出现错误退出
    fprintf(stderr,"usc_stop_synthesis error %d\n", ret);
}
```

6.2.8. 释放合成接口

释放合成接口实例。

```
void  usc_tts_release_service(TTSHANDLE handle)
```

参数

`handle`

合成接口实例句柄。

说明

释放合成接口实例回收占用资源。调用些方法后 `handle` 接口实例句柄不能再用于合成，否则会产生异常。

示例

```
// 释放合成接口实例
usc_tts_release_service(handle);
handle = 0;
```

附 1：错误代码表

错误代码	代码解释
-91002	调用顺序错误（请参照 demo 示例）
-91003	请求超时
-91004	http 协议错误
-91005	音频解码错误
-91007	未联网 或 网络未连接成功
-91008/-91009	请求过程中网络错误
-91131/-91132	合成 文本文件为空 / 文本文件过长
-91134	获取信息错误
-91138	handle 设置错误
-91154	加密校验错误
-91158	未设置 secret
其他-911XX	参数设置错误
-91725	Appkey 验证错误
-91735	服务器超时
其他-917XX	http 请求错误
-918XX	语义请求错误

附 2：语音合成状态表

宏定义	值	说明
RECEIVING_AUDIO_DATA	1	还有语音结果需要接收
AUDIO_DATA_RECV_DONE	2	语音结果全部接收完毕
SYNTH_PROCESS_ERROR	3	错误