

# **iFly Mobile Speech Platform**

## **Mobile Speech Client 5.0**

### **开发手册**

安徽科大讯飞信息科技股份有限公司  
ANHUI USTC IFLYTEK CO., LTD.

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。未得到安徽科大讯飞信息科技股份有限公司明确的书面许可，不得以任何目的、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。

本文档可能涉及安徽科大讯飞信息科技股份有限公司的专利（或正在申请的专利）、商标、版权或其他知识产权，除非得到安徽科大讯飞信息科技股份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。

本手册提及的其它产品和公司名称均可能是各自所有者的商标。

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，如果您使用本产品，表明您已阅读并接受了 EULA 的条款。

版权所有© 安徽科大讯飞信息科技股份有限公司  
Copyright © Anhui USTC iFLYTEK CO., LTD.

# 目 录

目 录.....	1
前言.....	3
第 1 章 概述.....	5
1.1 MSP 系统网络拓扑结构说明.....	5
1.2 名词和缩略语.....	6
1.3 开发包组件.....	6
1.4 文档说明.....	7
第 2 章 通用接口.....	8
2.1 通用接口简介.....	8
2.1.1 通用接口函数列表.....	8
2.1.2 返回值说明.....	8
2.2 函数调用.....	8
2.2.1 MSPLOGIN.....	8
2.2.2 MSPUPLOADDATA.....	9
2.2.3 MSPDOWNLOADDATA.....	11
2.2.4 MSPSEARCH.....	11
2.2.5 MSPLOGOUT.....	13
第 3 章 QTTS 开发接口说明.....	14
3.1 QTTS 接口简介.....	14
3.1.1 QTTS 接口函数列表.....	14
3.1.2 返回值说明.....	14
3.2 函数调用.....	14
3.2.1 QTTSSESSIONBEGIN.....	14
3.2.2 QTTSTEXTPUT.....	15
3.2.3 QTTSAUDIOGET.....	16
3.2.4 QTTSAUDIOINFO.....	18
3.2.5 QTTSSESSIONEND.....	19
3.2.6 QTTSGETPARAM.....	19
第 4 章 QISR 开发接口说明.....	22
4.1 QISR 接口简介.....	22
4.1.1 QISR 接口函数列表.....	22

4.1.2 返回值说明.....	22
4.2 函数调用.....	22
4.2.1 QISRSESSIONBEGIN.....	22
4.2.2 QISRGRAMMARACTIVATE.....	24
4.2.3 QISRAUDIOWRITE.....	25
4.2.4 QISRGETRESULT.....	28
4.2.5 QISRSESSIONEND.....	29
4.2.6 QISRGETPARAM.....	30
<b>第5章 错误码的定义.....</b>	<b>32</b>
5.1 宏.....	32
5.2 错误码列表.....	32
<b>第6章 开发例程.....</b>	<b>36</b>
6.1 语音合成开发例程.....	36
6.2 语音识别开发例程.....	38
6.3 语音听写开发例程.....	43
<b>第7章 常见问题解答.....</b>	<b>48</b>
<b>第8章 技术支持.....</b>	<b>50</b>

## 前言

欢迎使用讯飞移动语音平台 (iFly Mobile Speech Platform, MSP)!

讯飞移动语音平台是基于讯飞公司已有的 ISP 和 IMS 产品, 开发出的一款符合移动互联网用户使用的语音应用开发平台, 提供语音合成、语音听写、语音识别、声纹识别等服务, 为语音应用开发爱好者提供方便易用的开发接口, 使得用户能够基于该开发接口进行多种语音应用开发。其主要功能有:

- 1) 实现基于 HTTP 协议的语音应用服务器, 集成讯飞公司最新的语音引擎, 支持语音合成、语音听写、语音识别、声纹识别等服务;
- 2) 提供基于移动平台和 PC 上的语音客户端子系统, 内部集成音频处理和音频编解码模块, 提供关于语音合成、语音听写、语音识别和声纹识别完善的 API。

科大讯飞作为中国最大的智能语音技术提供商, 在智能语音技术领域有着长期的研究积累, 并在语音合成、语音识别、口语评测等多项技术上拥有国际领先的成果。科大讯飞是我国唯一以语音技术为产业化方向的“国家 863 计划成果产业化基地”、“国家规划布局内重点软件企业”、“国家火炬计划重点高新技术企业”、“国家高技术产业化示范工程”, 并被信息产业部确定为中文语音交互技术标准工作组组长单位, 牵头制定中文语音技术标准。2003 年, 科大讯飞获迄今中国语音产业唯一的“国家科技进步二等奖”, 2005 年获中国信息产业自主创新最高荣誉“信息产业重大技术发明奖”。2006 年至 2012 年连续七届在英文语音合成国际大赛 (Blizzard Challenge) 中蝉联大赛第一名。

基于拥有自主知识产权的世界领先智能语音技术, 科大讯飞已推出从大型电信级应用到小型嵌入式应用, 从电信、金融等行业到企业和家庭用户, 从 PC 到手机到 MP3/MP4/PMP 和玩具, 能够满足不同应用环境的多种产品。科大讯飞占有中文语音技术 70% 以上市场份额, 在电信、金融、电力、社保等主流行业的份额更达 80% 以上, 开发伙伴超过 800 家, 以讯飞为核心的中文语音产业链已初具规模。

本手册主要讲述如何利用讯飞移动语音平台客户端子系统 (Mobile Speech Client 5.0, MSC5.0) 提供的 API 进行语音合成、语音听写/识别和声纹识别等方面的应用开发, 其适用的读者有:

### □ 语音服务的提供商

如果您希望根据现有资源扩大某种特定服务的规模, 但又担心为某一种服务去开发应用平台代价太大, 通过阅读本手册, 您会发现, 只需要较少的编程就可以充分利用讯飞移动语音平台的诸多优势, 实现服务拓展, 缩短项目周期。

### □ 语音应用的开发商

如果您从事语音系统的二次开发, 面向行业级、家庭级等最终用户提供语音合成产品, 讯飞移动语音平台简洁明了而又功能全面的 API 将帮助您迅速完成语音应用系统的开发,

从而大大节约开发投入，丰富语音服务的内容。

## □ 语音技术的爱好者

如果您对语音应用系统具有浓厚的兴趣，并且具有一定的编程基础，您可以在本手册指导下学习语音方面的应用开发，体验语音世界的无穷乐趣。

本手册基本内容：

### 第 1 章 概述

简要概括讯飞移动语音平台的原理、特色、功能，以及一些名词、缩略语介绍等。

### 第 2 章 通用接口

介绍 **MSC5.0** 通用性较强的接口功能与应用，并详细列出开发步骤与例程。

### 第 3 章 QTTS 接口

介绍 **QTTS** 接口函数功能与应用，并详细列出开发步骤与例程。

### 第 4 章 QISR 接口

介绍 **QISR** 接口函数功能与应用，并详细列出开发步骤与例程。

### 第 5 章 QISV 接口

介绍 **QISV** 接口函数功能与应用，并详细列出开发步骤与例程。

### 第 6 章 错误码定义

介绍讯飞移动语音平台开发过程中返回的错误码及其定义。

### 第 7 章 开发例程

给出了语音合成、语音识别/听写和声纹识别的编程示例。

### 第 8 章 常见问题解答

### 第 9 章 技术支持

介绍本公司提供的技术支持方式，便于用户迅速获取帮助，提高应用效率。

通过阅读本手册，读者可以：

- 1) 了解科大讯飞语音应用技术的基本功能与特色
- 2) 掌握语音合成、语音识别系统接口规范
- 3) 了解语音合成系统、语音识别应用开发的基本思想和方法
- 4) 利用语音应用系统接口规范地进行二次开发

## 第 1 章 概述

### 1.1 MSP 系统网络拓扑结构说明

下图为 MSP 系统网络拓扑结构：

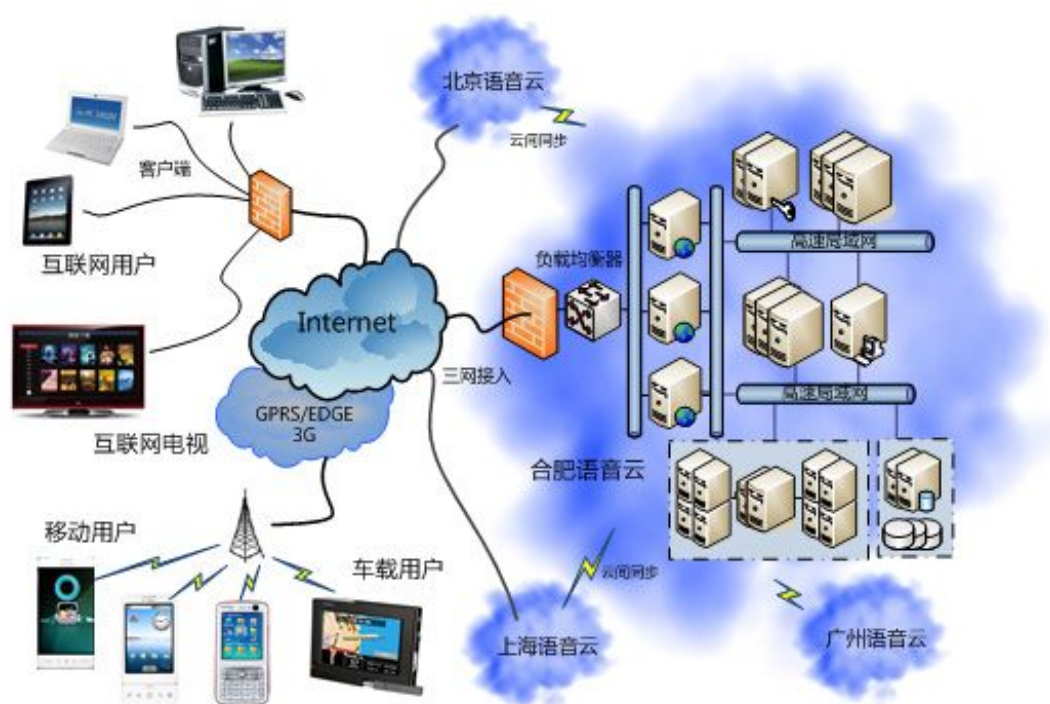


图 1 MSP 系统网络结构

从图中可以看到，完整的 MSP 平台架构在 Internet 上，分为分布式服务器端和客户端两大部分。

服务器端为 MSP 平台的核心部分，提供 HTTP、用户管理、语音服务以及云存储等服务，位于局域网内，对外统一接入 Internet，为客户端提供访问入口。其中：HTTP 服务器负责将客户端发送的服务请求发送至业务服务器，然后由业务服务器按照具体的服务类型，选择语音引擎获取语音服务，而后把结果返回给 HTTP 服务器，再回复客户端。云存储负责存储服务器端的运行日志以及用户的一些个性化数据如个性化音库等。

语音云客户端工作在用户设备上，主要功能是收集用户数据、数据简单加工处理，构建解析通信消息和网络交互等。

## 1.2 名词和缩略语

### □ TTS ( Text to Speech )

语音合成 (Text To Speech, TTS) 技术能够自动将任意文字实时转换为连续的自然语音, 是一种能够在任何时间、任何地点, 向任何人提供语音信息服务的高效便捷手段, 非常符合信息时代海量数据、动态更新和个性化查询的需求。

### □ IAT( iFly Auto Transform ) & ASR ( Automatic Speech Recognition )

语音听写和语音识别技术是一种使计算机能够识别人通过麦克风或者电话输入的词语或语句的技术, 简单的说就是能够让计算机听懂人说话。它的最终目标是使得计算机不受词汇量限制, 在各种噪声环境、语音信道下, 能够实时、准确地识别不同方言、口音等特点的说话人的语句。

让机器听懂人类的语音, 这是人们长期以来梦寐以求的事情。语音识别是一门交叉学科, 关系到多学科的研究领域, 不同领域上的研究成果都对语音识别的发展作了贡献。语音识别技术就是让机器通过识别和理解过程把语音信号转变为相应的文本或命令的技术。

### □ ISV

iFLY Speaker Verification, 讯飞说话者识别 (或称声纹识别) 系统。

### □ IVP

iFLY Voiceprint Password, 讯飞声纹密码。

### □ NLP

Nature Language Proccession, 自然语言理解。

### □ MSP

iFLY Mobile Speech Platform, 或称为 IMSP, 讯飞移动语音平台的缩写, 是讯飞面向移动互联网领域开发的语音服务平台。

### □ MSSP

Mobile Speech Service Protocol, 移动语音服务协议, 基于 HTTP1.1 协议扩展的语音应用协议。

## 1.3 开发包组件

### □ Windows 平台

开发组件	组件组成	说明
头文件	qtts.h qjsr.h qisv.h msp_cmn.h	接口函数声明, 如果不使用对应接口的功能, 可以不包含对应的头文件。



	msh_errors.h msh_types.h	
动态引入库	msh.lib	包含接口函数引入
运行时刻库	msh.dll	接口函数具体实现
客户端配置文件	msh.cfg	运行库相关配置项

#### □ Linux 平台

开发组件	组件组成	说明
头文件	qtts.h qisr.h qisv.h msh_cmn.h msh_errors.h msh_types.h	接口函数声明
运行时刻库	libmsh.so	接口函数具体实现
客户端配置文件	msh.cfg	运行库相关配置项

## 1.4 文档说明

本文针对的读者是具有 Windows、Linux 或者嵌入式平台开发经验的 C/C++ 程序员。

文档中使用的符号约定：

符号	含义
[in]	表明该参数是调用时赋值的参数——输入参数
[out]	该参数在函数返回时被赋值——输出参数
[in/out]	该参数在函数调用时作为输入、函数返回时作为输出参数

## 第 2 章 通用接口

### 2.1 通用接口简介

#### 2.1.1 通用接口函数列表

MSC5.0 提供如下通用接口：

函数名称	功能简介
MSPLogin	登录接口
MSPUploadData	数据上传接口
MSPDownloadData	数据下载接口
MSPLogout	登录退出接口

#### 2.1.2 返回值说明

对于开发接口，如果调用成功，返回值为 `int` 型的接口都会返回 `MSP_SUCCESS`，否则返回错误代码，错误代码参见 `msh_errors.h`；返回值为指针类型的接口，函数执行失败返回空指针 `0`，错误代码可以通过相关的回传参数查看，函数执行成功但无数据时返回空指针 `0`，有数据时返回非 `0` 指针。

### 2.2 函数调用

#### 2.2.1 MSPLogin

##### □ 函数原型

```
int MSPAPI MSPLogin( const char* usr, const char* pwd, const char* params )
```

##### □ 功能

用户登录，每一项语音业务必须先调用 `MSPLogin`，如果登录失败则无法获取任何语音服务。

##### □ 参数

◆ `usr [in]`

用户名，用户可以从网站 <http://dev.voicecloud.cn/index.php?vt=1> 申请。

◆ `pwd [in]`

密码。

◆ params [in]

登录时传入的参数，可以设置的参数及其取值范围请参考《可设置参数列表\_MSC50.xls》。

□ 返回值

如果函数调用成功返回 **MSP\_SUCCESS**，否则返回错误代码，错误代码参见 **msp\_errors.h**，几个主要的返回值如下：

返回值	意义
MSP_ERROR_INVALID_PARA	无效的参数
MSP_ERROR_DB_INVALID_USER	无效的用户名
MSP_ERROR_DB_INVALID_PWD	无效的密码

□ 说明

本接口为通用登录接口。**MSP** 系统所有功能都需要用户在登录的状态下使用。本接口应当在应用程序中仅调用一次，多次调用本函数（中间没有调 **MSPLogout**）时只有第一次调用此函数会进行实际的登录操作，后面的调用会返回成功，但不会完成实际的登录工作。同样 **MSPLogout** 也应当在应用程序退出时仅调用一次。

□ 用法示例

```
const char*   usr = NULL;
const char*   pwd =NULL;
const char*   params="appid = ***** , server_url =http://dev.voicecloud.cn:1028/index.htm";
int   ret = MSPLogin( usr, pwd, params );
if( MSP_SUCCESS != ret )
{
    printf( "MSPLogin failed, error code is: %d", ret );/* No speech serviec will be got if MSPLogin
failed */
}
```

□ 参见

**MSPLogout**。

## 2.2.2 MSPUploadData

□ 函数原型

```
const char* MSPAPI MSPUploadData(const char* dataName, void* data, unsigned
int dataLen, const char* params, int* errorCode)
```

□ 功能

用户数据上传。

□ 参数

## ◆ dataName [in]

数据名称。

## ◆ data [in]

数据缓冲区起始地址。

## ◆ dataLen [in]

数据长度。

## ◆ params [in]

数据描述参数。

## ◆ errorCode [out]

如果函数调用成功返回 0，否则返回错误代码，错误代码参见 `misp_errors.h`，几个主要的返回值如下：

返回值	意义
MISP_ERROR_INVALID_PARA	无效的参数
MISP_ERROR_BUSY	忙碌

## □ 说明

无

## □ 用法示例

```
const char*   dataname = userword;
FILE*        data = fopen("resource\\userwords.txt ", "rb");
const char*   params= "sub=uup,dtf=userword";
int dataLen = 0;
fseek(data, 0, SEEK_END);
dataLen = ftell(data);
fseek(data, 0, SEEK_SET);
MSPUploadData( dataname, data, dataLen, params, &errorcode);
if( MISP_SUCCESS != errorcode )
{
    printf( "MSPUploadData failed, error code is: %d", ret );
}
```

## □ 参见

无

### 2.2.3 MSPDownloadData

#### □ 函数原型

```
const void* MSPAPI MSPDownloadData(const char* params, unsigned int* dataLen,  
int* errorCode)
```

#### □ 功能

用户数据下载。

#### □ 参数

##### ◆ params [in]

可以设置的参数及其取值范围请参考《可设置参数列表\_MSC50.xls》。

##### ◆ dataLen [out]

下载数据长度。

##### ◆ errorCode [out]

如果函数调用成功返回 0，否则返回错误代码，错误代码参见 msp\_errors.h，几个主要的返回值如下：

返回值	意义
MSP_ERROR_INVALID_PARA	无效的参数
MSP_ERROR_BUSY	忙碌

#### □ 说明

无

#### □ 用法示例

```
const char*   params= "dt=userid";  
int dataLen = 0;  
MSPDownloadData (params, &dataLen, &errorcode);  
if( MSP_SUCCESS != errorcode )  
{  
    printf( "MSPDownloadData failed, error code is: %d", ret );  
}
```

#### □ 参见

无

### 2.2.4 MSPSearch

#### □ 函数原型

```
const void* MSPAPI MSPSearch(const char* params, const char* text, unsigned int*
contentLen, int* errorCode)
```

## □ 功能

语义。

## □ 参数

### ◆ params [in]

可以设置的参数及其取值范围请参考《可设置参数列表\_MSC50.xls》。

### ◆ text [in]

上传文本

### ◆ contentLen [in]

上传文本长度

### ◆ errorCode [out]

如果函数调用成功返回 0，否则返回错误代码，错误代码参见 msp\_errors.h，几个主要的返回值如下：

返回值	意义
MSP_ERROR_INVALID_PARA	无效的参数
MSP_ERROR_BUSY	忙碌

## □ 说明

无

## □ 用法示例

```
const char* text = "附近的美食";
int textLen = 0;
MSPSearch("wap_proxy=wifi,extid=null,aue=speex-wb,ssm=1,auf=audio/L16;rate=16000,ent=sm
s16k,sch=1,vaver=1.1,vascn=app,along=117.1484463,alat=31.8322467,vapos=中国安徽省合肥
市望江西路,date=2012-02-09,time=10:12:22", text, textLen, &errorcode);
if( MSP_SUCCESS != errorcode )
{
    printf( "MSPSearch failed, error code is: %d", ret );
}
```

## □ 参见

无

## 2. 2. 5 MSPLogout

### □ 函数原型

int MSPAPI MSPLogout()

### □ 功能

退出登录。

### □ 参数

无。

### □ 返回值

如果函数调用成功返回 `MSP_SUCCESS`，否则返回错误代码，错误代码参见 `mvp_errors.h`。

### □ 说明

本接口对应于 `MSPLogin` 接口，在使用通用接口相关功能时最后一个被调用，用来对 `MSC` 的通用接口部分进行逆初始化。没有调用 `MSPLogin` 直接调用本接口，则不会有任何效果，调用了 `MSPLogin` 不调用本接口而直接结束线程，则 `MSC` 会产生资源泄漏。

### □ 用法示例

```
int ret = MSPLogout( );
if( MSP_SUCCESS != ret )
{
    printf( "MSPLogout failed, error code is: %d", ret );
}
```

### □ 参见

`MSPLogin`。

## 第 3 章 QTTTS 开发接口说明

### 3.1 QTTTS 接口简介

#### 3.1.1 QTTTS 接口函数列表

MSC5.0 关于 TTS 提供如下函数调用：

函数名称	功能简介
QTTSSessionBegin	开始一个TTS会话
QTTSTextPut	输入要合成的文本
QTTSAudioGet	获取合成语音
QTTSAudioInfo	获取音频相关信息
QTTSSessionEnd	结束一路会话
QTTSGetParam	获取合成业务相关参数

#### 3.1.2 返回值说明

对于开发接口，如果调用成功，返回值为 int 型的接口都会返回 **MSP\_SUCCESS**，否则返回错误代码，错误代码参见 **misp\_errors.h**；返回值为指针类型的接口，函数执行失败返回空指针 0，错误代码可以通过相关的回传参数查看，函数执行成功但无数据时返回空指针 0，有数据时返回非 0 指针。

### 3.2 函数调用

#### 3.2.1 QTTSSessionBegin

##### □ 函数原型

```
const char* MSPAPI QTTSSessionBegin( const char* params, int* errorCode )
```

##### □ 功能

MSPLogin 成功后开始一路 TTS 会话，并在参数中指明本路会话用到的参数。

##### □ 参数

◆ params [in]

本路 TTS 会话使用的参数，可以设置的参数及其取值范围请参考《可设置参数列表\_MSC50.xls》。



## ◆ errorCode [out]

如果函数调用成功返回 0，否则返回错误代码，错误代码参见 `msp_errors.h`，几个主要的返回值如下：

返回值	意义
MSP_ERROR_NOT_INIT	未初始化
MSP_ERROR_INVALID_PARA	无效的参数
MSP_ERROR_CREATE_HANDLE	创建会话实例失败

## □ 返回值

MSC 为本路会话建立的 ID，用来唯一的标识本路会话，供以后调用其他函数时使用。函数调用失败则会返回 NULL。

## □ 说明

此处设定的参数在本路会话中一直有效。此函数需要和接口 `QTTSSessionEnd` 配对使用，在这两个接口之间可以调用实际完成合成功能的函数如写入合成文本和取音频数据等。没有调用此函数则后续的函数调用会因为没有合法的会话 ID 而无法完成对应的功能。

## □ 用法示例

```
/* 可设置的参数及含义请参考《可设置参数列表_MSC50》*/
const char*   params= "vcn=xiaoyan, aue=speex-wb;7,auf=audio/L16;rate=16000, ent=intp65";
int           ret = 0;
const char*   session_id = QTTSSessionBegin( params, &ret );
if( 0 != ret )
{
    printf( "QTTSSessionBegin failed, error code is: %d", ret );
}
```

## □ 参见

`QTTSSessionEnd`。

### 3.2.2 QTTSTextPut

## □ 函数原型

```
int MSPAPI QTTSTextPut( const char* sessionID, const char* textString, unsigned int
textLen, const char* params )
```

## □ 功能

写入要合成的文本。

## □ 参数

## ◆ sessionID [in]

由 QTTSSessionBegin 返回过来的会话 ID。

◆ textString [in]

将要进行合成的文本。

◆ textLen [in]

合成文本长度，单位字节。

◆ params [in]

本次合成所用的参数，只对本次合成的文本有效。

### 返回值

如果函数调用成功返回 0，否则返回错误代码，错误代码参见 msp\_errors.h。几个主要的值如下：

返回值	意义
MSP_ERROR_NOT_INIT	未初始化
MSP_ERROR_INVALID_HANDLE	无效的会话 ID
MSP_ERROR_INVALID_PARA	无效的参数
MSP_ERROR_INVALID_PARA_VALUE	无效的参数值
MSP_ERROR_NO_ENOUGH_BUFFER	发送的文本长度超过最大允许长度

### □ 说明

本接口不支持连续被调用。调用本接口写入合成文本后，用户需要反复调用 QTTSAudioGet 接口来获取音频。

### □ 用法示例

```
const char*   text_str = "安徽科大讯飞信息科技股份有限公司";
unsigned int   text_len = strlen( textString );           //textLen 参数为合成文本所占字节数
int ret = QTTSTextPut( session_id, text_str, text_len, NULL );
if( 0 != ret )
{
    printf( "QTTSTextPut failed, error code is: %d", ret );
}
```

### □ 参见

QTTSAudioGet。

## 3.2.3 QTTSAudioGet

### □ 函数原型

```
const void* MSPAPI QTTSAudioGet( const char* sessionID, unsigned int* audioLen,
int* synthStatus, int* errorCode )
```

## □ 功能

获取合成音频。

## □ 参数

### ◆ sessionID [in]

由 QTTSSessionBegin 返回过来的会话 ID。

### ◆ audioLen [out]

合成音频长度，单位字节。

### ◆ synthStatus [out]

合成音频状态，可能的值如下：

枚举常量	简介
MSP_TTS_FLAG_STILL_HAVE_DATA = 1	音频还没取完，还有后继的音频
MSP_TTS_FLAG_DATA_END = 2	音频已经取完

### ◆ errorCode [out]

如果函数调用成功返回 0，否则返回错误代码，错误代码参见 msp\_errors.h，几个主要的值如下：

返回值	意义
MSP_ERROR_NOT_INIT	未初始化
MSP_ERROR_INVALID_HANDLE	无效的会话 ID
MSP_ERROR_INVALID_PARA	无效的参数
MSP_ERROR_NO_MORE_DATA	没有更多的数据（音频已经取完）

## □ 返回值

如果函数调用失败返回 0 指针，调用成功且有音频数据时返回非 0 指针，无音频数据时返回 0 指针。

## □ 说明

用户需要反复调用此接口，在 errorCode 返回值为 0 的情况下直到 synthStatus 返回 MSP\_TTS\_FLAG\_DATA\_END（音频已经取完）为止。

在使用中，如果函数调用成功但没有获取到音频数据，用户需要将当前线程 sleep 一段时间后再次调用，以防止频繁调用浪费 CPU 资源。

## □ 用法示例

```
f_pcm = fopen("tts.pcm", "wb");
while (1) {
    const void *data = QTTSAudioGet(sessionID, &audioLen, &status, &errorCode);
    if (NULL != data)
```

```
fwrite(data, audioLen, 1, f_pcm);
if (status == 2 || errorCode != 0)
    break;
}
fclose(f_pcm);
```

□ 参见

无。

### 3.2.4 QTTSAudioInfo

□ 函数原型

```
const char* MSPAPI QTTSAudioInfo( const char* sessionID, int* errorCode )
```

□ 功能

获取关于合成音频的某些信息如合成音频对应的当前文本位置等。

□ 参数

◆ sessionID [in]

由 QTTSSessionBegin 返回过来的会话 ID。

◆ errorCode [out]

如果函数调用成功返回 0，否则返回错误代码，错误代码参见 `mvp_errors.h`，几个主要的返回值如下：

返回值	意义
MSP_ERROR_NOT_INIT	未初始化
MSP_ERROR_INVALID_HANDLE	无效的会话 ID

□ 返回值

如果函数调用成功返回字符串指针，否则返回 `NULL`。

□ 说明

本接口用在调用 `QTTSAudioGet` 后，获取关于此段音频的描述信息，目前支持的信息有：此次合成音频对应文本结束位置，用户可以利用此信息来做实时高亮度显示合成的文本等应用，格式为“ced=xxx”。

□ 用法示例

```
int          ret = 0;
const char*  audio_info = QTTSAudioInfo( session_id, &ret );
if( 0 != ret )
{
    printf( "QTTSAudioInfo failed, error code is: %d", ret );
}
```

□ 参见

QTTSSAudioGet。

### 3.2.5 QTTSSessionEnd

□ 函数原型

```
int MSPAPI QTTSSessionEnd( const char* sessionID, const char* hints )
```

□ 功能

结束一路 TTS 会话。

□ 参数

◆ sessionID [in]

由 QTTSSessionBegin 返回过来的会话 ID。

◆ hints [in]

结束本次会话的原因描述，用于记录日志，便于用户查阅或者跟踪某些问题。

□ 返回值

如果函数调用成功返回 0，否则返回错误代码，错误代码参见 `mss_errors.h`。几个主要的返回值如下：

返回值	意义
MSP_ERROR_NOT_INIT	未初始化
MSP_ERROR_INVALID_HANDLE	无效的会话 ID

□ 说明

本接口对应于 QTTSSessionBegin 接口，用来结束一路 TTS 会话。调用本函数后，关于当前会话的所有资源（参数，合成文本，会话实例等）都会被释放，所以用户不应该再针对该实例做任何操作（比如使用其会话 ID 等）。

□ 用法示例

```
int ret = QTTSSessionEnd ( session_id, "normal end" );
if( 0 != ret )
{
    printf( "QTTSSessionEnd failed, error code is: %d", ret );
}
```

□ 参见

QTTSSessionBegin

### 3.2.6 QTTSGetParam

□ 函数原型

```
int MSPAPI QTTSGetParam( const char* sessionID, const char* paramName, char*
paramValue, unsigned int* valueLen )
```

## □ 功能

查询 MSC 记录下来的一些信息如上行流量和下行流量等。

## □ 参数

### ◆ sessionID [in]

由 QISRSessionBegin 返回过来的会话 ID，可以为 NULL。

### ◆ paramName [in]

要获取的参数名称。支持同时查询多个参数，查询多个参数时，参数名称以“,”或“\n”分开，MSC5.0 支持的参数如下：

参数名称	意义
upflow	上行数据量。
downflow	下行数据量。

### ◆ paraValue [out]

获取到的参数值，以字符串形式返回；查询多个参数时，参数值之间以“;”分开，不支持的参数将返回空的值。

### ◆ valueLen [in/out]

输入是存放参数值字符串 paraValue 缓冲区的长度，输出为参数值字符串的长度。

## □ 返回值

如果函数调用成功返回 0，否则返回错误代码，错误代码参见 msp\_errors.h。几个主要的返回值如下：

返回值	意义
MSP_ERROR_NOT_INIT	未初始化
MSP_ERROR_INVALID_HANDLE	无效的会话 ID
MSP_ERROR_INVALID_PARA	无效的参数

## □ 说明

调用本接口时，以获取上行流量为例，如果用户将参数 sessionID 设为 NULL，则获取到的是用户进程启动后到本路会话开始之前产生的总上行流量；如果用户在一路会话的中间调用本接口并将参数 sessionID 设为本路会话的 sessionID，则获取到的是本路会话到此时为止产生的上行流量。下行流量与此相似。

## □ 用法示例

```
const char* para_name = "upflow;downflow";
```

```
char*      para_value[ 32 ] = "";  
unsigned int  value_len = 32;  
int  ret = QTTSGetParam ( session_id, para_name, para_value, &value_len );  
if( 0 != ret )  
{  
    printf( "QTTSGetParam failed, error code is: %d", ret );  
}
```

□ 参见

无。

## 第 4 章 QISR 开发接口说明

### 4.1 QISR 接口简介

#### 4.1.1 QISR 接口函数列表

MSC5.0 关于 ISR 提供如下函数调用：

函数名称	功能简介
QISRSessionBegin	开始一个ISR会话
QISRGrammarActivate	传入语法
QISRAudioWrite	写入用来识别的语音
QISRGetResult	获取识别结果
QISRSessionEnd	结束一路会话
QISRGetParam	获取识别/语音听写业务相关参数

本组接口不是线程安全的，用户需要自行保证某些数据的线程安全性，如 `sessionID`。

#### 4.1.2 返回值说明

对于开发接口，如果调用成功，返回值为 `int` 型的接口都会返回 `MSP_SUCCESS`，否则返回错误代码，错误代码参见 `misp_errors.h`；返回值为指针类型的接口，函数执行失败返回空指针 `0`，错误代码可以通过相关的回传参数查看，函数执行成功但无数据时返回空指针 `0`，有数据时返回非 `0` 指针。

### 4.2 函数调用

#### 4.2.1 QISRSessionBegin

##### ☐ 函数原型

```
const char* MSPAPI QISRSessionBegin( const char* grammarList, const char*  
params, int* errorCode )
```

##### ☐ 功能

本接口用来开始一路 ISR 会话，并在参数中指定本路 ISR 会话用到的语法列表，本次会话所用的参数等。

##### ☐ 参数



## ◆ grammarList [in]

uri-list 格式的语法，可以是一个语法文件的 URL 或者一个引擎内置语法列表。可以同时指定多个语法，不同的语法之间以 “,” 隔开。进行语音听写时不需要语法，此参数设定为 NULL 或空串即可；进行语音识别时则需要语法，语法可以在此参数中指定，也可以随后调用 QISRGrammarActivate 指定识别所用的语法。

## ◆ params [in]

本路 ISR 会话使用的参数，可设置的参数及其取值范围请参考《可设置参数列表\_MSC50.xls》。

## ◆ errorCode [out]

如果函数调用成功则其值为 0，否则返回错误代码，错误代码参见 msp\_errors.h。几个主要的返回值如下：

返回值	意义
MSP_ERROR_NOT_INIT	未初始化
MSP_ERROR_INVALID_PARA	无效的参数
MSP_ERROR_CREATE_HANDLE	创建会话实例失败

## □ 返回值

MSC 为本路会话建立的 ID，用来唯一的标识本路会话，供以后调用其他函数时使用。函数调用失败则会返回 NULL。

## □ 说明

此处设定的参数在本路会话中一直有效。此函数需要和接口 QISRSessionEnd 配对使用，在这两个接口之间可以调用实际完成识别功能的函数如激活语法、写入音频和获取结果等。没有调用此函数则 MSC 不会建立和当前线程有关的会话实例，后续的函数调用会因为合法的会话 ID 而无法完成对应的功能。

## □ 用法示例

```
/* vad_timeout 和 vad_speech_tail 两个参数只有在打开 VAD 功能时才生效 */
const char*   params = "sub=iat,aue=speex-wb;7,auf=audio/L16;rate=16000,ent=sms16k, "
                      "rst=plain,vad_timeout=1000,vad_speech_tail=1000";

int   ret = 0;
const char*   session_id = QISRSessionBegin( NULL, params, &ret );
if( 0 != ret )
{
    printf( "QISRSessionBegin failed, error code is: %d", ret );
}
```

## □ 参见

QISRSessionEnd。

#### 4.2.2 QISRGrammarActivate

##### □ 函数原型

```
int MSPAPI QISRGrammarActivate( const char* sessionID, const char* grammar,
                                const char* type, int weight )
```

##### □ 功能

本函数用来激活一个指定的语法，语法类型可以是任何一种合法的语法。

##### □ 参数

###### ◆ sessionID [in]

由 QISRSessionBegin 返回过来的会话 ID。

###### ◆ grammar [in]

语法字符串。

###### ◆ type [in]

语法类型，可以是 uri-list、abnf 等。

###### ◆ weight [in]

本次传入语法的权重，本参数在 MSC5.0 中会被忽略。

##### □ 返回值

如果函数调用成功返回 0，否则返回错误代码，错误代码参见 msp\_errors.h。几个主要的返回值如下：

返回值	意义
MSP_ERROR_NOT_INIT	未初始化
MSP_ERROR_INVALID_HANDLE	无效的会话 ID
MSP_ERROR_INVALID_PARA	无效的参数
MSP_ERROR_NO_DATA	没有（语法）数据

##### □ 说明

本函数只用在语音识别会话中，在语音听写会话中不需要。在一路会话中，用户可以调用此接口激活一个或多个语法，多个语法之间以“,” 隔开，此接口定义的语法和 QISRSessionBegin 指定的语法彼此之间相互独立，地位平等，互不干扰，用户可以通过查看识别结果中的 grammar 字段来了解识别结果对应的语法。激活的语法从激活成功时刻到会话结束时一直有效。

具体的识别语法编写规范可以参阅：

[Speech Recognition Grammar Specification Version 1.0](#)

<http://www.w3.org/TR/speech-grammar/>。

□ 用法示例

```
/* 激活引擎内置的歌曲搜索的语法 */
const char* grammar= get_grammar( "gm_continuous_digit.abnf" );
int  ret = QISRGrammarActivate( session_id, grammar, "abnf", 1 );
if( 0 != ret )
{
    printf( "QISRGrammarActivate failed, error code is: %d", ret );
}
```

□ 参见

无。

### 4.2.3 QISRAudioWrite

□ 函数原型

```
int MSPAPI QISRAudioWrite( const char* sessionID, const void* waveData, unsigned
int waveLen, int audioStatus, int *epStatus, int *rsltStatus )
```

□ 功能

写入本次识别的音频，本接口需要反复调用直到音频写完为止。

□ 参数

◆ sessionID [in]

由 QISRSessionBegin 返回过来的会话 ID。

◆ waveData [in]

音频数据缓冲区起始地址。

◆ waveLen [in]

音频数据长度，单位字节，其大小不能超过设定的 max\_audio\_size。

◆ audioStatus [in]

用来指明用户本次识别的音频是否发送完毕，可能的值如下：

枚举常量	简介
MSP_AUDIO_SAMPLE_FIRST = 1	第一块音频
MSP_AUDIO_SAMPLE_CONTINUE = 2	还有后继音频
MSP_AUDIO_SAMPLE_LAST = 4	最后一块音频

在 MSC5.0 中，ISR\_AUDIO\_SAMPLE\_LAST（0x04）用来指明当前的音频已经发送

完毕，除此之外的任何值都将被 MSC 视为还有后继的音频。

#### ◆ epStatus [out]

端点检测（End-point detected）器所处的状态，可能的值如下：

枚举常量	简介
MSP_EP_LOOKING_FOR_SPEECH = 0	还没有检测到音频的前端点。
MSP_EP_IN_SPEECH = 1	已经检测到了音频前端点，正在进行正常的音频处理。
MSP_EP_AFTER_SPEECH = 3	检测到音频的后端点，后继的音频会被 MSC 忽略。
MSP_EP_TIMEOUT = 4	超时。
MSP_EP_ERROR = 5	出现错误。
MSP_EP_MAX_SPEECH = 6	音频过大。

当 epStatus 大于等于 3 时，用户应当停止写入音频的操作，否则写入 MSC 的音频会被忽略。

#### ◆ rsltStatus [out]

识别器所处的状态，可能的值如下：

枚举常量	简介
MSP_REC_STATUS_SUCCESS = 0	识别成功，此时用户可以调用 QISRGetResult 来获取（部分）结果。
MSP_REC_STATUS_NO_MATCH = 1	识别结束，没有识别结果
MSP_REC_STATUS_INCOMPLETE = 2	正在识别中
MSP_REC_STATUS_SPEECH_COMPLETE = 5	识别结束

#### □ 返回值

如果函数调用成功返回 0，否则返回错误代码，错误代码参见 msp\_errors.h。几个主要的返回值如下：

返回值	意义
MSP_ERROR_NOT_INIT	未初始化
MSP_ERROR_INVALID_HANDLE	无效的会话 ID
MSP_ERROR_NULL_HANDLE	空句柄
MSP_ERROR_INVALID_PARA	无效的参数
MSP_ERROR_INVALID_PARA_VALUE	无效的参数值
MSP_ERROR_NO_ENOUGH_BUFFER	缓冲区溢出

#### □ 说明

MSC 处理音频的策略是边接收、边压缩（如果音频编码格式不为 raw）、边发送。由于音频压缩速度和网络速度的限制，如果音频写入太快太急（如 20 倍于音频码率），可

能会造成原始音频或压缩音频在 **MSC** 中积累过多，从而造成缓冲区无法再容纳更多的数据而产生 **MSP\_ERROR\_NO\_ENOUGH\_BUFFER**（10117，缓冲区溢出）的错误。

调用本接口时，推荐用户在写入音频时采取“边录边发”的方式，即每隔一小段时间将采集到的音频通过本接口写入 **MSC**。这种“边录边发”的方式可以加快结果返回的速度：发送靠后的音频时，前面的音频或许已经被服务器处理过并将部分结果返回了。

调用接口时请设置好 **audioStatus** 的值，并检查返回型参数 **epStatus** 的值，以便及时了解音频的前后端点等信息。如果当前写入的不是最后一块音频，需要将 **audioStatus** 的值设为 2（**ISR\_AUDIO\_SAMPLE\_CONTINUE**）。如果在音频写入过程中检测到 **epStatus** 的值为 3（**ISR\_EP\_AFTER\_SPEECH**），说明系统已经检测到音频的后端点，则应该立即结束写入音频；如果用户要在检测到音频后端点之前结束音频的写入，需要将最后一块音频数据的 **audioStatus** 设为 4（**ISR\_AUDIO\_SAMPLE\_LAST**）。

如果识别状态 **recogStatus** 值为 0（**ISR\_REC\_STATUS\_SUCCESS**）表示已经有部分或全部识别结果缓存在 **MSC** 中了，用户可以调用 **QISRGetResult** 获取这部分结果，再继续调用 **QISRAudioWrite** 以写入后续的音频数据，这种两个接口混调的方式，可以很快的获得（部分）识别结果，特别是使用较大音频进行语音听写时。

#### □ 用法示例

```
char          audio_data[ 5120 ] = "";  
unsigned int   audio_len = 0;  
int           audio_status = 2;  
int           ep_status = 0;  
int           recog_status = 0;  
int           ret = 0;  
while(MSP_AUDIO_SAMPLE_LAST != audio_status )  
{  
    ... // 读取音频到缓冲区 audio_data 中，设置音频长度 audio_len，音频状态 audio_status。  
    ret = QISRAudioWrite( session_id, audio_data, audio_len, audio_status  
                        , &ep_status, &recog_status );  
  
    if( 0 != ret )  
    {  
        printf( "QISRAudioWrite failed, error code is: %d", ret );  
        break;  
    }  
    else if(MSP_EP_AFTER_SPEECH == ep_status ) /* 检测到音频后端点，停止写入音频 */  
    {  
        printf( "end point of speech has been detected!" );  
        break;  
    }  
}  
  
/* 如果是实时采集音频，可以省略此操作。5KB 大小的 16KPCM 持续的时间是 160 毫秒 */
```

```
Sleep( 160 );  
}
```

□ 参见

QISRGetResult。

#### 4.2.4 QISRGetResult

□ 函数原型

```
const char * MSPAPI QISRGetResult( const char* sessionID, int* rsltStatus, int  
waitTime, int *errorCode )
```

□ 功能

获取识别结果。

□ 参数

◆ sessionID [in]

由 QISRSessionBegin 返回过来的会话 ID。

◆ rsltStatus [out]

识别结果的状态，其取值范围和含义请参考 QISRAudioWrite 的参数 recogStatus。

◆ waitTime [in]

与服务器交互的间隔时间，MSC5.0 中，此参数做保留用。

◆ errorCode [out]

如果函数调用成功返回 0，否则返回错误代码，错误代码参见 msp\_errors.h。几个主要的返回值如下：

返回值	意义
MSP_ERROR_NOT_INIT	未初始化
MSP_ERROR_INVALID_HANDLE	无效的会话 ID
MSP_ERROR_INVALID_PARA	无效的参数
MSP_ERROR_NO_DATA	没有数据（如没有写入识别所用的音频等）

□ 返回值

函数执行失败返回 NULL。函数执行成功并且获取到识别结果时返回识别结果，函数执行成功没有获取到识别结果时返回 NULL。

□ 说明

本接口一般是在音频写入完毕后调用，以获取语音识别结果。实际应用中，为了及时获取（部分）结果，本接口也可以在写入音频过程中调用，写入音频过程中的调用方法请

参考 QISRAudioWrite 接口的说明部分。在音频写入完毕后，用户需要反复调用此接口，直到识别结果获取完毕（**rlstStatus** 值为 5）或返回错误码。使用此接口时请注意，如果某次成功调用后没有获得识别结果，请将当前线程 **sleep** 一段时间后再次调用，以防止频繁调用浪费 CPU 资源。

#### □ 用法示例

```
char          rslt_str[ 2048 ] = "";  
const char*   rec_result = NULL;  
int           rslt_status = 0;  
int           ret = 0;  
while(MSP_REC_STATUS_SPEECH_COMPLETE != rslt_status )  
{  
    rec_result = QISRGetResult ( session_id, &rslt_status, 5000, &ret );  
    if( 0 != ret )  
    {  
        printf( "QISRGetResult failed, error code is: %d", ret );  
        break;  
    }  
  
    if( NULL != rec_result )  
    {  
        /* 用户可以用其他的方式保存识别结果 */  
        strcat( rslt_str, rec_result );  
        continue;  
    }  
  
    /* sleep 一下很有必要，防止 MSC 端无缓存的识别结果时浪费 CPU 资源 */  
    Sleep( 200 );  
}
```

#### □ 参见

QISRAudioWrite。

### 4.2.5 QISRSessionEnd

#### □ 函数原型

```
int MSPAPI QISRSessionEnd( const char* sessionID, const char* hints )
```

#### □ 功能

结束一路 ISR 会话。

#### □ 参数

◆ sessionID [in]

由 QISRSessionBegin 返回过来的会话 ID。

◆ hints [in]

结束本次会话的原因描述，用于记录日志，便于用户查阅或者跟踪某些问题。

□ 返回值

如果函数调用成功返回 0，否则返回错误代码，错误代码参见 msp\_errors.h 和 msp\_error.h。几个主要的返回值如下：

返回值	意义
MSP_ERROR_NOT_INIT	未初始化
MSP_ERROR_INVALID_HANDLE	无效的会话 ID

□ 说明

本接口需要和 QISRSessionBegin 配合使用，用来结束一路 ISR 会话。

调用本函数后，关于当前会话的所有资源（参数，语法，音频，会话实例等）都会被释放，所以用户不应该再针对该实例做任何操作（比如使用其 SessionID 等）。

□ 用法示例

```
int ret = QISRSessionEnd ( session_id, "normal end" );
if( 0 != ret )
{
    printf( "QISRSessionEnd failed, error code is: %d", ret );
}
```

□ 参见

QISRSessionBegin

#### 4.2.6 QISRGetParam

□ 函数原型

```
int MSPAPI QISRGetParam( const char* sessionID, const char* paramName, char*
paramValue, unsigned int* valueLen )
```

□ 功能

查询 MSC 记录下来的一些信息如上行流量和下行流量，输入语音的音量等。

□ 参数

◆ sessionID [in]

由 QISRSessionBegin 返回过来的会话 ID。

◆ paramName [in]

要获取的参数名称；支持同时查询多个参数，查询多个参数时，参数名称按“,”或“\n”



分隔开来，MSC5.0 支持的参数如下：

参数名称	意义
upflow	上行数据量。
downflow	下行数据量。
volume	最近一次写入的音频的音量。

◆ paraValue [out]

获取的参数值，以字符串形式返回；查询多个参数时，参数值之间以“;”分开，不支持的参数将返回空的值。

◆ valueLen [out]

参数值的长度。

□ 返回值

如果函数调用成功返回 0，否则返回错误代码，错误代码参见 `misp_errors.h`。几个主要的返回值如下：

返回值	意义
MSP_ERROR_NOT_INIT	未初始化
MSP_ERROR_INVALID_HANDLE	无效的会话 ID
MSP_ERROR_INVALID_PARA	无效的参数

□ 说明

流量信息的获取方法和含义请参考 3.2.6 节 QTTSGgetParam 接口的说明部分，由于音量的获取只能在一路会话中进行，所以使用本接口获取音量信息时参数 `sessionID` 不能为 NULL。

□ 用法示例

参见 3.2.6 节 QTTSGgetParam。

□ 参见

QTTSGgetParam。

## 第 5 章 错误码的定义

MSP5.0 客户端子系统返回的错误码都在 msp\_errors.h 中定义，大致可以分为一般错误、网络错误、资源错误、登录错误和 HTTP 错误等。

### 5.1 宏

和错误码有关的宏定义：

```
#define MSP_HTTP_ERROR (x) ((x) + MSP_ERROR_HTTP_BASE)
```

将 HTTP 错误码  $\times\times\times$  映射为  $12\times\times\times$ ，即关于 HTTP 的错误码后三位同 HTTP 协议中定义的错误码是一致的。

### 5.2 错误码列表

错误码	错误值	意义
MSP_SUCCESS	0	函数执行成功
MSP_ERROR_FAIL	-1	失败
MSP_ERROR_EXCEPTION	-2	异常
MSP_ERROR_GENERAL	10100	基码
MSP_ERROR_OUT_OF_MEMORY	10101	内存越界
MSP_ERROR_FILE_NOT_FOUND	10102	文件没有发现
MSP_ERROR_NOT_SUPPORT	10103	不支持
MSP_ERROR_NOT_IMPLEMENT	10104	没有实现
MSP_ERROR_ACCESS	10105	没有权限
MSP_ERROR_INVALID_PARA	10106	无效的参数
MSP_ERROR_INVALID_PARA_VALUE	10107	无效的参数值
MSP_ERROR_INVALID_HANDLE	10108	无效的句柄
MSP_ERROR_INVALID_DATA	10109	无效的数据
MSP_ERROR_NO_LICENSE	10110	没有授权许可
MSP_ERROR_NOT_INIT	10111	没有初始化
MSP_ERROR_NULL_HANDLE	10112	空句柄
MSP_ERROR_OVERFLOW	10113	溢出
MSP_ERROR_TIME_OUT	10114	超时
MSP_ERROR_OPEN_FILE	10115	打开文件出错
MSP_ERROR_NOT_FOUND	10116	没有发现
MSP_ERROR_NO_ENOUGH_BUFFER	10117	没有足够的内存
MSP_ERROR_NO_DATA	10118	没有数据
MSP_ERROR_NO_MORE_DATA	10119	没有更多的数据
MSP_ERROR_SKIPPED	10120	跳过
MSP_ERROR_ALREADY_EXIST	10121	已经存在

MSP_ERROR_LOAD_MODULE	10122	加载模块失败
MSP_ERROR_BUSY	10123	忙碌
MSP_ERROR_INVALID_CONFIG	10124	无效的配置项
MSP_ERROR_VERSION_CHECK	10125	版本错误
MSP_ERROR_CANCELED	10126	取消
MSP_ERROR_INVALID_MEDIA_TYPE	10127	无效的媒体类型
MSP_ERROR_CONFIG_INITIALIZE	10128	初始化 <b>Config</b> 实例
MSP_ERROR_CREATE_HANDLE	10129	建立句柄
MSP_ERROR_CODING_LIB_NOT_LOAD	10130	编解码库未加载
MSP_ERROR_USER_CANCELLED	10131	用户取消
MSP_ERROR_INVALID_OPERATION	10132	无效的操作
MSP_ERROR_NET_GENERAL	10200	网络一般错误
MSP_ERROR_NET_OPENSOCK	10201	打开套接字
MSP_ERROR_NET_CONNECTSOCK	10202	套接字连接
MSP_ERROR_NET_ACCEPTSOCK	10203	套接字接收
MSP_ERROR_NET_SENDSOCK	10204	发送错误
MSP_ERROR_NET_RECVSOCK	10205	接收错误
MSP_ERROR_NET_INVALIDSOCK	10206	无效的套接字
MSP_ERROR_NET_BADADDRESS	10207	无效的地址
MSP_ERROR_NET_BINDSEQUENCE	10208	绑定次序
MSP_ERROR_NET_NOTOPENSOCK	10209	套接字没有打开
MSP_ERROR_NET_NOTBIND	10210	没有绑定
MSP_ERROR_NET_NOTLISTEN	10211	没有监听
MSP_ERROR_NET_CONNECTCLOSE	10212	连接关闭
MSP_ERROR_NET_NOTDGRAMSOCK	10213	非数据报套接字
MSP_ERROR_NET_DNS	10214	<b>DNS</b> 解析错误
MSP_ERROR_NET_INIT	10215	网络初始化错误
MSP_ERROR_MSG_GENERAL	10300	消息一般错误
MSP_ERROR_MSG_PARSE_ERROR	10301	解析出错
MSP_ERROR_MSG_BUILD_ERROR	10302	构建出错
MSP_ERROR_MSG_PARAM_ERROR	10303	参数出错
MSP_ERROR_MSG_CONTENT_EMPTY	10304	<b>Content</b> 为空
MSP_ERROR_MSG_INVALID_CONTENT_TYPE	10305	<b>Content</b> 类型无效
MSP_ERROR_MSG_INVALID_CONTENT_LENGTH	10306	<b>Content</b> 长度无效
MSP_ERROR_MSG_INVALID_CONTENT_ENCODING	10307	<b>Content</b> 编码无效
MSP_ERROR_MSG_INVALID_KEY	10308	<b>Key</b> 无效
MSP_ERROR_MSG_KEY_EMPTY	10309	<b>Key</b> 为空

MSP_ERROR_MSG_SESSION_ID_EMPTY	10310	会话 ID 为空
MSP_ERROR_MSG_LOGIN_ID_EMPTY	10311	登录 ID 为空
MSP_ERROR_MSG_SYNC_ID_EMPTY	10312	同步 ID 为空
MSP_ERROR_MSG_APP_ID_EMPTY	10313	应用 ID 为空
MSP_ERROR_MSG_EXTERN_ID_EMPTY	10314	扩展 ID 为空
MSP_ERROR_MSG_INVALID_CMD	10315	无效的命令
MSP_ERROR_MSG_INVALID_SUBJECT	10316	无效的主题
MSP_ERROR_MSG_INVALID_VERSION	10317	无效的版本
MSP_ERROR_MSG_NO_CMD	10318	没有命令
MSP_ERROR_MSG_NO_SUBJECT	10319	没有主题
MSP_ERROR_MSG_NO_VERSION	10320	没有版本号
MSP_ERROR_MSG_MSSP_EMPTY	10321	消息为空
MSP_ERROR_MSG_NEW_RESPONSE	10322	新建响应消息失败
MSP_ERROR_MSG_NEW_CONTENT	10323	新建 Content 失败
MSP_ERROR_MSG_INVALID_SESSION_ID	10324	无效的会话 ID
MSP_ERROR_DB_GENERAL	10400	数据库一般错误
MSP_ERROR_DB_EXCEPTION	10401	异常
MSP_ERROR_DB_NO_RESULT	10402	没有结果
MSP_ERROR_DB_INVALID_USER	10403	无效的用户
MSP_ERROR_DB_INVALID_PWD	10404	无效的密码
MSP_ERROR_DB_CONNECT	10405	连接出错
MSP_ERROR_DB_INVALID_SQL	10406	无效的 SQL
MSP_ERROR_DB_INVALID_APPID	10407	无效的应用 ID
MSP_ERROR_RES_GENERAL	10500	资源一般错误
MSP_ERROR_RES_LOAD	10501	没有加载
MSP_ERROR_RES_FREE	10502	空闲
MSP_ERROR_RES_MISSING	10503	缺失
MSP_ERROR_RES_INVALID_NAME	10504	无效的名称
MSP_ERROR_RES_INVALID_ID	10505	无效的 ID
MSP_ERROR_RES_INVALID_IMG	10506	无效的映像
MSP_ERROR_RES_WRITE	10507	写操作错误
MSP_ERROR_RES_LEAK	10508	泄露
MSP_ERROR_RES_HEAD	10509	资源头部错误
MSP_ERROR_RES_DATA	10510	数据出错
MSP_ERROR_RES_SKIP	10511	跳过
MSP_ERROR_TTS_GENERAL	10600	合成一般错误
MSP_ERROR_TTS_TEXTEND	10601	文本结束
MSP_ERROR_TTS_TEXT_EMPTY	10602	文本为空

MSP_ERROR_REC_GENERAL	10700	一般错误
MSP_ERROR_REC_INACTIVE	10701	处于不活跃状态
MSP_ERROR_REC_GRAMMAR_ERROR	10702	语法错误
MSP_ERROR_REC_NO_ACTIVE_GRAMMARS	10703	没有活跃的语法
MSP_ERROR_REC_DUPLICATE_GRAMMAR	10704	语法重复
MSP_ERROR_REC_INVALID_MEDIA_TYPE	10705	无效的媒体类型
MSP_ERROR_REC_INVALID_LANGUAGE	10706	无效的语言
MSP_ERROR_REC_URI_NOT_FOUND	10707	没有对应的 URI
MSP_ERROR_REC_URI_TIMEOUT	10708	获取 URI 内容超时
MSP_ERROR_REC_URI_FETCH_ERROR	10709	获取 URI 内容时出错
MSP_ERROR_EP_GENERAL	10800	(EP) 一般错误
MSP_ERROR_EP_NO_SESSION_NAME	10801	(EP) 链接没有名字
MSP_ERROR_EP_INACTIVE	10802	(EP) 不活跃
MSP_ERROR_EP_INITIALIZED	10803	(EP) 初始化出错
MSP_ERROR_LOGIN_SUCCESS	11000	登录成功
MSP_ERROR_LOGIN_NO_LICENSE	11001	无授权
MSP_ERROR_LOGIN_SESSIONID_INVALID	11002	无效的 SessionID
MSP_ERROR_LOGIN_SESSIONID_ERROR	11003	错误的 SessionID
MSP_ERROR_LOGIN_UNLOGIN	11004	未登录
MSP_ERROR_LOGIN_INVALID_USER	11005	无效的用户
MSP_ERROR_LOGIN_INVALID_PWD	11006	无效的密码
MSP_ERROR_LOGIN_SYSTEM_ERROR	11099	系统错误
MSP_ERROR_HTTP_BASE	12000	HTTP 错误基码

## 第 6 章 开发例程

本章节以 Windows 下的开发为例，给出了语音合成、语音识别和语音听写的编程示例。

### 6.1 语音合成开发例程

```
#include "stdlib.h"
#include "stdio.h"
#include <string.h>
#include "msp_cmh.h"
#include "qsr.h"
#include "qtts.h"
#include "msp_errors.h"
#include "Windows.h"

#define END_SYNTH( reason ) \
{ \
    ret = QTTSsessionEnd( session_id, #reason ); \
    if( 0 != ret ) \
    { \
        printf("QTTSsessionEnd failed, error code is %d", ret ); \
    } \
    \
    ret = MSPLogout(); \
    if( 0 != ret ) \
    { \
        printf("QTTSFinis failed, error code is %d", ret ); \
    } \
}

int main()
{
    const char*      configs = NULL;
    const char*      session_id = NULL;
    const char*      synth_params = NULL;
    const char*      synth_text = NULL;
    unsigned int     text_len = 0;
    const char*      synth_speech = NULL;
    unsigned int     synth_speech_len = 0;
    FILE*            f_speech = NULL;
```

```
int          synth_status = 0;
int          ret = 0;

/* 用户登录 */
const char* lgi_params = "appid = *****, server_url = http://dev.voicecloud.cn/index.htm";
ret = MSPLogin( "usr_name", "*****", lgi_params );
if( 0 != ret )
{
    printf( "MSPLogin failed, error code is %d\n", ret );
    return -1;
}

/* 开始一路会话 */
synth_params = "vcn=xiaoyan, auf=audio/L16;rate=16000, aue=speex-wb;7, ent=intp65";
session_id = QTTSSessionBegin( synth_params, &ret );
if( 0 != ret )
{
    printf( "QTTSSessionBegin failed, error code is %d", ret );
    return -1;
}

/* 写入合成文本 */
synth_text = "讯飞语音云为您提供了最新最好的语音技术体验， "
             "我们在互联网上开放科大讯飞最新研发的各种语音技术， "
             "包含世界领先的语音合成技术、语音识别技术和声纹识别技术等。";
text_len = strlen( synth_text );
ret = QTTSTextPut( session_id, synth_text, text_len, NULL );
if( 0 != ret )
{
    printf( "QTTSTextPut failed, error code is %d", ret );
    END_SYNT( QTTSTextPut failed! );
    return -1;
}

/* 获取合成音频 */
f_speech = fopen( "synth_speech.pcm", "wb" );
if( NULL == f_speech )
{
    printf( "Can not open file \"synth_speech.pcm\" );
    END_SYNT( open file );
    return -1;
}
while( MSP_MSP_TTS_FLAG_DATA_END != synth_status )
{
```

```
synth_speech = QTTSAudioGet( session_id, &synth_speech_len
                             , &synth_status, &ret );

if( 0 != ret )
{
    printf( "QTTSAudioGet failed, error code is: %d", ret );
    break;
}
printf( "QTTSAudioGet ok, speech length = %d\n", synth_speech_len );

if( NULL != synth_speech && 0 != synth_speech_len )
{
    fwrite( synth_speech, 1, synth_speech_len, f_speech );
}
else
{
    /* 无音频, 当前线程 sleep 一下, 防止其过度占用 CPU 资源 */
    Sleep( 160 );
}
}
fclose( f_speech );

/* 结束会话, 释放资源 */
ret = QTTSSessionEnd( session_id, "normal end" );
if( NULL == f_speech )
{
    printf( "QTTSSessionEnd failed, error code is %d", ret );
}
session_id = NULL;
/* 退出登录 */
ret = MSPLogout();
if( MSP_SUCCESS != ret )
{
    printf( "MSPLogout failed, error code is: %d", ret );
}

return 0;
}
```

## 6.2 语音识别开发例程

```
#include "stdlib.h"
#include "stdio.h"
```



```
#include <string.h>
#include "msp_cmh.h"
#include "qisr.h"
#include "qtts.h"
#include "msp_errors.h"
#include "Windows.h"

#define END_RECOG( reason ) \
{ \
    ret = QISRSessionEnd( session_id, #reason ); \
    if( 0 != ret ) \
    { \
        printf("QISRSessionEnd failed, error code is %d", ret ); \
    } \
    \
    ret = MSPLogout(); \
    if( 0 != ret ) \
    { \
        printf("QISRFini failed, error code is %d", ret ); \
    } \
}

#define BLOCK_LEN 5 * 1024

const char* get_grammar( const char* filename )
{
    int ret = MSP_SUCCESS;
    int file_len = 0;
    char* grammar = NULL;
    FILE *fp=NULL;
    fp=fopen(filename,"rb");
    if (NULL == fp)
    {
        printf("get_grammar| open file \"%s\" failed.\n",filename ? filename : "");
        return NULL;
    }
    fseek(fp,0L,SEEK_END);
    file_len=ftell(fp);
    fseek(fp,0L,SEEK_SET);

    grammar = (char*)malloc(file_len+1); //从文件中读取语法，注意文本编码为 GB2312
    fread((void*)grammar,1,file_len,fp);
    grammar[file_len]='\0'; //字符串收尾
    fclose(fp);
}
```

```
        return grammar;
    }

int main()
{
    const char*      configs = NULL;
    const char*      session_id = NULL;
    const char*      recog_grammar = NULL;
    const char*      recog_params = NULL;
    char             recog_audio[ BLOCK_LEN ];
    FILE*            f_speech = NULL;
    FILE*            result_txt = NULL;
    int              audio_status = 0;
    int              ep_status = 0;
    int              rec_status = 0;
    int              rslt_status = 0;
    const char*      rec_result = NULL;
    unsigned int      audio_len = 0;
    int              ret = 0;

    /* 用户登录 */
    const char* lgi_params = "appid=50287829, server_url = http://dev.voicecloud.cn/index.htm";
    ret = MSPLogin( NULL, NULL, lgi_params );
    if( 0 != ret )
    {
        printf( "MSPLogin failed, error code is %d\n", ret );
        return -1;
    }

    /* 开始一路会话，使用引擎内置的语法进行识别 */
    recog_grammar = get_grammar( "gm_continuous_digit.abnf" );
    recog_params = "rst=plain,sub=asr,ssm=1,aue=speex,auf=audio/L16;rate=16000,grt=abnf";
    session_id = QISRSessionBegin( recog_grammar, recog_params, &ret );
    if( 0 != ret )
    {
        printf( "QISRSessionBegin failed, error code is %d\n", ret );
        return -1;
    }

    /* 打开用来进行识别的语音文件，用户可以采用其他的获取音频的方式比如实时采集音频 */
    f_speech = fopen( "一三四五六七八九十.wav", "rb" );
    if( NULL == f_speech )
    {
        printf( "Can not open file \"一三四五六七八九十.wav\"\n" );
    }
}
```

```
END_RECOG( open file );
return -1;
}

/* 发送音频数据，获取语音听写结果 */
while( MSP_AUDIO_SAMPLE_LAST != audio_status )
{
    audio_len = fread( recog_audio, 1, BLOCK_LEN, f_speech );
    audio_status = ( audio_len == BLOCK_LEN ) ?
MSP_AUDIO_SAMPLE_CONTINUE :MSP_AUDIO_SAMPLE_LAST;
    ret = QISRAudioWrite( session_id, recog_audio, audio_len, audio_status, &ep_status,
&rslt_status );
    if( 0 != ret )
    {
        printf( "QISRSessionBegin failed, error code is %d\n", ret );
        rslt_status = MSP_REC_STATUS_COMPLETE;
        break;
    }
    printf( "write audio data ok! len=%d, status=%d\n", audio_len, audio_status );

    /* 已经有结果缓存在 MSC 中了，可以获取了 */
    if( MSP_REC_STATUS_SUCCESS == rslt_status )
    {
        rec_result = QISRGetResult( session_id, &rslt_status, 5000, &ret );
        if( 0 != ret )
        {
            printf( "QISRGetResult failed, error code is %d\n", ret );
            rslt_status = MSP_REC_STATUS_COMPLETE;
            break;
        }

        if( NULL != rec_result )
        {
            printf( "got a result: %s\n", rec_result );
            result_txt = fopen("asr_result.txt", "a+");
            if (result_txt)
            {
                fwrite(rec_result, 1, strlen(rec_result), result_txt);
                fclose(result_txt);
            }
        }

        /* 全部结果已经取完了 */
        if( MSP_REC_STATUS_COMPLETE == rslt_status )
```

```
        {
            printf( "the result has been got completely!\n" );
            break;
        }
    }

    /* 检测到音频后 endpoint, 结束音频发送 */
    if( MSP_EP_AFTER_SPEECH == ep_status )
    {
        printf( "end point of speech has been detected!\n" );
        break;
    }

    Sleep( 160 );
}
fclose( f_speech );

/* 获取余下的识别结果 */
while( MSP_REC_STATUS_COMPLETE != rslt_status )
{
    rec_result = QISRGetResult( session_id, &rslt_status, 5000, &ret );
    if( 0 != ret )
    {
        printf( "QISRGetResult failed, error code is: %d\n", ret );
        break;
    }

    if( NULL != rec_result )
    {
        printf( "got a result: %s\n", rec_result );
    }

    /* sleep 一下很有必要, 防止 MSC 端无缓存的识别结果时浪费 CPU 资源 */
    Sleep( 160 );
}

/* 结束会话, 释放资源 */
ret = QISRSessionEnd( session_id, "normal end" );
if( NULL == f_speech )
{
    printf( "QISRSessionEnd failed, error code is %d\n", ret );
}
session_id = NULL;
/* 退出登录 */
ret = MSPLogout();
```

```
if( MSP_SUCCESS != ret )
{
    printf( "MSPLogout failed, error code is: %d", ret );
}

return 0;
}
}
```

## 6.3 语音听写开发例程

```
#include "stdlib.h"
#include "stdio.h"
#include <string.h>
#include "msp_cmn.h"
#include "qisr.h"
#include "qtts.h"
#include "msp_errors.h"
#include "Windows.h"

#define END_RECOG( reason ) \
{ \
    ret = QISRSessionEnd( session_id, #reason ); \
    if( 0 != ret ) \
    { \
        printf("QISRSessionEnd failed, error code is %d", ret ); \
    } \
    \
    ret = MSPLogout(); \
    if( 0 != ret ) \
    { \
        printf("QISRFinis failed, error code is %d", ret ); \
    } \
}

#define BLOCK_LEN 5 * 1024

int main()
{
    const char*      configs = NULL;
    const char*      session_id = NULL;
```

```
const char*      recog_params = NULL;
char             recog_audio[ BLOCK_LEN ];
FILE*           f_speech = NULL;
FILE*           result_txt = NULL;
int             audio_status = 0;
int             ep_status = 0;
int             rec_status = 0;
int             rslt_status = 0;
const char*     rec_result = NULL;
unsigned int    audio_len = 0;
int            ret = 0;

/* 用户登录 */
const char* lgi_params = "appid = *****, server_url = http://dev.voicecloud.cn/index.htm";
ret = MSPLogin( "usr_name", "*****", lgi_params );
if( 0 != ret )
{
    printf( "MSPLogin failed, error code is %d\n", ret );
    return -1;
}

/* 开始一路会话 */
recog_params = "sub=iat, aue=speex-wb;7, auf=audio/L16;rate=16000, “
               “ent=sms16k, rst=plain, vad_speech_tail=1500”;
session_id = QISRSessionBegin( NULL, recog_params, &ret );
if( 0 != ret )
{
    printf( "QISRSessionBegin failed, error code is %d\n", ret );
    return -1;
}

/* 打开用来进行识别的语音文件，用户可以采用其他的获取音频的方式比如实时采集音频 */
f_speech = fopen( "IAT_16KPCM_10s_0.pcm", "rb" );
if( NULL == f_speech )
{
    printf( "Can not open file \"IAT_16KPCM_10s_0.pcm\"\n" );
    END_RECOG( open file );
    return -1;
}

/* 发送音频数据，获取语音听写结果 */
while( MSP_AUDIO_SAMPLE_LAST != audio_status )
{
    audio_len = fread( recog_audio, 1, BLOCK_LEN, f_speech );
```

```
audio_status = ( audio_len == BLOCK_LEN ) ?
    MSP_AUDIO_SAMPLE_CONTINUE :MSP_AUDIO_SAMPLE_LAST;
ret = QISRAudioWrite( session_id, recog_audio, audio_len
    , audio_status, &ep_status, &rslt_status );
if( 0 != ret )
{
    printf( "QISRSessionBegin failed, error code is %d\n", ret );
    rslt_status = MSP_REC_STATUS_COMPLETE;
    break;
}
printf( "write audio data ok! len=%d, status=%d\n", audio_len, audio_status );

/* 已经有结果缓存在 MSC 中了，可以获取了 */
if( MSP_REC_STATUS_SUCCESS == rslt_status )
{
    rec_result = QISRGetResult( session_id, &rslt_status, 5000, &ret );
    if( 0 != ret )
    {
        printf( "QISRGetResult failed, error code is %d\n", ret );
        rslt_status = MSP_REC_STATUS_COMPLETE;
        break;
    }

    if( NULL != rec_result )
    {
        printf( "got a result: %s\n", rec_result );
        result_txt = fopen("result.txt", "at+");
        if (result_txt)
        {
            fwrite(rec_result, 1, strlen(rec_result), result_txt);
            fclose(result_txt);
        }
    }

    /* 全部结果已经取完了 */
    if( MSP_REC_STATUS_COMPLETE == rslt_status )
    {
        printf( "the result has been got completely!\n" );
        break;
    }
}

/* 检测到音频后 endpoint，结束音频发送 */
if( MSP_EP_AFTER_SPEECH == ep_status )
```

```
{
    printf( "end point of speech has been detected!\n" );
    break;
}

Sleep( 160 );
}
fclose( f_speech );

/* 获取余下的识别结果 */
while( MSP_REC_STATUS_COMPLETE != rslt_status )
{
    rec_result = QISRGetResult( session_id, &rslt_status, 5000, &ret );
    if( 0 != ret )
    {
        printf( "QISRGetResult failed, error code is: %d\n", ret );
        break;
    }

    if( NULL != rec_result )
    {
        printf( "got a result: %s\n", rec_result );
    }

    /* sleep 一下很有必要，防止 MSC 端无缓存的识别结果时浪费 CPU 资源 */
    Sleep( 160 );
}

/* 结束会话，释放资源 */
ret = QISRSessionEnd( session_id, "normal end" );
if( NULL == f_speech )
{
    printf( "QISRSessionEnd failed, error code is %d\n", ret );
}
session_id = NULL;
/* 退出登录 */
ret = MSPLogout();
if( MSP_SUCCESS != ret )
{
    printf( "MSPLogout failed, error code is: %d", ret );
}

return 0;
```



}
---

## 第 7 章 常见问题解答

**问题 1：调用 QISRInit/QTTSInit 返回 10106-无效的参数。**

答：原因可能是：

- 1) Configs 字符串没有按照格式书写，正确的是以逗号隔开的参数对（参数名=参数值）组成的字符串。
- 2) Configs 字符串不是合法的 C 风格字符串，有些平台（如 Android）需要手动在字符串末尾加 '\0' 字符。

**问题 2：拿到了合成音频但不知道如何来播放。**

答：合成拿到的音频是没有音频头的，音频头中含有音频格式、采样率、音频长度等播放音频所需信息。拿到合成音频后，用户可以添加音频头，然后使用常规播放器来播放；也可以使用 Cool Edit 等软件手动选择音频参数来播放。

**问题 3：如何进行大文本的合成**

答：语音云一次语音合成允许的合成文本大小不超过配置项 `max_text_size` 指定的值，最大不超过 4096 个字节，所以对于长度超过此值的大合成文本，用户可以采用“分段合成”的方式，即先将大文本按照标点符号如句号进行切分，然后对每一段文本分别进行合成。进行分段合成时，用户既可以在一路会话中循环使用 `QTTSTextPut+QTTSAudioGet` 组合完成合成，也可以为每一段文本使用一路独立的会话完成合成。

**问题 4：调用 QISRSessionBegin 返回 10130-对应的动态库没有加载。**

答：原因可能是：

- 1) MSC 所在目录中没有包含相应的音频编解码动态库，如使用了 `aue=amr-wb;7`，但是目录中没有 `amr_wb.dll`。
- 2) 目录中有相应的动态库但没有加载。MSC 默认只加载 `speex.dll`，要使用别的编解码算法需要在配置文件或 `configs` 字符串中指明加载的编解码动态库，如想同时使用 `speex` 和 `amr-wb` 编解码算法，需要将配置文件 `msc.cfg` 中的配置项 `coding_libs` 指定为 `speex.dll;amr_wb.dll` 或在 `configs` 字符串中使用参数对 `coding_libs=speex.dll;amr_wb.dll`。

**问题 5：获取不到识别/听写结果。**

答：原因可能是：

- 1) QISRSessionBegin 的参数设置不正确，如没有设置好正确的引擎类型等。
- 2) 音频格式不对，客户端支持的音频编解码算法只支持 16 位 Intel PCM 格式的音频。

**问题 6：能获取到语音听写结果但是不全。**

答：此问题主要是在调用 QISRAudioWrite 时没有正确设置参数 `audioStatus` 所致，此参数在写入非最后一个音频数据块时需要设置为 2，写入最后一个数据块时需要设置为 4，以告诉 MSC 音频写入完毕。如果只有一个音频数据块，`audioStatus` 也需要设置为 4。

**问题 7：可以拿到识别或转写结果但是响应很慢。**

答：此问题可以尝试如下方法来解决：

- 1) 调用 QISRAudioWrite 接口写音频数据时，尽量做到“匀速发送”——周期性的发送定长数据，做到边录边发，避免一次发送数据量过大的音频。
- 2) 采用 QISRAudioWrite 接口和 QISRGetResult 接口混调的方式。在调用 QISRAudioWrite 接口时，可以检查 out 型参数 `recogStatus`，如果其值为 0，表明已经有（部分）识别结果缓存在 MSC 中了，此时可以调用 QISRGetResult 来获取结果。

## 第 8 章 技术支持

如果您在安装、使用或开发过程中遇到任何问题或者建议，请与我们联系！

联系时对问题的描述请尽量包含以下内容：

- 系统配置（包括 CPU、内存、硬盘、操作系统及产品版本等信息）
- 问题细节（包括问题的重现过程及合成的文本内容、识别音频等）
- 问题重现（包括详细的操作过程和运行日志等）

科大讯飞提供以下方式的技术支持：

### □ 电话支持

请于周一～周五，北京时间 9：00～17：00 间，拨打电话： 0551—65331813 获得技术支持信息。

### □ 电子邮件支持

请将问题的详细描述发至：msp\_support@iflytek.com。

### □ 在线支持

请登录我们的论坛 <http://dev.voicecloud.cn/bbs/index.php?vt=1>。

### □ 信件支持

请将问题的详细描述发至：

中国安徽省合肥市望江西路 666 号科大讯飞语音产业基地 邮编 230088

或传真至： 0551—65331801 65331802