ECx00U&EGx00U 系列
QuecOpen 音频调试 API 参考手册

LTE Standard 模块系列

版本：1.0

日期：2021-09-01

状态：受控文件

上海移远通信技术股份有限公司（以下简称“移远通信”）始终以为客户提供最及时、最全面的服务为宗旨。如需任何帮助，请随时联系我司上海总部，联系方式如下：

上海移远通信技术股份有限公司
上海市闵行区田林路 1016 号科技绿洲 3 期（B 区）5 号楼　　邮编：200233
电话：+86 21 5108 6236　　邮箱：info@quectel.com

或联系我司当地办事处，详情请登录：http://www.quectel.com/cn/support/sales.htm。

如需技术支持或反馈我司技术文档中的问题，请随时登陆网址：
http://www.quectel.com/cn/support/technical.htm 或发送邮件至：support@quectel.com。

前言

移远通信提供该文档内容以支持客户的产品设计。客户须按照文档中提供的规范、参数来设计产品。同时，您理解并同意，移远通信提供的参考设计仅作为示例。您同意在设计您目标产品时使用您独立的分析、评估和判断。在使用本文档所指导的任何硬软件或服务之前，请仔细阅读本声明。您在此承认并同意，尽管移远通信采取了商业范围内的合理努力来提供尽可能好的体验，但本文档和其所涉及服务是在“可用”基础上提供给您的。移远通信可在未事先通知的情况下，自行决定随时增加、修改或重述本文档。

使用和披露限制

许可协议

除非移远通信特别授权，否则我司所提供硬软件、材料和文档的接收方须对接收的内容保密，不得将其用于除本项目的实施与开展以外的任何其他目的。

版权声明

移远通信产品和本协议项下的第三方产品可能包含受移远通信或第三方材料、硬软件和文档版权保护的相关资料。除非事先得到书面同意，否则您不得获取、使用、向第三方披露我司所提供的文档和信息，或对此类受版权保护的资料进行复制、转载、抄袭、出版、展示、翻译、分发、合并、修改，或创造其衍生作品。移远通信或第三方对受版权保护的资料拥有专有权，不授予或转让任何专利、版权、商标或服务商标权的许可。为避免歧义，除了正常的非独家、免版税的产品使用许可，任何形式的购买都不可被视为授予许可。对于任何违反保密义务、未经授权使用或以其他非法形式恶意使用所述文档和信息的违法侵权行为，移远通信有权追究法律责任。

商标

除另行规定，本文档中的任何内容均不授予在广告、宣传或其他方面使用移远通信或第三方的任何商标、商号及名称，或其缩略语，或其仿冒品的权利。

第三方权利

您理解本文档可能涉及一个或多个属于第三方的硬软件和文档（“第三方材料”）。您对此类第三方材料的使用应受本文档的所有限制和义务约束。

移远通信针对第三方材料不做任何明示或暗示的保证或陈述，包括但不限于任何暗示或法定的适销性或特定用途的适用性、平静受益权、系统集成、信息准确性以及与许可技术或被许可人使用许可技术相关的不侵犯任何第三方知识产权的保证。本协议中的任何内容都不构成移远通信对任何移远通信产品或任何其他硬软件、设备、工具、信息或产品的开发、增强、修改、分销、营销、销售、提供销售或以其他方式维持生产的陈述或保证。此外，移远通信免除因交易过程、使用或贸易而产生的任何和所有保证。

免责声明

1) 移远通信不承担任何因未能遵守有关操作或设计规范而造成损害的责任。

2) 移远通信不承担因本文档中的任何因不准确、遗漏、或使用本文档中的信息而产生的任何责任。

3) 移远通信尽力确保开发中功能的完整性、准确性、及时性，但不排除上述功能错误或遗漏的可能。除非另有协议规定，否则移远通信对开发中功能的使用不做任何暗示或法定的保证。在适用法律允许的最大范围内，移远通信不对任何因使用开发中功能而遭受的损害承担责任，无论此类损害是否可以预见。

4) 移远通信对第三方网站及第三方资源的信息、内容、广告、商业报价、产品、服务和材料的可访问性、安全性、准确性、可用性、合法性和完整性不承担任何法律责任。

文档历史

修订记录

版本	日期	作者	变更表述
-	2021-03-08	Kevin WANG	文档创建
1.0	2021-09-01	Kevin WANG	受控版本

目录

表格索引

1 引言

移远通信 ECx00U 系列和 EGx00U 模块支持 QuecOpen®方案；QuecOpen®是基于 RTOS 的嵌入式开发平台，可简化 IoT 应用的软件设计和开发过程。有关 QuecOpen®的详细信息，请参考***文档 [1]***。

本文档主要介绍在 QuecOpen®方案下，ECx00U 系列和 EGx00U 模块的音频系统、外置 Codec NV 机制、音频相关 API（包括通用音频类、PCM 类、音频通道类、铃声类、TTS 类和音频增益类）以及音频开发示例。

1.1. 适用模块

表 1：适用模块

模块系列	模块
ECx00U	EC200U 系列
	EC600U 系列
EGx00U	EG500U-CN
	EG700U-CN

2 音频系统概述

展锐平台音频系统的参考设计包括如下功能：

- 控制模拟外设：打开或关闭外设、调节音量、设置静音等功能。
- 通讯语音功能：2G/3G CS 呼叫、4G VoLTE 呼叫、以及通话中的 PCM 数据流功能。
- 多媒体音频功能：MP3/WAV/AMR 等文件的编解码功能。

3 外置 Codec NV 机制

模块外置 Codec 采用 delta NV 机制，用户可通过修改 Codec NV 来控制外置 Codec 的默认参数。但只有在模块开机时，CP 侧才能读取该 NV，并在开始/结束播放音频时控制外置 Codec 寄存器。因此在代码中修改 Codec NV 后，相关配置需重启模块才能生效。

Codec NV 由基本参数、I2C 参数、AIF 参数、外置 Codec 寄存器参数四部分构成。

3.1. 头文件

本章所述 API 的头文件为 *ql_audio.h*，位于 SDK 包的 *sdk\components\ql-kernel\inc* 目录下；若无特别说明，本文档所述头文件均位于该目录下。

备注

本文所述 Codec delta NV API 与音频调试相关 API 的头文件均为 *ql_audio.h*，路径为 *sdk\components\ql-kernel\inc*。

3.2. Codec NV 结构

3.2.1. AUD_BASIC_CFG_T

基本参数的结构体定义如下：

```
typedef struct
{
    uint32_t codecAddr;
    uint16_t codecIsMaster;
    uint16_t dataFormat;
    uint16_t reserved;
    uint16_t iic_data_width;
    uint16_t initFlag;
    uint16_t externalCodec;
}AUD_BASIC_CFG_T;
```

- **参数**

类型	参数	描述
uint32_t	*codecAddr*	外置 Codec 访问地址
uint16_t	*codecIsMaster*	外置 Codec 为主设备或从设备 *1* 外置 Codec 为主设备（默认值） *0* 外置 Codec 为从设备 该参数不可修改
uint16_t	*dataFormat*	数据传输模式 *0* I2S（默认值） *1* PCM（此模式下不支持双声道，及 16 kHZ 以上采样率）
uint16_t	*reserved*	暂不支持
uint16_t	*iic_data_width*	I2C 写入数据时支持的数据长度 *1* 9-bit（如 NAU88C22） *2* 8-bit（默认值）（如 ES8374） *3* 16-bit，高字节优先（如 ALC5616） *4* 16-bit，低字节优先
uint16_t	*initFlag*	是否需要初始化 I2C *0* 需要（默认值） *1* 不需要 使用外置 Codec 时，不可修改该参数。
uint16_t	*externalCodec*	选择使用内置或外置 Codec *0* 使用内置 Codec（默认值） *1* 使用外置 Codec

3.2.2. AUD_I2C_CFG_T

I2C 参数的结构体定义如下：

```
typedef struct
{
    int32_t id;
    uint32_t freq;
    int32_t clockMode;
} AUD_I2C_CFG_T;
```

- **参数**

类型	参数	描述
int32_t	*id*	I2C 通路

		0 I2C 通路 1（通路 1 由摄像头使用，不可选） *1* I2C 通路 2（默认）
uint32_t	*freq*	I2C 系统时钟，保持默认值 200000 kHz
int32_t	*clockMode*	I2C 工作时钟频率 *0* 100 kHZ（默认） *1* 400 kHZ

3.2.3. AUD_AIF_SERIAL_CFG_T

AIF 参数的结构体定义如下：

```
typedef struct
{
    AUD_SERIAL_MODE_E mode;
    bool aifIsMaster;
    bool lsbFirst;
    bool polarity;
    AUD_AIF_RX_DELAY_E rxDelay;
    AUD_AIF_TX_DELAY_E txDelay;
    AUD_AIF_RX_MODE_E rxMode;
    AUD_AIF_TX_MODE_E txMode;
    uint32_t fs;
    uint32_t bckLrckRatio;
    bool invertBck;
    bool outputHalfCycleDelay;
    bool inputHalfCycleDelay;
    bool enableBckOutGating;
}AUD_AIF_SERIAL_CFG_T;
```

- **参数**

类型	参数	描述
AUD_SERIAL_MODE_E	*mode*	数据传输模式 *0* I2S（默认） *1* PCM *AUD_SERIAL_MODE_E* 枚举信息详情请参考 *ql_audio.h*。
bool	*aifIsMaster*	外置 Codec 为主设备或从设备 *0* 从设备（默认） *1* 主设备 该参数不可修改。
bool	*lsbFirst*	数据帧格式 *0* 第一帧数据为 MSB（默认）

		1 第一帧数据为 LSB
bool	*polarity*	LRCK 极性设置 *0* 高电平为左声道，低电平为右声道 *1* 高电平为右声道，低电平为左声道
AUD_AIF_RX_DELAY_E	*rxDelay*	输入串行数据 MSB 与 LRCK 边缘之间的延迟 *0* 不延迟，对齐（默认） *1* 延迟一个周期 *2* 延迟两个周期 *3* 延迟三个周期 *AUD_AIF_RX_DELAY_E* 枚举信息详情请参考 *ql_audio.h*。
AUD_AIF_TX_DELAY_E	*txDelay*	输出串行数据 MSB 与 LRCK 边缘之间的延迟 *0* 不延迟，对齐（默认） *1* 延迟一个周期 *AUD_AIF_TX_DELAY_E* 枚举信息详情请参考 *ql_audio.h*。
AUD_AIF_RX_MODE_E	*rxMode*	输入串行数据声道配置 *0* 立体声从引脚输入，立体声输出到 IFC *1* 立体声从引脚输入，单声道从左声道输出到 IFC（默认，CP 会根据实际情况配置） *AUD_AIF_RX_MODE_E* 枚举信息详情请参考 *ql_audio.h*。
AUD_AIF_TX_MODE_E	*txMode*	输出串行数据声道配置 *0* 立体声从 IFC 输入，立体声输出到引脚 *1* 单声道从 IFC 输入，立体声从左声道输出到引脚（默认，代码中会根据实际情况进行配置） *2* 单声道从 IFC 输入，立体声从左右声道重复输出到引脚 *AUD_AIF_TX_MODE_E* 枚举信息详情请参考 *ql_audio.h*。
uint32_t	*fs*	LRCK 采样率。默认 8 kHz，CP 会根据实际情况配置。
uint32_t	*bckLrckRatio*	BCLK/LRCK 比值。默认 32，CP 会根据实际情况配置。
bool	*invertBck*	是否翻转 BCLK *0* 否（默认） *1* 是
bool	*outputHalfCycleDelay*	是否延迟半个周期输出数据 *0* 否（默认） *1* 是
bool	*intputHalfCycleDelay*	是否延迟半个周期输入数据 *0* 否（默认）

		1 是
bool	*enableBckOutGating*	是否设置 BCLK 门限 *0* 否（默认） *1* 是

3.2.4. AUD_CODEC_REG_T

外置 Codec 寄存器配置分为 5 类：初始化配置寄存器、采样率配置寄存器、输入配置寄存器、输出配置寄存器和关闭配置寄存器。每个寄存器的配置包含 3 个值：寄存器地址、寄存器配置值和时延。外置 Codec 寄存器参数的结构体定义如下：

```
typedef struct
{
    uint32_t regAddr;
    uint16_t val;
    uint16_t delay;
} AUD_CODEC_REG_T;
```

- **参数**

类型	参数	描述
uint32_t	*regAddr*	寄存器地址
uint16_t	*val*	寄存器配置值
uint16_t	*delay*	时延。单位：毫秒

3.2.4.1. 初始化配置寄存器

该配置共有 100 组。播放音频/录音开始时，将按照每组配置的寄存器地址将寄存器配置值写入外置 Codec 并延迟一段时间，用不完的数组默认填 0。代码中对应为：*AUD_CODEC_REG_T initRegCfg[100]*。

3.2.4.2. 采样率配置寄存器

采样率相关的寄存器配置支持 8 k、11.025 k、12 k、16 k、22.05 k、24 k、32 k、44.1 k、48 k 和 64 k 共 10 个数组，每个数组下有 12 组寄存器。*initRegCfg* 配置完成后，将按照每组配置的寄存器地址将寄存器配置值写入外置 Codec 并延迟一段时间，用不完的数组默认填 0。代码中对应为：*AUD_CODEC_REG_T sampleRegCfg[10][12]*。

3.2.4.3. 输入配置寄存器

输入路径相关寄存器配置目前预留了 6 个数组（0：main mic，1：aux mic，2：dual mic，3：hp mic_l，4：hp mic_r，5：暂时保留），最后一个数组目前无作用。代码中会根据当前的输入路径进行配置。数组 *inpathRegCfg0* 为主 MIC 的配置寄存器。每个数组下都有 20 组寄存器配置，包含输入路径配置。*sampleRegCfg* 设置完成后，若此传输方向为输入，将按照每组配置的寄存器地址将寄存器配置值写入外置 Codec 并延迟一段时间，用不完的数组默认填 0。代码中对应为：*AUD_CODEC_REG_T inpathRegCfg[6][20]*。

3.2.4.4. 输出配置寄存器

输出路径相关寄存器配置目前预留了 4 个数组（0：RECEIVER，1：EAR_PIECE，2：LOUD_SPEAKER，3：LOUD_SPEAKER_EAR_PIECE），代码中会根据当前的输出路径进行配置。每个数组下有 20 组寄存器配置，包含输出路径配置。*sampleRegCfg* 设置完成后，若此传输方向为输出，将按照每组配置的寄存器地址将寄存器配置值写入外置 Codec 并延迟一段时间，用不完的数组默认填 0。代码中对应为：*AUD_CODEC_REG_T outpathRegCfg[4][20]*。

3.2.4.5. 关闭配置寄存器

该配置共有 50 组。播放音频/录音结束后，将按照每组配置的寄存器地址将寄存器配置值写入外置 Codec 并延迟一段时间，用不完的数组默认填 0。代码中对应为：*AUD_CODEC_REG_T closeRegCfg[50]*。

3.3. Codec delta NV API 介绍

3.3.1. ql_aud_codec_write_nv

该函数用于将数据写入外置 Codec delta NV，写入 NV 后重启模块生效。

- **函数原型**

```
ql_audio_errcode_e ql_aud_codec_write_nv(QL_HAL_CODEC_CFG_T *nv_cfg)
```

- **参数**

nv_cfg：
[In] Codec delta NV 结构体。请参考***第 3.3.1.1 章***。

- **返回值**

请参考***第 3.3.1.2 章***。

3.3.1.1. QL_HAL_CODEC_CFG_T

Codec delta NV 的结构体定义如下：

```
typedef struct
{
    AUD_BASIC_CFG_T  basicCfg;
    uint16_t reserved[24];
    AUD_I2C_CFG_T i2cCfg;
    AUD_AIF_SERIAL_CFG_T i2sAifcfg;
    AUD_CODEC_REG_T initRegCfg[100];
    AUD_CODEC_REG_T closeRegCfg[50];
    AUD_CODEC_REG_T sampleRegCfg[10][12];
    AUD_CODEC_REG_T inpathRegCfg[6][20];
    AUD_CODEC_REG_T outpathRegCfg[4][20];
} QL_HAL_CODEC_CFG_T;
```

- **参数**

类型	参数	描述
AUD_BASIC_CFG_T	*basicCfg*	基本参数。请参考***第 3.2.1 章***。
uint16_t	*reserved*	暂不支持
AUD_I2C_CFG_T	*i2cCfg*	I2C 参数。请参考***第 3.2.2 章***。
AUD_AIF_SERIAL_CFG_T	i2sAifcfg	AIF 参数。请参考***第 3.2.3 章***。
AUD_CODEC_REG_T	*initRegCfg*	初始化配置寄存器。请参考***第 3.2.4.1 章***。
AUD_CODEC_REG_T	*closeRegCfg*	关闭配置寄存器。请参考***第 3.2.4.5 章***。
AUD_CODEC_REG_T	*sampleRegCfg*	采样率配置寄存器。请参考***第 3.2.4.2 章***。
AUD_CODEC_REG_T	*inpathRegCfg*	输入配置寄存器。请参考***第 3.2.4.3 章***。
AUD_CODEC_REG_T	*outpathRegCfg*	输出配置寄存器。请参考***第 3.2.4.4 章***。

3.3.1.2. ql_audio_errcode_e

音频错误码表示函数是否执行成功，若失败则返回错误原因。枚举信息定义如下：

```
typedef enum
{
```

```
    QL_AUDIO_SUCCESS                = 0,

    QL_AUDIO_UNKNOWN_ERROR          = (QL_COMPONENT_AUDIO << 16) | 901,
    QL_AUDIO_INVALID_PARAM,
    QL_AUDIO_OPER_NOT_SUPPORTED,
    QL_AUDIO_DEVICE_BUSY,
    QL_AUDIO_FILE_NOT_EXIST,            //905
    QL_AUDIO_FREE_SIZE_NOT_ENOUGH,
    QL_AUDIO_NO_MEMORY,
    QL_AUDIO_SET_PATH_FAIL,
    QL_AUDIO_FILE_IN_OPERATION,
    QL_AUDIO_ERROR_AUDIO_PATH,          //910
    QL_AUDIO_FILE_TYPE_NOT_SUPPORT,
    QL_AUDIO_DEVICE_NOT_EXIST,
    QL_AUDIO_PLAY_FAIL,
    QL_AUDIO_PARAM_SET_FAIL,
    QL_AUDIO_OPEN_FILE_FAIL,            //915
    QL_AUDIO_RECORD_SYS_FAIL,
    QL_AUDIO_PLAYER_SYS_FAIL,
    QL_AUDIO_END_FAIL,
    QL_AUDIO_NOT_INIT,
    QL_AUDIO_CODEC_WR_FAIL,
    QL_AUDIO_CODEC_RD_FAIL ,
    QL_AUDIO_CODEC_INIT_FAIL
}ql_audio_errcode_e
```

- **参数**

参数	描述
QL_AUDIO_SUCCESS	函数执行成功
QL_AUDIO_UNKNOWN_ERROR	未知错误
QL_AUDIO_INVALID_PARAM	无效参数
QL_AUDIO_OPER_NOT_SUPPORTED	不支持该操作
QL_AUDIO_DEVICE_BUSY	正在进行录音/播放音频
QL_AUDIO_FILE_NOT_EXIST	文件不存在
QL_AUDIO_FREE_SIZE_NOT_ENOUGH	文件系统空间不足
QL_AUDIO_NO_MEMORY	内存 RAM 空间不足
QL_AUDIO_SET_PATH_FAIL	切换通道失败

QL_AUDIO_FILE_IN_OPERATION	文件已被打开
QL_AUDIO_ERROR_AUDIO_PATH	无效的音频通道
QL_AUDIO_FILE_TYPE_NOT_SUPPORT	文件格式不支持
QL_AUDIO_DEVICE_NOT_EXIST	没有正在播放的音频
QL_AUDIO_PLAY_FAIL	播放失败
QL_AUDIO_PARAM_SET_FAIL	参数设置失败
QL_AUDIO_OPEN_FILE_FAIL	文件打开失败
QL_AUDIO_RECORD_SYS_FAIL	录音系统错误
QL_AUDIO_PLAYER_SYS_FAIL	播放系统错误
QL_AUDIO_END_FAIL	录音/播放音频结束失败
QL_AUDIO_NOT_INIT	音频未初始化
QL_AUDIO_CODEC_WR_FAIL	写入数据失败
QL_AUDIO_CODEC_RD_FAIL	读取数据失败
QL_AUDIO_CODEC_INIT_FAIL	Codec 初始化失败

3.3.2. ql_aud_codec_read_nv

该函数用于从外置 Codec delta NV 读取数据。

- **函数原型**

```
ql_audio_errcode_e ql_aud_codec_read_nv(QL_HAL_CODEC_CFG_T *nv_cfg)
```

- **参数**

nv_cfg：
[Out] Codec delta NV 结构体。请参考***第 3.3.1.1 章***。

- **返回值**

请参考***第 3.3.1.2 章***。

3.3.3. ql_audio_iic_write

该函数用于将数据写入外置 Codec 寄存器。

- **函数原型**

```
ql_audio_errcode_e ql_audio_iic_write(uint8 dev_addr, uint16 reg_addr, uint8 size, uint16 val)
```

- **参数**

dev_addr：
[In] 外置 Codec 访问地址。暂时只支持 Codec ALC5616 与 Codec NAU8810；地址分别为 0X1B 和 0x1A。

reg_addr：
[In] 外置 Codec 寄存器地址。

size：
[In] 写入数据的长度。
1 单字节
2 2 个字节

val：
[In] 需要写入外置 Codec 的数据。

- **返回值**

请参考***第 3.3.1.2 章***。

3.3.4. ql_audio_iic_read

该函数用于从外置 Codec 寄存器读取数据。

- **函数原型**

```
ql_audio_errcode_e ql_audio_iic_read(uint8 dev_addr, uint16 reg_addr, uint8 size, uint16 *p_val)
```

- **参数**

dev_addr：
[In] 外置 Codec 访问地址。暂时只支持 ALC5616 与 NAU8810；地址分别为 0X1B 和 0x1A。

reg_addr：
[In] 外置 Codec 寄存器地址。

size：
[In] 读取数据的长度。
1 单字节
2 2 个字节

p_val：
[Out] 读取数据的存放地址。

- **返回值**

请参考***第 3.3.1.2 章***。

3.3.5. ql_aud_codec_clk_enable

该函数用于使能 Codec 26 MHz 时钟，为外置 Codec 提供时钟。

- **函数原型**

```
void ql_aud_codec_clk_enable(void)
```

- **参数**

无

- **返回值**

无

3.3.6. ql_aud_codec_clk_disable

该函数用于关闭 Codec 26 MHz 时钟。

- **函数原型**

```
void ql_aud_codec_clk_disable(void)
```

- **参数**

无

- **返回值**

无

4 API 介绍

4.1. 通用音频类 API

4.1.1. ql_aud_play_file_start

该函数用于播放音频文件。

- **函数原型**

```
ql_audio_errcode_e ql_aud_play_file_start(char *file_name, AudPlayType_e type, cb_on_player play_cb)
```

- **参数**

file_name：
[In] 音频文件。

type：
[In] 音频播放方式。请参考***第 4.1.1.1 章***。

play_cb：
[In] 音频播放回调函数。在播放开始和完成时被调用，可为 NULL。请参考***第 4.1.1.2 章***。

- **返回值**

请参考***第 3.3.1.2 章***。

4.1.1.1. AudPlayType_e

音频播放方式，枚举信息定义如下：

```
typedef enum
{
    QL_AUDIO_PLAY_TYPE_NONE = 0,    //invalid
    QL_AUDIO_PLAY_TYPE_LOCAL,
    QL_AUDIO_PLAY_TYPE_VOICE,
```

```
    QL_AUDIO_PLAY_TYPE_POC,
    QL_AUDIO_PLAY_TYPE_MAX,   //invalid
} AudPlayType_e;
```

- **参数**

参数	描述
QL_AUDIO_PLAY_TYPE_LOCAL	本地播放
QL_AUDIO_PLAY_TYPE_VOICE	远端播放
QL_AUDIO_PLAY_TYPE_POC	POC 播放

4.1.1.2. cb_on_player

该函数为音频播放回调函数，用于上报音频播放状态。

- **函数原型**

```
typedef int(*cb_on_player)(char *p_data, int len, enum_aud_player_state state)
```

- **参数**

p_data：
[In] 暂时保留。回调函数被调用时，该变量不会被赋值。

len：
[In] 暂时保留。回调函数被调用时，该变量不会被赋值。

state：
[In] 音频播放状态。请参考***第 4.1.1.3 章***。

- **返回值**

0 函数执行成功
其他值 函数执行失败

4.1.1.3. enum_aud_player_state

在播放过程中产生该枚举中的状态时，将会触发回调函数。音频播放状态枚举信息定义如下：

```
typedef enum
{
    AUD_PLAYER_ERROR = -1,
    AUD_PLAYER_START = 0,
    AUD_PLAYER_PAUSE,
    AUD_PLAYER_FINISHED,
    AUD_PLAYER_CLOSE,
    AUD_PLAYER_RESUME,
}enum_aud_player_state;
```

- **参数**

参数	描述
AUD_PLAYER_ERROR	发生错误
AUD_PLAYER_START	播放开始
AUD_PLAYER_PAUSE	播放暂停
AUD_PLAYER_FINISHED	播放完成/结束
AUD_PLAYER_CLOSE	播放通道关闭
AUD_PLAYER_RESUME	播放恢复

4.1.2. ql_aud_player_stop

该函数用于停止播放音频文件并释放音频资源。

- **函数原型**

```
ql_audio_errcode_e ql_aud_player_stop(void)
```

- **参数**

无

- **返回值**

请参考*第 3.3.1.2 章*。

4.1.3. ql_aud_player_pause

该函数用于暂停播放音频文件。

- **函数原型**

```
ql_audio_errcode_e ql_aud_player_pause(void)
```

- **参数**

无

- **返回值**

请参考*第 3.3.1.2 章*。

4.1.4. ql_aud_player_resume

该函数用于恢复播放音频文件。

- **函数原型**

```
ql_audio_errcode_e ql_aud_player_resume(void)
```

- **参数**

无

- **返回值**

请参考*第 3.3.1.2 章*。

4.1.5. ql_aud_wait_play_finish

该函数用于等待音频播放完成。

- **函数原型**

```
ql_audio_errcode_e ql_aud_wait_play_finish(int timeout)
```

- **参数**

timeout：
[In] 等待时间。

- **返回值**

请参考*第 3.3.1.2 章*。

4.1.6. ql_aud_get_play_state

该函数用于获取音频播放状态。

- **函数原型**

```
AudioStatus_e ql_aud_get_play_state(void)
```

- **参数**

无

- **返回值**

请参考*第 4.1.6.1 章*。

4.1.6.1. AudioStatus_e

音频播放/录音状态枚举信息定义如下：

```
typedef enum
{
    QL_AUDIO_STATUS_IDLE,
    QL_AUDIO_STATUS_RUNNING,
    QL_AUDIO_STATUS_PAUSE,
    QL_AUDIO_STATUS_FINISHED,
    } AudioStatus_e;
```

- **参数**

参数	描述
QL_AUDIO_STATUS_IDLE	空闲状态
QL_AUDIO_STATUS_RUNNING	运行状态
QL_AUDIO_STATUS_PAUSE	暂停状态
QL_AUDIO_STATUS_FINISHED	完成状态

4.1.7. ql_aud_get_playing_func

该函数用于获取音频播放的接口类型。

- **函数原型**

```
QL_PLAYING_FUNCTION_E ql_aud_get_playing_func(void)
```

- **参数**

无

- **返回值**

请参考*第 4.1.7.1 章*。

4.1.7.1. QL_PLAYING_FUNCTION_E

音频播放的接口类型枚举信息定义如下：

```
typedef enum
{
    QL_AUDIO_PLAYER_IDLE,
    QL_AUDIO_IS_PLAYING_FILE,
    QL_AUDIO_IS_PLAYING_STREAM,
    QL_AUDIO_IS_PLAYING_PCM,
    QL_AUDIO_IS_PLAYING_DTMF,
}QL_PLAYING_FUNCTION_E;
```

- **参数**

参数	描述
QL_AUDIO_PLAYER_IDLE	不在播放音频文件
QL_AUDIO_IS_PLAYING_FILE	正在播放音频文件（由 *ql_aud_play_file_start()*发起）
QL_AUDIO_IS_PLAYING_STREAM	正在播放音频流（建议使用 PCM 类 API 播放音频流）
QL_AUDIO_IS_PLAYING_PCM	正在使用 PCM 类 API 播放音频（*ql_aud_pcm_open()*等）
QL_AUDIO_IS_PLAYING_DTMF	正在使用 DTMF 类 API 播放音频（暂不支持）

4.1.8. ql_get_audio_state

该函数用于获取音频初始化状态。

- **函数原型**

```
int ql_get_audio_state (void)
```

- **参数**

无

- **返回值**

1　　函数执行成功
0　　函数执行失败

4.1.9. ql_aud_set_volume

该函数用于设置音频播放音量。

- **函数原型**

```
ql_audio_errcode_e ql_aud_set_volume(AudPlayType_e type, AUDIOHAL_SPK_LEVEL_T volume)
```

- **参数**

type：
[In] 音频播放方式。请参考***第 4.1.1.1 章***。

volume：
[In] 音量大小。请参考***第 4.1.9.1 章***。

- **返回值**

请参考***第 3.3.1.2 章***。

4.1.9.1. AUDIOHAL_SPK_LEVEL_T

音频播放音量枚举信息定义如下：

```
typedef enum
{
```

```
    AUDIOHAL_SPK_MUTE          = 0,
    AUDIOHAL_SPK_VOL_1,
    AUDIOHAL_SPK_VOL_2,
    AUDIOHAL_SPK_VOL_3,
    AUDIOHAL_SPK_VOL_4,
    AUDIOHAL_SPK_VOL_5,
    AUDIOHAL_SPK_VOL_6,
    AUDIOHAL_SPK_VOL_7,
    AUDIOHAL_SPK_VOL_8,
    AUDIOHAL_SPK_VOL_9,
    AUDIOHAL_SPK_VOL_10,
    AUDIOHAL_SPK_VOL_11,
    AUDIOHAL_SPK_VOL_QTY,
} AUDIOHAL_SPK_LEVEL_T;
```

- **参数**

参数	描述
AUDIOHAL_SPK_MUTE	静音
AUDIOHAL_SPK_VOL_1	音量 1 级
AUDIOHAL_SPK_VOL_2	音量 2 级
AUDIOHAL_SPK_VOL_3	音量 3 级
AUDIOHAL_SPK_VOL_4	音量 4 级
AUDIOHAL_SPK_VOL_5	音量 5 级
AUDIOHAL_SPK_VOL_6	音量 6 级
AUDIOHAL_SPK_VOL_7	音量 7 级
AUDIOHAL_SPK_VOL_8	音量 8 级
AUDIOHAL_SPK_VOL_9	音量 9 级
AUDIOHAL_SPK_VOL_10	音量 10 级
AUDIOHAL_SPK_VOL_11	音量 11 级（最大音量）
AUDIOHAL_SPK_VOL_QTY	暂不支持

4.1.10. ql_aud_get_volume

该函数用于获取音频播放音量。

- **函数原型**

```
AUDIOHAL_SPK_LEVEL_T ql_aud_get_volume(AudPlayType_e type)
```

- **参数**

type：
[In] 音频播放方式。请参考***第 4.1.1.1 章***。

- **返回值**

请参考***第 4.1.9.1 章***。

4.1.11. ql_aud_record_file_start

该函数用于以文件方式录音，并将音频文件存储到文件系统中。

- **函数原型**

```
ql_audio_errcode_e ql_aud_record_file_start(char *file_name, ql_aud_config *config, AudRecType_e type, cb_on_record record_cb)
```

- **参数**

file_name：
[In] 音频文件。

config：
[In] 录音数据参数。请参考***第 4.1.11.1 章***。

type：
[In] 录音方式。请参考***第 4.1.11.2 章***。

record_cb：
[In] 录音回调函数。在录音开始、完成和异常时被调用，不可为空。请参考***第 4.1.11.3 章***。

- **返回值**

请参考***第 3.3.1.2 章***。

4.1.11.1. ql_aud_config

录音数据参数结构体信息定义如下：

```
typedef struct
{
    int channels;
    int samplerate;
    int len_size;
}ql_aud_config;
```

- **参数**

类型	参数	描述
int	*channels*	通道数 *1* 单声道 *2* 双声道
int	*samplerate*	采样率。支持 8 kHz 和 16 kHz。
int	*len_size*	暂不支持

4.1.11.2. AudRecType_e

录音方式枚举信息定义如下：

```
typedef enum
{
    QL_REC_TYPE_NONE = 0,    //invalid
    QL_REC_TYPE_MIC,
    QL_REC_TYPE_VOICE,
} AudRecType_e;
```

- **参数**

参数	描述
QL_REC_TYPE_MIC	MIC 录音（本地录音）
QL_REC_TYPE_VOICE	远端录音

4.1.11.3. cb_on_record

该函数为录音回调函数。以音频流方式录音时，底层产生录音数据后，该回调函数用于上报录音状态，并将录音数据通过 *p_data* 参数上报给用户。通过文件方式录音时，此回调函数仅用于上报录音状态，不传送数据。

- **函数原型**

```
typedef int(*cb_on_record)(char *p_data, int len, enum_aud_record_state state);
```

- **参数**

p_data：
[In] 录音数据的存储地址。每隔 20 ms 底层会将录音数据打包，通过回调函数上报给用户。

len：
[In] 录音数据的长度。

state：
[In] 录音状态。请参考***第 4.1.11.4 章***。

- **返回值**

0	函数执行成功
其他值	函数执行失败

4.1.11.4. enum_aud_record_state

在录音过程中产生该枚举中的状态时，将会触发回调函数。录音状态枚举信息定义如下：

```
typedef enum
{
    AUD_RECORD_ERROR = -1,
    AUD_RECORD_START = 0,
    AUD_RECORD_DATA,
    AUD_RECORD_PAUSE,
    AUD_RECORD_CLOSE,
    AUD_RECORD_RESUME,
}enum_aud_record_state;
```

- **参数**

参数	描述
AUD_RECORD_ERROR	发生错误
AUD_RECORD_START	录音开始
AUD_RECORD_DATA	上报数据
AUD_RECORD_PAUSE	录音暂停
AUD_RECORD_CLOSE	录音结束
AUD_RECORD_RESUME	录音恢复

4.1.12. ql_aud_record_stream_start

该函数用于以音频流方式录音，并将录音数据通过回调函数发送到本地。

- **函数原型**

```
ql_audio_errcode_e ql_aud_record_stream_start(ql_aud_config *config, AudRecType_e type, cb_on_record record_cb)
```

- **参数**

config：
[In] 指定录音数据参数。请参考***第 4.1.11.1 章***。

type：
[In] 录音方式。请参考***第 4.1.11.2 章***。

record_cb：
[In] 录音回调函数。在录音开始、完成和异常时被调用，不可为空。请参考***第 4.1.11.3 章***。

- **返回值**

请参考***第 3.3.1.2 章***。

4.1.13. ql_aud_record_stop

该函数用于结束录音。

- **函数原型**

```
ql_audio_errcode_e ql_aud_record_stop(void)
```

- **参数**

无

- **返回值**

请参考*第 3.3.1.2 章*。

4.1.14. ql_aud_get_record_state

该函数用于获取录音状态。

- **函数原型**

```
AudioStatus_e ql_aud_get_record_state(void)
```

- **参数**

无

- **返回值**

请参考*第 4.1.6.1 章*。

4.1.15. ql_aud_get_recroding_func

该函数用于获取录音的接口类型。

- **函数原型**

```
QL_RECORDING_FUNCTION_E ql_aud_get_recording_func(void)
```

- **参数**

无

- **返回值**

请参考*第 4.1.15.1 章*。

4.1.15.1. QL_RECORDING_FUNCTION_E

录音的接口类型枚举信息定义如下：

```
typedef enum
{
        QL_AUDIO_RECORDER_IDLE,
      QL_AUDIO_IS_RECORDING_FILE,
      QL_AUDIO_IS_RECORDING_STREAM,
      QL_AUDIO_IS_RECORDING_PCM,
} QL_RECORDING_FUNCTION_E;
```

- **参数**

参数	描述
QL_AUDIO_RECORDER_IDLE	未录音
QL_AUDIO_IS_RECORDING_FILE	正在以文件方式录音（由 *ql_aud_record_file_start()*发起）
QL_AUDIO_IS_RECORDING_STREAM	正在以音频流方式录音（由 *ql_aud_record_stream_start()* 发起）
QL_AUDIO_IS_RECORDING_PCM	正在使用 PCM 类 API 录音

4.2. PCM 类 API

4.2.1. ql_aud_pcm_open

该函数用于打开并初始化 PCM 设备，为录音和播放 PCM 数据流做准备。音频数据格式可选择 WAV、PCM、AMRNB、AMRWB 和 MP3（仅播放时可用）。选择非 PCM 格式时，底层硬件编/解码器会自动根据输入的格式进行编/解码，实现数据流录音和播放。

- **函数原型**

```
PCM_HANDLE_T ql_aud_pcm_open(QL_PCM_CONFIG_T *config, AudStreamFormat_e format, unsigned int flags, pcm_path_e type)
```

- **参数**

config：
[In] PCM 数据流参数。请参考***第 4.2.1.2 章***。

format：
[In] PCM 数据流格式。请参考***第 4.2.1.3 章***。

type：
[In] PCM 数据流录音/播放方式。请参考***第 4.2.1.4 章***。

flags：
[In] 打开的 PCM 标志。请参考***第 4.2.1.1 章***。

- **返回值**

NULL	函数执行失败
其他值	函数执行成功，返回 PCM 句柄

备注

若打开 PCM 数据标志 *QL_PCM_READ_FLAG*，则录音将在调用此函数后开始，且需在短时间内调用 *ql_pcm_read()*读取音频数据。系统缓存最大为 16 KB，当缓存区满后，录音结束，且所有数据会被丢弃。

4.2.1.1. PCM 标志

PCM 标志定义如下：

```
#define QL_PCM_BLOCK_FLAG      (0x01)
#define QL_PCM_NONBLOCK_FLAG   (0x02)
#define QL_PCM_READ_FLAG       (0x04)
#define QL_PCM_WRITE_FLAG      (0x08)
```

- **参数**

参数	描述
QL_PCM_BLOCK_FLAG	阻塞方式录音和播放数据流
QL_PCM_NONBLOCK_FLAG	非阻塞方式录音和播放数据流
QL_PCM_READ_FLAG	录音 PCM 数据
QL_PCM_WRITE_FLAG	播放 PCM 数据

4.2.1.2. QL_PCM_CONFIG_T

PCM 数据流参数，枚举信息定义如下：

```
typedef struct {
    unsigned int channels;
    unsigned int samplerate;
    unsigned int periodcnt;
} QL_PCM_CONFIG_T;
```

- **参数**

类型	参数	描述
unsigned int	*channels*	声道数 播放：1 和 2 录音：仅能为 1
unsigned int	*samplerate*	采样率 录音：8000 Hz、16000 Hz 播放：8000~44100 Hz
unsigned int	*periodcnt*	暂无作用

4.2.1.3. AudStreamFormat_e

PCM 数据流格式枚举信息定义如下：

```
typedef enum
{
    QL_AUDIO_FORMAT_UNKNOWN,
    QL_AUDIO_FORMAT_PCM,
    QL_AUDIO_FORMAT_WAVPCM,
    QL_AUDIO_FORMAT_MP3,
    QL_AUDIO_FORMAT_AMRNB,
    QL_AUDIO_FORMAT_AMRWB,
    QL_AUDIO_FORMAT_MAX
} AudStreamFormat_e;
```

- **参数**

参数	描述
QL_AUDIO_FORMAT_UNKNOWN	未知格式

QL_AUDIO_FORMAT_PCM	PCM 格式
QL_AUDIO_FORMAT_WAVPCM	WAV 格式
QL_AUDIO_FORMAT_MP3	MP3 格式
QL_AUDIO_FORMAT_AMRNB	AMRNB 格式
QL_AUDIO_FORMAT_AMRWB	AMRWB 格式

4.2.1.4. pcm_path_e

PCM 数据流录音/播放方式枚举信息定义如下：

```
typedef enum
{
    QL_PCM_NONE = 0,
    QL_PCM_LOCAL,
    QL_PCM_VOICE_CALL,
    QL_PCM_POC,
    QL_PCM_MAX,
}pcm_path_e;
```

- **参数**

参数	描述
QL_PCM_LOCAL	本地进行 PCM 数据流录音/播放
QL_PCM_VOICE_CALL	远端进行 PCM 数据流录音/播放
QL_PCM_POC	POC 方式进行 PCM 数据流录音/播放 （音质高，但只支持 PCM 编码格式）

4.2.2. ql_pcm_read

该函数用于读取 PCM 数据流。

- **函数原型**

```
int ql_pcm_read(PCM_HANDLE_T handle, void *data, unsigned int count)
```

- **参数**

handle：
[In] *ql_aud_pcm_open()*返回的句柄。

data：
[In] 播放数据存放的缓存地址。

count：
[In] 播放的 PCM 数据的字节数。

- **返回值**

0	无可读数据
小于 0	数据读取失败
大于 0	实际读取数据的字节数

4.2.3. ql_pcm_write

该函数用于播放 PCM 数据流。

- **函数原型**

```
int ql_pcm_write(PCM_HANDLE_T handle, void *data, unsigned int count)
```

- **参数**

handle：
[In] *ql_aud_pcm_open()*返回的句柄。

data：
[In] 播放数据存放的缓存地址。

count：
[In] 播放的 PCM 数据的字节数。

- **返回值**

0	数据缓冲区已满
小于 0	播放失败
大于 0	实际播放数据的字节数

4.2.4. ql_aud_data_done

此函数用于停止写入 PCM 数据流，CP 侧将在缓存中数据为空后，停止从 AP 侧播放 PCM 数据流。调用此函数后，可调用 *ql_aud_wait_play_finish()*等待音频播放结束，且不可再调用 *ql_pcm_write()*向内核写数据；不调用此函数的情况下，不会停止播放 PCM 数据流，且可继续调用 *ql_pcm_write()*播放 PCM 数据流。

- **函数原型**

```
void ql_aud_data_done(void)
```

- **参数**

无

- **返回值**

无

4.2.5. ql_pcm_close

该函数用于关闭 PCM 设备，释放系统资源。

- **函数原型**

```
int ql_pcm_close(PCM_HANDLE_T handle)
```

- **参数**

handle：
[In] *ql_aud_pcm_open()*返回的句柄。

- **返回值**

0　　函数执行成功
其他　　函数执行失败

4.2.6. ql_pcm_buffer_reset

该函数用清空当前 PCM 数据流的音频数据缓冲区。

- **函数原型**

```
ql_audio_errcode_e ql_pcm_buffer_reset(PCM_HANDLE_T handle)
```

- **参数**

handle：
[In] *ql_aud_pcm_open()*返回的句柄。

- **返回值**

请参考***第 3.3.1.2 章***。

4.2.7. ql_pcm_buffer_used

该函数用获取 PCM 数据流的音频数据缓冲区已使用的大小，缓冲区总大小均为 16 KB。

- **函数原型**

```
int ql_pcm_buffer_used(PCM_HANDLE_T handle)
```

- **参数**

handle：
[In] *ql_aud_pcm_open()*返回的句柄。

- **返回值**

其他值	函数执行失败，参数无效
大于 0	缓冲区剩余大小

4.3. 音频通道类 API

4.3.1. ql_set_audio_path_earphone

该函数用于打开耳机通道。输入通道为耳机自带 MIC，输出通道为耳机自带扬声器。

- **函数原型**

```
ql_audio_errcode_e ql_set_audio_path_earphone (void)
```

- **参数**

无

- **返回值**

请参考***第 3.3.1.2 章***。

4.3.2. ql_set_audio_path_receiver

该函数用于打开听筒通道。输入通道为麦克风，输出通道为听筒。

- **函数原型**

```
ql_audio_errcode_e ql_set_audio_path_receiver (void)
```

- **参数**

无

- **返回值**

请参考***第 3.3.1.2 章***。

4.3.3. ql_set_audio_path_speaker

该函数用于打开扬声器通道。输入通道为麦克风，输出通道为扬声器。

- **函数原型**

```
ql_audio_errcode_e ql_set_audio_path_speaker (void)
```

- **参数**

无

- **返回值**

请参考***第 3.3.1.2 章***。

4.3.4. ql_aud_set_output_type

该函数用于设置音频输出通道。

- **函数原型**

```
ql_audio_errcode_e ql_aud_set_output_type(AudOutputType_e output_type)
```

- **参数**

output_type：
[In] 输出通道。请参考***第 4.3.4.1 章***。

- **返回值**

请参考*第 3.3.1.2 章*。

4.3.4.1. AudOutputType_e

音频输出通道枚举信息定义如下：

```
typedef enum
{
    QL_OUTPUT_RECEIVER = 0,    //< receiver
    QL_OUTPUT_HEADPHONE = 1, //< headphone
    QL_OUTPUT_SPEAKER = 2,     //< speaker
    QL_OUTPUT_MAX,
} AudOutputType_e;
```

- **参数**

参数	描述
QL_OUTPUT_RECEIVER	音频输出到听筒通道
QL_OUTPUT_HEADPHONE	音频输出到耳机通道
QL_OUTPUT_SPEAKER	音频输出到扬声器通道

4.3.5. ql_aud_get_output_type

该函数用于获取当前音频输出通道。

- **函数原型**

```
AudOutputType_e ql_aud_get_output_type(void)
```

- **参数**

无

- **返回值**

请参考*第 4.3.4.1 章*。

4.3.6. ql_aud_set_input_type

该函数用于设置音频输入通道。

- **函数原型**

```
ql_audio_errcode_e ql_aud_set_input_type(AudInputType_e input_type)
```

- **参数**

input_type：
[In] 输入通道。请参考***第 4.3.6.1 章***。

- **返回值**

请参考***第 3.3.1.2 章***。

4.3.6.1. AudInputType_e

音频输入通道，枚举信息定义如下：

```
typedef enum
{
    QL_INPUT_MAINMIC = 0, //< main mic
    QL_INPUT_AUXMIC   = 1,   //< auxilary mic
    QL_INPUT_DUALMIC = 2, //< dual mic
    QL_INPUT_HPMIC_L = 3, //< headphone mic left
    QL_INPUT_HPMIC_R = 4, //< headphone mic right
} AudInputType_e;
```

- **参数**

参数	描述
QL_INPUT_MAINMIC	音频输入通道为主 MIC
QL_INPUT_AUXMIC	音频输入通道为辅助 MIC
QL_INPUT_DUALMIC	音频输入通道为双 MIC（主 MIC+辅助 MIC）
QL_INPUT_HPMIC_L	音频输入通道为耳机 MIC（MIC 位于耳机左通道上）
QL_INPUT_HPMIC_R	音频输入通道为耳机 MIC（MIC 位于耳机右通道上）

4.3.7. ql_aud_get_input_type

该函数用于获取当前音频输入通道。

- **函数原型**

```
AudInputType_e ql_aud_get_input_type(void)
```

- **参数**

无

- **返回值**

请参考***第 4.3.6.1 章***。

4.3.8. ql_aud_set_pa_type

该函数用于设置默认内置音频功放类型，该设置重启模块后生效。

- **函数原型**

```
ql_audio_errcode_e ql_aud_set_pa_type(QL_AUD_SPKPA_TYPE_E pa_type)
```

- **参数**

pa_type：
[In] 内置功放类型。请参考***第 4.3.8.1 章***。

- **返回值**

请参考***第 3.3.1.2 章***。

4.3.8.1. QL_AUD_SPKPA_TYPE_E

内置功放类型枚举信息定义如下：

```
typedef enum
{
    QL_AUD_SPKPA_TYPE_CLASSAB,
    QL_AUD_SPKPA_TYPE_CLASSD,
    QL_AUD_SPKPA_TYPE_MAX
} QL_AUD_SPKPA_TYPE_E;
```

- **参数**

参数	描述
QL_AUD_SPKPA_TYPE_CLASSAB	内置功放为 AB 类
QL_AUD_SPKPA_TYPE_CLASSD	内置功放为 D 类

4.3.9. ql_aud_get_pa_type

该函数用于获取默认内置音频功放类型。

- **函数原型**

```
QL_AUD_SPKPA_TYPE_E ql_aud_get_pa_type(void)
```

- **参数**

无

- **返回值**

请参考***第 4.3.8.1 章***。

4.4. 铃声类 API

4.4.1. ql_aud_set_ringtone_type

该函数用于设置默认来电铃声类型。

- **函数原型**

```
ql_audio_errcode_e ql_aud_set_ringtone_type(QL_AUD_RING_TYPE_E type)
```

- **参数**

type：
[In] 来电铃声类型。请参考***第 4.4.1.1 章***。
当选择客户自定义方式时，可以使用 *ql_bind_ring_tone_cb()*绑定回调函数进行来电铃声播放。详情请参考***第 4.4.3 章***。

- **返回值**

请参考*第 3.3.1.2 章*。

4.4.1.1. QL_AUD_RING_TYPE_E

来电铃声类型枚举信息定义如下：

```
typedef enum
{
    QL_AUD_RING_NONE   =   0,      //no ring tone
    QL_AUD_RING_NOKIA,             //invalid now
    QL_AUD_RING_DIAL_TONE,
    QL_AUD_RING_CUSTOMER_DEF,
    QL_AUD_RING_MAX,
}QL_AUD_RING_TYPE_E;
```

- **参数**

参数	描述
QL_AUD_RING_NONE	无来电铃声
QL_AUD_RING_DIAL_TONE	默认铃声
QL_AUD_RING_CUSTOMER_DEF	自定义来电铃声

4.4.2. ql_aud_set_ringtone_type

该函数用于获取默认来电铃声类型。

- **函数原型**

```
QL_AUD_RING_TYPE_E ql_aud_get_ringtone_type(void)
```

- **参数**

无

- **返回值**

请参考*第 4.4.1.1 章*。

4.4.3. ql_bind_ring_tone_cb

该函数用于绑定来电铃声的回调函数，用于自定义播放来电铃声。需要配合使用*ql_aud_set_ringtone_type()*将来电铃声类型设置为客户自定义模式。

- **函数原型**

```
ql_audio_errcode_e ql_bind_ring_tone_cb(cb_on_ring_tone cb)
```

- **参数**

cb：
[In] 来电铃声的回调函数。当有电话呼入、本地接通电话或本地电话未接通而对方挂断电话时，将调用该回调函数。

- **返回值**

请参考***第 3.3.1.2 章***。

4.5. TTS 相关 API

4.5.1. ql_tts_engine_init

该函数用于初始化 TTS 引擎。

- **函数原型**

```
ql_errcode_tts_e ql_tts_engine_init(pUserCallback mCallback)
```

- **参数**

mCallback：
[In] 回调函数。用于将解析得到的 PCM 音频数据发送至应用层，在调用 *ql_tts_start()*播放 TTS 后就会进入此回调函数。

- **返回值**

请参考***第 4.5.2 章***。

4.5.2. ql_errcode_tts_e

TTS 错误码表示函数是否执行成功，若失败则返回错误原因，枚举信息定义如下：

```
typedef enum
{
QL_TTS_SUCCESS                    = QL_SUCCESS,
QL_TTS_UNKNOWN_ERROR              = 901 | (QL_COMPONENT_AUDIO_TTS << 16),
QL_TTS_INVALID_PARAM              = 902 | (QL_COMPONENT_AUDIO_TTS << 16),
QL_TTS_OPERATION_NOT_SUPPORT      = 903 | (QL_COMPONENT_AUDIO_TTS << 16),
QL_TTS_DEVICE_BUSY                = 904 | (QL_COMPONENT_AUDIO_TTS << 16),
QL_TTS_INIT_ENGINE_ERR            = 2001 | (QL_COMPONENT_AUDIO_TTS << 16),
QL_TTS_INIT_SOURCE_ERR            = 2002 | (QL_COMPONENT_AUDIO_TTS << 16),

QL_TTS_START_ERR                  = 2003 | (QL_COMPONENT_AUDIO_TTS << 16),
QL_TTS_STOP_ERR                   = 2004 | (QL_COMPONENT_AUDIO_TTS << 16),
QL_TTS_EXIT_ERR                   = 2005 | (QL_COMPONENT_AUDIO_TTS << 16),
}ql_errcode_tts_e;
```

- **参数**

参数	描述
QL_TTS_SUCCESS	函数执行成功
QL_TTS_UNKNOWN_ERROR	未知错误
QL_TTS_INVALID_PARAM	无效参数
QL_TTS_OPERATION_NOT_SUPPORT	不支持该操作
QL_TTS_DEVICE_BUSY	TTS 正在播放
QL_TTS_INIT_ENGINE_ERR	TTS 引擎初始化失败
QL_TTS_INIT_SOURCE_ERR	TTS 数据库异常
QL_TTS_START_ERR	TTS 开始播放失败
QL_TTS_STOP_ERR	TTS 停止播放失败
QL_TTS_EXIT_ERR	TTS 退出失败

4.5.3. ql_tts_set_config_param

该函数用于在播放 TTS 前设置配置选项。

- **函数原型**

```
ql_errcode_tts_e ql_tts_set_config_param(ql_tts_config_e type, int value)
```

- **参数**

type：
[In] TTS 的配置类型。请参考***第 4.5.3.1 章***。

value：
[In] 对应播放速度：-32768~32767；
播放音量大小：-32768~32767；
解码格式：*0* GBK
1 UTF-8。

- **返回值**

请参考***第 4.5.2 章***。

4.5.3.1. ql_tts_config_e

TTS 的配置类型，枚举信息定义如下：

```
typedef enum
{
    QL_TTS_CONFIG_SPEED = 1,
    QL_TTS_CONFIG_VOLUME,
    QL_TTS_CONFIG_ENCODING,
    QL_TTS_CONFIG_MAX
}ql_tts_config_e;
```

- **参数**

参数	描述
QL_TTS_CONFIG_SPEED	设置播放 TTS 的速度
QL_TTS_CONFIG_VOLUME	设置播放 TTS 的音量
QL_TTS_CONFIG_ENCODING	设置输入文本的编码格式。请参考***第 4.5.3.2 章***。

4.5.3.2. ql_tts_encoding_e

输入文本的编码格式，枚举信息定义如下：

```
typedef enum
{
    QL_TTS_GBK=0,
    QL_TTS_UTF8,
    QL_TTS_UCS2,
}ql_tts_encoding_e;
```

- **参数**

参数	描述
QL_TTS_GBK	输入播放的 TTS 文本为 GBK 编码
QL_TTS_UTF8	输入播放的 TTS 文本为 UTF-8 编码
QL_TTS_UCS2	输入播放的 TTS 文本为 UCS-2 编码

4.5.4. ql_tts_get_config_param

该函数用于获取 TTS 的配置选项。

- **函数原型**

```
int ql_tts_get_config_param(ql_tts_config_e type)
```

- **参数**

type：
[In] TTS 的配置类型。请参考***第 4.5.3.1 章***。

- **返回值**

返回对应播放速度、播放音量大小和解码格式。

4.5.5. ql_tts_is_running

该函数用于获取当前是否正在播放 TTS。

- **函数原型**

```
int ql_tts_is_running(void)
```

- **参数**

无

- **返回值**

0 当前未播放 TTS
1 当前正在播放 TTS

4.5.6. ql_utf8_to_gbk_str

该函数用于将 UTF-8 格式的字符串转换为 GBK 格式的字符串。

- **函数原型**

```
void ql_utf8_to_gbk_str(void *utf8, int inputlen, int *outputlen, void *gbk)
```

- **参数**

utf8：
[In] 输入的 UTF-8 格式的字符串。

Inputlen：
[In] 输入的 UTF-8 格式的字符串的长度。

outputlen：
[Out] 转换为 GBK 格式的字符串的长度。

gbk：
[Out] 转换成 GBK 格式的字符串。

- **返回值**

无

4.5.7. ql_tts_start

该函数用于开始播放 TTS。该函数会阻塞式进行 TTS 解析，解析过程中会不断调用 *ql_tts_engine_init()* 注册的回调函数，将解析得到的 PCM 音频数据送至应用层。当输入的 TTS 字符串均已完成解析后，自动退出 *ql_tts_start()*。若需要提前退出 *ql_tts_start()*，可设置 *ql_tts_engine_init()*函数注册的回调函数返回-1，或者在应用层调用 *ql_tts_exit()*函数。

- **函数原型**

```
ql_errcode_tts_e ql_tts_start(const char *textString, unsigned int textLen)
```

- **参数**

textString：
[In] 输入 UTF-8 或者 GBK 格式的字符串。

textLen：
[In] 输入 UTF-8 或者 GBK 格式的字符串的长度。

- **返回值**

请参考***第 4.5.2 章***。

4.5.8. ql_tts_end

该函数用于在 TTS 播放完成时释放占用资源。

- **函数原型**

```
ql_errcode_tts_e ql_tts_end(void)
```

- **参数**

无

- **返回值**

请参考***第 4.5.2 章***。

4.5.9. ql_tts_exit

该函数用于中断 TTS 播放并退出 TTS。

- **函数原型**

```
ql_errcode_tts_e ql_tts_exit(void)
```

- **参数**

无

- **返回值**

请参考*第 4.5.2 章*。

备注

以下备注针对本文涉及的所有音频调试 API：
1. 通话模式下，禁止进行本地录入/播放音频。
2. 非通话模式下，禁止进行远端录入/播放音频。

4.6. 音频增益类 API

4.6.1. ql_aud_set_calib_real_time

该函数用于打开/关闭音频参数的实时校准功能。调用函数进行音频参数调整之前，须调用该函数关闭实时校准功能。

- **函数原型**

```
ql_audio_errcode_e ql_aud_set_calib_real_time(bool real_time)
```

- **参数**

real_time：
[In] 打开/关闭实时校准功能。
0 关闭
1 打开

- **返回值**

请参考*第 3.3.1.2 章*。

4.6.2. ql_aud_set_icvolume_level_gain

该函数用于设置不同场景下，音量等级对应的数字增益和算法增益。

- **函数原型**

```
ql_audio_errcode_e   ql_aud_set_icvolume_level_gain
(
        AudPlayType_e                play_mode,
    AudOutputType_e          out_type,
    AUDIOHAL_SPK_LEVEL_T     level,
    uint16                   dac_gain,
    uint16                   alg_gain
)
```

- **参数**

play_mode：
[In] 音频播放方式。请参考***第 4.1.1.1 章***。

out_type：
[In] 输出通道。请参考***第 4.3.4.1 章***。

level：
[In] 音量等级。请参考***第 4.1.9.1 章***。

dac_gain：
[In] 数字增益，范围为 0~99。对应的真实增益为：

0 -26 dB
1 -25 dB
2 -25 dB
3 -24.5 dB
98 23 dB
99 23.5 dB

alg_gain:
[In] 算法增益，范围为 0~15。对应的真实增益为：

0 MUTE（静音）
1 -42 dB
2 -39 dB
3 -36 dB
15 0 dB

- **返回值**

请参考***第 3.3.1.2 章***。

4.6.3. ql_aud_get_icvolume_level_gain

该函数用于获取不同场景下，音量等级对应的数字增益和算法增益。

- **函数原型**

```
ql_audio_errcode_e    ql_aud_get_icvolume_level_gain
(
        AudPlayType_e           play_mode,
    AudOutputType_e         out_type,
    AUDIOHAL_SPK_LEVEL_T    level,
    uint16                  *dac_gain,
    uint16                  *alg_gain
)
```

- **参数**

play_mode：
[In] 音频播放方式。请参考***第 4.1.1.1 章***。

out_type：
[In] 输出通道。请参考***第 4.3.4.1 章***。

level：
[In] 音量等级。请参考***第 4.1.9.1 章***。

dac_gain：
[Out] 数字增益，范围为 0~99。对应的真实增益为：
0 -26 dB
1 -25 dB
2 -25 dB
3 -24.5 dB
98 23 dB
99 23.5 dB

alg_gain：
[Out] 算法增益，范围为 0~15。对应的真实增益为：
0 MUTE（静音）
1 -42 dB
2 -39 dB
3 -36 dB

15 0 dB

- **返回值**

请参考*第 3.3.1.2 章*。

5 音频开发示例

移远通信 ECx00U 系列和 EGx00U 模块 QuecOpen SDK 代码中提供了音频调试的示例，即 *audio_demo.c* 文件，路径为 *sdk\ components\ql-application\audio*。有关如何在 APP 侧使用上述 API 进行音频开发以及简单调试，请参考 *audio_demo.c*。

6 常见问题

1. 问：录音为何异常停止？
   答：(1) 录音时，要注意需在短时间内不断读取外置 Codec 生成的音频数据；目前音频数据最大缓存为 16 KB，若编码器中的音频缓存数据超过 16 KB，录音将异常停止。该情况为异常停止的常见原因。
   (2) 来电时，录音将会被直接打断。
   (3) 录音时，若文件系统已满，将直接停止录音。

2. 问：播放为何异常停止？
   答：播放异常停止，一般是因为来电打断，或者是其他线程调用了音频播放结束函数 *ql_pcm_close()* 或 *ql_aud_player_stop()*。

3. 问：为何出现播放无声或录音无声的情况？
   答：此情况一般是由于外置 Codec 或音频通道未选择正确导致的。

7 附录 参考文档和术语缩写

表 2：参考文档

文档名称
[1] Quectel_ECx00U&EGx00U 系列_QuecOpen_CSDK_快速开发指导

表 3：术语缩写

缩写	英文全称	中文全称
AIF	Audio Interface	声卡
API	Application Programming Interface	应用程序编程接口
AP	Application Processor	应用处理器
APP	Application	应用
AMR	Adaptive Multi Rate	自适应多速率
CP	Cellular Processor	基带芯片加协处理器
IoT	Internet of Things	物联网
NV	Non-Volatile Random Access Memory	非易失性随机访问存储器
PCM	Pulse Code Modulation	脉冲编码调制
SDK	Software Development Kit	软件开发工具包
TTS	Text To Speech	文本转语音