度秘SDK（嵌入式版）V2.0.2

百度在线网络技术（北京）有限公司

(版权所有,翻版必究)

修改记录

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | 修改后  版本号 | 修改内容简介 | 修改日期 | 修改人 |
| 1 | V1.0 | 全文 |  | [jianchuanli@baidu.com](mailto:jianchuanli@baidu.com) |
| 2 | V2.0 | 全文 |  | [jianchuanli@baidu.com](mailto:jianchuanli@baidu.com) |
| 3 | v2.0.1 |  |  | [jianchuanli@baidu.com](mailto:jianchuanli@baidu.com) |
| 4 | V2.0.1 | 添加第三部分内容—音箱speaker设备相关 |  | changxiantang |
| 5 | V2.0.2 | 修改device event,status接口参数   1. 设置可写文件根路径目录地址 2. 设置可写文件名 3. device\_event,device\_status接口及参数优化 4. tts合成，增加透传参数 |  | changxiantang |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

目 录

度秘SDK（嵌入式版）V2.0.2 1

1、度秘SDK说明 5

2、相关接口 5

2.1、SDK初始化 5

2.2、SDK回调事件设置 6

2.3、SDK方法调用post 6

2.3.1、post接口定义 7

2.3.2、启动SDK 7

2.3.2.1、启动参数定义 8

2.3.3、推送录音数据 11

2.3.3.1、推送录音参数 11

2.3.4、取消语音引擎 12

2.3.5、打开语音识别 12

2.3.6、设置用户BDUSS（登录态参数） 13

2.3.6.1、设置用户BDUSS参数定义 13

2.3.7、清除用户BDUSS（登录态参数） 14

2.3.8、设置可写日志文件目录路径 14

2.3.9、设置可写日志文件名称 14

2.3.10、TTS合成（字符串转化为音频流） 15

2.3.10.1、TTS合成参数定义 15

2.3.10.2、切换TTS合成透传参数 15

2.3.11、切换TTS引擎声音 (tts合成音色切换) 16

2.3.12、语音引擎切换（适用于唤醒识别引擎） 16

2.3.12.1、语音引擎切换参数定义 17

2.3.13、度秘能力事件调用（事件上传、请求垂类） 17

2.3.13.1、度秘能力事件调用参数（事件上传、请求垂类） 18

2.4、SDK回调本地回调事件（message\_callback） 19

2.4.1、注册事件回调 19

2.4.2、语音识别事件回调 20

2.4.2.1、语音识别回调状态参数 20

2.4.3、语音唤醒回调 22

2.4.3.1、语音唤醒回调状态参数 22

2.4.4、度秘后端解析返回结果 23

2.4.4.1、度秘后端解析返回结果参数 23

2.4.5、TTS合成事件回调 23

2.4.5.1、TTS合成事件返回结果参数 24

2.4.6、度秘能力事件回调（事件上传、请求垂类） 25

2.4.6.1、度秘能力事件返回结果参数 25

2.4.7、一轮交互结束事件回调 26

2.4.7.1、一轮交互结束事件回调参数 27

3、音箱speaker设备相关 28

3.1 第三方需要提供的speaker设备相关信息 28

3.2 调用接口 29

3.3 调用时机 30

4、SDK设备相关 31

4.1设备提供者 31

4.2 设备定义 32

4.3 自定义设备接口 32

5、状态同步和事件上传相关 32

5.1、支持device类型 33

5.2、设备的状态同步 33

5.2.1、播放器(AudioPlayer)状态 33

5.2.2、扬声器(Speaker)状态 35

5.2.3、TTS合成播放(SpeechSynthesizer)状态 36

5.3、上传事件类型 37

5.3.1、播放器(AudioPlayer)事件数据 38

5.3.2、扬声器(Speaker)事件数据 39

5.3.3、TTS合成(SpeechSynthesizer)事件数据 40

5.3.4、Alerts(闹钟/提醒/定时器)事件数据 41

# 1、度秘SDK说明

度秘SDK集成了度秘具备的语音唤醒、语音识别、语音合成、多轮对话的人机交互能力。

说明：sdk只接收录音数据做识别，不提供录音功能，同时语音合成只有提供文字转化成原始pcm流数据，不提供播放功能。

# 2、相关接口

## 2.1、SDK初始化

在main函数方法中调用初始化接口，初始化BDuerSDK。

public int init(const char \*appId, const char \*appKey, const char \*deviceId, const char \*bduss = NULL);

|  |
| --- |
| 调用示例 |
| int req;  req = BDuerSDK::get\_instance()->initSDK(“appid”,”appkey”,”deviceId”,”bduss”);  if(req == BDUER\_POST\_SUCCESS)  {  //调用成功  } |

注意：appId，appKey需要向度秘方申请,bduss为用户登录的信息（如果没有可以传空或者不传入参数）。

## 2.2、SDK回调事件设置

设置sdk处理后的回调事件。

|  |
| --- |
| 调用示例 |
| class Req:public IBDuerSDKCallback  {  Req() {  }  ~Req() {  }  void remote\_callback(BDuerMessage &message) {  }  void message\_callback(BDuerMessage &message) {  }  }  IBDuerSDKCallback \*callback = new Req();  BDuerSDK::get\_instance()->set\_event\_listener(callback); |
|  |

## 2.3、SDK方法调用post

通过post(BDuermessage &msg,int \*req)可以调用tts合成，语音唤醒，语音识别等功能。

### 2.3.1、post接口定义

|  |
| --- |
| post(BDuerMessage &message, int \*req); |
| /\*\*  \* 向sdk发送消息指令  \* @param message 具体消息内容  \* @param req 返回值，调用成功返回BDUER\_POST\_SUCCESS  \*/  void post(BDuerMessage &message, int \*req); |

### 2.3.2、启动SDK

启动SDK的工作需要在注册成功的回调以后才能调用。

|  |
| --- |
| 调用示例 |
| class Req:public IBDuerSDKCallback  {  Req() {  }  ~Req() {  }  void remote\_callback(BDuerMessage &message) {  //mateApp远端回调(长链接回调)  }  void message\_callback(BDuerMessage &message) {  //本地事件回调  if (strcmp(BDUER\_CALLBACK\_REGISTER\_SUCCESS, message.get\_name()) == 0) {  cout << "注册成功" << endl;  int req;  BDuerMessage message;  message.set\_name(BDUER\_CMD\_START);  message.set\_parameter(BDUER\_PARAM\_KEY\_SPEECH\_ENGINE\_SUPPORT, SPEECH\_SUPPORT\_WAKEUP\_AND\_ASR);  message.set\_parameter(BDUER\_PARAM\_KEY\_AFTER\_WAKE\_UP, AFTER\_FINISH\_WAITING);  duerSDK->post(message, &req);  }  }  } |

#### 2.3.2.1、启动参数定义

其中在启动SDK时，需要设置message的name为BDUER\_CMD\_START和一部分parameter参数。

|  |
| --- |
| message.set\_name(const char \*cmd)命令设置 |
| BDUER\_CMD\_START |
| /\*\*  \*启动SDK命令(只能调用一次)  \*/  BDUER\_CMD\_START |

语音引擎类型的参数设置

|  |
| --- |
| message.set\_parameter(const char \*key, int value) 语音引擎类型的参数设置 |
| BDUER\_PARAM\_KEY\_SPEECH\_ENGINE\_SUPPORT |
| //开始设置语音引擎的类型  typedef enum SpeechEngineSupport {  //只支持语音识别  SPEECH\_SUPPORT\_ASR,  //仅支持语音唤醒  SPEECH\_SUPPORT\_WAKEUP,  //支持语音唤醒识别(如果不设置默认为此类型)  SPEECH\_SUPPORT\_WAKEUP\_AND\_ASR,  } SpeechEngineSupport; |

唤醒后的工作状态（只适用于唤醒识别引擎下使用）

|  |
| --- |
| message.set\_parameter(const char \*key, int value) 唤醒后的工作状态设置 |
| BDUER\_PARAM\_KEY\_AFTER\_WAKE\_UP |
| //唤醒后的工作状态（需要在设置了支持SPEECH\_SUPPORT\_WAKEUP\_AND\_ASR之后才有效）  typedef enum AfterFinishWakeUp {  //唤醒后进入等待引擎（需要外部启动开启后续工作）  AFTER\_FINISH\_WAITING,  //唤醒后直接进入语音识别引擎  AFTER\_FINISH\_OPEN\_ASR\_ENGINE  } AfterFinishWakeUp; |

语音识别和唤醒需要的配置文件路径参数

|  |
| --- |
| message.set\_parameter(const char \*key, int value) |
| BDUER\_PARAM\_KEY\_SPEECH\_RES\_FILE\_PATH |
| //设置语音识别和唤醒需要的配置文件路径，用于语音识别和唤醒使用如何不设置默认路径为./resources/ |

sdkconfigs配置文件路径设置

|  |
| --- |
| message.set\_parameter(const char \*key, int value) |
| BDUER\_PARAM\_KEY\_SDK\_CONFIG\_FILE\_PATH |
| //设置整个sdk的配置文件目录，默认设置为当前目录的./sdkconfigs/  配置文件内容：  {  "voice\_pid": “713", //语音识别需要的产品号  "voice\_package": “com.baidu.dumi.lenovo.box", //语音识别的需要的app名字  "voice\_server\_url": “http://vse.baidu.com/v2", //语音识别链接的服务器地址  "debug\_switch": “1", //debug开关  "sdk\_version": “V0.7.0-Lenovo\_speaker-R" //当前version版本号  } |

### 2.3.3、推送录音数据

|  |
| --- |
| message.set\_name(const char\* cmd) 命令设置 |
| BDUER\_CMD\_AUDIORECORDER\_PCM\_STREAM\_BUFFER |
| /\*\*  \* 向SDK中推送录音的数据命令  \*/  BDUER\_CMD\_AUDIORECORDER\_PCM\_STREAM\_BUFFER |

#### 2.3.3.1、推送录音参数

|  |
| --- |
| 调用示例 |
| int req;  BDuerMessage message;  message.set\_name(BDUER\_CMD\_AUDIORECORDER\_PCM\_STREAM\_BUFFER);  message.set\_parameter(BDUER\_PARAM\_KEY\_AUDIORECORDER\_PCM\_STREAM\_DATA, stream, streamSize);  duerSDK->post(message, &req); |

定义设置录音数据

|  |
| --- |
| message.set\_parameter(const char \*key, const char \*value, int valueLen) 设置录音数据 |
| BDUER\_PARAM\_KEY\_AUDIORECORDER\_PCM\_STREAM\_DATA |

### 2.3.4、取消语音引擎

取消语音识别，则放弃已经产生的音频输入数据，无返回结果。

|  |
| --- |
| message.set\_name(const char\* cmd) 命令设置 |
| BDUER\_CMD\_CANCEL\_SPEECH\_ENGINE |
| /\*\*  \* 通知SDK关闭语音引擎  \*/  BDUER\_CMD\_CANCEL\_SPEECH\_ENGINE |

### 2.3.5、打开语音识别

打开当前语音引擎。

|  |
| --- |
| message.set\_name(const char\* cmd) 命令设置 |
| 如果设置的语音引擎的类型是 SPEECH\_SUPPORT\_ASR， 则设置如下参数：  message.set\_name(BDUER\_CMD\_START\_SPEECH\_ENGINE); |
| 如果设置的语音引擎的类型是 SPEECH\_SUPPORT\_WAKEUP\_AND\_ASR,则设置如下参数：  message.set\_name(BDUER\_CMD\_OPEN\_SPEECH\_ENGINE);  message.set\_parameter(BDUER\_PARAM\_KEY\_OPEN\_SPEECH\_ENGINE,SPEECH\_ENGINE\_ASR); |

### 2.3.6、设置用户BDUSS（登录态参数）

|  |
| --- |
| message.set\_name(const char\* cmd) 命令设置 |
| BDUER\_CMD\_SET\_USER\_BDUSS |
| /\*\*  \* 设置用户BDUSS（登录态参数）  \*/ BDUER\_CMD\_SET\_USER\_BDUSS |

#### 2.3.6.1、设置用户BDUSS参数定义

设置bduss参数

|  |
| --- |
| message.set\_parameter(const char \*key, const char \*value, int valueLen) 设置用户登录态参数 |
| BDUER\_PARAM\_KEY\_USER\_BDUSS |

### 2.3.7、清除用户BDUSS（登录态参数）

|  |
| --- |
| message.set\_name(const char\* cmd) 命令设置 |
| BDUER\_CMD\_CLEAR\_USER\_BDUSS |
| /\*\*  \* 清除用户BDUSS（登录态参数）  \*/ BDUER\_CMD\_CLEAR\_USER\_BDUSS |

### 2.3.8、设置可写日志文件目录路径

|  |
| --- |
| BDuerMessage msg;  msg.set\_name(BDUER\_CMD\_START);  string local\_write\_file\_path = “”;  msg.set\_parameter(BDUER\_PARAM\_KEY\_LOCAL\_WRITE\_FILE\_PATH, local\_write\_file\_path, local\_write\_file\_path.length()); |
|  |

### 2.3.9、设置可写日志文件名称

|  |
| --- |
| BDuerMessage msg;  msg.set\_name(BDUER\_CMD\_START);  string local\_write\_file\_name = “”;  msg.set\_parameter(BDUER\_PARAM\_KEY\_LOCAL\_WRITE\_FILE\_NAME, local\_write\_file\_name, local\_write\_file\_name.length()); |
|  |

### 2.3.10、TTS合成（字符串转化为音频流）

|  |
| --- |
| message.set\_name(const char\* cmd) 命令设置 |
| BDUER\_CMD\_TTS\_STR |
| /\*\*  \* TTS合成  \*/ BDUER\_CMD\_TTS\_STR |

#### 2.3.10.1、TTS合成参数定义

设置TTS合成的字符串参数

|  |
| --- |
| set\_parameter(const char\* key, const char \*value, int valueLen) 设置TTS字符传参数 |
| BDUER\_PARAM\_KEY\_TTS\_STR |

#### 2.3.10.2、切换TTS合成透传参数

设置TTS合成透传参数

|  |
| --- |
| set\_parameter(const char\* key,const char\*value int value\_length); |
| BDUER\_PARAM\_KEY\_TTS\_STRING\_INDEX |

### 2.3.11、切换TTS引擎声音 (tts合成音色切换)

|  |
| --- |
| message.set\_name(const char\* cmd) 命令设置  message.set\_parameter(const char\* key,int vlaue); |
| BDUER\_CMD\_SWITCH\_TTS\_ENGINE\_TYPE  BDUER\_PARAM\_KEY\_SWITCH\_TTS\_ENGINE\_TYPE |
| /\*\*  \* TT引擎切换类型  \*/ typedef enum \_ttsEngineType {  ttsEngineType\_Female = 0,  ttsEngineType\_male\_normal = 1,  ttsEngineType\_male\_special = 2,  ttsEngineType\_male\_emotion = 3,  ttsEngineType\_children\_emotion = 4  } ttsEngineType; |

### 2.3.12、语音引擎切换（适用于唤醒识别引擎）

该命令只适用于使用唤醒识别的时起作用。

|  |
| --- |
| message.set\_name(const char\* cmd) 命令设置 |
| BDUER\_CMD\_OPEN\_SPEECH\_ENGINE |
| /\*\*  \* 语音引擎切换  \*/ BDUER\_CMD\_OPEN\_SPEECH\_ENGINE |

#### 2.3.12.1、语音引擎切换参数定义

引擎类型参数设置

|  |
| --- |
| set\_parameter(const char\* key, int value) 命令设置 |
| BDUER\_PARAM\_KEY\_OPEN\_SPEECH\_ENGINE |
| /\*\*  \* 当前工作的引擎  \*/ typedef enum SpeechEngineState {  //语音引擎  SPEECH\_ENGINE\_ASR,  //等待引擎（不接受任何录音数据）  SPEECH\_ENGINE\_WAITING,  //唤醒引擎  SPEECH\_ENGINE\_WAKEUP,  } SpeechEngineState; |

### 2.3.13、度秘能力事件调用（事件上传、请求垂类）

|  |
| --- |
| message.set\_name(const char\* cmd) 命令设置 |
| BDUER\_CMD\_SEND\_HTTP\_REQUEST |
| /\*\*  \* 度秘能力事件调用  \*/ BDUER\_CMD\_SEND\_HTTP\_REQUEST |

#### 2.3.13.1、度秘能力事件调用参数（事件上传、请求垂类）

|  |
| --- |
| set\_parameter(const char\* key, int value) 请求类型参数设置 |
| BDUER\_PARAM\_KEY\_HTTP\_REQUEST\_TYPE |
| /\*\*  \* 请求类型  \*/ typedef enum HttpRequestType {  //事件上报请求  HTTP\_REQUEST\_TYPE\_EVENT,  //闹钟拉取请求  HTTP\_REQUEST\_TYPE\_ALARM  } HttpRequestType; |

请求body参数设置

|  |
| --- |
| set\_parameter(const char\* key, const char \*value, int valueLen);请求参数中的body设置 |
| BDUER\_PARAM\_KEY\_HTTP\_REQUEST\_BODY |

## 2.4、SDK回调本地回调事件（message\_callback）

|  |
| --- |
| char\* message.get\_name() 回调命令 |
| BDUER\_CALLBACK\_REGISTER\_SUCCESS |
| /\*\*  \*注册事件回调  \*/ BDUER\_CALLBACK\_REGISTER\_SUCCESS |

### 2.4.1、注册事件回调

### 2.4.2、语音识别事件回调

|  |
| --- |
| char\* message.get\_name() 回调命令 |
| BDUER\_CALLBACK\_ASR |
| /\*\*  \*语音识别回调命令  \*/ BDUER\_CALLBACK\_ASR |

#### 2.4.2.1、语音识别回调状态参数

|  |
| --- |
| message.get\_parameter(const char\* key,int &value) 获取语音回调的状态 |
| BDUER\_PARAM\_KEY\_ASR\_STATUS |
| int status;  message.get\_parameter(BDUER\_PARAM\_KEY\_ASR\_STATUS, status);  /\*\*  \*语音识别枚举状态  \*/  typedef enum ASRStatus {  ASRStatusStart = 0, // 检测到用户开始说话  ASRStatusFlushData, // 连续上屏  ASRStatusFinish, // 语音识别功能完成，服务器返回正确结果  ASRStatusMeterLevel, // 当前音量回调  ASRStatusCancel // 用户取消  //ASRStatusError, // 发生错误  } ASRStatus; |

语音识别返回的字符串参数

|  |
| --- |
| message.get\_parameter(const char\* key,char\* &value) 获取语音回调的字符串参数 |
| BDUER\_PARAM\_KEY\_ASR\_DATA |
| char \*word;  message.get\_parameter(BDUER\_PARAM\_KEY\_ASR\_DATA, word); |

### 2.4.3、语音唤醒回调

|  |
| --- |
| char\* message.get\_name() 回调命令 |
| BDUER\_CALLBACK\_WAKEUP |
| /\*\*  \*语音唤醒回调命令  \*/ BDUER\_CALLBACK\_WAKEUP |

#### 2.4.3.1、语音唤醒回调状态参数

|  |
| --- |
| char\* message.get\_parameter(const char\* key,int &value) 语音唤醒回调状态参数 |
| BDUER\_PARAM\_KEY\_WAKEUP\_STATUS |
| int status;  message.get\_parameter(BDUER\_PARAM\_KEY\_WAKEUP\_STATUS, status);  /\*\*  \*语音唤醒枚举状态  \*/  typedef enum WakeupStatus {  WakeupStatusTriggered, // 命中唤醒词  } WakeupStatus; |

### 2.4.4、度秘后端解析返回结果

|  |
| --- |
| char\* message.get\_name() 回调命令 |
| BDUER\_CALLBACK\_DUER |
| /\*\*  \*度秘后端解析返回结果  \*/ BDUER\_CALLBACK\_DUER |

#### 2.4.4.1、度秘后端解析返回结果参数

|  |
| --- |
| char\* message.get\_parameter(const char\* key,char\* &value) 度秘后端解析返回结果参数 |
| BDUER\_PARAM\_KEY\_DUER\_SERVER\_CONTENT |
| char \*content;  message.get\_parameter(BDUER\_PARAM\_KEY\_DUER\_SERVER\_CONTENT, content); |

### 2.4.5、TTS合成事件回调

|  |
| --- |
| char\* message.get\_name() 回调命令 |
| BDUER\_CALLBACK\_TTS |
| /\*\*  \*TTS合成事件回调  \*/ BDUER\_CALLBACK\_TTS |

#### 2.4.5.1、TTS合成事件返回结果参数

|  |
| --- |
| char\* message.get\_parameter(const char\* key,int &value) TTS合成事件返回结果参数 |
| BDUER\_PARAM\_KEY\_TTS\_STATUS |
| int status;  message.get\_parameter(BDUER\_PARAM\_KEY\_TTS\_STATUS, status);  //TTS合成枚举状态  typedef enum TTSStatus {  TTSStatusBegin, // 语音合成开始  TTSStatusAudioData, // 正在合成语音  TTSStatusFinish, // 语音合成结束  } TTSStatus; |

TTS合成的pcm流数据获取

|  |
| --- |
| char\* message.get\_parameter(const char\* key, const char \*&outValue, int &outValueLen);  TTS合成的pcm流数据获取 |
| BDUER\_PARAM\_KEY\_TTS\_AUDIO\_DATA |
| /\*\*  \* char \*stream　是返回的tts数据  \* int length 是返回的tts的数据长度  \*/  const char \*stream;  int length;  message.get\_parameter(BDUER\_PARAM\_KEY\_TTS\_AUDIO\_DATA, stream, length); |

### 2.4.6、度秘能力事件回调（事件上传、请求垂类）

|  |
| --- |
| char\* message.get\_name() 回调命令 |
| BDUER\_CALLBACK\_HTTP\_RESPONSE |
| /\*\*  \*度秘能力事件回调  \*/ BDUER\_CALLBACK\_HTTP\_RESPONSE |

#### 2.4.6.1、度秘能力事件返回结果参数

获取度秘能力返回的结果

|  |
| --- |
| char\* message.get\_parameter(const char\* key, const char \*&outValue, int &outValueLen);  获取度秘能力返回的结果 |
| BDUER\_PARAM\_KEY\_HTTP\_RESPONSE |
| const char \*content;  int length;  message.get\_parameter(BDUER\_PARAM\_KEY\_HTTP\_RESPONSE, content, length); |

获取返回的度秘能力的类型(事件上传、垂类返回)

|  |
| --- |
| char\* message.get\_parameter(const char\* key, int &value);  获取返回的度秘能力的类型(事件上传、垂类返回) |
| BDUER\_PARAM\_KEY\_HTTP\_REQUEST\_TYPE |
| int request\_type;  message.get\_parameter(BDUER\_PARAM\_KEY\_HTTP\_REQUEST\_TYPE, length); |

### 2.4.7、一轮交互结束事件回调

识别结束并且请求度秘后端服务结束

|  |
| --- |
| char\* message.get\_name() 回调命令 |
| BDUER\_CALLBACK\_FINAL |
| /\*\*  \*一轮交互结束事件回调  \*/ BDUER\_CALLBACK\_FINAL |

#### 2.4.7.1、一轮交互结束事件回调参数

获取错误信息参数

|  |
| --- |
| char\* message.get\_parameter(const char\* key,char\* &value) 回调获取错误信息参 |
| BDUER\_PARAM\_KEY\_ERROR\_DESC |
| char \*errorMsg;  message.get\_parameter(BDUER\_PARAM\_KEY\_ERROR\_DESC, errorMsg); |

获取错误code参数

|  |
| --- |
| char\* message.get\_parameter(const char\* key,char\* &value) 回调获取错误code参数 |
| BDUER\_PARAM\_KEY\_ERROR\_CODE |
| int errorCode;  message.get\_parameter(BDUER\_PARAM\_KEY\_ERROR\_CODE, errorCode);  /\*\*  \* 度秘错误状态  \*/  typedef enum BDuerErrorCode {  BDuerTTSErrorCode = 10001,//TTS错误  BDuerRegisterErrorCode, //注册错误  BDuerWakeupErrorCode,//唤醒  BDuerASRErrorCode,//语音识别  BDuerSDKErrorCode,//SDK内部错误  BDuerDUMIServerErrorCode,//度秘后台  BDuerSDKFinishCode, //SDK一次交互成功完成  } BDuerErrorCode; |

# 3、音箱speaker设备相关

## 3.1 第三方需要提供的speaker设备相关信息

|  |  |
| --- | --- |
| message字段key | 解释说明 |
| BDUER\_DEVICE\_ID | 音箱设备id ,string |
| BDUER\_DEVICE\_NAME | 音箱名称,string |
| BDUER\_DEVICE\_VERSION | 音箱设备版本,string |
| BDUER\_DEVICE\_SOFTWARE\_VERSION | 音箱系统软件版本,string |
| BDUER\_DEVICE\_SYS\_VERSION | 与上面的保持一致,string |
| BDUER\_DEVICE\_MAC\_ADDRESS | 音箱设备mac地址,string |
| BDUER\_DEVICE\_SN | 音箱设备序列号,string |
| BDUER\_DEVICE\_CUID | 与device id保持一致,string |
| BDUER\_DEVICE\_IP | 音箱端ip地址,string |
| BDUER\_DEVICE\_SSID | Wifi的ssid参数, string |
| BDUER\_DEVICE\_OWNER\_USER\_ID | 有家庭地址，这是用户自己填写的，不具有强制的准确性,string |
| BDUER\_DEVICE\_USER\_ID | 针对Lenovo speaker,就是LenovoID,string |
| BDUER\_DEVICE\_VOLUME | 音箱设备当前音量, int: 0 -100 |
| BDUER\_DEVICE\_MUTE | 音箱设备是否静音 int: 1-静音 0-非静音 |
| BDUER\_DEVICE\_CHARGING | 是否充电状态 int: 1- 充电状态 0- 非充电状态 |
| BDUER\_DEVICE\_CHARGE\_LEVEL | 充电进度 int: 0-100 |

## 3.2 调用接口

|  |
| --- |
| 调用示例 |
| BDuerMessage message;  message.set\_name(BDUER\_DEVICE\_INFO);  message.set\_parameter(BDUER\_DEVICE\_ID,”123456”);  message.set\_parameter(BDUER\_DEVICE\_NAME,”客厅的音箱”);  message.set\_parameter(BDUER\_DEVICE\_VERSION,”1.0.0”);  message.set\_parameter(BDUER\_DEVICE\_ SOFTWARE\_VERSION,”2.0.0”);  message.set\_parameter(BDUER\_DEVICE\_SYS\_VERSION,”1.1.0”);  message.set\_parameter(BDUER\_DEVICE\_MAC\_ADDRESS,”xxxxx”);  message.set\_parameter(BDUER\_DEVICE\_SN,”XXXXX”);  message.set\_parameter(BDUER\_DEVICE\_CUID,”123456”);  message.set\_parameter(BDUER\_DEVICE\_IP,”127.0.0.1”);  message.set\_parameter(BDUER\_DEVICE\_OWNER\_USER\_ID,”xxxxx”);  message.set\_parameter(BDUER\_DEVICE\_USER\_ID,”lenovoId”);  message.set\_parameter(BDUER\_DEVICE\_VOLUME, 30);  message.set\_parameter(BDUER\_DEVICE\_MUTE,1);  message.set\_parameter(BDUER\_DEVICE\_CHARGING,1);  message.set\_parameter(BDUER\_DEVICE\_CHARGE\_LEVEL,70);  duerSDK->post(message,&req); |
|  |

## 3.3 调用时机

init初始化注册init()后，调用一次；以后每当speaker端上面任何一个信息改变时，调用一次通知SDK .

# 4、SDK设备相关

## 4.1设备提供者

|  |
| --- |
| DeviceProvider |
| public:  //获取单例  static DeviceProvider \*get\_instance();  //添加支持的设备  bool add\_device(DeviceName device);  //删除支持的设备  void remove\_device(DeviceName device);  //删除所有设备  void remove\_all();  //销毁设备管理  void destroy();  //为设备同步状态  void add\_device\_status(DeviceName device,const struct DeviceStatus& device\_status);  //转化设备事件  void get\_device\_event(DeviceName name,const struct DeviceEvent& device\_event,string& event);  //获取设备当前的状态  void get\_device\_status(string& result);  //获取支持的设备信息  void get\_device\_interface(string& result);  //设置用户自定以设备接口  void set\_device\_creater(DeviceCreater \*creater); |

## 4.2 设备定义

|  |
| --- |
| Device |
| public:  Device(char \*name);  Device();  //同步状态函数  virtual void insert\_status(const struct DeviceStatus& device\_status) = 0;  //获取设备状态  virtual void get\_status(string& status) = 0;  //解析事件函数  virtual void parse\_event(const struct DeviceEvent& device\_event,string& event) = 0;  //获取设备名称  virtual char \*get\_name(); |

## 4.3 自定义设备接口

|  |
| --- |
| DeviceCreater |
| public:  //创建用户自定义设备  virtual Device\* createCustomDevice(DeviceName name) = 0; |

# 5、状态同步和事件上传相关

## 5.1、支持device类型

|  |
| --- |
| DeviceName |
| typedef enum DeviceName {  NAME\_AUDIOPLAYER, //播放器设备  NAME\_SPEAKER, //扬声器设备  NAME\_SPEECHSYNTHESIZER, //TTS合成播放设备  NAME\_ALERTS // 闹钟提醒设备  }DeviceName; |
|  |

## 5.2、设备的状态同步

注册的所有设备在每次本地态改变时需要通知到sdk中，状态同步需要在事件上传之前完成。

### 5.2.1、播放器(AudioPlayer)状态

|  |
| --- |
| AudioPlayer 播放器设备 |
| struct AudioPlayer { //播放器device  long offset\_ms; //当前播放的歌曲进度  const char\* token; //当前播放的歌曲id  AudioPlayerState player\_activity; //播放器状态  AudioPlayer() {  token = "\0";  offset\_ms = -1;  player\_activity = AUDIOPLAYER\_IDEL;  }  }; |

AudioPlayer状态

|  |
| --- |
| AudioPlayerState 播放器设备状态 |
| typedef enum AudioPlayerState { //播放器状态  AUDIOPLAYER\_IDEL, //初始化  AUDIOPLAYER\_PLAYING, //正咋播放  AUDIOPLAYER\_PAUSED, //播放暂停  AUDIOPLAYER\_FINISHED, //播放完成  AUDIOPLAYER\_STOPPED // 停止  }AudioPlayerState; |

调用事例

|  |
| --- |
| AudioPlayer 播放器状态同步 |
| struct DeviceStatus status;  status.audioPlayer.player\_activity = AUDIOPLAYER\_PLAYING;  status.audioPlayer.token = token.c\_str();  status.audioPlayer.offset\_ms = offset\_ms;  DeviceProvider::get\_instance()->add\_device\_status(NAME\_AUDIOPLAYER, status); |

### 5.2.2、扬声器(Speaker)状态

|  |
| --- |
| Speaker 扬声器设备 |
| struct Speaker { //扬声器device  int volume; //扬声器当前音量  bool muted; //静音选项  Speaker() {  volume = 0;  muted = false;  }  }; |

调用事例

|  |
| --- |
| Speaker 扬声器状态同步 |
| struct DeviceStatus status;  status.speaker.volume = volume;  status.speaker.muted = false;  DeviceProvider::get\_instance()->add\_device\_status(NAME\_SPEAKER, status); |

### 5.2.3、TTS合成播放(SpeechSynthesizer)状态

|  |
| --- |
| SpeechSynthesizer TTS合成播放设备 |
| struct SpeechSynthesizer { //TTS合成device  const char\* token;  long offset\_ms;  SpeechSynthesizerState player\_activity;  SpeechSynthesizer() {  token = "";  offset\_ms = -1;  player\_activity = SPEECHSYNTHESIZER\_IDEL;  }  }; |

SpeechSynthesizer 状态

|  |
| --- |
| SpeechSynthesizerState TTS合成设备状态 |
| typedef enum SpeechSynthesizerState { //TTS合成deivce状态  SPEECHSYNTHESIZER\_IDEL,  SPEECHSYNTHESIZER\_PLAYING,  SPEECHSYNTHESIZER\_FINISHED  }SpeechSynthesizerState; |

调用事例

|  |
| --- |
| SpeechSynthesizerState TTS合成状态同步 |
| struct DeviceStatus device\_status;  device\_status.speechSynthesizer.player\_activity = SPEECHSYNTHESIZER\_FINISHED;  device\_status.speechSynthesizer.token = token.c\_str();  device\_status.speechSynthesizer.offset\_ms = -1;  DeviceProvider::get\_instance()->add\_device\_status(NAME\_SPEECHSYNTHESIZER, device\_status); |

## 5.3、上传事件类型

上传事件之前需要先同步好设备的状态

事件类型定义

|  |
| --- |
| DeviceEvent 事件结构 |
| struct DeviceEvent {  struct EventHeader header; //事件头  struct AudioPlayerPayload audioPlayerPayload; //播放器事件数据  struct SpeechSynthesizerPayload speechSynthesizerPayload; //tts合成播放事件数据  struct SpeakerPayload speakerPayload; //扬声器事件数据  struct AlertsPayload alertsPayload;// 闹钟提醒事件数据  }; |
| EventHeader 事件头结构 |
| struct EventHeader {  DeviceName name\_space; //设备名称  const char\* name; // 事件名称  const char\* message\_id; //对应的消息id  }; |
| DeviceEventType Device事件类型 |
| typedef enum DeviceEventType { //Device事件类型  AUDIOPLAYER\_PLAYBACK\_STARTED, //播放器开始播放  AUDIOPLAYER\_PLAYBACK\_STOPPED, //播放器停止播放  AUDIOPLAYER\_PLAYBACK\_FINISHED, //播放器播放完成  AUDIOPLAYER\_PROGRESS\_REPORT\_INTERVAL\_ELAPSED, //播放器进度  AUDIOPLAYER\_PLAYBACK\_NEARLY\_FINISHED, //播放完成获取下一首  SPEAKER\_VOLUME\_CHANGED, //扬声器声音变化  SPEAKER\_MUTE\_CHANGED, //扬声器静音变化  SPEECHSYNTHESIZER\_SPEECH\_STARTED, //tts合成开始播放  SPEECHSYNTHESIZER\_SPEECH\_FINISHED , //tts完成播放  SPEECHSYNTHESIZER\_SPEECH\_STOPPED, //tts播放停止  ALERTS\_SET\_ALERT\_SUCCEEDED, //设置alert成功  ALERTS\_SET\_ALERT\_FAILED, // 设置alert失败  ALERTS\_DELETE\_ALERT\_SUCCEEDED, //删除alert成功  ALERTS\_DELETE\_ALERT\_FAILED, // 删除alert失败  ALERTS\_ALERT\_STARTED, // alert开始  ALERTS\_ALERT\_STOPED // alert 结束  }; |

### 5.3.1、播放器(AudioPlayer)事件数据

|  |
| --- |
| AudioPlayerPayload 数据结构 |
| struct AudioPlayerPayload {  const char\* token; //当前播放歌曲唯一id  long offset\_ms; //当前播放歌曲进度  }; |

调用事例：

|  |
| --- |
| AudioPlayer 播放事件上传事例 |
| struct DeviceEvent device\_event;  device\_event.header.name\_space = NAME\_AUDIOPLAYER;  device\_event.header.name = AUDIOPLAYER\_PLAYBACK\_STARTED;  device\_event.header.message\_id = messsage\_id.c\_str();  device\_event.audioPlayerPayload.offset\_ms = offset\_ms;  device\_event.audioPlayerPayload.token = token.c\_str();  string event;  DeviceProvider::get\_instance()->get\_device\_event(NAME\_AUDIOPLAYER,device\_event,event);  int req = 0;  BDuerMessage message;  message.set\_name((char \*) BDUER\_CMD\_SEND\_HTTP\_REQUEST);  message.set\_parameter(BDUER\_PARAM\_KEY\_HTTP\_REQUEST\_TYPE, HTTP\_REQUEST\_TYPE\_EVENT);  message.set\_parameter(BDUER\_PARAM\_KEY\_HTTP\_REQUEST\_BODY, event.c\_str(), len);  duerSDK->post(message, &req); |

### 5.3.2、扬声器(Speaker)事件数据

|  |
| --- |
| SpeakerPayload 数据结构 |
| struct SpeakerPayload { //扬声器device的事件上传内容  int volume;  }; |

调用事例

|  |
| --- |
| Speaker 播放事件上传事例 |
| struct DeviceEvent device\_event;  device\_event.header.name\_space = NAME\_SPEAKER;  device\_event.header.name = SPEAKER\_VOLUME\_CHANGED;  device\_event.header.message\_id = messsage\_id.c\_str();  device\_event.speakerPayload.volume = volume;  string event;  DeviceProvider::get\_instance()->get\_device\_event(NAME\_SPEAKER,device\_event,event);  nt req = 0;  BDuerMessage message;  message.set\_name((char \*) BDUER\_CMD\_SEND\_HTTP\_REQUEST);  message.set\_parameter(BDUER\_PARAM\_KEY\_HTTP\_REQUEST\_TYPE, HTTP\_REQUEST\_TYPE\_EVENT);  message.set\_parameter(BDUER\_PARAM\_KEY\_HTTP\_REQUEST\_BODY, event.c\_str(), len);  duerSDK->post(message, &req); |

### 5.3.3、TTS合成(SpeechSynthesizer)事件数据

|  |
| --- |
| SpeechSynthesizerPayload 数据结构 |
| struct SpeechSynthesizerPayload { //TTS合成device的事件上传内容  const char\* token; //合成的tts的唯一id  }; |

调用事例

|  |
| --- |
| SpeechSynthesizer TTS合成播放事件上传事例 |
| struct DeviceEvent device\_event;  device\_event.header.name\_space = NAME\_SPEECHSYNTHESIZER;  device\_event.header.name = SPEECHSYNTHESIZER\_SPEECH\_STARTED;  device\_event.header.message\_id = messsage\_id.c\_str();  device\_event.speechSynthesizerPayload.token = token.c\_str();  string event;  DeviceProvider::get\_instance()->get\_device\_event(NAME\_SPEECHSYNTHESIZER,device\_event,event);  int req = 0;  BDuerMessage message;  message.set\_name((char \*) BDUER\_CMD\_SEND\_HTTP\_REQUEST);  message.set\_parameter(BDUER\_PARAM\_KEY\_HTTP\_REQUEST\_TYPE, HTTP\_REQUEST\_TYPE\_EVENT);  message.set\_parameter(BDUER\_PARAM\_KEY\_HTTP\_REQUEST\_BODY, event.c\_str(), len);  duerSDK->post(message, &req); |

### 5.3.4、Alerts(闹钟/提醒/定时器)事件数据

|  |
| --- |
| AlertsPayload数据结构 |
| struct AlertsPayload{ //TTS合成device的事件上传内容  const char\* token; //alert的唯一id  }; |

调用事例

|  |
| --- |
| Alerts 事件上传事例 |
| struct DeviceEvent device\_event;  device\_event.header.name\_space = NAME\_ALERTS;  device\_event.header.name = ALERTS\_SET\_ALERT\_SUCCEEDED;  device\_event.header.message\_id = messsage\_id.c\_str();  device\_event.alertsPayload.token = token.c\_str();  string event;  DeviceProvider::get\_instance()->get\_device\_event(NAME\_ALERTS,device\_event,event);  int req = 0;  BDuerMessage message;  message.set\_name((char \*) BDUER\_CMD\_SEND\_HTTP\_REQUEST);  message.set\_parameter(BDUER\_PARAM\_KEY\_HTTP\_REQUEST\_TYPE, HTTP\_REQUEST\_TYPE\_EVENT);  message.set\_parameter(BDUER\_PARAM\_KEY\_HTTP\_REQUEST\_BODY, event.c\_str(), len);  duerSDK->post(message, &req); |