README

工作模式

在init的时候通过config->effect_mode进行配置，支持工作过程中通过hrsc_set_paras进行设置

hrsc_effect_mode_t	说明
HRSC_EFFECT_MODE_NONE	不需要音效处理，只需要根据输入输出配置决定是否需要进行多通道mix和resample
HRSC_EFFECT_MODE_ASR	语音识别模式
HRSC_EFFECT_MODE_VOIP	语音通话模式

事件通知

通过回调函数event_callback(void* cookie, hrsc_event_t event)来通知业务层

hrsc_event_t	说明
HRSC_EVENT_WAKE_NORMAL	唤醒后再下发命令，如: "小爱同学" -唤醒-> "今天天气怎么样"
HRSC_EVENT_WAKE_ONESHOT	唤醒词命令一起下发，如: "小爱同学，今天天气怎么样"
HRSC_EVENT_WAIT_ASR_TIMEOUT	唤醒后 T 时间内没有检测到声音，发送该事件，同时算法模块进入未唤醒状态; 其中 T 通过HRSC_PARAS_TYPE_WAIT_ASR_TIMEOUT设置
HRSC_EVENT_VAD_BEGIN	唤醒后只要检测到ASR数据，发送该事件
HRSC_EVENT_VAD_MID	暂未使用
HRSC_EVENT_VAD_END	ASR数据结束

参数设置

通过hrsc_set_paras(const hrsc_paras_t* para)来动态设置工作参数，参数为key-value的形式

hrsc_paras_type_t	说明
HRSC_PARAS_TYPE_EFFECT_MODE	设置工作模式，比如由语音识别模式切换到voip模式
HRSC_PARAS_TYPE_WAIT_ASR_TIMEOUT	设置等待ASR的超时时间，比如设置为2s，则唤醒后2秒内没有检测到用户命令，发送HRSC_EVENT_WAIT_ASR_TIMEOUT事件通知业务层
HRSC_PARAS_TYPE_VAD_TIMEOUT	设置vad检测超时时间，比如设置为0.3秒，如果连续0.3秒内没有语音输入，则认为命令输入结束;特别说明一点，这个值应该是算法给出的一个经验值
HRSC_PARAS_TYPE_WAKEUP_DATA_SWITCH	这个设置为0后，只需要调用回调event_callback通知唤醒事件，不需要调用回调wakeup_data_callback回传唤醒数据。
HRSC_PARAS_TYPE_VAD_SWITCH	是否开启VAD检测，设置为0后，不需要进行vad检测，通过hrsc_set_status(HRSC_STATUS_ASR_PAUSE),hrsc_set_status(HRSC_STATUS_ASR_RESUME)来控制asr数据的回传;设置为1后，只有当检测

到有语音命令后才需要调用asr_data_callback回传数据。

工作状态

通过hrsc_set_status(hrsc_status_t st)来设置算法模块工作状态

hrsc_status_t	说明
HRSC_STATUS_WAKEUP	设置算法模块为唤醒状态，按键唤醒、调试、多轮会话时使用
HRSC_STATUS_UNWAKEUP	设置算法为未唤醒状态，调试，多轮会话使用

回传数据分包类型

asr数据长度不定，就有了分包回传数据的需要

hrsc_callback_data_type_t	说明
HRSC_CALLBACK_DATA_ALL	一次回传包含全部数据，例如wakeup和short asr
HRSC_CALLBACK_DATA_HEADER	需要分包回传数据的第一包，必须设置hrsc_callback_data_t::buffer::start_time
HRSC_CALLBACK_DATA_MIDDLE	分包回传数据的中间包，可以不设置hrsc_callback_data_t::buffer::start_time,即hrsc_callback_data_t::buffer::start_time = 0
HRSC_CALLBACK_DATA_TAIL	分包回传数据的最后一包，可以不设置hrsc_callback_data_t::buffer::start_time

audio数据结构

需要根据回传数据找到与之对应的原始录音数据段，所以增加时间戳start_time.

hrsc_audio_buffer_t	说明
unsigned int size	数据长度
hrsc_time_stamp_t start_time	第一帧数据的时间，ms
void *raw	buffer 指针

回传数据结构

hrsc_callback_data_t	说明
hrsc_callback_data_type_t type	见 hrsc_callback_data_type_t
hrsc_audio_buffer_t buffer	见 hrsc_audio_buffer_t
float angle	0.0 ～ 360.0， 语音唤醒和语音asr识别角度
float score	本次唤醒的得分

hrsc_sdk.h

```
/*
 * Copyright (c) 2019 horizon.ai.
 *
 * Unpublished copyright. All rights reserved. This material
contains
 * proprietary information that should be used or copied only
within
 * horizon.ai, except with written permission of horizon.ai.
 * @author: Horizon
 * @file: hrsc_sdk.cc
 * @date: 2019-04-24
 * @brief:
 */
#ifndef __HRSC_SDK_H__
#define __HRSC_SDK_H__

#if defined(__cplusplus)
```

```
extern "C"
{
#endif
#define HRSC_SDK_VERSION 3

typedef unsigned long long hrsc_time_stamp_t;

typedef enum {
    HRSC_LOG_LVL_ERROR = 0,
    HRSC_LOG_LVL_WARNING,
    HRSC_LOG_LVL_INFO,
    HRSC_LOG_LVL_DEBUG,
    HRSC_LOG_LVL_TRACE,

    HRSC_LOG_LVL_MAX
} hrsc_log_lvl_t;

typedef enum {
    HRSC_AUDIO_FORMAT_PCM_16_BIT = 0,
    HRSC_AUDIO_FORMAT_PCM_8_BIT,
    HRSC_AUDIO_FORMAT_PCM_24_BIT,
    HRSC_AUDIO_FORMAT_PCM_32_BIT,
} hrsc_audio_format_t;

typedef enum {
    HRSC_EVENT_WAKE_NORMAL = 0,
    HRSC_EVENT_WAKE_ONESHOT,
    HRSC_EVENT_WAIT_ASR_TIMEOUT,
    HRSC_EVENT_VAD_BEGIN,
    HRSC_EVENT_VAD_MID,
    HRSC_EVENT_VAD_END,
} hrsc_event_t;

typedef enum {
    HRSC_STATUS_WAKEUP = 0,
    HRSC_STATUS_UNWAKEUP,

} hrsc_status_t;

typedef enum {
    /**
    * @brief: just channel mix & resample
    */
    HRSC_EFFECT_MODE_NONE = 0,
    /**
    * @brief: voice recognition
    */
    HRSC_EFFECT_MODE_ASR,
```

```
    HRSC_EFFECT_MODE_VOIP,
} hrsc_effect_mode_t;

typedef enum {
    HRSC_PARAS_TYPE_EFFECT_MODE = 0,
    HRSC_PARAS_TYPE_WAIT_ASR_TIMEOUT,
    HRSC_PARAS_TYPE_VAD_TIMEOUT,
    HRSC_PARAS_TYPE_WAKEUP_DATA_SWITCH, //0 disable, 1 enable
    HRSC_PARAS_TYPE_VAD_SWITCH, //0 disable, 1 enable
    HRSC_PARAS_TYPE_PROCESSED_DATA_SWITCH, //0, disable, 1 enable
    HRSC_PARAS_TYPE_VOIP_DATA_SWITCH, //0, disable, 1 enable
} hrsc_paras_type_t;

typedef enum {
    HRSC_CALLBACK_DATA_ALL = 0,
    HRSC_CALLBACK_DATA_HEADER,
    HRSC_CALLBACK_DATA_MIDDLE,
    HRSC_CALLBACK_DATA_TAIL
} hrsc_callback_data_type_t;


typedef struct paras_data_s {
    hrsc_paras_type_t type;
    void *value;
} hrsc_paras_t;


/**
 * @brief: specifies the input or output audio format to be used by
the effect module
 */
typedef struct {
    unsigned int samplingRate;
    unsigned int channels;
    hrsc_audio_format_t format;
} hrsc_audio_config_t;

typedef struct {
    /**
    * @brief: number of bytes in buffer
    */
    unsigned int size;
    /**
    * @brief: ms, timestamp of first frame
    */
    hrsc_time_stamp_t start_time;
    union {
        /**
```

```
         * @brief: raw pointer to start of buffer, get buffer by
hrsc_get_buffer
         */
        void *raw;
        unsigned int *s32;
        unsigned short *s16;
        unsigned char *u8;
    };
} hrsc_audio_buffer_t;

typedef struct callback_data_s {
    hrsc_callback_data_type_t type;
    hrsc_audio_buffer_t buffer;
    float angle;

    /**
    * @brief: just for wakeup, the score of wakup
    */
    float score;

    /**
     * @brief wakeup for asr recognize result
     */
    char *result;

    /**
     * @brief the length of result
     */
    unsigned int result_len;
} hrsc_callback_data_t;

/**
 * @brief: is used to configure audio parameters and callback
functions.
 */
typedef struct {

    /**
     * @brief : input auido format
     */
    hrsc_audio_config_t inputCfg;
    /**
     * @brief : target output format
     */
    hrsc_audio_config_t outputCfg;
    /**
     * @brief : channel index of reference
     */
```

```
    unsigned char ref_ch_index;
    /**
     * @brief : ms, pcm data before wakeup word
     */
    unsigned int wakeup_prefix;
    /**
     * @brief : ms, pcm data after wakeup word
     */
    unsigned int wakeup_suffix;
    /**
    * @brief : ms, how much time for waitting asr data after wakeup
    */
    unsigned int wait_asr_timeout;
    /**
     * @brief : ms, threshold time for judgment of vad end
     */
    unsigned int vad_timeout;
    /**
     * @brief : voice active detect switch, 0 disable, 1 enable
     */
    unsigned int vad_switch;
    /**
     * @brief : 0: do not send wakeup data to user, 1: send
     */
    unsigned int wakeup_data_switch;
    /**
     * @brief : 0: do not send processed data to user, 1: send
     */
    unsigned int processed_data_switch;
    /**
      * @brief : 0.0 ~ 1.0, do not send wakeup event when score <
target_score
      */
    float target_score;
    /**
     * @brief : work mode, see hrsc_effect_mode_t
     */
    hrsc_effect_mode_t effect_mode;
    /**
     * @brief : config file path if needed, e.g.,
"/vendor/etc/hrsc_cfg.cfg"
     */
    const char *cfg_file_path;
    /**
     * @brief : private data pointer
     */
    void *priv;
```

```
    /**
    * @brief log callback
    */
    void (*log_callback)(hrsc_log_lvl_t lvl, const char *tag, const
char *format, ...);

    /**
     * @brief notify user when a new event happen
     * @param cookie, hrsc_effect_config_t->priv
     * @param event, see hrsc_event_t
     */
    void (*event_callback)(const void *cookie, hrsc_event_t event);

    /**
     * @brief notify user the sound zone when wake up
     * @param cookie
     * @param type
     */
//  void (*zone_callback)(const void *cookie, SoundZoneType type);

    /**
     * @brief send the wake up data to user
     * @param cookie, hrsc_effect_config_t->priv
     * @param data, see hrsc_callback_data_t
     * @param keyword_index: wakeup keyword index, ignore if just
support one keyword
     */
    void (*wakeup_data_callback)(const void *cookie,
                                 const hrsc_callback_data_t *data,
                                 const int keyword_index);

    /**
     * @brief asr data callback handler
     * @param cookie, hrsc_effect_config_t->priv
     * @param data, see hrsc_callback_data_t
     */
    void (*asr_data_callback)(const void *cookie, const
hrsc_callback_data_t *data);

    /**
     * @brief voip data callback handler
     * @param cookie, hrsc_effect_config_t->priv
     * @param data, see hrsc_callback_data_t
     */
    void (*voip_data_callback)(const void *cookie, const
hrsc_callback_data_t *data);

    /**
```

```
     * @brief send processed data to user
     * @param cookie, hrsc_effect_config_t->priv
     * @param data, see hrsc_callback_data_t
     */
    void (*processed_data_callback)(const void *cookie, const
hrsc_callback_data_t *data);
} hrsc_effect_config_t;


/**
 * @brief Initialize voice preprocess module
 * @param config: see hrsc_effect_config_t
 * @return 0 if successful, error code otherwise
 */
int hrsc_init(const hrsc_effect_config_t *config);

/**
 * @brief Start voice preprocess module
 * @return 0 if successful, error code otherwise
 */
int hrsc_start();

/**
 * @brief Stop voice preprocess module
 * @return 0 if successful, error code otherwise
 */
int hrsc_stop();

/**
 * @brief Release the sources of voice preprocess module
 * @return 0 if successful, error code otherwise
 */
int hrsc_release();

/**
 * @brief set status, e.g., user cancle the wakeup
 * @param st: see hrsc_status_t
 */
void hrsc_set_status(hrsc_status_t st);

/**
 * @brief get hrsc status
 * @return see hrsc_status_t
 */
hrsc_status_t hrsc_get_status(void);

/**
 * @brief send the record data to voice process module
```

```
 * @param input: record data buffer
 */
void hrsc_process(hrsc_audio_buffer_t *input);

/**
 * @brief set parameters
 * @param para: see hrsc_paras_t
 * @return 0 if successfull, error code otherwise
 */
int hrsc_set_paras(const hrsc_paras_t *para);

/**
 * @brief get parameters
 * @param type: see hrsc_paras_type_t
 * @param value:
 */
void hrsc_get_paras(hrsc_paras_type_t type, void *value);

#if HRSC_SDK_VERSION > 1

/**
 * @brief get hrsc sdk version index_sequence_for
 * @return: string of version info which contains factory info and
version info, size < 128
 */
const char *hrsc_get_version_info(void);

#endif
#if defined(__cplusplus)
}
#endif
#endif /*__HRSC_SDK_H__*/
```