Bluetooth SDK Documentation

This is the API Reference Manual for the A98 Bluetooth SDK.

Quick Start

This section shows you how to use the Bluetooth SDK to write a A2DP SNK/SRC application.

Initializing The Library

To initialize the library, use the function linkplay_bt_server_init(). The functions takes a struct of type linkplay_bt_config as a parameter.

One of the required fields of the struct bas_server_init_call_back, the application provides a callback function that start BSA server.

Here is a simplified example of using the linkplay_bt_server_init() function:

```
static void bt_start_bsa_server(int enable)
{

        if(enable) {
        system("echo 0 > /sys/class/rfkill/rfkill0/state");
        system("pkill bsa_server");
        system("echo 1 > /sys/class/rfkill/rfkill0/state");

        //AP6355 use bcm4345 firmware
        // system("bsa_server -d /dev/ttyS1 -p /etc/bluetooth/BCM4345C0.hcd -r13 -u /tmp/ > /dev/null &");
        //AP6212A use bcm43430 frimware
        system("bsa_server -d /dev/ttyS1 -p /etc/bluetooth/BCM43430.hcd -r13 -u /tmp/ > /dev/null &");
        } else {
        system("echo 0 > /sys/class/rfkill/rfkill0/state");
        system("pkill bsa_server");
        }
        return 0;

}

int main(int argc, char **argv)

{

        BtError bt_err;
        linkplay_bt_config bt_config = {0};

        strcpy(bt_config.bt_name, "linkplaybt_demo");
        bt_config.bas_server_init_call_back = bt_start_bsa_server;

        bt_err = linkplay_bt_server_init(&bt_config);
        if(bt_err != kBtErrorOk) {
        printf("linkplay_bt_server_init error \r\n");
        exit(1);
        }

...

}
```

The Main Event Loop

In order to do work, the library function linkplay_bluetooth_event_dump() must be called periodically. This is where the library performs all asynchronous operations, and this is where it invokes the callback functions that the application registers (such as the error callback that is specified in linkplay_bluetooth_registerEventCallbacks).

```
...

linkplay_bluetooth_registerEventCallbacks(bluetooth_event_callback, &g_bt_contex_test);

callback.pcm_start_call_fn = audio_render_pcm_open;
callback.pcm_write_data_fn = audio_render_pcm_write_data;

linkplay_bluetooth_registerAudioRenderCallBack(callback);

bt_load_history();

g_bt_contex_test.source_handle = bt_a2dp_source_init_demo_play();

while(1) {
bt_err = linkplay_bluetooth_event_dump(timeoutms);

if(bt_err == kBtErrorInitFailed) {
printf("bt server init error \r\n");
exit(1);
} else if(bt_err == KBtErrorInitProcessing) {
timeoutms = 50;
continue;
} else if(bt_err == kBtErrorOk) {
timeoutms = -1;
}

}
```

There are two major ways to write the main event loop:

- Call linkplay_bluetooth_event_dump() from the application's main event loop
- Call linkplay_bluetooth_event_dump() from a separate thread and communicate with the application's main thread

In both cases, you must make sure that you never invoke Bluetooth APIs from different threads at the same time. The Bluetooth SDK is **not** thre ad-safe.

Writing Callbacks

The application should not perform time-consuming tasks in linkplay_bluetooth_registerEventCallbacks(). Try to return from the callback as quickly as possible. If a time-consuming operation needs to be performed as a reaction to an event, the callback should trigger an asynchronous task.

Playing Audio

it is need to implement the callback linkplay_bluetooth_registerAudioRenderCallBack().

The following example playing audio by ALSA.

```
void *audio_render_pcm_open(UINT8 format, UINT16 sample_rate, UINT8 num_channel, UINT8 bit_per_sample)
{

        int status;
        printf("%s %d \r\n", __FUNCTION__, __LINE__);
        if(g_last_format == format && g_last_sample_rate == sample_rate
        && g_last_channel == num_channel && g_last_bitspersample == bit_per_sample
        && alsa_handle) {
        printf("pcm_open the same param, do nothing \r\n");
        return (void *)alsa_handle;
        }
        /* If ALSA PCM driver was already open => close it */
        if(alsa_handle != NULL) {
        printf("ALSA driver close ++++ \r\n");
        snd_pcm_close(alsa_handle);
        alsa_handle = NULL;
        printf("ALSA driver close ---- \r\n");
        }

        /* Open ALSA driver */
        printf("ALSA driver open ++++ \r\n");
        status = snd_pcm_open(&alsa_handle, "default", SND_PCM_STREAM_PLAYBACK, SND_PCM_NONBLOCK);
        if(status < 0) {
        printf("snd_pcm_open failed: %s", snd_strerror(status));
        } else {
        /* Configure ALSA driver with PCM parameters */
        printf("ALSA driver set params ++++ \r\n");
        status = _snd_pcm_set_params(alsa_handle, format,
        SND_PCM_ACCESS_RW_INTERLEAVED,
        num_channel,
        sample_rate, 1, 500000);/* 0.5sec */

        if(status < 0) {
        printf("snd_pcm_set_params failed: %s \r\n", snd_strerror(status));
        } else {
        g_last_format = format;
        g_last_channel = num_channel ;
        g_last_bitspersample = bit_per_sample;
        g_last_sample_rate = sample_rate;

        }

}

return (void *)alsa_handle;

}
void audio_render_pcm_write_data(void *context, int connection, UINT8 *p_buffer, int frames)
{

        snd_pcm_sframes_t alsa_frames;
        // printf("%s %d %d\r\n",__FUNCTION__,__LINE__,frames);

        if(alsa_handle != NULL && p_buffer && frames > 0) {
        alsa_frames = snd_pcm_writei(alsa_handle, p_buffer, frames);
        // printf("alsa_frames %d \n", alsa_frames);
        if(alsa_frames < 0) {
        alsa_frames = snd_pcm_recover(alsa_handle, alsa_frames, 0);
        }

        if(alsa_frames < 0) {
        // printf(" snd_pcm_writei failed %s\n", snd_strerror(alsa_frames));
        }
        }

        return ;

}



int main(int argc, char **argv)

{

        pcm_render_call_back callback;
        callback.pcm_start_call_fn = audio_render_pcm_open;
        callback.pcm_write_data_fn = audio_render_pcm_write_data;

        linkplay_bluetooth_registerAudioRenderCallBack(callback);

        ...

}
```

A2DP Source

The bluetooth SDK support A2DP Source profile,which was able to transmit stream to other Bluetooth speaker.

To start a stream, use the function linkplay_bluetooth_source_start_stream() , The functions takes a struct of type tLINKPLAY_AV_MEDIA_FEED_CFG_PCM as a parameter.

The following example shows how to do this:

```
tLINKPLAY_AV_MEDIA_FEED_CFG_PCM pcm_config;

pcm_config.sampling_freq = 44100;
pcm_config.num_channel = 2;
pcm_config.bit_per_sample = 16;

linkplay_bluetooth_source_start_stream(&pcm_config);
```

To send the stream, use the function linkplay_bluetooth_source_send_stream().

To stop the stream, use the function linkplay_bluetooth_stop_stream().

NoteSee the example in the SDK package for the full code.

In the example code, receive the pcm data and write into FIFO in ALSA library, start a thread to receive FIFO data and send it out by linkplay_bluetooth_source_send_stream().

GATT

The Bluetooth SDK included server APIs for users to create GATT server,

You must register a GATT service first ,use the function linkplay_bluetooth_ble_server_register() to register server app,

 Using the following API for GATT service.

/* Description: create service*/ linkplay_bluetooth_ble_server_create_service()
/* Description: Add character to service*/ linkplay_bluetooth_ble_server_add_character()
/* Description :Configure the BLE advertisng data*/ linkplay_bluetooth_set_ble_adv_data()
/* Description: Configure the BLE Advertisement interval*/ linkplay_bluetooth_ble_set_adv_param()
/* Description:Send indication to client*/ linkplay_ble_server_send_indication()