```
#include <inttypes.h>
//#define DEBUG1
//#define DEBUG2
//#define DEBUG3
#define NO_PACK
#define PACK_LITTLE_ENDIAN

#include <aci.h>
#include "pack_lib.h"
#include "utilities.h"

#include "ancs_base.h"
#include "ancs_data_source.h"
#include "ancs_notification_list.h"
#include "ancs_notification_source.h"


extern boolean command_send_enable;
extern unsigned long last_command_send;
static ancs_parsing_env_t ancs_parsing_env;

 ancs_notification_t* ancs_data_source_parser(const uint8_t* buffer) {
    debug_println(F("Notification Attribute Details"));
    debug_println(F("[ANCS DS] ancs_data_source_hook()"));

    debug2_print(F("[ANCS DS]  -> Buffer received: "));
    #ifdef DEBUG2
    for (uint8_t i=0; i<ANCS_DATA_LEN; ++i)
        serial_print_char(buffer[i]);
    debug_println();
    #endif

    uint8_t cmd;
    uint32_t nid;
    uint8_t aid;
    uint16_t len;

    ancs_notification_t* notif = NULL;

    // unpack command and uid
    if (ancs_parsing_env.ahead == 0) {
        unpack(buffer, "BIBH", &cmd, &nid, &aid, &len);
        ancs_parsing_env.nid = nid;
        ancs_parsing_env.aid = aid;
        ancs_parsing_env.len = len;
        debug2_print(F("[ANCS DS]  -> COMMAND: "));
        debug2_println(cmd, HEX);
```

```
        debug2_print(F("[ANCS DS]  -> NOTIFICATION: "));
        debug2_println(nid, DEC);
        debug2_print(F("[ANCS DS]  -> ATTRIBUTE: "));
        debug2_println(aid, HEX);
        debug2_print(F("[ANCS DS]  -> DATA LENGTH: "));
        debug2_println(len, DEC);
        if (len < ((ANCS_DATA_LEN - ANCS_HEADER_LEN) - ANCS_ATTR_REQ_LEN))
{
            debug2_println(F("[ANCS DS]  -> All data is in this datagram"));
            debug2_print(F("[ANCS DS]  -> DATA: "));
            ancs_parsing_env.buffer = (uint8_t*)malloc(len+1);
            buffer = buffer + ANCS_HEADER_LEN + ANCS_ATTR_REQ_LEN;
            for (uint8_t i=0; i<len; ++i) {
               #ifdef DEBUG2
               serial_print_char(buffer[i]);
               #endif
               ancs_parsing_env.buffer[i] = buffer[i];
            }
            debug2_println();
            ancs_parsing_env.buffer[len] = '\0';
            ancs_parsing_env.index = 0;
            ancs_parsing_env.ahead = 0;

            command_send_enable = true;
            debug2_println(F("[ANCS DS] Parsing is over!"));
            Serial.print(F("[ANCS DS] Data received: "));
            Serial.println((char*)ancs_parsing_env.buffer);
            notif =  ancs_cache_attribute(ancs_parsing_env.nid,
                           ancs_parsing_env.aid,
                           (char*)ancs_parsing_env.buffer,
                           ancs_parsing_env.len);
            free(ancs_parsing_env.buffer);
        } else {
            debug2_print(F("[ANCS DS]  -> Data continuing in next datagram Len:"));
            debug2_print(len);
            ancs_parsing_env.buffer = (uint8_t*)malloc(len+1);
            ancs_parsing_env.ahead = len-ANCS_FIRST_DATA_LEN;
            buffer = buffer + ANCS_HEADER_LEN + ANCS_ATTR_REQ_LEN;
            debug2_print(F("[ANCS DS]  -> DATA: "));
            for (uint8_t i=0; i<ANCS_FIRST_DATA_LEN; ++i) {
               #ifdef DEBUG2
               serial_print_char(buffer[i]);
               #endif
               ancs_parsing_env.buffer[i] = buffer[i];
            }
            debug2_println();
            ancs_parsing_env.index = ANCS_FIRST_DATA_LEN;
```

```
            debug2_println(F("[ANCS DS] There's more to come!"));
        }
    } else {
        if (ancs_parsing_env.ahead < ANCS_DATA_LEN) {
            debug2_println(F("[ANCS DS]  -> All data left is in this datagram"));
            debug2_print(F("[ANCS DS]  -> BUFFER IDX: "));
            debug2_println(ancs_parsing_env.index, DEC);
            debug2_print(F("[ANCS DS]  -> DATA: "));
            for (uint8_t i=0; i<ancs_parsing_env.ahead; ++i) {
                #ifdef DEBUG2
                serial_print_char(buffer[i]);
                #endif
                ancs_parsing_env.buffer[ancs_parsing_env.index+i] = buffer[i];
            }
            debug2_println();
            ancs_parsing_env.buffer[ancs_parsing_env.index+ancs_parsing_env.ahead] =
'\0';
            ancs_parsing_env.index = 0;
            ancs_parsing_env.ahead = 0;
            command_send_enable = true;
            debug2_println(F("[ANCS DS] Parsing is over!"));
            Serial.print(F("[ANCS DS] Data received: "));
            Serial.println((char*)ancs_parsing_env.buffer);
            notif = ancs_cache_attribute(ancs_parsing_env.nid,
                                ancs_parsing_env.aid,
                                (char*)ancs_parsing_env.buffer,
                                ancs_parsing_env.len);
            free(ancs_parsing_env.buffer);
        } else {
            debug2_println(F("[ANCS DS]  -> Data continuing in next datagram"));
            debug2_print(F("[ANCS DS]  -> BUFFER IDX: "));
            debug2_println(ancs_parsing_env.index, DEC);
            debug2_print(F("[ANCS DS]  -> DATA: "));
            for (uint8_t i=0; i<ANCS_DATA_LEN; ++i) {
                #ifdef DEBUG3
                serial_print_char(buffer[i]);
                #endif
                ancs_parsing_env.buffer[ancs_parsing_env.index+i] = buffer[i];
            }
            debug2_println();
            ancs_parsing_env.index += ANCS_DATA_LEN;
            ancs_parsing_env.ahead = ancs_parsing_env.ahead - ANCS_DATA_LEN;
            debug2_println(F("[ANCS DS] There's more to come!"));
        }
    }
    return notif;
}
```

```
extern void ancs_notifications_use_hook(ancs_notification_t* notif);
void ancs_notification_validation() {
    ancs_notification_t* notif = ancs_notification_list_pop();
    // Serial.print(F("ancs_notification_validation: "));
    // Serial.print(F("["));
    // if ((notif->flags & ANCS_EVT_FLAG_SILENT) == ANCS_EVT_FLAG_SILENT)
    //     Serial.print(F("-"));
    // else if ((notif->flags & ANCS_EVT_FLAG_IMPORTANT) ==
ANCS_EVT_FLAG_IMPORTANT)
    //     Serial.print(F("!"));
    // else
    //     Serial.print(F(" "));
    // Serial.print(F("]"));
    // switch (notif->category) {
    //     case ANCS_CATEGORY_OTHER:
    //         Serial.println(F("Other"));
    //         break;
    //     case ANCS_CATEGORY_INCOMING_CALL:
    //         Serial.println(F("Incoming call"));
    //         break;
    //     case ANCS_CATEGORY_MISSED_CALL:
    //         Serial.println(F("Missed call"));
    //         break;
    //     case ANCS_CATEGORY_VOICEMAIL:
    //         Serial.println(F("Voicemail"));
    //         break;
    //     case ANCS_CATEGORY_SOCIAL:
    //         Serial.println(F("Social"));
    //         break;
    //     case ANCS_CATEGORY_SCHEDULE:
    //         Serial.println(F("Schedule"));
    //         break;
    //     case ANCS_CATEGORY_EMAIL:
    //         Serial.println(F("Email"));
    //         break;
    //     case ANCS_CATEGORY_NEWS:
    //         Serial.println(F("News"));
    //         break;
    //     case ANCS_CATEGORY_HEALTH_FITNESS:
    //         Serial.println(F("Health & Fitness"));
    //         break;
    //     case ANCS_CATEGORY_BUSINESS_FINANCE:
    //         Serial.println(F("Business & Finance"));
    //         break;
    //     case ANCS_CATEGORY_LOCATION:
    //         Serial.println(F("Location"));
```

```
    //         break;
    //     case ANCS_CATEGORY_ENTERTAINMENT:
    //         Serial.println(F("Entertainment"));
    //         break;
    // }
    // Serial.print(F(": "));
    // Serial.println(notif->title);
}

ancs_notification_t* ancs_cache_attribute(uint32_t nid, uint8_t aid, const char* buffer,
uint16_t len) {
    char* datetime;
    ancs_notification_t* notif = ancs_notification_list_get(nid);
    debug3_print(F("ancs_cache_attribute("));
    debug3_print(nid, DEC);
    debug3_print(F(", 0x"));
    debug3_print(aid, HEX);
    debug3_print(F(", '"));
    debug3_print(buffer);
    debug3_println(F("')"));
    debug2_print(F(" Notif #"));
    debug2_print(nid, DEC);

    if (notif != NULL) {

    switch (aid) {
        #ifdef ANCS_USE_APP
        case ANCS_NOTIFICATION_ATTRIBUTE_APP_IDENTIFIER:
            debug2_print(F(", App: "));
            strncpy(notif->app, buffer, strlen(buffer));
            break;
#endif
        case ANCS_NOTIFICATION_ATTRIBUTE_TITLE:
            debug2_print(F(", Title: "));
            strncpy(notif->title, buffer, strlen(buffer));
            break;
        case ANCS_NOTIFICATION_ATTRIBUTE_DATE:
            debug_print(F(", Date: "));
            // YYYYMMDDTHHMM
            strncpy(datetime, buffer, 4);
            datetime[4] = '\0';
            notif->time.Y = atoi(datetime);
            buffer = buffer+4;
            strncpy(datetime, buffer, 2);
            datetime[2] = '\0';
            notif->time.M = atoi(datetime);
            buffer = buffer+2;
```

```
            strncpy(datetime, buffer, 2);
            datetime[2] = '\0';
            notif->time.D = atoi(datetime);
            buffer = buffer+3;
            strncpy(datetime, buffer, 2);
            datetime[2] = '\0';
            notif->time.h = atoi(datetime);
            buffer = buffer+2;
            strncpy(datetime, buffer, 2);
            datetime[2] = '\0';
            notif->time.m = atoi(datetime);
              buffer = buffer+2;
              strncpy(datetime, buffer, 2);
              datetime[2] = '\0';
              notif->time.s = atoi(datetime);
            free(datetime);
            break;
        case ANCS_NOTIFICATION_ATTRIBUTE_MESSAGE_SIZE:
            debug_print(F(", Msglen: "));
            notif->msg_len = atoi(buffer);
            break;
        #ifdef ANCS_USE_SUBTITLE
        case ANCS_NOTIFICATION_ATTRIBUTE_SUBTITLE:
            debug_print(F(", SubTitle: "));
            strncpy(notif->subtitle, buffer, strlen(buffer));
            break;
        #endif
        case ANCS_NOTIFICATION_ATTRIBUTE_MESSAGE:
            debug_print(F(", Message: "));
            strncpy(notif->message, buffer, strlen(buffer));
            debug_println(buffer);
            return notif;

            break;
        default:
            debug_print(F(", Attribute unknown 0x"));
            debug_print(aid, HEX);
            debug_print(F(": "));
    }
    } else {
        debug_println(F("ERROR: Notification not found in the Cache"));
    }
    debug_println(buffer);
    return NULL;
}
```