

Title	<b>UPS</b>
No.	

Rev.	Date	By	History
0.0.1	2003-4-22	UIS	Initial draft
0.0.2	2003-0502	KCY	revision

1.	.....	3
2.	.....	3
2.1.	ups_mon.c.....	3
2.2.	ups_app.c.....	4
2.3.	sena_cmd.h.....	8
2.4.	sena_cmd.c.....	9
2.5.	sena_util.h.....	9
2.6.	sena_util.c.....	10
2.7.	Makefile.....	24



```

if(queue_init(&upsq, 2048) < 0){
    fprintf(stderr, "queue init failed.. exiting..\n");
    exit(0);
}

// open serial device
if(ups_ser_open(<0){
    fprintf(stderr, "Can't open serial device.. exiting..\n");
    exit(0);
}
}

void ups_mon_sighandler(int sig){
    switch(sig){
        case SIGINT:
        case SIGTERM:
            ups_ser_close();
            finalize_upsdata();
            queue_free(&upsq);
            exit(0);
        case SIGCHLD:
            {
                int status;
                pid_t pid;

                for(;;){
                    pid = waitpid(-1, &status, WNOHANG);
                    if (pid <= 0)
                        break;
                }
            }
            break;
    }
}

extern void start_ups_app();

main(){
    printf("Starting ups_mon.. \r\n");
    if(!fork()) {
        init_ups_mon(true);
        start_ups_app();
    }
}

```

## 2.2. ups\_app.c

```

handle_upsmsg, catch_all_async_traps, UPS
catch_response, UPS
poll_device,
sena_cmd.c sena_util.c

```

```

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include "ups_ser.h"
#include "libups_data.h"
#include "ups_data.h"
#include "ups_msgq.h"
#include "strqueue.h"

```

```

// implementations specific includes..
#include "sena_cmd.h"
#include "sena_util.h"

#define UPS_TIME_INTERVAL 300000 // microsec
#define UPS_POLL_INTERVAL 1000000 // microsec
#define USER_APP_VER "Sena ups_mon v1.0"

extern queue upsq;
static int send_command(char *cmd);
static int read_buffer(int lastcmd);

// handle messages from the web or SNMP
int handle_upsmsg(int command, long val, char *strval){
    // send any set command to the UPS
    char cmdstr[100] = {0, };
    int lastcmd;

    // get set command from sena_util
    if(get_set_command(cmdstr, command, val, strval) != -1)
        send_command(cmdstr);

    // confirm set result
    bzero(&cmdstr[0], sizeof(cmdstr));
    lastcmd = get_poll_command(cmdstr, command, CONFIRM);
    if(lastcmd != -1){
        usleep(UPS_TIME_INTERVAL);
        send_command(cmdstr);
        usleep(UPS_TIME_INTERVAL);
        read_buffer(lastcmd);
    }
}

// catch async alarms #ALM
// #ALM009PRE1,3END
static int catch_all_async_traps(){
#define TRAP_HEADER "#ALM"
    char *tp = NULL;

    while((tp = strstr(upsq.buf, TRAP_HEADER))    {
        int start;
        int traplen;
        char trapstr[256] = {0, };
        char *ptmp;

        // async trap detected
        ptmp = tp;
        start = tp - upsq.buf;
        ptmp += strlen(TRAP_HEADER);

        if(strlen(ptmp) >= 3){ // length
            if((ptmp[0]>='0' && ptmp[0]<='9')&&
                (ptmp[1]>='0' && ptmp[1]<='9')&&(ptmp[2]>='0' && ptmp[2]<='9')){
                char temp[4] = {0, };
                strncpy(temp, ptmp, 3);
                ptmp += 3;
                traplen = atoi(temp);
                if(strlen(ptmp) >= traplen){
                    if(traplen < sizeof(trapstr)){
                        queue_read(&upsq, trapstr, start,
                            traplen+strlen(TRAP_HEADER)+3);
                        parse_trap(trapstr);
                        continue;
                    }
                }
            }
        }
    }
}

// delete the trap header so that the broken trap will be ignored next time

```

```

        queue_delete(&upsq, start, strlen(TRAP_HEADER));
    }
}

// catch response #RSP
// #RSP043UIDSena,DEMO,1.0.15,2.0.0,UPSLink,RouterEND
static int catch_response(int lastcmd){
    #define RESP_HEADER "#RSP"
    char *tr = NULL;
    int start;
    int resplen;
    char respstr[256] = {0, };
    char *ptmp;

    if(tr = strstr(upsq.buf, RESP_HEADER)){
        // response detected
        ptmp = tr;
        start = tr - upsq.buf;
        ptmp += strlen(RESP_HEADER);

        if(strlen(ptmp) >= 3){ // length
            if((ptmp[0]>='0' && ptmp[0]<='9')&&
                (ptmp[1]>='0' && ptmp[1]<='9')&&(ptmp[2]>='0' && ptmp[2]<='9')){
                char temp[4] = {0, };
                strncpy(temp, ptmp, 3);
                ptmp += 3;
                resplen = atoi(temp);
                if(strlen(ptmp) >= resplen){
                    if(resplen < sizeof(respstr)){
                        queue_read(&upsq, respstr, start,
                            resplen+strlen(RESP_HEADER)+3);
                        parse_response(lastcmd, respstr);
                        return 0;
                    }
                }
            }
        }
        // delete the response header so that the broken responses
        // will be ignored next time
        queue_delete(&upsq, start, strlen(RESP_HEADER));
    }
}

static int read_buffer(int lastcmd){
    int len;
    int totallen = 0;
    char buff[2048] = {0, };

    // read until there is no data remaining on the buffer
    do{
        len = ups_ser_read(buff, sizeof(buff));
        if(len > 0){
            // add read data to queue.
            queue_write (&upsq, buff, len);
            totallen += len;
        }
    }while(len > 0);

    if(totallen == 0){
        // sleep for a wile
        usleep(600000);
        // retry once more
        do{
            len = ups_ser_read(buff, sizeof(buff));
            if(len > 0){
                // add read data to queue.
                queue_write (&upsq, buff, len);
                totallen += len;
            }
        }while(len > 0);
    }
}

```

```

    }
    }while(len > 0);

    if(totalLEN == 0){
        _UPS_DATA_ID id;
        reset_upsdata();
        read_ups_identification(&id);
        strcpy(id.app_sw_ver, USER_APP_VER);
        save_ups_identification(&id);
        set_alarm(_ALARM_COMMLOST, ALARM_PRESENT, false);
        return -1;
    }
}
set_alarm(_ALARM_COMMLOST, ALARM_RELEASED, false);

// catch async trap if any
catch_all_async_traps();
// read the response and parse it
catch_response(lastcmd);
// clear queue (clear garbage or unknown)
queue_clear(&upsq);
return 0;
}

static int poll_device(){
    char cmdstr[100] = {0, };
    int lastcmd;

    lastcmd = get_poll_command(cmdstr, 0, SCHEDULED);
    if(lastcmd != -1){
        send_command(cmdstr);

        usleep(UPS_TIME_INTERVAL);
        read_buffer(lastcmd);
    }
}

static int send_command(char *cmd){
    int res;
    res = ups_ser_write(cmd, strlen(cmd));
    return res;
}

static void register_usrspec(){
    return;
}

void start_ups_app(){
    pid_t pid;

    // register user-specific alarm and test
    register_usrspec();

    // clear queue initially
    queue_clear(&upsq);

    pid = fork();

    if(pid == -1){
        fprintf(stderr, "process spawn fail. exiting..\r\n");
        exit(0);
    }
    else if(pid != 0){
        // parent process
        _UPS_DATA_ID id;
        read_ups_identification(&id);
        strcpy(id.app_sw_ver, USER_APP_VER);
        save_ups_identification(&id);

```

```

        usleep(500000);

        // poll ups
        while(1){{
            // lock to be thead safe
            lock_upssem();
            // send scheduled command, read response and asynchronous trap
            poll_device();
            // unlock
            unlock_upssem();

            // sleep for a while to yield.
            usleep(UPS_POLL_INTERVAL);
        }
    }
}
else{
    // child process
    // monitor ups messages from the snmpd or web
    while(1){
        int command;
        long val;
        char strval[256] = {0, };

        // this is a blocking function
        read_upsmsgq(&command, &val, strval);

        // lock to be thead safe
        lock_upssem();
        // received a command
        // the handling may be write and read serial.
        handle_upsmsg(command, val, strval);
        // unlock
        unlock_upssem();

        // sleep for a while to yield.
        usleep(UPS_TIME_INTERVAL);
    }
}
}
}

```

### 2.3. sena\_cmd.h

Get            ID            Enum            .

```

#ifndef _SENA_CMD_H
#define _SENA_CMD_H

typedef enum {SCHEDULED=1, CONFIRM=2} POLL_TYPE;

// Get Commands
enum {
    UID = 0,
    BAT = 1,
    INP = 2,
    OUT = 3,
    BYP = 4,
    ALM = 5,
    TST = 6,
    CTR = 7,
    CFG = 8,

    NULLCMD = 9
};

```



```
#endif //_SENA_CMD_H
```

## 2.4. sena\_cmd.c

Get Enum ,  
minor\_com major\_com 2가 .

```
#include <stdlib.h>
#include "sena_cmd.h"

char *poll_cmdstr[] = {
    "#GET006UIDEND",
    "#GET006BATEND",
    "#GET006INPEND",
    "#GET006OUTEND",
    "#GET006BYPEND",
    "#GET006ALMEND",
    "#GET006TSTEND",
    "#GET006CTREND",
    "#GET006CFGEND",

    NULL
};

int minor_com [] = {
    BAT,
    INP,
    OUT,
    BYP,
    ALM,
    CFG,
    NULLCMD
};

int major_com [] = {
    UID,
    BAT,
    INP,
    OUT,
    BYP,
    ALM,
    TST,
    CTR,
    CFG,
    NULLCMD
};
```

## 2.5. sena\_util.h

```
#ifndef _SENA_UTIL_H
#define _SENA_UTIL_H

int parse_trap(char *trapstr);
int parse_response(int lastcmd, char *respstr);
int get_poll_command(char *cmdstr, int cmd, int type);
int get_set_command(char *cmdstr, int cmd, long val, char *strval);

#endif //_SENA_UTIL_H
```

## 2.6. sena\_util.c

```

                                Delimiter
comtok,                                parse_trap, UPS
parse_response, UPS                    get_poll_command,
                                get_set_command   가
                                .
comtok  strtok                        Delimiter
    strtok가                            가 0
                                .
get_poll_command  SCHEDULED  CONFIRM  가                Get
. SCHEDULED                                minor_com 5    major_com
                                OUT  ALM                . CONFIRM
                                .
                                Set                    Get

```

```

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include "libups_data.h"
#include "ups_data.h"
#include "sena_cmd.h"
#include "ups_msgq.h"

// SENA type UPS poll_command set
extern char *poll_cmdstr[];
extern int minor_com[];
extern int major_com[];

// response parsing utility
// get token from a ','-delimited response string.
// this utility do not ignore the zero-length token
static char empty[1] = {0, };
static char tokbase[1024];
static char *tok;
static char *comtok(char *base, char *del){
    int i;
    char *p;
    char *pret;
    if(base) {
        bzero(tokbase, sizeof(tokbase));
        strncpy(tokbase, base, strlen(base));
        if(del) {
            if(strlen(tokbase) == 0) {
                tok = tokbase;
                return NULL;
            }
            for(i=0; i<strlen(del); i++) {
                if(tokbase[0] == del[i]) {
                    tok = tokbase+sizeof(char);
                    return empty;
                }
            }
            p = strchr(tokbase, del[i]);
            if(p) {
                pret = tokbase;
                tok = p+sizeof(char);
                *p = 0x00;
                return pret;
            }
        }
        pret = tokbase;
        tok = tokbase+strlen(tokbase);
    }
}

```

```

        return pret;
    }
    else {
        tok = tokbase;
        return NULL;
    }
}
else {
    if(tok == NULL) return NULL;
    if(del) {
        if(strlen(tok) == 0) {
            return NULL;
        }
        for(i=0; i<strlen(del); i++) {
            if(tok[0] == del[i]) {
                tok = tok+sizeof(char);
                return empty;
            }
        }
        p = strchr(tok, del[i]);
        if(p) {
            pret = tok;
            tok = p+sizeof(char);
            *p = 0x00;
            return pret;
        }
    }
    pret = tok;
    tok = tok+strlen(tok);
    return pret;
}
else {
    return NULL;
}
}
}
// end of response parsing utility
//

int parse_trap(char *trapstr){
    char trapcmd[4] = {0, };
    char trap[256] = {0, };
    strncpy(trapcmd, trapstr, 3);
    strncpy(trap, trapstr+3, strlen(trapstr)-6); // trim "PRE/REL" and "END"
    if(strcmp(trapcmd, "PRE") == 0){
        // alarm present
        char *palarmindex = NULL;
        int alarmindex = 0;
        palarmindex = comtok(trap, ",");
        if(palarmindex && (strlen(palarmindex) > 0)){
            // convert to zero-based alarm index in ups_alarm.h
            alarmindex = atoi(palarmindex)-1+_ALARM_BATTBAD;
            set_alarm(alarmindex, ALARM_PRESENT, false);
            while((palarmindex = comtok(NULL, ",")) != NULL){
                if(strlen(palarmindex) > 0){
                    // convert to zero-based alarm index in ups_alarm.h
                    alarmindex = atoi(palarmindex)-1+_ALARM_BATTBAD;
                    set_alarm(alarmindex, ALARM_PRESENT, false);
                }
            }
        }
    }
    else if(strcmp(trapcmd, "REL") == 0){
        // alarm released
        char *palarmindex = NULL;
        int alarmindex = 0;
        palarmindex = comtok(trap, ",");
        if(palarmindex && (strlen(palarmindex) > 0)){
            // convert to zero-based alarm index in ups_alarm.h

```

```

alarmindex = atoi(palarmindex)-1+_ALARM_BATTBAD;
set_alarm(alarmindex, ALARM_RELEASED, false);
while((palarmindex = comtok(NULL, ",")) != NULL){
    if(strlen(palarmindex) > 0){
        // convert to zero-based alarm index in ups_alarm.h
        alarmindex = atoi(palarmindex)-1+_ALARM_BATTBAD;
        set_alarm(alarmindex, ALARM_RELEASED, false);
    }
}
}
}
}

int parse_response(int lastcmd, char *respstr)
{
    char respcmd[4] = {0, };
    char resp[256] = {0, };
    strncpy(respcmd, respstr, 3);
    strncpy(resp, respstr+3, strlen(respstr)-6); // trim command and "END"
    switch(lastcmd){
        case UID:
            if(strcmp(respcmd, "UID") == 0){
                _UPS_DATA_ID data;
                char *manufacturer;
                char *model;
                char *upsswver;
                char *agentswver;
                char *name;
                char *attdevices;

                read_ups_identification(&data);
                manufacturer = comtok(resp, ",");
                if(manufacturer) {
                    if(strlen(manufacturer) > 0)
                        strncpy(data.manufacturer, manufacturer,
                                sizeof(data.manufacturer)-1);
                    model = comtok(NULL, ",");
                }
                if(model) {
                    if(strlen(model) > 0)
                        strncpy(data.model, model, sizeof(data.model)-1);
                    upsswver = comtok(NULL, ",");
                }
                if(upsswver) {
                    if(strlen(upsswver) > 0)
                        strncpy(data.sw_ver, upsswver, sizeof(data.sw_ver)-1);
                    agentswver = comtok(NULL, ",");
                }
                if(agentswver) {
                    if(strlen(agentswver) > 0)
                        strncpy(data.agent_sw_ver, agentswver,
                                sizeof(data.agent_sw_ver)-1);
                    name = comtok(NULL, ",");
                }
                if(name) {
                    if(strlen(name) > 0)
                        strncpy(data.identname, name, sizeof(data.identname)-1);
                    attdevices = comtok(NULL, ",");
                }
                if(attdevices) {
                    if(strlen(attdevices) > 0)
                        strncpy(data.attdev, attdevices, sizeof(data.attdev)-1);
                }
                save_ups_identification(&data);
            }
            break;
        case BAT:
            // battery

```

```

if(strcmp(respcmd, "BAT") == 0){
    _UPS_DATA_BATTERY data;
    char *status;
    char *secs_onbatt;
    char *est_min;
    char *est_chg;
    char *voltage;
    char *current;
    char *temperature;

    read_ups_battery(&data);
    status = comtok(resp, ",");
    if(status){
        if(strlen(status) > 0){
            int sts;
            sts = atoi(status);
            if(sts == 2){
                set_alarm(_ALARM_LOWBATT, ALARM_RELEASED, false);
                set_alarm(_ALARM_DEPBATT, ALARM_RELEASED, false);
                data.status = BATTSTS_NORMAL;
            }
            else if(sts == 3){
                set_alarm(_ALARM_LOWBATT, ALARM_PRESENT, false);
                set_alarm(_ALARM_DEPBATT, ALARM_RELEASED, false);
                data.status = BATTSTS_LOW;
            }
            else if(sts == 4){
                set_alarm(_ALARM_LOWBATT, ALARM_RELEASED, false);
                set_alarm(_ALARM_DEPBATT, ALARM_PRESENT, false);
                data.status = BATTSTS_DEP;
            }
            else{
                set_alarm(_ALARM_LOWBATT, ALARM_RELEASED, false);
                set_alarm(_ALARM_DEPBATT, ALARM_RELEASED, false);
                data.status = BATTSTS_UNKNOWN;
            }
        }
        secs_onbatt = comtok(NULL, ",");
    }
    if(secs_onbatt) {
        if(strlen(secs_onbatt) > 0)
            data.secs_onbatt= atoi(secs_onbatt);
        est_min = comtok(NULL, ",");
    }
    if(est_min) {
        if(strlen(est_min) > 0)
            data.est_min = atoi(est_min);
        est_chg = comtok(NULL, ",");
    }
    if(est_chg) {
        if(strlen(est_chg) > 0)
            data.est_chg = atoi(est_chg);
        voltage = comtok(NULL, ",");
    }
    if(voltage) {
        if(strlen(voltage) > 0)
            data.voltage = atoi(voltage) * 10;
        current = comtok(NULL, ",");
    }
    if(current) {
        if(strlen(current) > 0)
            data.current = atoi(current) * 10;
        temperature = comtok(NULL, ",");
    }
    if(temperature) {
        if(strlen(temperature) > 0)
            data.temperature= atoi(temperature);
    }
}

```

```

        save_ups_battery(&data);
    }
    break;
case INP:
    // input
    if(strcmp(respcmd, "INP") == 0){
        _UPS_DATA_INPUT data;
        char *bads;
        char *num_lines;
        char *frequency[MAX_PHASE];
        char *voltage[MAX_PHASE];
        char *current[MAX_PHASE];
        char *power[MAX_PHASE];

        int i;
        int linenum = 0;

        read_ups_input(&data);
        bads = comtok(resp, ",");
        if(bads) {
            if(strlen(bads) > 0)
                data.bads = atoi(bads);
            num_lines = comtok(NULL, ",");
        }
        if(num_lines) {
            if(strlen(num_lines) > 0){
                if(atoi(num_lines) <= MAX_PHASE && atoi(num_lines) >= 0){
                    data.num_lines = atoi(num_lines);
                    linenum = data.num_lines;
                }
            }
            frequency[0] = comtok(NULL, ",");
        }
        for(i=0; i<linenum; i++){
            if(frequency[i]) {
                if(strlen(frequency[i]) > 0)
                    data.frequency[i] = atoi(frequency[i]) * 10;
                voltage[i] = comtok(NULL, ",");
            }
            if(voltage[i]) {
                if(strlen(voltage[i]) > 0)
                    data.voltage[i] = atoi(voltage[i]);
                current[i] = comtok(NULL, ",");
            }
            if(current[i]) {
                if(strlen(current[i]) > 0)
                    data.current[i] = atoi(current[i]) * 10;
                power[i] = comtok(NULL, ",");
            }
            if(power[i]) {
                if(strlen(power[i]) > 0)
                    data.power[i] = atoi(power[i]);
                if(i < linenum-1)
                    frequency[i+1] = comtok(NULL, ",");
            }
        }
        save_ups_input(&data);
    }
    break;
case OUT:
    // output
    if(strcmp(respcmd, "OUT") == 0){
        _UPS_DATA_OUTPUT data;
        char *source;
        char *frequency;
        char *num_lines;
        char *voltage[MAX_PHASE];

```

```

char *current[MAX_PHASE];
char *power[MAX_PHASE];
char *load[MAX_PHASE];

int i;
int linenum = 0;

read_ups_output(&data);
source = comtok(resp, ",");
if(source) {
    if(strlen(source) > 0){
        int src;
        src = atoi(source);
        if(src == 1){
            set_alarm(_ALARM_ONBATT, ALARM_RELEASED, false);
            set_alarm(_ALARM_ONBYPASS, ALARM_RELEASED, false);
            data.source = OUTSRC_OTHER;
        }
        else if(src == 2){
            set_alarm(_ALARM_ONBATT, ALARM_RELEASED, false);
            set_alarm(_ALARM_ONBYPASS, ALARM_RELEASED, false);
            data.source = OUTSRC_NONE;
        }
        else if(src == 3){
            set_alarm(_ALARM_ONBATT, ALARM_RELEASED, false);
            set_alarm(_ALARM_ONBYPASS, ALARM_RELEASED, false);
            data.source = OUTSRC_NORMAL;
        }
        else if(src == 4){
            set_alarm(_ALARM_ONBATT, ALARM_RELEASED, false);
            set_alarm(_ALARM_ONBYPASS, ALARM_PRESENT, false);
            data.source = OUTSRC_BYPASS;
        }
        else if(src == 5){
            set_alarm(_ALARM_ONBYPASS, ALARM_RELEASED, false);
            set_alarm(_ALARM_ONBATT, ALARM_PRESENT, false);
            data.source = OUTSRC_BATTERY;
        }
        else if(src == 6){
            set_alarm(_ALARM_ONBATT, ALARM_RELEASED, false);
            set_alarm(_ALARM_ONBYPASS, ALARM_RELEASED, false);
            data.source = OUTSRC_BOOSTER;
        }
        else if(src == 7){
            set_alarm(_ALARM_ONBATT, ALARM_RELEASED, false);
            set_alarm(_ALARM_ONBYPASS, ALARM_RELEASED, false);
            data.source = OUTSRC_REDUCER;
        }
    }
    frequency = comtok(NULL, ",");
}
if(frequency) {
    if(strlen(frequency) > 0)
        data.frequency = atoi(frequency) * 10;
    num_lines = comtok(NULL, ",");
}
if(num_lines) {
    if(strlen(num_lines) > 0){
        if(atoi(num_lines) <= MAX_PHASE && atoi(num_lines) >= 0){
            data.num_lines = atoi(num_lines);
            linenum = data.num_lines;
        }
    }
    voltage[0] = comtok(NULL, ",");
}
for(i=0; i<linenum; i++){
    if(voltage[i]) {
        if(strlen(voltage[i]) > 0)

```

```

        data.voltage[i] = atoi(voltage[i]);
        current[i] = comtok(NULL, ",");
    }
    if(current[i]) {
        if(strlen(current[i]) > 0)
            data.current[i] = atoi(current[i]) * 10;
        power[i] = comtok(NULL, ",");
    }
    if(power[i]) {
        if(strlen(power[i]) > 0)
            data.power[i] = atoi(power[i]);
        load[i] = comtok(NULL, ",");
    }
    if(load[i]) {
        if(strlen(load[i]) > 0)
            data.load[i] = atoi(load[i]);
        if(i < linenum-1)
            voltage[i+1] = comtok(NULL, ",");
    }
    }
    save_ups_output(&data);
}
break;
case BYP:
    // bypass
    if(strcmp(respcmd, "BYP") == 0){
        _UPS_DATA_BYPASS data;
        char *frequency;
        char *num_lines;
        char *voltage[MAX_PHASE];
        char *current[MAX_PHASE];
        char *power[MAX_PHASE];

        int i;
        int linenum = 0;

        read_ups_bypass(&data);
        frequency = comtok(resp, ",");
        if(frequency) {
            if(strlen(frequency) > 0)
                data.frequency = atoi(frequency) * 10;
            num_lines = comtok(NULL, ",");
        }
        if(num_lines) {
            if(strlen(num_lines) > 0){
                if(atoi(num_lines) <= MAX_PHASE && atoi(num_lines) >= 0){
                    data.num_lines = atoi(num_lines);
                    linenum = data.num_lines;
                }
            }
            voltage[0] = comtok(NULL, ",");
        }
        for(i=0; i<linenum; i++){
            if(voltage[i]) {
                if(strlen(voltage[i]) > 0)
                    data.voltage[i] = atoi(voltage[i]);
                current[i] = comtok(NULL, ",");
            }
            if(current[i]) {
                if(strlen(current[i]) > 0)
                    data.current[i] = atoi(current[i]) * 10;
                power[i] = comtok(NULL, ",");
            }
            if(power[i]) {
                if(strlen(power[i]) > 0)
                    data.power[i] = atoi(power[i]);
                if(i < linenum-1)
                    voltage[i+1] = comtok(NULL, ",");
            }
        }
    }

```



```

    }
    }
    save_ups_bypass(&data);
}
break;
case ALM:
    // alarm
    if(strcmp(respcmd, "ALM") == 0){
        int status;
        char *pstatus = NULL;
        int alarmindex = 0;
        pstatus = comtok(resp, ",");
        if(pstatus){
            status = atoi(pstatus);
            if(status == 1)
                set_alarm(alarmindex, ALARM_PRESENT, false);
            else
                set_alarm(alarmindex, ALARM_RELEASED, false);
            alarmindex++;
            while((pstatus = comtok(NULL, ",")) != NULL){
                status = atoi(pstatus);
                if(status == 1)
                    set_alarm(alarmindex, ALARM_PRESENT, false);
                else
                    set_alarm(alarmindex, ALARM_RELEASED, false);
                alarmindex++;
            }
        }
    }
    break;
case TST:
    if(strcmp(respcmd, "TST") == 0){
        char *testid;
        char *summary;
        char *detail;

        testid = comtok(resp, ",");
        if(testid) {
            // do not call "save_ups_testid".
            // save_ups_testid must be called only when performing the test.
            // save_ups_testid is called in get_set_command function of
            // this file.
            // see the manual for details.
            summary = comtok(NULL, ",");
        }
        if(summary) {
            if(strlen(summary) > 0){
                int smry;
                smry = atoi(summary);
                if(smry == 1){
                    set_alarm(_ALARM_TESTINPROG, ALARM_RELEASED, false);
                    save_ups_test_summary(TESTRES_DONEPASS);
                }
                else if(smry == 2){
                    set_alarm(_ALARM_TESTINPROG, ALARM_RELEASED, false);
                    save_ups_test_summary(TESTRES_DONEWARN);
                }
                else if(smry == 3){
                    set_alarm(_ALARM_TESTINPROG, ALARM_RELEASED, false);
                    save_ups_test_summary(TESTRES_DONEERR);
                }
                else if(smry == 4){
                    set_alarm(_ALARM_TESTINPROG, ALARM_RELEASED, false);
                    save_ups_test_summary(TESTRES_ABORTED);
                }
                else if(smry == 5){
                    set_alarm(_ALARM_TESTINPROG, ALARM_PRESENT, false);
                    save_ups_test_summary(TESTRES_INPROG);
                }
            }
        }
    }
}

```

```

        }
        else if(smry == 6){
            set_alarm(_ALARM_TESTINPROG, ALARM_RELEASED, false);
            save_ups_test_summary(TESTRES_NOTEST);
        }
    }
    detail = comtok(NULL, ",");
}
if(detail) {
    if(strlen(detail) > 0)
        save_ups_test_detail(detail);
}
}
break;
case CTR:
if(strcmp(respcmd, "CTR") == 0){
    char *shutdowntype;
    char *shutdownad;
    char *startupad;
    char *rebootwd;
    char *autorestarttype;

    shutdowntype = comtok(resp, ",");
    if(shutdowntype) {
        if(strlen(shutdowntype) > 0){
            int type = atoi(shutdowntype);
            if(type == 1)
                save_ups_control_shutdowntype(SHUTDNTYPE_OUTPUT);
            else if(type == 2)
                save_ups_control_shutdowntype(SHUTDNTYPE_SYSTEM);
        }
        shutdownad = comtok(NULL, ",");
    }
    if(shutdownad) {
        // do not call "save_ups_control_shutdownafterdelay".
        // save_ups_control_shutdownafterdelay must be called only
        // when transmitting this command to the UPS.
        // save_ups_control_shutdownafterdelay is called in
        // get_set_command function of this file.
        // see the manual for details.
        startupad = comtok(NULL, ",");
    }
    if(startupad) {
        // do not call "save_ups_control_startupafterdelay".
        // save_ups_control_startupafterdelay must be called only
        // when transmitting this command to the UPS.
        // save_ups_control_startupafterdelay is called in
        // get_set_command function of this file.
        // see the manual for details.
        rebootwd = comtok(NULL, ",");
    }
    if(rebootwd) {
        // do not call "save_ups_control_rebootwithduration".
        // save_ups_control_rebootwithduration must be called only
        // when transmitting this command to the UPS.
        // save_ups_control_rebootwithduration is called in
        // get_set_command function of this file.
        // see the manual for details.
        autorestarttype = comtok(NULL, ",");
    }
    if(autorestarttype) {
        if(strlen(autorestarttype) > 0){
            int type = atoi(autorestarttype);
            if(type == 1)
                save_ups_control_autorestart(AUTORESTRT_ON);
            else if(type == 2)
                save_ups_control_autorestart(AUTORESTRT_OFF);
        }
    }
}
}

```

```

    }
}
break;
case CFG:
    if(strcmp(respcmd, "CFG") == 0){
        _UPS_DATA_CONFIG cdata;
        char *input_volt;
        char *input_freq;
        char *output_volt;
        char *output_freq;
        char *output_va;
        char *output_power;
        char *low_batt_time;
        char *audible_status;
        char *low_volt_trpt;
        char *high_volt_trpt;

        read_ups_config(&cdata);
        input_volt = comtok(resp, ",");
        if(input_volt) {
            if(strlen(input_volt) > 0)
                cdata.input_volt = atoi(input_volt);
            input_freq = comtok(NULL, ",");
        }
        if(input_freq) {
            if(strlen(input_freq) > 0)
                cdata.input_freq = atoi(input_freq) * 10;
            output_volt = comtok(NULL, ",");
        }
        if(output_volt) {
            if(strlen(output_volt) > 0)
                cdata.output_volt = atoi(output_volt);
            output_freq = comtok(NULL, ",");
        }
        if(output_freq) {
            if(strlen(output_freq) > 0)
                cdata.output_freq = atoi(output_freq) * 10;
            output_va = comtok(NULL, ",");
        }
        if(output_va) {
            if(strlen(output_va) > 0)
                cdata.output_va = atoi(output_va);
            output_power = comtok(NULL, ",");
        }
        if(output_power) {
            if(strlen(output_power) > 0)
                cdata.output_power = atoi(output_power);
            low_batt_time = comtok(NULL, ",");
        }
        if(low_batt_time) {
            if(strlen(low_batt_time) > 0)
                cdata.low_batt_time = atoi(low_batt_time);
            audible_status = comtok(NULL, ",");
        }
        if(audible_status) {
            if(strlen(audible_status) > 0){
                int status = atoi(audible_status);
                if(status == 1)
                    cdata.audible_status = ADBLALM_DISABLED;
                else if(status == 2)
                    cdata.audible_status = ADBLALM_ENABLED;
                else if(status == 3)
                    cdata.audible_status = ADBLALM_MUTED;
            }
            low_volt_trpt = comtok(NULL, ",");
        }
        if(low_volt_trpt) {
            if(strlen(low_volt_trpt) > 0)

```

```

        cdata.low_volt_trpt = atoi(low_volt_trpt);
        high_volt_trpt = comtok(NULL, ",");
    }
    if(high_volt_trpt) {
        if(strlen(high_volt_trpt) > 0)
            cdata.high_volt_trpt = atoi(high_volt_trpt);
    }
    save_ups_config(&cdata);
}
break;
defalut:
break;
}
}

#define MINOR_COM_INT 5
static int curcmd = 0;
static int poll_count = MINOR_COM_INT;
static int sched_cmdidx;

int get_poll_command(char *cmdstr, int cmd, int type){
    int pollcmdidx;
    int *cmdset;
    switch(type){
        case SCHEDULED:
            // insert OUT and ALM between every commands
            if(sched_cmdidx != OUT && sched_cmdidx != ALM){
                pollcmdidx = OUT;
                sched_cmdidx = pollcmdidx;
                break;
            }
            else if(sched_cmdidx == OUT){
                pollcmdidx = ALM;
                sched_cmdidx = pollcmdidx;
                break;
            }
            // normal routine
            if(poll_count == MINOR_COM_INT){
                if(major_com[curcmd] == NULLCMD){
                    // switch to minor command rotation
                    poll_count = 0;
                    curcmd = 0;
                    pollcmdidx = minor_com[curcmd];
                }
                else{
                    pollcmdidx = major_com[curcmd];
                }
            }
            else{
                if(minor_com[curcmd] == NULLCMD){
                    poll_count++;
                    curcmd = 0;
                    if(poll_count == MINOR_COM_INT){
                        // switch to major command rotation
                        pollcmdidx = major_com[curcmd];
                    }
                    else{
                        pollcmdidx = minor_com[curcmd];
                    }
                }
                else{
                    pollcmdidx = minor_com[curcmd];
                }
            }
            curcmd++;
            sched_cmdidx = pollcmdidx;
            break;

```

```

case CONFIRM:
    switch(cmd){
        // identification
        case _UPSMSG_SET_ID_IDENTNAME:
        case _UPSMSG_SET_ID_ATTDEVICE:
            pollcmdidx = UID;
            break;
        // configuration
        case _UPSMSG_SET_BATT_REPLACEDATE:
        case _UPSMSG_SET_CONF_INPUTVOLT:
        case _UPSMSG_SET_CONF_INPUTFREQ:
        case _UPSMSG_SET_CONF_OUTPUTVOLT:
        case _UPSMSG_SET_CONF_OUTPUTFREQ:
        case _UPSMSG_SET_CONF_OUTPUTVA:
        case _UPSMSG_SET_CONF_OUTPUTPOWER:
        case _UPSMSG_SET_CONF_LOWBATTIME:
        case _UPSMSG_SET_CONF_AUDIBLESTATUS:
        case _UPSMSG_SET_CONF_LOWVOLTTRPT:
        case _UPSMSG_SET_CONF_HIGHVOLTTRPT:
        case _UPSMSG_SET_CONF_BATTLIFE:
            pollcmdidx = CFG;
            break;
        // ups control
        case _UPSMSG_SET_CONT_SHUTDOWNNTYPE:
        case _UPSMSG_SET_CONT_SHUTDOWNAFDELAY:
        case _UPSMSG_SET_CONT_STARTUPAFDELAY:
        case _UPSMSG_SET_CONT_REBOOTWDURATION:
        case _UPSMSG_SET_CONT_AUTORESTARTTYPE:
            pollcmdidx = CTR;
            break;
        // test
        case _UPSMSG_SET_TEST_ABORT:
        case _UPSMSG_SET_TEST_GENERAL:
        case _UPSMSG_SET_TEST_QUICKBATT:
        case _UPSMSG_SET_TEST_DEEPBATT:
        case _UPSMSG_SET_TEST_USR:
            pollcmdidx = TST;
            break;
        // user specific controls
        case _UPSMSG_SET_CONT_USR1: // inverter
        case _UPSMSG_SET_CONT_USR2: // bypass
        case _UPSMSG_SET_CONT_USR3: // rectifier
        case _UPSMSG_SET_CONT_USR4:
        case _UPSMSG_SET_CONT_USR5:
        case _UPSMSG_SET_CONT_USR6:
        case _UPSMSG_SET_CONT_USR7:
        case _UPSMSG_SET_CONT_USR8:
        case _UPSMSG_SET_CONT_USR9:
        case _UPSMSG_SET_CONT_USR10:
        // user specific configurations
        case _UPSMSG_SET_CONF_USR1:
        case _UPSMSG_SET_CONF_USR2:
        case _UPSMSG_SET_CONF_USR3:
        case _UPSMSG_SET_CONF_USR4:
        case _UPSMSG_SET_CONF_USR5:
        case _UPSMSG_SET_CONF_USR6:
        case _UPSMSG_SET_CONF_USR7:
        case _UPSMSG_SET_CONF_USR8:
        case _UPSMSG_SET_CONF_USR9:
        case _UPSMSG_SET_CONF_USR10:
        default :
            return -1;
    }
    break;
default:
    return -1;
}

```

```

strcpy(cmdstr, poll_cmdstr[pollcmdidx]);
return pollcmdidx;
}

int get_set_command(char *cmdstr, int cmd, long val, char *strval)
{
char vals[10] = {0, };
sprintf(vals, "%u", val);
if(cmdstr == NULL)
return -1;
switch(cmd){
// ups identification
case _UPSMSG_SET_ID_IDENTNAME:
if(strval == NULL)
return -1;
sprintf(cmdstr, "#SET%03dNAM%sEND", 6+strlen(strval), strval);
break;
case _UPSMSG_SET_ID_ATTDEVICE:
if(strval == NULL)
return -1;
sprintf(cmdstr, "#SET%03dATD%sEND", 6+strlen(strval), strval);
break;
// nominal - battery
case _UPSMSG_SET_BATT_REPLACEDATE:
return -1;
// configuration
case _UPSMSG_SET_CONF_INPUTVOLT:
sprintf(cmdstr, "#SET%03dCIV%sEND", 6+strlen(vals), vals);
break;
case _UPSMSG_SET_CONF_INPUTFREQ:
sprintf(cmdstr, "#SET%03dCIF%sEND", 6+strlen(vals), vals);
break;
case _UPSMSG_SET_CONF_OUTPUTVOLT:
sprintf(cmdstr, "#SET%03dCOV%sEND", 6+strlen(vals), vals);
break;
case _UPSMSG_SET_CONF_OUTPUTFREQ:
sprintf(cmdstr, "#SET%03dCOF%sEND", 6+strlen(vals), vals);
break;
case _UPSMSG_SET_CONF_OUTPUTVA:
case _UPSMSG_SET_CONF_OUTPUTPOWER:
return -1;
case _UPSMSG_SET_CONF_LOWBATTTIME:
sprintf(cmdstr, "#SET%03dCLB%sEND", 6+strlen(vals), vals);
break;
case _UPSMSG_SET_CONF_AUDIBLESTATUS:
if(val == ADBLALM_DISABLED)
sprintf(cmdstr, "#SET007CAS1END");
else if(val == ADBLALM_ENABLED)
sprintf(cmdstr, "#SET007CAS2END");
else if(val == ADBLALM_MUTED)
sprintf(cmdstr, "#SET007CAS3END");
else
return -1;
break;
case _UPSMSG_SET_CONF_LOWVOLTTRPT:
sprintf(cmdstr, "#SET%03dCLT%sEND", 6+strlen(vals), vals);
break;
case _UPSMSG_SET_CONF_HIGHVOLTTRPT:
sprintf(cmdstr, "#SET%03dCHT%sEND", 6+strlen(vals), vals);
break;
case _UPSMSG_SET_CONF_BATTLIFE:
return -1;
// ups control
case _UPSMSG_SET_CONT_SHUTDOWNTYPE:
if(val == SHUTDNTYPE_OUTPUT)
sprintf(cmdstr, "#SET007SDT1END");
else if(val == SHUTDNTYPE_SYSTEM)
sprintf(cmdstr, "#SET007SDT2END");
}
}

```

```

else return -1;
break;
case _UPSMSG_SET_CONT_SHUTDOWNAFDELAY:
if(val == -1)
    sprintf(cmdstr, "#SET008SDA-1END");
else
    sprintf(cmdstr, "#SET%03dSDA%SEND", 6+strlen(vals), vals);
save_ups_control_shutdownafterdelay(val);
break;
case _UPSMSG_SET_CONT_STARTUPAFDELAY:
if(val == -1)
    sprintf(cmdstr, "#SET008SUA-1END");
else
    sprintf(cmdstr, "#SET%03dSUA%SEND", 6+strlen(vals), vals);
save_ups_control_startupafterdelay(val);
break;
case _UPSMSG_SET_CONT_REBOOTWDURATION:
if(val == -1)
    sprintf(cmdstr, "#SET008RWD-1END");
else
    sprintf(cmdstr, "#SET%03dRWD%SEND", 6+strlen(vals), vals);
save_ups_control_rebootwithduration(val);
break;
case _UPSMSG_SET_CONT_AUTORESTARTTYPE:
if(val == AUTORESTRT_ON)
    sprintf(cmdstr, "#SET007ARS1END");
else if(val == AUTORESTRT_OFF)
    sprintf(cmdstr, "#SET007ARS2END");
else
    return -1;
break;
// test
case _UPSMSG_SET_TEST_ABORT:
save_ups_testid(TEST_ABORT, 0);
sprintf(cmdstr, "#SET007TID2END");
break;
case _UPSMSG_SET_TEST_GENERAL:
save_ups_testid(TEST_GENERAL, 0);
sprintf(cmdstr, "#SET007TID3END");
break;
case _UPSMSG_SET_TEST_QUICKBATT:
save_ups_testid(TEST_QUICKBATT, 0);
sprintf(cmdstr, "#SET007TID4END");
break;
case _UPSMSG_SET_TEST_DEEPBATT:
save_ups_testid(TEST_DEEPBATTCALIB, 0);
sprintf(cmdstr, "#SET007TID5END");
break;
case _UPSMSG_SET_TEST_USR:
save_ups_testid(TEST_USER_SPECIFIC, 0);
return -1;
// user specific controls
case _UPSMSG_SET_CONT_USR1: // inverter
case _UPSMSG_SET_CONT_USR2: // bypass
case _UPSMSG_SET_CONT_USR3: // rectifier
case _UPSMSG_SET_CONT_USR4:
case _UPSMSG_SET_CONT_USR5:
case _UPSMSG_SET_CONT_USR6:
case _UPSMSG_SET_CONT_USR7:
case _UPSMSG_SET_CONT_USR8:
case _UPSMSG_SET_CONT_USR9:
case _UPSMSG_SET_CONT_USR10:
// user specific configurations
case _UPSMSG_SET_CONF_USR1:
case _UPSMSG_SET_CONF_USR2:
case _UPSMSG_SET_CONF_USR3:
case _UPSMSG_SET_CONF_USR4:
case _UPSMSG_SET_CONF_USR5:

```

```
case _UPSMSG_SET_CONF_USR6:
case _UPSMSG_SET_CONF_USR7:
case _UPSMSG_SET_CONF_USR8:
case _UPSMSG_SET_CONF_USR9:
case _UPSMSG_SET_CONF_USR10:
default :
    return -1;
}
return cmd;
}
```

## 2.7. Makefile

```
                sena_util.c  sena_cmd.c가  가                UPSAPP_OBJ
sena_util.o  sena_cmd.o  가                .
```

```
CROSS_COMPILE = /opt/hardhat/devkit/ppc/8xx/bin/ppc_8xx-
CC = $(CROSS_COMPILE)gcc
AR = $(CROSS_COMPILE)ar

INCLUDEDIRS = -I. -I./include
LDFLAGS = -L./lib
CFLAGS = ${INCLUDEDIRS} -D_REENTERANT

UPSAPP = upsapp
PROG_LIST = ${UPSAPP}

UPSAPP_OBJ = ./sena_util.o ./sena_cmd.o ./ups_mon.o ./ups_app.o

all : ${PROG_LIST}

${UPSAPP} : ${UPSAPP_OBJ}
    ${CC} -o $@ ${LDFLAGS} ${UPSAPP_OBJ} -lupslink

c.o :
    ${CC} -c ${CFLAGS} $<

clean :
    rm -f *.o ${UPSAPP_OBJ} ${PROG_LIST} core
```