

**Writing station specific FS code**

**September 2003 TOW**

**Ed Himwich, GSFC/NVI**

# Code: Basic

September 2003 TOW

Ed Himwich, NVI/GSFC

## 1. Overview of FS Architecture

- 1.1. Diagrams from FS manual, attached at end of write-up
  - 1.1.1. Initialization
  - 1.1.2. Normal Execution
  - 1.1.3. Pointing Programs
  - 1.1.4. Module Checking
  - 1.1.5. Data logging
- 1.2. Only “ddout” writes to disk
- 1.3. Except for “boss” reading the schedule and procedures, there is no disk reading except during initialization

## 2. Control Files For Station Coding

- 2.1. “stpgm.ctl”
  - 2.1.1. Defines station programs to run and stop at FS start and stop
  - 2.1.2. Trailing ampersand “&” means it is run in background
  - 2.1.3. No ampersand “&” means wait for return before proceeding
  - 2.1.4. Programs
    - 2.1.4.1. Most stations only have a subset
    - 2.1.4.2. “stcom”
      - 2.1.4.2.1. Initializes station shared memory
      - 2.1.4.2.2. Reads some control files
      - 2.1.4.2.3. Listed first in control file, so runs first
      - 2.1.4.2.4. No ampersand, “wait”
    - 2.1.4.3. Other programs are run in background, with ampersand
      - 2.1.4.3.1. stqkr            station SNAP commands
      - 2.1.4.3.2. antcn        antenna interface
      - 2.1.4.3.3. cheks        station module checking
      - 2.1.4.3.4. sterp        station error reporting
- 2.2. sterr.ctl
  - 2.2.1. Station program error messages.
  - 2.2.2. See Error Messages in Code: Intermediate write-up for more details
- 2.3. stcmd.ctl
  - 2.3.1. Station SNAP commands access control
- 2.4. mdlpo.ctl
  - 2.4.1. Pointing model file, usually read by “antcn”

### 3. Resource allocation

- 3.1. For FS internals this is done by fsalloc: shared memory, semaphores, and message queue
- 3.2. For station software this can be done with sample stalloc. Example provides only shared memory: C and two areas of FORTRAN shared memory

### 4. Emulation Services

- 4.1. Class I/O passes binary buffers between programs
- 4.2. Scheduling allows programs to pass control back and forth between "parent" and "child" processes
- 4.3. Resource numbers or semaphores allows coordinated access to resources
- 4.4. "Break" allows a signal to be sent to program to initiate some special action, usually aborting some behavior
- 4.5. Suspending and Resuming - Allows a program to suspend execution until some other action is taken, usually by the operator who must tell the program to resume again, with a "go".
- 4.6. Shared memory, straightforward in C, complicated in FORTRAN because there is no direct support

### 5. More Information

- 5.1. Code Intermediate and Advanced Chief Meeting write-up for overview
- 5.2. End of Volume 2 of the FS Manuals has several relevant sections but some out of date
- 5.3. See examples of use in FS code

### 6. "antcn"

- 6.1. Must be coded to not cause delays
- 6.2. Modes
  - 6.2.1. 0 initialization
  - 6.2.2. 1 new source
  - 6.2.3. 2 new offsets
  - 6.2.4. 3 check onsource status, with error logging
  - 6.2.5. 4 antenna= command
  - 6.2.6. 5 onsource with no error logging
  - 6.2.7. 6 reserved for focus control
  - 6.2.8. 7 onsource with additional information
  - 6.2.9. 8 station specific detectors (new)
- 6.3. Example "antcn"

```
/* antcn

   This is the antcn (ANTenna CoNtrol program) for Tsukuba 32.
*/

#define MINMODE 0
#define MAXMODE 8
#define MDLPO "/usr2/control/mdlpo.ct1"

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

#include "../fs/include/dpi.h"
#include "../fs/include/fs_types.h"
```

```

#include "../..fs/include/shm_addr.h"      /* shared memory pointer */
#include "../..fs/include/params.h"
#include "../..fs/include/fscm.h"
#include "../..fs/include/pmodel.h"

#include "../include/stparams.h"
#include "../include/stcom.h"
#include "../include/stm_addr.h"          /* shared memory pointer */

struct stcom *st;
struct fscm *fs;

void st_setup(), st_command(), st_onsource(), st_antenna();
void setup_ids(), setup_st();
void putpname();
int gmodl();
void corrq(), equn(), tracq(), pmdlq();
void skd_run(), cls_clr();
int nsem_test();
void logit();
long idum[] = {0,0,0,0,0};
long cls_alc();

main()
{
    int ierr;
    int imode;
    long ip[5];
    int iy,id;
    double eqofeq;
    char idev[64],oldlog[8]; /* test */
    char buf [80];
    int imem; /* test */
    int nrec, nrecr;
    long class, classr;
    int i;

    /* Set up IDs for shared memory, then assign the pointer to
       "fs", for readability.
    */
    setup_ids();
    fs = shm_addr;
    setup_st();
    st = stm_addr;

    /* Put our program name where logit can find it. */

    putpname("antcn");

    /* allocate class box for message from trakl */

    if( -1 == (stm_addr->antbox=cls_alc())) {
        fprintf( stderr," antbox allocation failed\n");
        exit( -1);
    }

    /* Return to this point to wait until we are called again */

Continue:
    skd_wait("antcn",ip,(unsigned)0);

    imode = ip[0];
    class = ip[1];
    nrec = ip[2];
    nrecr = 0;
    classr = 0;
    if (imode < MINMODE || imode > MAXMODE) {
        ierr = -1;
        goto End;
    }
}

```

```

switch (imode) {

case 0:          /* initialize */
    ierr = 0;
    if (gmodl(MDLPO,&st->pmodel) < 0) {
        ierr = -6;
        goto End;
    }
    st_setup();
    skd_run("trakl", 'n', idum);
    break;
case 1:          /* source= command */
case 2:          /* offsets          */
    if (nsem_test("trakl") != 1) {
        logit(NULLPTR, -8, "an");
        goto End;
    }
    if (memcmp(st->point.oldlog, fs->LLOG, sizeof(st->point.oldlog)) != 0)
        pmdlq(&st->pmodel); /* log the model */
    st_command(imode);
    fs->ionsor=0;
    break;

case 4:          /* direct antenna= command */
    if (class == 0)
        goto End;
    if (nsem_test("trakl") != 1) {
        logit(NULLPTR, -8, "an");
        cls_clr(class);
        goto End;
    }
    st_antenna(class, nrec, &classr, &nrecr, &ierr);
    break;

case 6:          /* reserved */
    ierr = -1;
    goto End;

case 3:          /* onsource command with error message */
case 5:          /* onsource command with no error logging */
case 7:          /* onsource command with additional info */
    if (nsem_test("trakl") != 1) {
        logit(NULLPTR, -8, "an");
        goto End;
    }
    st_onsource();
    if (st->error.konsor)
        fs->ionsor=1;
    else {
        fs->ionsor=0;
        if (imode == 3 && st->point.itype < 5 )
            logit(NULLPTR, -103, "an");
    }
    if (imode == 7) {
        tracq(&st->pos_old, &st->ercr_old);
        iy = st->pos_old.t[5] - 1900;
        id = st->pos_old.t[4];
        equn(iy, id, &eqofeq);
        corrq(&st->ercr_old, eqofeq);
    }

    break;

case 8:
    if (strncmp(shm_addr->user_dev1_name, " ", 2) != 0) {
        char *meter;
        for (i=0; i<5; i++)
            ip[i]=0;
        if (strncmp(shm_addr->user_dev1_name, "u5", 2) == 0) {
            meter="p2";
            ib_req2(ip, meter, "AP");
        } else if (strncmp(shm_addr->user_dev1_name, "u6", 2) == 0) {

```

```

        meter="p1";
        ib_req2(ip,meter,"AP");
    }
    skd_run("ibcon", 'w', ip);
    skd_par(ip);
    if(ip[2]<0)
        goto Continue;
}
if(strncmp(shm_addr->user_dev2_name," ",2)!=0) {
    char *meter;
    for (i=0;i<5;i++)
        ip[i]=0;
    if(strncmp(shm_addr->user_dev2_name,"u5",2)==0) {
        meter="p2";
        ib_req2(ip,meter,"AP");
    } else if(strncmp(shm_addr->user_dev2_name,"u6",2)==0) {
        meter="p1";
        ib_req2(ip,meter,"AP");
    }
    skd_run("ibcon", 'w', ip);
    skd_par(ip);
    if(ip[2]<0)
        goto Continue;
}
rte_sleep(100);
if(strncmp(shm_addr->user_dev1_name," ",2)!=0) {
    char *meter;
    for (i=0;i<5;i++)
        ip[i]=0;
    if(strncmp(shm_addr->user_dev1_name,"u5",2)==0) {
        meter="p2";
        ib_req5(ip,meter,20);
    } else if(strncmp(shm_addr->user_dev1_name,"u6",2)==0) {
        meter="p1";
        ib_req5(ip,meter,20);
    }
    skd_run("ibcon", 'w', ip);
    skd_par(ip);
    if(ip[2]<0)
        goto Continue;

    i=20;
    ib_res_ascii(buf,&i,ip);
    {
        float pwr;
        sscanf(buf,"%f",&pwr);
        shm_addr->user_dev1_value=pwr*1e6;
    }
}
if(strncmp(shm_addr->user_dev2_name," ",2)!=0) {
    char *meter;
    for (i=0;i<5;i++)
        ip[i]=0;
    if(strncmp(shm_addr->user_dev2_name,"u5",2)==0) {
        meter="p2";
        ib_req5(ip,meter,20);
    } else if(strncmp(shm_addr->user_dev2_name,"u6",2)==0) {
        meter="p1";
        ib_req5(ip,meter,20);
    }
    skd_run("ibcon", 'w', ip);
    skd_par(ip);
    if(ip[2]<0)
        goto Continue;

    i=20;
    ib_res_ascii(buf,&i,ip);
    {
        float pwr;
        sscanf(buf,"%f",&pwr);
        shm_addr->user_dev2_value=pwr*1e6;
    }
}

```

```
    }  
  }  
  break;  
default:  
  ierr = -1;  
} /* end of switch */  
  
End:  
ip[0] = classr;  
ip[1] = nrecr;  
ip[2] = ierr;  
memcpy(ip+3,"an",2);  
  
ip[4] = 0;  
goto Continue;  
  
}
```

# Code: Intermediate

September 2003 TOW

Ed Himwich, NVI/GSFC

## 1. Connecting to FS resources

- 1.1. Necessary for FS libraries and utilities to work
- 1.2. The first thing a program should do is
  - 1.2.1. In C:
    - 1.2.1.1. setup\_ids()
  - 1.2.2. In FORTRAN:
    - 1.2.2.1. call setup\_fscom
    - 1.2.2.2. call read\_fscom
    - 1.2.2.3. When waking up must call "read\_quikr" to refresh fscom\_quik

## 2. Program Scheduling

- 2.1. Form of inter-program communication.
- 2.2. Flow
  - 2.2.1. All programs initialize by connecting to FS resources (see above)
  - 2.2.2. Most wait to be scheduled, skd\_wait().
- 2.3. In C:
  - 2.3.1. skd\_run() schedule a program with run parameters
  - 2.3.2. skd\_par() retrieve run parameters from a returning child
  - 2.3.3. skd\_run\_arg() schedule program with an ASCII string
  - 2.3.4. skd\_wait() wait for some one to schedule me with optional time-out and return run parameters
  - 2.3.5. skd\_arg() retrieve n-th ASCII blank delimited argument from father
  - 2.3.6. skd\_chk() check whether I've been scheduled
  - 2.3.7. skd\_end() wake up my father if I have one, send run parameters
  - 2.3.8. dad\_pid() get my father's pid, 0 if none
- 2.4. In FORTRAN:
  - 2.4.1. run\_prog() accesses skd\_run
  - 2.4.2. wait\_prog() skd\_wait with no time-out
  - 2.4.3. wait\_abst() skd\_wait with time-out at absolute time
  - 2.4.4. wait\_abstd() skd\_wait with time-out at absolute time including doy
  - 2.4.5. wait\_relt() skd\_wait with relative wait time
  - 2.4.6. get\_arg() skd\_arg
  - 2.4.7. rmpar() skd\_par



### **3. Error handling and class buffer exchange**

- 3.1. Structured around long (integer\*4) 5 element interprogram communication array
  - 3.1.1. First Element - class number holding messages
  - 3.1.2. Second - number of class records
  - 3.1.3. Third - Error number if non-zero
  - 3.1.4. Fourth - first two characters of error type
  - 3.1.5. Fifth - first two characters of additional error type or additional binary information
- 3.2. Errors are detected at the lowest level and the routines and process return until the highest level routine, usually boss or chekr logs the error
- 3.3. Asynchronous - errors exist without context, this has good and bad points
- 3.4. If there is an error don't leave data in class numbers, unless you want it logged, it is ascii and "boss" is the executive.
- 3.5. Normal path
  - 3.5.1. BOSS passes command to QUIKR in a class buffer with save bits on
  - 3.5.2. QUIKR parses command, generates class message for MATCN
  - 3.5.3. MATCN processes buffers, generating responses which go into class buffers, and or an error
  - 3.5.4. QUIKR parses response from MATCN and generates log entries in class buffers and sends them or MATCN errors to BOSS
  - 3.5.5. BOSS accepts log entries or errors and sends them to the log entry system, an error automatically removes a command for time list

### **4. Setting up station help pages and error messages**

- 4.1. Text files go in /usr2/st/help and are in the form xxx.\_\_\_, where xxx corresponds to the xxx the help=xxx, could be a command name or other word
- 4.2. Error messages go in /usr2/control/sterr.ctl, compare to /usr2/fs/control/fserr.ctl
- 4.3. Three lines per error message, where XX is the two letter error mnemonic (in caps), nnn is the numeric error:
  - 4.3.1. ""
  - 4.3.2. XX -nnn
  - 4.3.3. message

### **5. Station SNAP Commands**

- 5.1. Linked to the station specific stqkr program
- 5.2. See example code in /usr2/fs/st.default/st-1.0.0./stqkr
- 5.3. FORTRAN should be avoided
- 5.4. Selection of code in stqkr for each command is determined by values in stcmd.ctl file
- 5.5. Coding of simple C based SNAP command bbc in /usr2/fs/quikv/bbc.c provides an example of how to implement
- 5.6. "stqkr" example

**stqkr.c**

```

/* stqkr - C version of station command controller
*/
#include <string.h>
#include <stdio.h>
#include <sys/types.h>

#include "../fs/include/fs_types.h"
#include "../fs/include/shm_addr.h"          /* shared memory pointer */
#include "../fs/include/params.h"
#include "../fs/include/fscom.h"
#include "../fs/include/pmodel.h"

#include "../include/stparams.h"
#include "../include/stcom.h"
#include "../include/stm_addr.h"          /* shared memory pointer */

struct stcom *st;
struct fscom *fs;

#define MAX_BUF    257

main()
{
    long ip[5];
    int isub,itask,idum,ierr,nchars,i;
    char buf[MAX_BUF];
    struct cmd_ds command;
    int cls_rcv(), cmd_parse();
    void skd_wait();

    /* Set up IDs for shared memory, then assign the pointer to
       "fs", for readability.
    */
    setup_ids();
    fs = shm_addr;
    setup_st();
    st = stm_addr;

loop:
    skd_wait("stqkr",ip,(unsigned) 0);
    if(ip[0]==0) {
        ierr=-1;
        goto error;
    }
    ierr=0;

    nchars=cls_rcv(ip[0],buf,MAX_BUF,&idum,&idum,0,0);
    if(nchars==MAX_BUF && buf[nchars-1] != '\0' ) { /*does it fit?*/
        ierr=-2;
        goto error;
    }
    /* null terminate to be sure */
    if(nchars < MAX_BUF && buf[nchars-1] != '\0') buf[nchars]='\0';

    if(0 != (ierr = cmd_parse(buf,&command))) { /* parse it */
        ierr=-3;
        goto error;
    }

    isub = ip[1]/100;
    itask = ip[1] - 100*isub;

    switch (isub) {
        case 1:
/*
                                antenna echo function */
            ierr=0;
            aecho(&command,ip);
            break;

```

```

        case 2:
            /*                      WX function */
            ierr=0;
            wx(&command,ip);
            break;

        case 3:
            /*                      IF Attenuator function */
            ierr=0;
            ifatt(&command,itask,ip);
            break;

        default:
            ierr=-4;
            goto error;
    }
    goto loop;

error:
    for (i=0;i<5;i++) ip[i]=0;
    ip[2]=ierr;
    memcpy(ip+3,"st",2);
    goto loop;
}

```

## ifatt.c

```

/* Tsukuba if att snap command */

#include <stdio.h>
#include <string.h>
#include <sys/types.h>

#include "../fs/include/params.h"
#include "../fs/include/fs_types.h"
#include "../fs/include/fscom.h"
#include "../fs/include/shm_addr.h"
#include "../fs/include/pmodel.h"
/* shared memory definition */
/* shared memory pointer */

#include "../include/stparams.h"
#include "../include/stcom.h"
#include "../include/stm_addr.h"

void ifatt(command,itask,ip)
struct cmd_ds *command;          /* parsed command structure */
int itask;
long ip[5];                      /* ipc parameters */
{
    int ilast, ierr, ichold, i, count, type;
    char *ptr;
    struct ifatt_cmd lcl;

    char *arg_next();

    void ifatt_dis();

    void skd_run(), skd_par();    /* program scheduling utilities */
    ichold= -99;                  /* check vlaue holder */

    ip[0]=ip[1]=0;

    if (command->equal != '=') {          /* read module */
        ifatt_req_q(ip);
        goto k4con;
    }
    else if (command->argv[0]==NULL) goto parse; /* simple equals */
    else if (command->argv[1]==NULL) /* special cases */
        if (*command->argv[0]=='?') {

```

```

        ifatt_dis(command,itask,ip);
        return;
    }

/* if we get this far it is a set-up command so parse it */

parse:
    ilast=0;
    memcpy(&lcl,&stm_addr->ifatt,sizeof(lcl));

    count=1;
    while( count>= 0) {
        ptr=arg_next(command,&ilast);
        ierr=ifatt_dec(&lcl,&count, ptr);
        if(ierr !=0 ) goto error;
    }

/* all parameters parsed okay, update common */
/*
    ichold=shm_addr->check.k4rec.check;
    shm_addr->check.k4rec.check=0;
    */
    memcpy(&stm_addr->ifatt,&lcl,sizeof(lcl));

/* format buffers for k4con */

    ifatt_req_c(ip,&lcl);

k4con:
    skd_run("ibcon",'w',ip);
    skd_par(ip);
    /*
    if (ichold != -99) {
        shm_addr->check.k4rec.state=TRUE;
        if (ichold >= 0)
            ichold=ichold % 1000 + 1;
        shm_addr->check.k4rec.check=ichold;
    }
    */
    if(ip[2]<0) {
        cls_clr(ip[0]);
        ip[0]=ip[1]=0;
        return;
    }

    ifatt_dis(command,itask,ip);
    return;

error:
    ip[0]=0;
    ip[1]=0;
    ip[2]=ierr;
    memcpy(ip+3,"st",2);
    return;
}

```

## ifatt\_dis.c

```

/* Tsukuba if att display */

#include <stdio.h>
#include <string.h>
#include <sys/types.h>

#include "../fs/include/params.h"
#include "../fs/include/fs_types.h"
#include "../fs/include/fscom.h"
#include "../fs/include/shm_addr.h"
#include "../fs/include/pmodel.h"

```

```

#include "../include/stparams.h"
#include "../include/stcom.h"
#include "../include/stm_addr.h"

#define MAX_OUT 256

void ifatt_dis(command,itask,ip)
struct cmd_ds *command;
int itask;
long ip[5];
{
    struct ifatt_cmd lclc;
    int kcom, i, ierr, count;
    char output[MAX_OUT];

    kcom= command->argv[0] != NULL &&
        *command->argv[0] == '?' && command->argv[1] == NULL;

    if ((!kcom) && command->equal == '=') {
        if(ip[0]!=0) {
            cls_clr(ip[0]);
            ip[0]=0;
        }
        ip[1]=0;
        return;
    } else if (kcom){
        memcpy(&lclc,&stm_addr->ifatt,sizeof(lclc));
    } else {
        ifatt_res_q(&lclc,ip);
        if(ip[1]!=0) {
            cls_clr(ip[0]);
            ip[0]=ip[1]=0;
        }
        if(ip[2]!=0) {
            ierr=ip[2];
            goto error;
        }
    }

    /* format output buffer */

    strcpy(output,command->name);
    strcat(output,"/");

    count=0;
    while( count>= 0) {
        if (count > 0) strcat(output,",");
        count++;
        ifatt_enc(output,&count,&lclc);
    }
    if(strlen(output)>0) output[strlen(output)-1]='\0';

    for (i=0;i<5;i++) ip[i]=0;
    cls_snd(&ip[0],output,strlen(output),0,0);
    ip[1]=1;

    return;

error:
    ip[0]=0;
    ip[1]=0;
    ip[2]=ierr;
    memcpy(ip+3,"ki",2);
    return;
}

```

## ifatt\_util.c

```

/* Tsukuba IF ATT buffer parsing utilities */

```

```

#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <limits.h>
#include <math.h>

#include "../fs/include/macro.h"
#include "../fs/include/params.h"
#include "../fs/include/fs_types.h"
#include "../fs/include/fscom.h"          /* shared memory definition */
#include "../fs/include/shm_addr.h"      /* shared memory pointer */
#include "../fs/include/pmodel.h"

#include "../include/stparams.h"
#include "../include/stcom.h"
#include "../include/stm_addr.h"

static char device[]={"if"};           /* device menemonics */

#define MAX_BUF 512

int ifatt_dec(lcl,count,ptr)
struct ifatt_cmd *lcl;
int *count;
char *ptr;
{
    int ierr, arg_int();

    ierr=0;
    if(ptr == NULL) ptr="";

    if (*count >0 && *count <5) {
        ierr=arg_int(ptr,&lcl->atten[*count-1],0,FALSE);
        if(ierr==0 && (lcl->atten[*count-1]<0 || lcl->atten[*count-1]>81))
            ierr=-200;
    } else
        *count=-1;

    if(ierr!=0) ierr-=*count;
    if(*count>0) (*count)++;
    return ierr;
}

void ifatt_enc(output,count,lcl)
char *output;
int *count;
struct ifatt_cmd *lcl;
{
    int ivalue, type;

    output=output+strlen(output);

    if(*count > 0 && *count < 5 ) {
        sprintf(output,"%02d",lcl->atten[*count-1]);
    } else
        *count=-1;

    return;
}

ifatt_req_q(ip)
long ip[5];
{
    ib_req7(ip,device,31*2-1+2,"rout:clos? (@1(0:15),2(0:14))");
}

ifatt_req_c(ip,lclc)
long ip[5];
struct ifatt_cmd *lclc;
{
    char buffer[120];

```

```

unsigned short word[2];
int i;

ib_req2(ip,device,"ROUT:DRIV:ON:ALL");

word[0]=word[1]=0;

for(i=0;i<4;i++) {
    int value=lclcl->atten[i];
    int low,up;
    if(value<80) {
        if(value%10 <8)
            low=value%10;
        else
            low=0x8 | (value%10-4);
        up=value/10;
    } else if(value == 80){
        low=0xE;
        up=0x7;
    } else if(value == 81) {
        low=0xF;
        up=0x7;
    }
    word[0]|=low << (i*4);
    word[1]|=up << (i*4);
}

if(word[0]!=0xFFFF || word[1] !=0x7777) {
    strcpy(buffer,"ROUT:OPEN (@");
    if(word[0]!=0xFFFF) {
        strcat(buffer,"1(");
        for (i=0;i<16;i++)
            if((word[0] & (1<<i)) == 0)
                sprintf(buffer+strlen(buffer),"%d",i);
        strcpy(buffer+strlen(buffer)-1,"");
        if(word[1]!=0x7777)
            strcat(buffer,",");
    }

    if(word[1]!=0x7777) {
        strcat(buffer,"2(");
        for (i=0;i<16;i++)
            if((word[1] & (1<<i)) == 0)
                sprintf(buffer+strlen(buffer),"%d",i);
        strcpy(buffer+strlen(buffer)-1,"");
    }
    strcat(buffer,")");
    ib_req2(ip,device,buffer);
}

if(word[0]!=0 || word[1] !=0) {
    strcpy(buffer,"ROUT:CLOS (@");
    if(word[0]!=0) {
        strcat(buffer,"1(");
        for (i=0;i<16;i++)
            if((word[0]& (1<<i)) != 0)
                sprintf(buffer+strlen(buffer),"%d",i);
        strcpy(buffer+strlen(buffer)-1,"");
        if(word[1]!=0)
            strcat(buffer,",");
    }

    if(word[1]!=0) {
        strcat(buffer,"2(");
        for (i=0;i<16;i++)
            if((word[1]& (1<<i)) != 0)
                sprintf(buffer+strlen(buffer),"%d",i);
        strcpy(buffer+strlen(buffer)-1,"");
    }
}

```

```

        strcat(buffer,"");
        ib_req2(ip,device,buffer);
    }
}

ifatt_res_q(lclc,ip)
struct ifatt_cmd *lclc;
long ip[5];
{
    char buffer[MAX_BUF];
    int i,max;
    unsigned short word[2];

    max=sizeof(buffer);
    ib_res_ascii(buffer,&max,ip);
    if(max < 0) {
        ip[2]=-1;
        return;
    }

    word[0]=word[1]=0;

    for(i=0;i<31;i++) {
        int ibit;
        sscanf(buffer+i*2,"%d",&ibit);
        if(ibit!=0)
            word[i/16]|=1<<(i%16);
    }

    for (i=0;i<4;i++) {
        unsigned char x;
        x=(word[0]>>(i*4)) & 0xf;
        lclc->atten[i]=x &0x7;
        if((x&0x8)!=0)
            lclc->atten[i]+=4;
        x=(word[1]>>(i*4)) & 0x7;
        lclc->atten[i]+=x*10;
    }

    return;
}

```

## WX.C

```

/* wx command
*/

#include <stdio.h>
#include <string.h>
#include <sys/types.h>

#include "../fs/include/fs_types.h"
#include "../fs/include/shm_addr.h" /* shared memory pointer */
#include "../fs/include/params.h"
#include "../fs/include/fscom.h"
#include "../fs/include/pmodel.h"

#include "../include/stparams.h"
#include "../include/stcom.h"
#include "../include/stm_addr.h" /* shared memory pointer */

extern struct stcom *st;
extern struct fscom *fs;

#define MAX_OUT 256

void wx(command,ip)
struct cmd_ds *command;
long ip[5];

```



```

{
    char  output[MAX_OUT];
    int  which,i;
    long time;

    which=stm_addr->wx.which;
    if(which <0 || 1 < which) {
        ip[2]=-96;
        goto error;
    }

    rte_rawt(&time);
    if(stm_addr->wx.time[which]< time-6001) {
        ip[2]=-95;
        goto error;
    }
    strcpy(output,command->name);
    strcat(output,"/");

    sprintf(output+strlen(output),"% .1f,% .1f,% .1f",
            stm_addr->wx.temp[which],
            stm_addr->wx.pres[which],
            stm_addr->wx.humi[which]);

    shm_addr->tempwx=stm_addr->wx.temp[which];
    shm_addr->humiwx=stm_addr->wx.humi[which];
    shm_addr->preswx=stm_addr->wx.pres[which];

    for (i=0;i<5;i++) ip[i]=0;
    cls_snd(&ip[0],output,strlen(output),0,0);
    ip[1]=1;

    return;

error:
    ip[0]=0;
    ip[1]=0;
    memcpy(ip+3,"st",2);
    return;
}

```

# Code: Advanced

September 2003 TOW

Ed Himwich, NVI/GSFC

## 1. Class I/O

- 1.1. Emulation of HP system feature with System V messages
- 1.2. Class numbers are FIFO mailboxes,
- 1.3. In C:
  - 1.3.1. cls\_snd() send a class message
  - 1.3.2. cls\_rcv() receive a class message
  - 1.3.3. cls\_clr() clear a class number
- 1.4. In FORTRAN:
  - 1.4.1. put\_buf() send a class message
  - 1.4.2. get\_buf() receive a class message
  - 1.4.3. clrcl() clear a class number
- 1.5. Class number
- 1.6. word:
  - 1.6.1. long (integer\*4) variable
  - 1.6.2. Bits Value
  - 31-16 zero
  - 15 1=no-wait
  - 14 1=save buffer
  - 13 1=save class
  - 12-0 class number

## 2. LOGIT routines

- 2.1. Used for reporting errors, send text to log
- 2.2. Not normally needed since BOSS normally logs these errors
- 2.3. See examples in existing code
- 2.4. In C:
  - 2.4.1. putpname() insert external program name for external messages
  - 2.4.2. logit() ASCII message OR Error number, mnemonic
  - 2.4.3. logita() ASCII message OR Error number, mnemonic, plus additional character code
  - 2.4.4. logite() ASCII message OR Error number, mnemonic, plus additional text for S2 errors
- 2.5. In FORTRAN:
  - 2.5.1. pname() insert external program name for external messages
  - 2.5.2. logit2() log text message from external program
  - 2.5.3. logit2\_ch() log text message from external program input is character variable

- 2.5.4. logit3()log ASCII text, used primarily by newlg.f for log header information
- 2.5.5. logit4()log ASCII message with calling procedure name
- 2.5.6. logit4d() log ASCII message with date information
- 2.5.7. logit4\_ch() log character type message
- 2.5.8. logit5()log the opening line of a new log
- 2.5.9. logit6()log error, all args except the last two are zero.

The last two arguments are:

- 2.5.9.1. error integer
- 2.5.9.2. error mnemonic ASCII

- 2.5.10. logit6c() log error, all args except the last two are zero.

The last two arguments are:

- 2.5.10.1. error integer
- 2.5.10.2. error mnemonic, character value

- 2.5.11. logit7()log error, all args except the last four are zero.

The last four arguments are:

- 2.5.11.1. + or -/0 for last argument is 2 characters
- 2.5.11.2. error number
- 2.5.11.3. ASCII mnemonic
- 2.5.11.4. last arg (0 is ignored)

- 2.5.12. logit7cc() same as logit7(), but last two args are character
- 2.5.13. logit7ci() same as logit7(), but next to last argument is character
- 2.5.14. logit7ic() same as logit7(), but last argument is character

### 3. shared memory for the FS

#### 3.1. FORTRAN

Defined in fs/include/fscom.i, includes:

- 3.1.1. fscom\_init.i
  - 3.1.1.1. initialization values from sincom
  - 3.1.1.2. set only once and is read in by read\_fscom
- 3.1.2. fscom\_quik.i
  - 3.1.2.1. quikr defined values

3.1.2.2. may be changed every time quikr runs and needs to be refreshed with read\_quikr every time a program is scheduled

3.1.3. fscom\_dum.i

3.1.3.1. FORTRAN copy of C stored data

3.1.3.2. C data accessed by fs\_set/get routines in /usr2/fs/newlb/prog.c

3.2. In C:

The shared memory area is available through the pointer

```
extern struct fscom *shm_addr;
```

defined in /usr2/fs/include/shm\_addr.h

#### **4. Shared memory for the station software**

4.1. Example in /usr2/fs/st.default/stlib/stm\_util.c

4.2. Needs initialization by stalloc

4.3. Supports one C area and two FORTRAN areas

4.4. In C:

The shared memory area is available through the pointer

```
extern struct stcom *stm_addr;
```

defined in /usr2/fs/st.default/st-1.0.0/include/stm\_addr.h

4.5. FORTRAN

4.6. Please avoid using FORTRAN shared memory

4.7. For FORTRAN primitive C-based functions are provided

4.7.1. stm\_map() defines up to 2 FORTRAN areas to managed

4.7.2. stm\_read() refreshes from C area

4.7.3. stm\_write() copies to C area

4.7.4. These can be used to build up higher level function like those for the FS found in /usr2/fs/flib/: setup\_fscom.f, read\_\*.f write\_\*.f

#### **5. Semaphores**

5.1. Provide a means for controlling access to resource, literal or "virtual".

5.2. In C there are two levels of semaphores:

5.2.1. Numbered

5.2.1.1. Fixed number SEM\_NUM (32)

5.2.1.2. Access functions

5.2.1.2.1. sem\_take() take a semaphore, wait until available if necessary

5.2.1.2.2. sem\_put() release a semaphore

- 5.2.1.2.3. sem\_nb()            take if available, return -1 if  
                                 not available
- 5.2.1.2.4. sem\_value()        return value

5.2.2. Named

5.2.2.1. Fixed number SEM\_NUM (32), disjoint from Numbered semaphores uses a 5 character name to identify

5.2.2.2. once a name is defined, it is defined until next boot

5.2.2.3. Access functions

- 5.2.2.3.1. nsem\_take()        take a semaphore, wait until  
                                 available if necessary
- 5.2.2.3.2. nsem\_put()         release a semaphore
- 5.2.2.3.3. sem\_test()         return state, 1 = taken

5.3. In FORTRAN

5.3.1. only the named semaphores are supported

5.3.2. Access functions:

- 5.3.2.1. rn\_take()            take a semaphore, wait until  
                                 available if necessary
- 5.3.2.2. rn\_put()             release a semaphore
- 5.3.2.3. rn\_test()            return state, .true. = taken

5.4. Defined named semaphores

- 5.4.1. "fs "                    FS is active
- 5.4.2. "fsctl"                schedule needs control with the next 2 seconds,  
                                 coordinates hardware access of boss, chekr, setcl, and fmset
- 5.4.3. "fivpt"                 FIVPT is active
- 5.4.4. "onoff"                 FIVPT is active
- 5.4.5. "pfmed"                PFMED is active
- 5.4.6. "pcalr"                PCALR is active
- 5.4.7. "lvdt "                 access to the LVDT

6. Pcald

- 6.1. Used to collect phase-cal data
- 6.2. Sample stub in /usr2/fs/st.default/pcald for next release
- 6.3. Should use "fsctl" named semaphore before accessing hardware, details TBD

**7. Asynchronous programs**

7.1. Three options:

- 7.1.1. Started at boot time
- 7.1.2. Started from stpgm.ctl at FS start time
- 7.1.3. Started by "antcn"

- 7.2. Option (2) is preferred because if the FS is restarted the system is completely re-initialized
- 7.3. Option (1) is necessary if the program needs to be running even when the FS isn't
- 7.4. Option (3) can simulate option (2) if it runs periodically, say every second, by checking the "fs" named semaphores to see if the FS is running
- 7.5. WX Example

```

/* wxget - retrieve wx data asynchronously
*/

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

#include "../fs/include/dpi.h"
#include "../fs/include/fs_types.h"
#include "../fs/include/shm_addr.h" /* shared memory pointer */
#include "../fs/include/params.h"
#include "../fs/include/fscm.h"
#include "../fs/include/pmodel.h"

#include "../include/stparams.h"
#include "../include/stcom.h"
#include "../include/stm_addr.h" /* shared memory pointer */

main(argc,argv)
int argc;
char **argv;
{
    int max,which;
    long ip[5],time;
    char buffer[28], bufs[10];
    float temp,pres,humi;

    setup_ids();
    setup_st();

    stm_addr->wx.which=-1;

    while(TRUE) {
        ip[0]=ip[1]=0;
        ib_req12(ip,"wx");
        skd_run("ibcon",'w',ip);
        skd_par(ip);
        if(ip[2] < 0) {
            if(ip[0]!=0)
                cls_clr(ip[0]);
            logita(NULL,ip[2],ip+3,ip+4);
            continue;
        }
        rte_sleep(3);

        ip[0]=ip[1]=0;
        ib_req2(ip,"wx","S1D000X0/0*");
        skd_run("ibcon",'w',ip);
        skd_par(ip);
        if(ip[2] < 0) {
            if(ip[0]!=0)
                cls_clr(ip[0]);
            logita(NULL,ip[2],ip+3,ip+4);
            continue;
        }
        rte_sleep(1001);

        ip[0]=ip[1]=0;
        ib_req5(ip,"wx",28);
    }
}

```

```

skd_run("ibcon",'w',ip);
skd_par(ip);
if(ip[2] < 0) {
    if(ip[0]!=0)
        cls_clr(ip[0]);
    logita(NULL,ip[2],ip+3,ip+4);
    continue;
}
rte_rawt(&time);

max=sizeof(buffer);
ib_res_ascii(buffer,&max,ip);

memcpy(bufs,buffer+11,6);
bufs[6]=0;
if(1!=sscanf(bufs,"%f",&temp)) {
    logita(NULL,-99,"st"," ");
    continue;
}
memcpy(bufs,buffer+17,3);
bufs[3]=0;
if(1!=sscanf(bufs,"%f",&humi)) {
    logita(NULL,-98,"st"," ");
    continue;
}
memcpy(bufs,buffer+20,6);
bufs[6]=0;
if(1!=sscanf(bufs,"%f",&pres)) {
    logita(NULL,-97,"st"," ");
    continue;
}

if(stm_addr->wx.which < 0 || 1 < stm_addr->wx.which )
    which=0;
else
    which=1-stm_addr->wx.which;

stm_addr->wx.temp[which]=temp;
stm_addr->wx.pres[which]=pres;
stm_addr->wx.humi[which]=humi;
stm_addr->wx.time[which]=time;
stm_addr->wx.which=which;
/*
printf(" max %d buffer %s time %d which %d \n",max,buffer,time,which);

printf(" temp %f pres %f humi %f\n",temp,pres,humi);
*/
}
}

```