TOW2025 - Seminar

FS Station Code

Alexander Neidhardt (TUM Wettzell)

Experience level: Beginners, Advanced.

Description: This course describes how to write station specific code with C. We discuss how other programs can easily interact with the FS shared memory and how to manage them.

Code: FSb1, FSb2

TOW2025 - Seminar

FS Station Code

What about FS?

What does a station has to offer to the FS?
How to control your antenna from FS?
How to control your equipment from FS?
How to fill data sets of the FS?
How to add functionality to the FS?
How to extend number of parallel devices?
How to combine functionalities to the FS?

What about FS?

For general FS basics see:

TOW2023 - Maintenance Workshops

FS Operations

Alexander Neidhardt (TUM Wettzell)

Experience level: Beginners.

Description: This course describes the general structure of the NASA Field System, including important control files, program locations, handling, and so on. We will take a look into installation and setup. Main part is the use of the FS and the adaption of the PC for the Field System.

Thanks for input from Simon Seidl (TUM Wettzell),
Katherine Pazamickas (PERATON), and Ed Himwich (NVI)

Code: FSo1, FSo2

TOW2025 - Seminar

FS Station Code

What about FS?

What does a station has to offer to the FS?

How to control your antenna from FS?

How to control your equipment from FS?

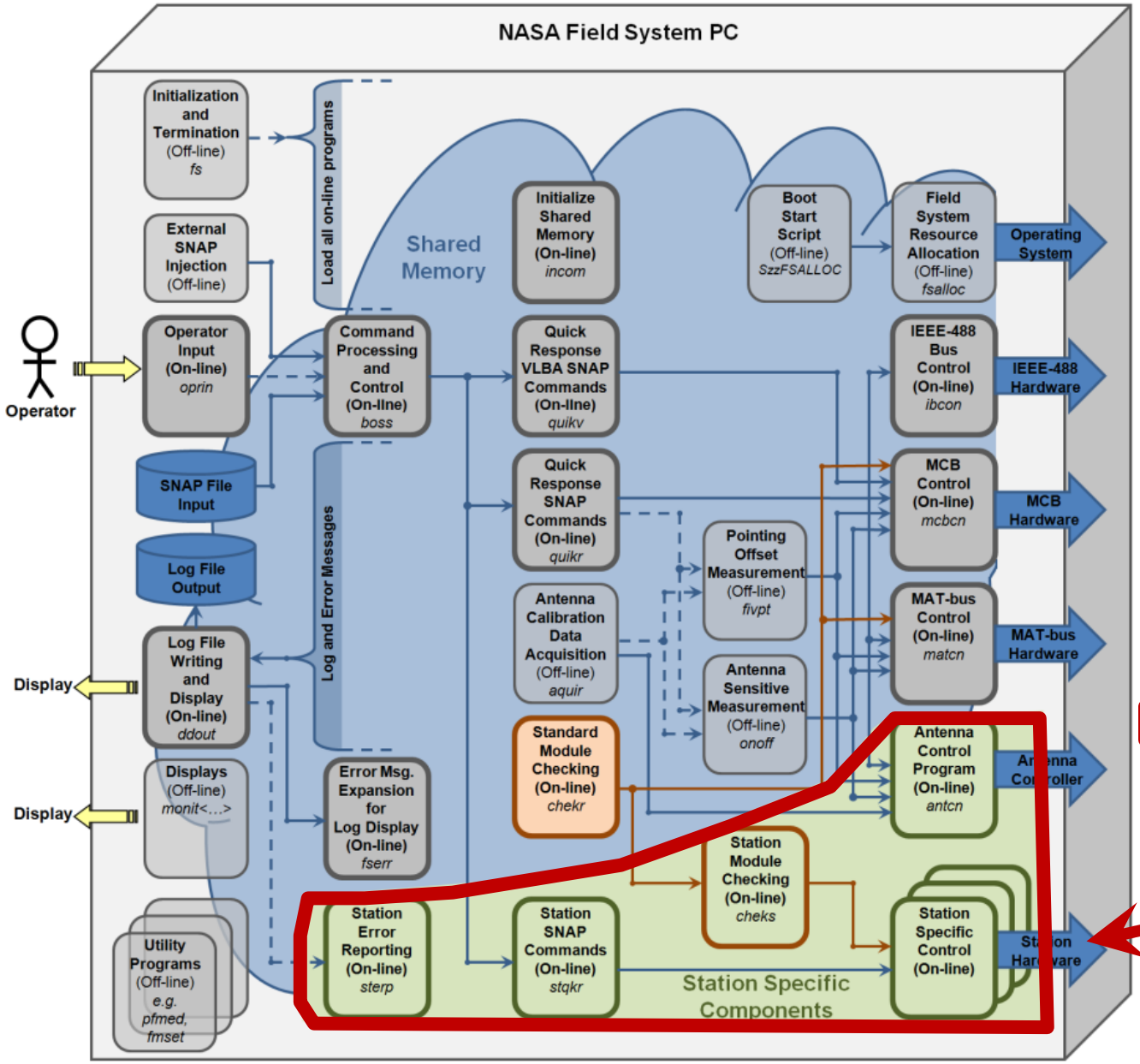
How to fill data sets of the FS?

How to add functionality to the FS?

How to extend number of parallel devices?

How to combine functionalities to the FS?

What does a station has to offer to the FS?



The NASA Field System can be split into six main layers:

1. Programs for hardware control (hardware driving)
2. Programs for (module) checking (monitoring)
3. Programs for the SNAP command interpretation
4. Programs for Command Processing and Control (coordination: «boss» or, e.g., the Antenna Calibration Data Acquisition «aquir»)
5. Programs for error reporting
6. Programs for user interfacing

The NASA Field System can be split into two categories, according to where the code is developed:

1. the general Field System programs from NASA/NVI (Himwich, Horsley, et. al.)
2. station code, individually programmed by station staff

What does a station has to offer to the FS?

Station-specific programs

Antenna Control („`antcn`“)

Activated in `dev.ctl`

Station specific commands („`stqkr`“)

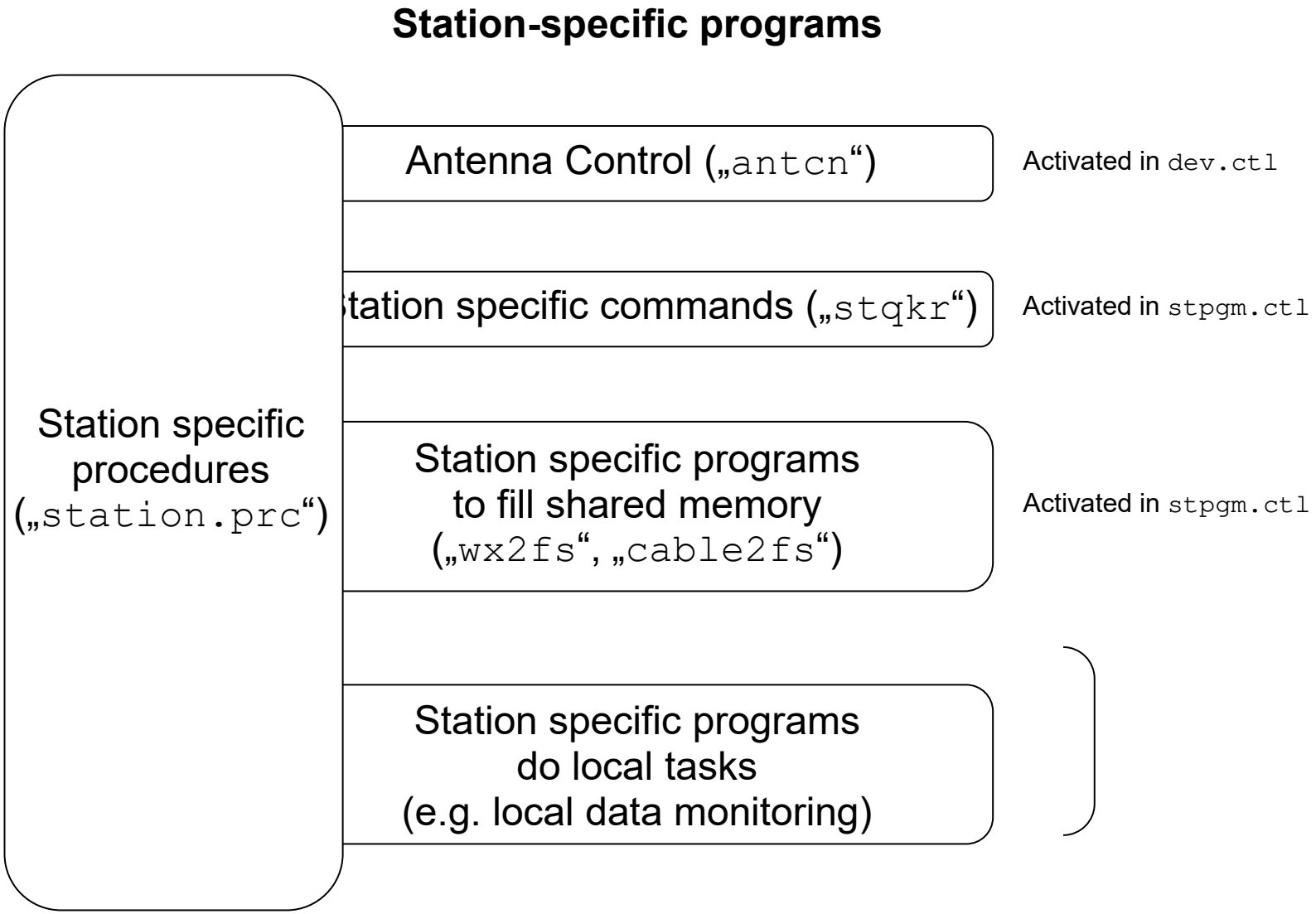
Activated in `stpgm.ctl`

Station specific programs
to fill shared memory
(„`wx2fs`“, „`cable2fs`“)

Activated in `stpgm.ctl`

Station specific programs
do local tasks
(e.g. local data monitoring)

What does a station has to offer to the FS?



What does a station has to offer to the FS?

Station-specific programs

```
cd /usr2/fs/st.default
      ➔ "Copied" to /usr2/

control    oper    proc    prog    sched    st-0.0.0    tle_files
                                             Station code
```

What does a station has to offer to the FS?

Station-specific programs

```
cd /usr2/fs/st.default
      ➔ "Copied" to /usr2/

control    oper    proc    prog    sched    st-0.0.0    tle_files
```

Station code

```
/usr2/fs-9.11.19/st.default/st-0.0.0/
      ➔ Copied to /usr2/st-x.x.x/ linked with /usr2/st/

Makefile    autoftp    cheks       include      metserver   ntpq_dummy
stalloc     stlib      tacdclient  antcn        bin         help
metclient   misc       pcald       sterp        stqkr
```

Antenna control

Station command interpreter

What does a station has to offer to the FS?

Station-specific programs

```
cd /usr2/fs/st.default
      ➔ "Copied" to /usr2/

control    oper    proc    prog    sched    st-0.0.0    tle_files
```

Station code

```
/usr2/fs-9.11.19/st.default/st-0.0.0/
      ➔ Copied to /usr2/st-x.x.x/ linked with /usr2/st/

Makefile    autoftp    cheks       include      metserver   ntpq_dummy
stalloc     stlib      tacdclient  antcn        bin         help
metclient   misc       pcald       sterp        stqkr
```

Antenna control

Station command interpreter

Station code can be very individual. Default programs are principally suggestions.
At least, you need:
- Antenna control program
  (usually „antcn“, but can have any name, must just fit to entries in /usr2/control/stpgm.ctl)
- Station QKR
  (usually „stqkr“, but can have any name, must just fit to entries in /usr2/control/stpgm.ctl)
- Meteo program
- Dotmon, cable (counter-reading) programs

Useful is:
- A directory with help pages
- A directory for binaries
- Maybe a directory for local control (configuration) files

TOW2025 - Seminar

FS Station Code

What about FS?
What does a station has to offer to the FS?
How to control your antenna from FS?
How to control your equipment from FS?
How to fill data sets of the FS?
How to add functionality to the FS?
How to extend number of parallel devices?
How to combine functionalities to the FS?

How to control your antenna from FS?

Station-specific programs

Antenna Control („antcn“) — Activated in dev.ctl

tation specific commands („stqkr“) — Activated in stpgm.ctl

Station specific procedures („station.prc“)

Station specific programs to fill shared memory („wx2fs“, „cable2fs“) — Activated in stpgm.ctl

Station specific programs do local tasks (e.g. local data monitoring)

How to control your equipment from FS?

Station QKR („`stqkr`“)

Sample „`/usr2/control/stcmd.ctl`“:

```
* /usr2/control/dev.ctl
********** Field System LU control file
/dev/null      GPIB board device name, gpib0, or serial port /dev/ttyS* for GPIB-232CT
/dev/null      Mark III MAT device name
9600           Mark III MAT baud rate
/dev/null      Mark III Data Buffer device name
9600           Mark III Data Buffer baud rate
antcn          Antenna interface
/dev/null      Barcode reader device name
/dev/null     VLBA MCB device name
57600          VLBA MCB baud rate
/dev/null      ATNF Dataset device name
38400          ATNF Dataset baud rate
```

Activate „antcn“ program

How to control your antenna from FS?

Antenna Control („antcn“)

Include section

```
/* antcn.c
 *
 * This is the stub version of antcn (ANTenna CoNtrol program).
 * This version sends a log message whenever it is called.
 */

...
/* Defined variables */
#define MINMODE 0   /* min,max modes for our operation */
#define MAXMODE 10

/* Include files */

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

#include "../../fs/include/params.h" /* FS parameters
*/
#include "../../fs/include/fs_types.h" /* FS header files
*/
#include "../../fs/include/fscom.h"  /* FS shared mem. structure
*/
#include "../../fs/include/shm_addr.h" /* FS shared mem. pointer
*/
```

Better in your program:
-I with path
in compiler call
and just file names
here, e.g.
#include <params.h>

Shared memory

```
struct fscom *fs;    = NULL;
```

Corresponding Functions to include section

```
/* Subroutines called */
void setup_ids();
void putpname();
void skd_run(), cls_clr();
int nsem_test();
void logit();
```

Functions have arguments => better define them
=> or directly use .h/.c file modules

>>>

How to control your antenna from FS?

Antenna Control („antcn“)

Init shared memory (just once!!!)

Define program name in FS environment

```
/* antcn main program starts here */
main()
{
  int ierr, nrec, nrecr;
  int dum = 0;
  int r1, r2;
  int imode,i,nchar;
  long ip[5], class, clasr;
  char buf[80], buf2[100];

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

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

  putpname("antcn");

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

>>>

How to control your antenna from FS?

Antenna Control („antcn“)

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

  imode = ip[0];
  class = ip[1];
  nrec = ip[2];


  nrecr = 0;
  clasr = 0;


  if (imode < MINMODE || imode > MAXMODE) {
    ierr = -1;
    goto End;
  }
```

Wait for new command and receive command ID

Split command ID:
- **Mode (com. number)**
- **Class (for com. receive)**
- **Number of msg. parts**

Init return values

Check error in command ID

>>>

```
/* Handle each mode in a separate section */

  switch (imode) {

    case 0:         /* initialize */
      ...
    case 1:         /* source= command */
      ...
    case 2:         /* offsets         */
      ...
    case 3:         /* onsource command with error message */
      ...
    case 4:         /* direct antenna= command */
      ...
    case 5:         /* onsource command with no error logging */
      ...
    case 6:         /* reserved */
      ...
    case 7:         /* onsource command with additional info  */
      ...
    case 8:         /* Station dependent detectors access */
      ...
    case 9:         /* Satellite tracking mode */
      ...
    case 10:       /* normally triggered on FS termination if
                      evironment variable FS_ANTCN_TERMINATION
                      has been defined */
      ...
  default:
      ...
  }  /* end of switch */
```

Command switch and Call of antenna interface

>>>

Command Processing Loop

Better: while (true)

Prepare answer
- **Class**
- **Number of msg. parts**
- **Error number**
- **Application letter code**

```
End:
  ip[0] = clasr;
  ip[1] = nrecr;
  ip[2] = ierr;
  memcpy(ip+3,"AN",2);
  ip[4] = 0;
  goto Continue;
}
```

How to control your antenna from FS?

Antenna Control („antcn“)

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

  imode = ip[0];
  class = ip[1];
  nrec = ip[2];



  nrecr = 0;
  clasr = 0;


  if (imode < MINMODE || imode > MAXMODE) {
    ierr = -1;
    goto End;
  }
```

>>>

Command Processing Loop

```
/* Hand

  switc

    cas
      ...
    case 1:        /* source= command */
      ...
    case 2:        /* offsets         */
      ...
    case 3:        /* onsource command with error message */
      ...
    case 4:        /* direct antenna= command */
      ...
    case 5:        /* onsource command with no error logging */
      ...
    case 6:        /* reserved */
      ...
    case 7:        /* onsource command with additional info  */
      ...
    case 8:        /* Station dependent detectors access */
      ...
    case 9:        /* Satellite tracking mode */
      ...
    case 10:      /* normally triggered on FS termination if
                     evironment variable FS_ANTCN_TERMINATION
                     has been defined */
      ...
  default:
      ...
}  /* end of switch */
```

>>>

On-source must be very conservative and not say the antenna is on-source when it isn’t. This especially important when a new source or offset has been requested. We don’t want the status for the previous source/offset, we want the latest commands taken into account (better say off-source sometimes when on-source than every say on when off).

```
End:
  ip[0] = clasr;
  ip[1] = nrecr;
  ip[2] = ierr;
  memcpy(ip+3,"AN",2);
  ip[4] = 0;
  goto Continue;
}
```

How to control your antenna from FS?

Antenna Control („antcn“)

Sample:

```
case 0:         /* initialize */
{
    strncpy(acAnswerText, "Initializing Vertex ACU antenna interface", 79);
    logit(acAnswerText, 0, NULL);
    iFSErrorNumber = 0;

    if (iInitError)
    {
        strncpy(acAnswerText, "[ERROR] FS shared memory: init pointer is NULL", 79);
        logit(acAnswerText, 0, NULL);
        logit("", -5, "AN");
        exit(1);
    }
    fs->ionsor = 0;
    if (usCOpenInterface ("127.0.0.1", 0,
                          &ACUDescriptor) == CACUNOK)
    {
        strncpy(acAnswerText, "[ERROR] ACU: can't open interface", 79);
        logit(acAnswerText, 0, NULL);
        logit("", -5, "AN");
        ACUDescriptor = NULL;
    }
    if (usCStopAllAxis (&ACUDescriptor) == CACUNOK)
    {
        logit("", -5, "AN");
        strncpy(acAnswerText, "[ERROR] ACU: can't stop movement", 79);
        logit(acAnswerText, 0, NULL);
    }
    usFSTrackingMode = FSTRACKINGMODE_IDLE;
    if (usCMeteoOpenInterface ("127.0.0.1", 0,
                               &MeteoDescriptor) == CMETEO_NOK)
    {
        strncpy(acAnswerText, "[ERROR] Meteo: can't open interface", 79);
        logit(acAnswerText, 0, NULL);
        logit("", -5, "AN");
        MeteoDescriptor = NULL;
        usUseMeteo = 0;
    }
    break;
}
```

Write message to log

Write message to log defined in „/usr2/fs/control/fserr.ctl“

```
...
""
AN    -5
Error returned from antenna
...
```

or in „/usr2/control/sterr.ctl“

How to control your antenna from FS?

Antenna Control („antcn“)

Sample:

```
case 1:         /* source= command */
{
    ...
    /* Convert RADEC string */

    ...
    /* Get parameters for refraction correction */

    ...
    if (usCMoveToRaDecPosition (&ACUDescriptor,
                                (short)SFSTime.iYear,
                                (short)SFSTime.iDoY,
                                (short)SFSTime.iHour,
                                (short)SFSTime.iMinute,
                                (short)SFSTime.iSecond,
                                0.0,
                                dRightAscensionHour,
                                dDeclinationDegree,
                                SSourceStatus.acSourceStatus,
                                usWrapIdentifier,
                                usEpochIdentifier) == CACUNOK)
    {
        logit("", -5, "AN");
        strncpy(acAnswerText, "[ERROR] ACU: can't command position", 79);
        logit(acAnswerText, 0, NULL);
        goto Continue;
    }

    ...
    break;
}
```

SNAP:
source=1909+161,191158.26,161146.9,2000.0,cw
source RA DEC YEAR Cable wrap
Catalog

Please also read cable-wrap memo to get cable-wrap right:
https://ivscc.gsfc.nasa.gov/meetings/tow2013/Himwich.Sem2.pdf

How to control your antenna from FS?

Antenna Control („antcn“)

Sample:

```
case 4:         /* direct antenna= command */
{
     if (class == 0)
       goto End;
     for (i=0; i<nrec; i++) {
       strcpy(buf2,"Received message for antenna: ");
       nchar = cls_rcv(class,buf,sizeof(buf),&r1,&r2,dum,dum);
       buf[nchar] = '\0';  /* make into a string */
       strcat(buf2,buf);
       logit(buf2,0,NULL);
       ...
       for (iCommandCharIndex = 0; iCommandCharIndex < strlen(buf); ++iCommandCharIndex)
       {
           acCommand[iCommandCharIndex] = (char)toupper((int)buf[iCommandCharIndex]);
       }
       ...
       /*********************************************/
       /* antenna=halt or antenna=stop              */
       /*********************************************/
       if (strlen(acCommand) == 4 &&
           (!strncmp ("HALT", acCommand, 4) || !strncmp ("STOP", acCommand, 4)) &&
           nrec == 1)
       {
           usFSTrackingMode = FSTRACKINGMODE_IDLE;
           strncpy(acAnswerText, "ACU: stop all axis", 79);
           logit(acAnswerText, 0, NULL);
           if (usCStopAllAxis (&ACUDescriptor) == CACUNOK)
           {
               logit("", -105, "AN");
           }
           else
           {
               strcpy(buf,"ACK");
               cls_snd(&clasr,buf,3,0,0);
               nrecr += 1;
           }
       }
       /* OR: cls_clr(class);  */
     break;
}
```

Receive command

Prepare command

SNAP: antenna=halt

Process command

Reply command

TOW2025 - Seminar

FS Station Code

What about FS?
What does a station has to offer to the FS?
How to control your antenna from FS?
How to control your equipment from FS?
How to fill data sets of the FS?
How to add functionality to the FS?
How to extend number of parallel devices?
How to combine functionalities to the FS?

How to control your equipment from FS?

Station-specific programs

Antenna Control („antcn“) — Activated in dev.ctl

Station specific commands („stqkr“) — Activated in stpgm.ctl

Station specific procedures („station.prc“)

Station specific programs to fill shared memory („wx2fs“, „cable2fs“) — Activated in stpgm.ctl

Station specific programs do local tasks (e.g. local data monitoring)

How to control your equipment from FS?

Classic C-Code

Station QKR („stqkr“): **stqkr.c**

```
/* stqkr - C version of station command controller */

#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"
             /* shared memory pointer */

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

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();
```

Initialize

Loop

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

      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:
/*          call routine here to handle a task */
            break;
         case 2:
/*          call routine here to handle next task */
            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;
}
```

Wait for incoming orders

Receive command and arguments

Parse (interprete) command and arguments

Performe action according to command => Call function

Prepare return Values => ACK

➔ **Principle similar to „antcn“**

How to control your equipment from FS?

C/C++-Code

Station QKR („stqkr“): **stqkr.cpp**

```
#include <stdio.h>
#include <stdlib.h>
#include "fsshm.h„
#include "simple_structured_conf.hpp"
#include "meteo.hpp"
#include "rxmon.hpp"
#include "testequip.hpp"

int main (int iArgc, char * pcArgv[])
{
    BOSSCOM BOSSCOMIdentifier = NULL;  /// BOSSCOMIdentifier = Identification of current communication (it is a pointer to the data set
                                       /// "ip" of the field system)
    char acProgramName[6] = "stqkr";   /// pcProgramName = name of the program, which waits for a message
    char acReceivedMessage[4096];      /// acReceivedMessage = complete message sent from boss
    CSimpleStructuredConf CConfiguration; /// CConfiguration = the whole configuration parameters of the stqkr program
    unsigned long ulCurrentLineNumber; /// ulCurrentLineNumber = line number of the configuration file, where an error occured
    std::string strCurrentTag;         /// strCurrentTag = tag in the configuration file, where the error occured

    printf ("stqkr: Startup ...\n");

    /// Open communication to NASA FS boss
    if (usOpenBossCommunication (&BOSSCOMIdentifier))
    {
        printf ("[ERROR] stqkr: Cannot open connection to NASA FS boss\n");
        return 1;
    }

    /// Check program parameters
    if (iArgc != 2)
    {
        (void) usPrintError2Log (&BOSSCOMIdentifier, acProgramName, "SQ", -1, "");
        goto CloseBossCommunication;
    }

    /// Read configuration
    if (CConfiguration.usReadConfig (pcArgv[1], ulCurrentLineNumber, strCurrentTag))
    {
        if (strCurrentTag.empty())
        {
            (void) usPrintError2Log (&BOSSCOMIdentifier, acProgramName, "SQ", -2, "No file found");
        }
        else
        {
            (void) usPrintError2Log (&BOSSCOMIdentifier, acProgramName, "SQ", -2, "Error in line %ld arround tag '%s'",
                                     ulCurrentLineNumber, strCurrentTag.c_str());
        }
        goto CloseBossCommunication;
    }
```

Initialize

How to control your equipment from FS?

C/C++-Code

Station QKR („stqkr“): **stqkr.cpp**

```
#include <stdio.h>
#include <stdlib.h>
#include "fsshm.h„
#include "simple_structured_conf.hpp"
#include "meteo.hpp"
#include "rxmon.hpp"
#include "testequip.hpp"

int main (int iArgc, char * pcArgv[])
{
    BOSSCOM BOSSCOMIdentifier = NULL;  /// BOSSCOMIdentifier
                                       /// "ip" of the field
    char acProgramName[6] = "stqkr";   /// pcProgramName = n
    char acReceivedMessage[4096];      /// acReceivedMessage
    CSimpleStructuredConf CConfiguration; /// CConfiguration
    unsigned long ulCurrentLineNumber; /// ulCurrentLineNumb
    std::string strCurrentTag;         /// strCurrentTag = t

    printf ("stqkr: Startup ...\n");

    /// Open communication to NASA FS boss
    if (usOpenBossCommunication (&BOSSCOMIdentifier))
    {
        printf ("[ERROR] stqkr: Cannot open connection to NA
        return 1;
    }

    /// Check program parameters
    if (iArgc != 2)
    {
        (void) usPrintError2Log (&BOSSCOMIdentifier, acProgr
        goto CloseBossCommunication;
    }

    /// Read configuration
    if (CConfiguration.usReadConfig (pcArgv[1], ulCurrentLin
    {
        if (strCurrentTag.empty())
        {
            (void) usPrintError2Log (&BOSSCOMIdentifier, acP
        }
        else
        {
            (void) usPrintError2Log (&BOSSCOMIdentifier, acProgramName, "SQ", -2, "Error in line %ld arround tag '%s'",
                                     ulCurrentLineNumber, strCurrentTag.c_str());
        }
        goto CloseBossCommunication;
    }
```

Module „fsmonitor“ (Wettzell)

Communication with NASA FS boss

```
typedef long * BOSSCOM;
unsigned short usOpenBossCommunication (BOSSCOM * pBOSSCOMIdentifier);
unsigned short usCloseBossCommunication (BOSSCOM * pBOSSCOMIdentifier);
unsigned short usWaitForMessageFromBoss (BOSSCOM * pBOSSCOMIdentifier,
                                         char * pcProgramName,
                                         char acReceivedMessage[4096]);
long lGetCommandIdentifierOfIncomingCommand (BOSSCOM * pBOSSCOMIdentifier);
long lGetIPCClassNumberForIncomingMessage (BOSSCOM * pBOSSCOMIdentifier);
long lGetNumberOfElementsInIncomingMessage (BOSSCOM * pBOSSCOMIdentifier);
unsigned short usAcknowledgeMessageProcessing (BOSSCOM * pBOSSCOMIdentifier);
unsigned short usPrintMessage2Log (BOSSCOM * pBOSSCOMIdentifier,
                                   const char acProgramNameInput[6],
                                   const char * pcFormat,
                                   ...);
unsigned short usPrintError2Log (BOSSCOM * pBOSSCOMIdentifier,
                                 const char acProgramNameInput[6],
                                 const char acFSErrorIdentCodeInput[3],
                                 int iErrorCode,
                                 const char * pcFormat,
                                 ...);
unsigned short usReplyMessageToBoss (BOSSCOM * pBOSSCOMIdentifier,
                                     const char * pcFormat,
                                     ...);
unsigned short usWriteSatEphemToFile(void);
```

How to control your equipment from FS?

C/C++-Code

Station QKR („stqkr“): **stqkr.cpp**

```
    /// Start processing loop
    while (1)
    {
        /// Wait for incoming messages from boss
        if (usWaitForMessageFromBoss (&BOSSCOMIdentifier,
                                      acProgramName,
                                      acReceivedMessage))
        {
            continue;
        }

        /// Switch between the different commands according to the class number
        switch (lGetIPCClassNumberForIncomingMessage (&BOSSCOMIdentifier))
        {
            // *********************************************************
            // Command "wx"
            // *********************************************************
            case 100: /* wx */
            {
            }
            // *********************************************************
            // Command "rx" and "rxall"
            // *********************************************************
            case 200: /* rx */
            case 201: /* rxall */
            {
            }
            ...

        if (usAcknowledgeMessageProcessing (&BOSSCOMIdentifier))
        {
            continue;
        }
    }

CloseBossCommunication:
    /// Close communication to NASA FS boss
    if (usCloseBossCommunication (&BOSSCOMIdentifier))
    {
        printf ("[ERROR] stqkr: Cannot close connection\n");
        return 1;
    }

    printf ("[ERROR] stqkr: While loop failed\n");
    return 1;
}
```

Wait for incoming Orders and Receive command and arguments

Identify command

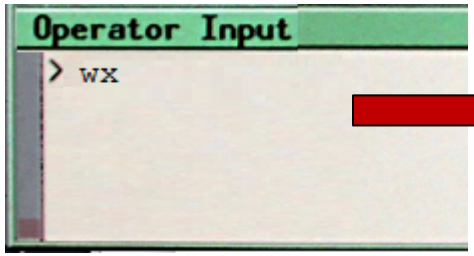
Interprete and Performe action according to command => Call function

Prepare return values

How to control your equipment from FS?

Station QKR („stqkr“)

Sample „/usr2/control/stcmd.ctl“:




```
**********STATION SPECIFIC COMMANDS***************
*COMMAND      SEG SUBPA BO
sthelp       stq 00001 01 FFFFFFFFFFFF
wx           stq 00100 01 FFFFFFFFFFFF
wxreport     stq 00101 01 FFFFFFFFFFFF
wxreference  stq 00102 01 FFFFFFFFFFFF
wxwind       stq 00103 01 FFFFFFFFFFFF
wxwindstow   stq 00104 01 FFFFFFFFFFFF
rx           stq 00200 01 FFFFFFFFFFFF
dotmon       stq 00210 01 FFFFFFFFFFFF
cable        stq 00212 01 FFFFFFFFFFFF
maser        stq 00213 01 FFFFFFFFFFFF
```

Classic C-Code

```
switch (isub) {
   ...
   case 100:
   ...
}
```

C/C++-Code

```
switch (lGetIPCClassNumberForIncomingMessage
                (&BOSSCOMIdentifier))
{
    case 100: /* wx */
    {
        ...
    }
```

How to control your equipment from FS?

Station QKR („stqkr“)

Sample „/usr2/control/sterr.ctl“:

```
""
AN -200
antenna warning (see antenna=status).
""
CA -1
Cable data are not correct
""
SQ -1
stqkr startup failed, because of missing program argument for the configuration file.
""
SQ -2
stqkr startup failed, because of errorneous configuration file.
""
SQ -3
there is no procedure defined for the entered station command.
""
SQ -4
help output does not work.
""
SQ -100
command wx failed.
""
SQ -101
meteo call false.
""
SQ -102
meteo open failed.
""
SQ -103
meteo close failed.
""
SQ -104
meteo read failed.
""
--More--(35%)
```

Classic C-Code

```
logit("", -101, "SQ");
```

```
or: usPrintError2Log (…)
```

C/C++-Code

Log file

```
2023.111.09:58:37.02?ERROR sq -101 meteo call false.
```

How to control your equipment from FS?

Difference 32-bit to 64-bit in station code

32-bit	===>	64-bit
long	===>	int

See:

https://nvi-inc.github.io/fs/misc/64-bit_conversion.html

For automatic conversion of station code see:

https://github.com/dehorsley/unlongify

But better (my personal opinion):

- Install a completely new computer with 64-bit Debian
- Install FSL10 or greater for 64-bit
- Copy your station code
- Go manually through your code and change it manually to int, where required (so that you can also validate address operations etc.)

How to control your equipment from FS?

Difference 32-bit to 64-bit in station code

32-bit ===> 64-bit
long ===> int

```
/* from /usr2/fs/clib/skd_util.c */
void skd_wait (char name[ 5],
               long ip[5],
               unsigned centisec);
void skd_run (char name[5],
              char w,
              long ip[5]);
/* from /usr2/fs/clib/cls_util.c */
int cls_rcv  long lIPCClassNumberForIncomingCommand,
             char * acAnswerTextfer,
             int length,
             int * rtn1,
             int * rtn2,
             int msgflg,
             int save);
void cls_snd  long * lIPCClassNumberForIncomingCommand,
              char * acAnswerTextfer,
              int length,
              int parm3,
              int parm4);
void cls_clr (long lIPCClassNumberForIncomingCommand);
```

```
/* from /usr2/fs/clib/skd_util.c */
void skd_wait (char name[ 5],
               int ip[5],
               unsigned centisec);
void skd_run (char name[5],
              char w,
              int ip[5]);
/* from /usr2/fs/clib/cls_util.c */
int cls_rcv (int iIPCClassNumberForIncomingCommand,
             char * acAnswerTextfer,
             int iength,
             int * rtn1,
             int * rtn2,
             int msgflg,
             int save);
void cls_snd (int * iIPCClassNumberForIncomingCommand,
              char * acAnswerTextfer,
              int iength,
              int parm3,
              int parm4);
void cls_clr (int iIPCClassNumberForIncomingCommand);
```

```
/* antcn main program starts here */
main()
{
  int ierr, nrec, nrecr;
  int dum = 0;
  int r1, r2;
  int imode,i,nchar;
  long ip[5], class, clasr;
  char buf[80], buf2[100];
```

```
/* antcn main program starts here */
main()
{
  int ierr, nrec, nrecr;
  int dum = 0;
  int r1, r2;
  int imode,i,nchar;
  int ip[5], class, clasr;
  char buf[80], buf2[100];
```

How to control your equipment from FS?

Difference 32-bit to 64-bit in station code

32-bit	===>	64-bit
long	===>	int

Expect more warnings

```
../src/antcn.c:2171:25: warning: âstrncpyâ output truncated before terminating nul copying 79 bytes from a s
tring of the same length [-Wstringop-truncation]
                        strncpy(acAnswerText, "**********************************************************************
********************", 79);
                        ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
../srcext/fsmonitor/fsshm.c:687:5: warning: âmemsetâ used with length equal to number of elements without mu
ltiplication by element size [-Wmemset-elt-size]
    memset (it, 0, 6);
    ^~~~~~
```

```
../srcext/fsmonitor/fsshm.c:2061:9: warning: unused variable âipâ [-Wunused-variable]
     int ip[5];
         ^~
```

```
../srcext/fsmonitor/fsshm.c:2589:70: warning: format â%liâ expects argument of type âlong intâ, but argument
 9 has type âintâ [-Wformat=]
        fprintf(pFilePointer, "   %4d %02d %02d   %02d:%02d:%02d %13li %17.8f %10.5f %10.5f %10.5f %10.5f\n
", sYear, sMon, sDay, sHour, sMin, sSec, shm_addr->ephem[iIndex].t, dJD,
                                                              ~~~~^
                                         ~~~~~~~~~~~~~~~~~~~~~~~~~~
                                                             %13i
```

```
../srcext/fsmonitor/fsshm.c:1440:13: note: âsnprintfâ output 8 bytes into a destination of size 7
            snprintf(SFSMark5->acCheckTime[iBank], 7, "       ");
            ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
../srcext/fsmonitor/fsshm.c:1452:60: warning: âsnprintfâ output truncated before the last format character [
-Wformat-truncation=]
            snprintf(SFSMark5->acGB[iBank], 10, "         ");
                                                ^
```

…

➔ Take warnings seriously and fix all of them!

TOW2025 - Seminar

FS Station Code

What about FS?
What does a station has to offer to the FS?
How to control your antenna from FS?
How to control your equipment from FS?
How to fill data sets of the FS?
How to add functionality to the FS?
How to extend number of parallel devices?
How to combine functionalities to the FS?

How to fill data sets of the FS?

Station-specific programs

Antenna Control („antcn“)

Activated in dev.ctl

tation specific commands („stqkr“)

Activated in stpgm.ctl

Station specific
procedures
(„station.pr

Station specific programs
to fill shared memory
(„wx2fs“, „cable2fs“)

Activated in stpgm.ctl

Station specific programs
do local tasks
(e.g. local data monitoring)

How to fill data sets of the FS?

Sample for an own meteorological sensor

```
/* Include section similar to stqkr.c */

...

struct stcom *st;
struct fscom *fs;


main()
{
  ...

  setup_ids();
  fs = shm_addr;
  if (nsem_test(NSEMNAME) != 1)
  {
      /* ERROR */
  }
  while (1==1)
  {
    ...
    if (usGetMeteoFromReference (fTempWX,
                                 fPresWX,
                                 fHumiWX) != METEO_OK)
     {
         logit("", -1, „WX");
         ...
     }
     ...

     shm_addr->tempwx = fTempWX;
     shm_addr->humiwx = fHumiWX;
     shm_addr->preswx = fPresWX;

     ...
  }
}
```

How to fill data sets of the FS?

Sample for an own meteorological sensor

Img. source:
https://www.bkg.bund.de/

```
/* Include section similar to stqkr.c */

...

struct stcom *st;
struct fscom *fs;


main()
{
  ...

  setup_ids();
  fs = shm_addr;
  if (nsem_test(NSEMNAME) != 1)
  {
      /* ERROR */
  }
  while (1==1)
  {
    ...
    if (usGetMeteoFromReference (fTempWX,
                                 fPresWX,
                                 fHumiWX) != METEO_OK)

    {
        logit("", -1, „WX");
        ...
    }
    ...

    shm_addr->tempwx = fTempWX;
    shm_addr->humiwx = fHumiWX;
    shm_addr->preswx = fPresWX;

    ...
  }
}
```

System Status Monitor

WETTZELL		2023.111.23:55:15	UT		TEMP	10.8	2000+472		SLEWING			
MODE	RATE	07:15:00	NEXT		HUMID	53.9	RA	20h 02m 10.42s				
		SCHED=	none	LOG=	station	PRES	942.2	DEC	47d 25m		(2000)	
		TSYS:	IFA	IFB	IFC	IFD	CABLE	0.006370	AZ	59.7837	EL	39.7989
			37	55	132	-47	WIND	5.04	DIR	74		
NO CHECK: rx												

How to add functionality to the FS?

Control files `/usr2/control/stpgm.ctl`

FS programs

Station programs

```
* Put site-specific programs here that should
* be started by the Field System.
* antcn should not be here.
erchk x xterm -name erchk -e erchk &
moni2 x xterm -name monit2 -e monit2 &
scnch x xterm -name scnch  -e 'fsclient -n -w -s | grep /!\*scan_check..' &
wx2fs n wx2fs > /dev/null 2> /dev/null &
stqkr n stqkr /usr2/st/control/stqkr.conf &
cable2fs n cable2fs /usr2/st/control/cable2fs.conf &
patch_mark6.sh x /usr2/st/bin/patch_mark6.sh init:192.168.1.1:14  &
```

TOW2025 - Seminar

FS Station Code

What about FS?
What does a station has to offer to the FS?
How to control your antenna from FS?
How to control your equipment from FS?
How to fill data sets of the FS?
How to add functionality to the FS?
How to extend number of parallel devices?
How to combine functionalities to the FS?

How to add functionality to the FS?

Station-specific programs

Station specific procedures („station.prc“)

Antenna Control („antcn“) — Activated in dev.ctl

tation specific commands („stqkr“) — Activated in stpgm.ctl

Station specific programs to fill shared memory („wx2fs“, „cable2fs“) — Activated in stpgm.ctl

Station specific programs do local tasks (e.g. local data monitoring)

How to command the FS in your code?

Command injection

`/usr2/fs/bin/inject_snap`

`/usr2/fs/bin/inject_snap -w log` ➔ Output of current log filename
e.g. `log/station`

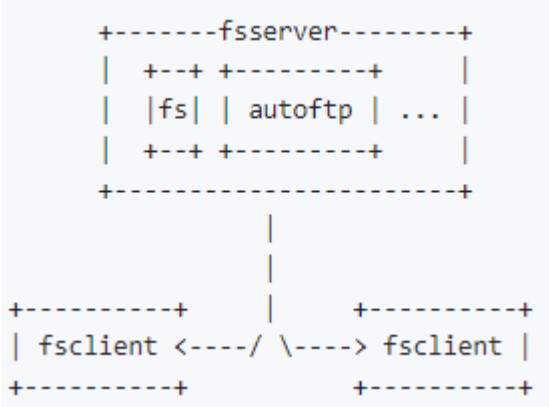
`/usr2/fs/bin/inject_snap <command>` ➔ Send command to FS
`e.g.`
`/usr2/fs/bin/inject_snap wx`

`/usr2/fs/bin/inject_snap '" Test'` ➔ Send comment to FS

How to read answers from the FS in your code?

Command reply catching („streamlog“)

FS Display Server to get log messages



“streamlog” is recommended for FS 10.2 or greater to get log messages.

https://github.com/nvi-inc/fs/blob/main/misc/display_server.md

How to read answers from the FS in your code?

Command reply catching („streamlog“)

`/usr2/fs/bin/fsclient`

`/usr2/fs/bin/fsclient -n` ➔ Get all log messages on standard out

`/usr2/fs/bin/fsclient -s -n` ➔ Get all log messages on standard out inclusively some historic lines

```
12:46:47;rx=dewar?
12:46:47/rx/dewar,NOK,19.00,0.00,292.00,1.0900e+01
15:14:14;-help
15:14:14?ERROR sp   -4 Unrecognized name (not a function or procedure).
15:17:12;log
15:17:12/log/station
15:19:02;wx
15:19:02#stqkr#wx/Used meteo site: GOW Meteo Database (Wetterstation Wettzell)
15:19:02#stqkr#wx/Height of pressure sensor: 656.025 m
15:19:02/wx/10.30,937.80,52.00
15:20:51;" Test
15:29:51;wx
15:29:51#stqkr#wx/Used meteo site: GOW Meteo Database (Wetterstation Wettzell)
15:29:51#stqkr#wx/Height of pressure sensor: 656.025 m
15:29:51/wx/9.90,938.00,53.60
15:29:55;wx
15:29:55#stqkr#wx/Used meteo site: GOW Meteo Database (Wetterstation Wettzell)
15:29:55#stqkr#wx/Height of pressure sensor: 656.025 m
15:29:55/wx/9.90,938.00,53.60
```

==> pipe the output to a program which can read it as standard in so that you can use it in other programs or scripts e.g.

`fsclient -s -n | grep "wx"`

End with Ctrl - C

How to add functionality to the FS?

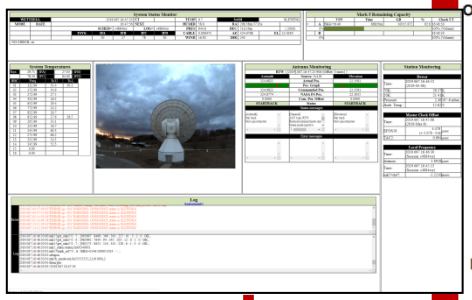
SNAP system calls

Sample SNAP file of schedule r4999wz

```
scan_name=140-1933,r4999,wz,60,60
source=3c418,203837.03,511912.7,2000.0,ccw
setupsx
!2021.140.19:33:07
preob
!2021.140.19:33:17
sy=cmd2flexbuff.py net2file=open:/raid/r4999wz/r4999_wz_140-1933,n ;
data_valid=on
midob
!2021.140.19:34:17
data_valid=off
sy=cmd2flexbuff.py net2file = close
postob
```

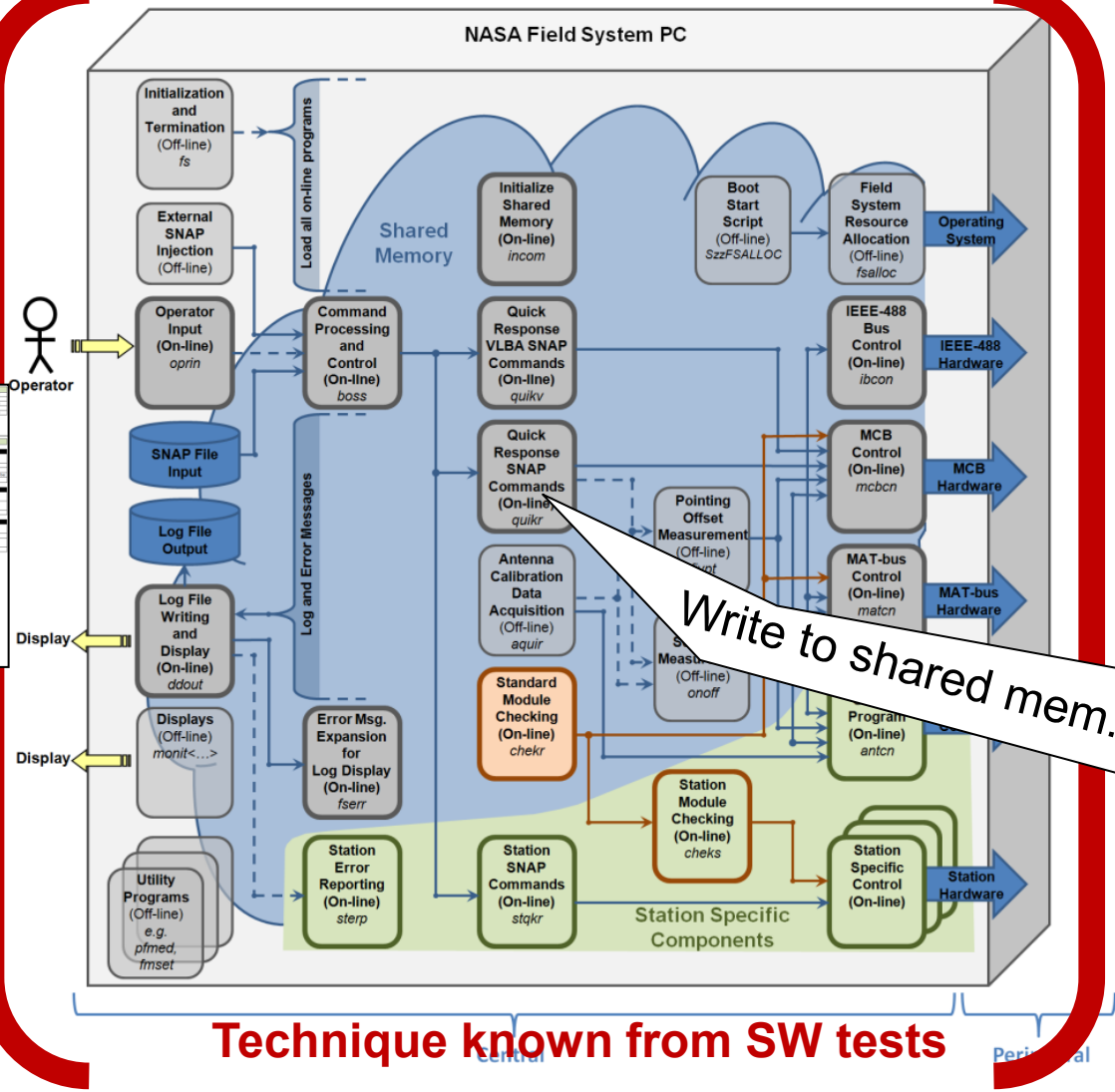
How to add functionality to the FS?

Modern controlling and monitoring



NASA FS Output remotely accessable

e.g. Logs Errors …



e.g. Antenna Timing Meteo Rx …

TOW2025 - Seminar

FS Station Code

What about FS?
What does a station has to offer to the FS?
How to control your antenna from FS?
How to control your equipment from FS?
How to fill data sets of the FS?
How to add functionality to the FS?
How to extend number of parallel devices?
How to combine functionalities to the FS?

How to extend number of parallel devices?

Sample 2 DBBCs:

`/usr2/control/dbbad.ctl`

➔ **Commands**

```
...
dbbc=power=1
...
fila10g=arp off
...
```

`/usr2/control/dbba2.ctl`

➔ **Commands**

```
...
dbbc2=power=1
...
fila10g2=arp off
...
```

But what if you have more than the allowed standard devices?

Attention … here comes a hack!!!

How to extend number of parallel devices?

But what if you have more than the allowed standard devices?

`socat` – Multipurpose relay

e.g. `socat TCP-LISTEN:142,fork,reuseaddr TCP:192.168.1.1:142`

Extend `station.prc`

```
define mydev3        00000000000x
sy=/usr2/st/bin/myscript_device3.sh > /dev/null 2> /dev/null
enddef
```

Script `myscript_device3.sh`

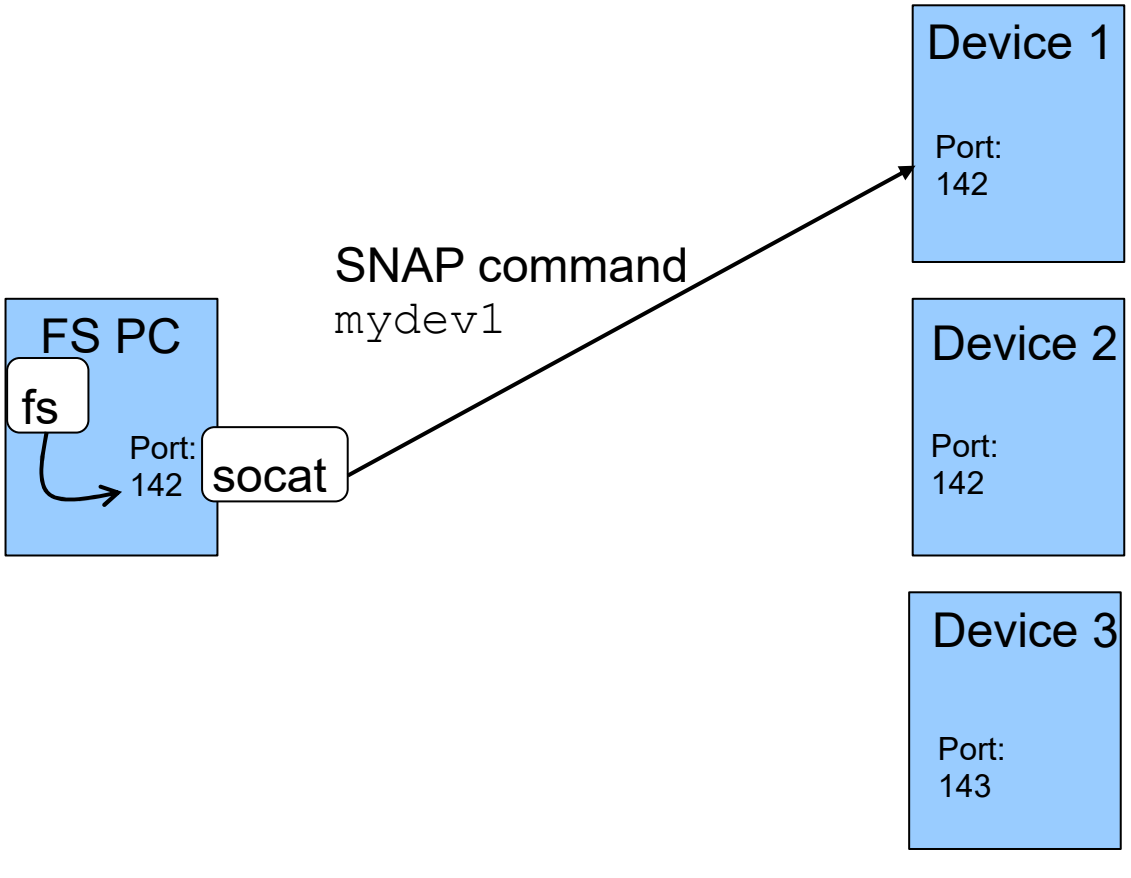
```
socat_basic_call="socat TCP-LISTEN:140,fork,reuseaddr TCP:"
# Kill previous patching
COMMAND="ps ax | grep \"${socat_basic_call}\" | grep -v grep | grep -o -E \"^[ ]*[0-9]+\" | tr '\n' ' '"
SOCAT_PIDS=`eval $COMMAND`
if [[ -n $SOCAT_PIDS ]]; then
    kill -9 $SOCAT_PIDS
    #echo -e "Killing processes with \"kill -9 $SOCAT_PIDS\""
fi
# Start patching of communication with socat
SOCAT_CALL="${socat_basic_call}192.168.1.1:143 > /dev/null 2> /dev/null &"
eval $SOCAT_CALL
```

How to extend number of parallel devices?

But what if you have more than the allowed standard devices?

`socat` – Multipurpose relay

e.g. `socat TCP-LISTEN:142,fork,reuseaddr TCP:192.168.1.1:142`

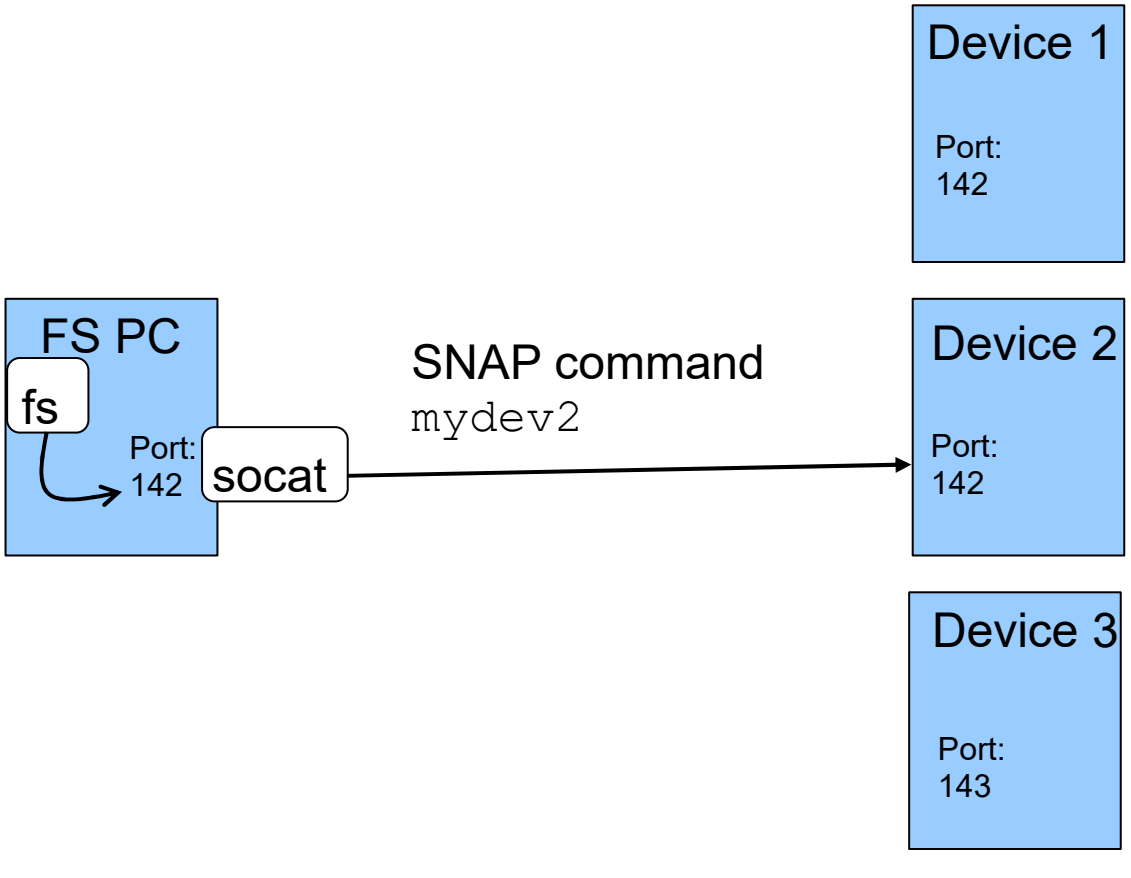


How to extend number of parallel devices?

But what if you have more than the allowed standard devices?

`socat` – Multipurpose relay

e.g. `socat TCP-LISTEN:142,fork,reuseaddr TCP:192.168.1.2:142`

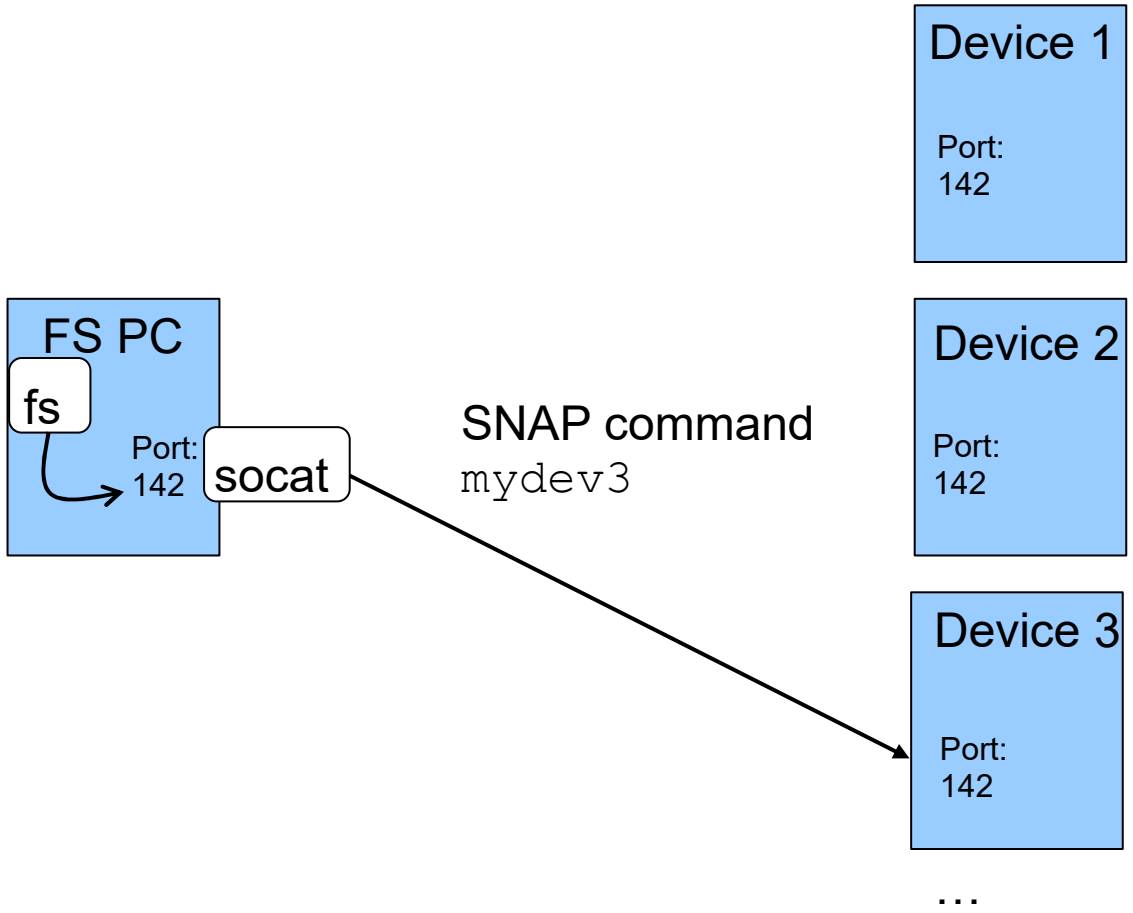


How to extend number of parallel devices?

But what if you have more than the allowed standard devices?

`socat` – Multipurpose relay

e.g. `socat TCP-LISTEN:142,fork,reuseaddr TCP:192.168.1.3:142`

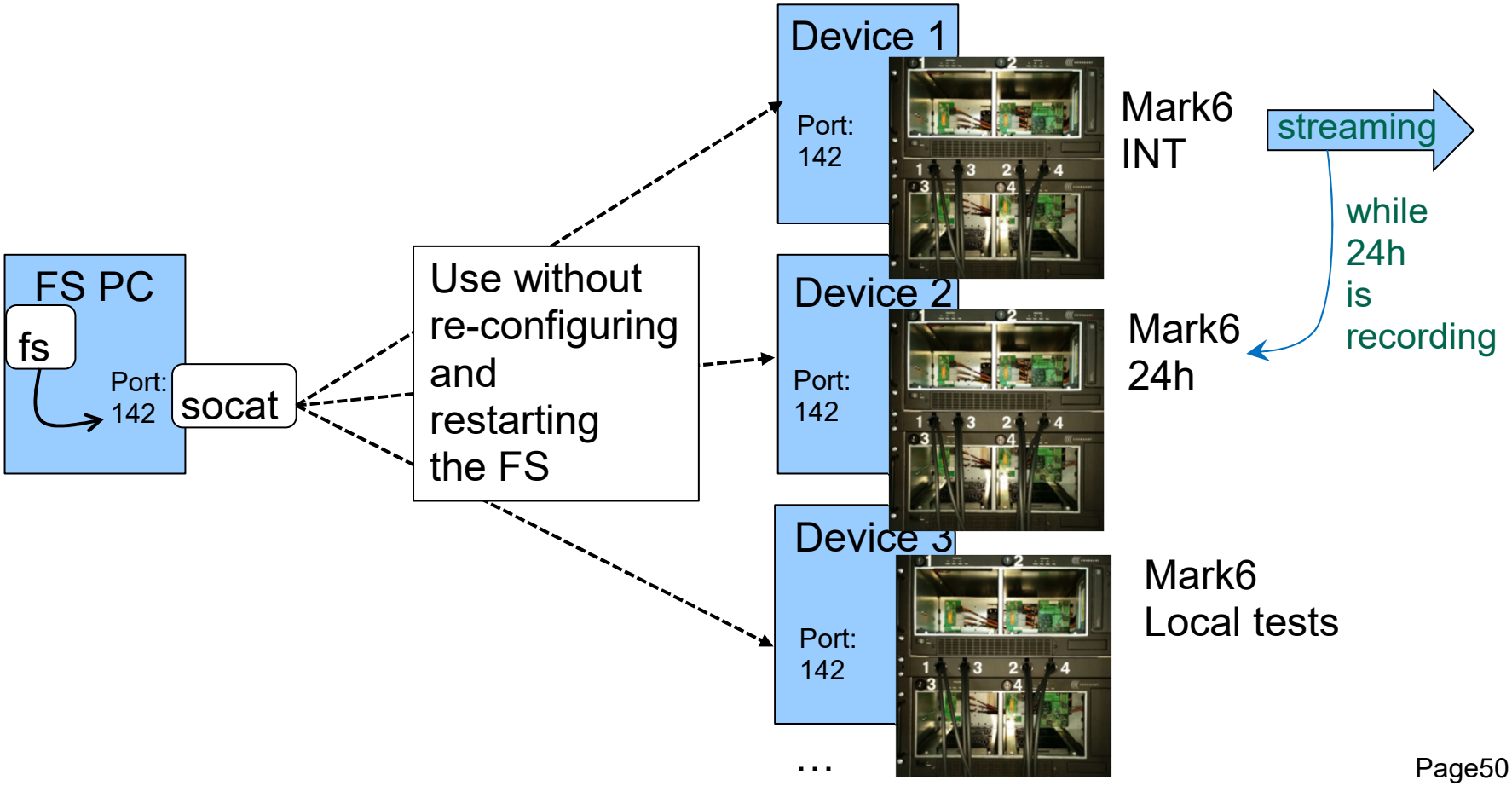


How to extend number of parallel devices?

But what if you have more than the allowed standard devices?

`socat` – Multipurpose relay

e.g. `socat TCP-LISTEN:142,fork,reuseaddr TCP:192.168.1.1:142`

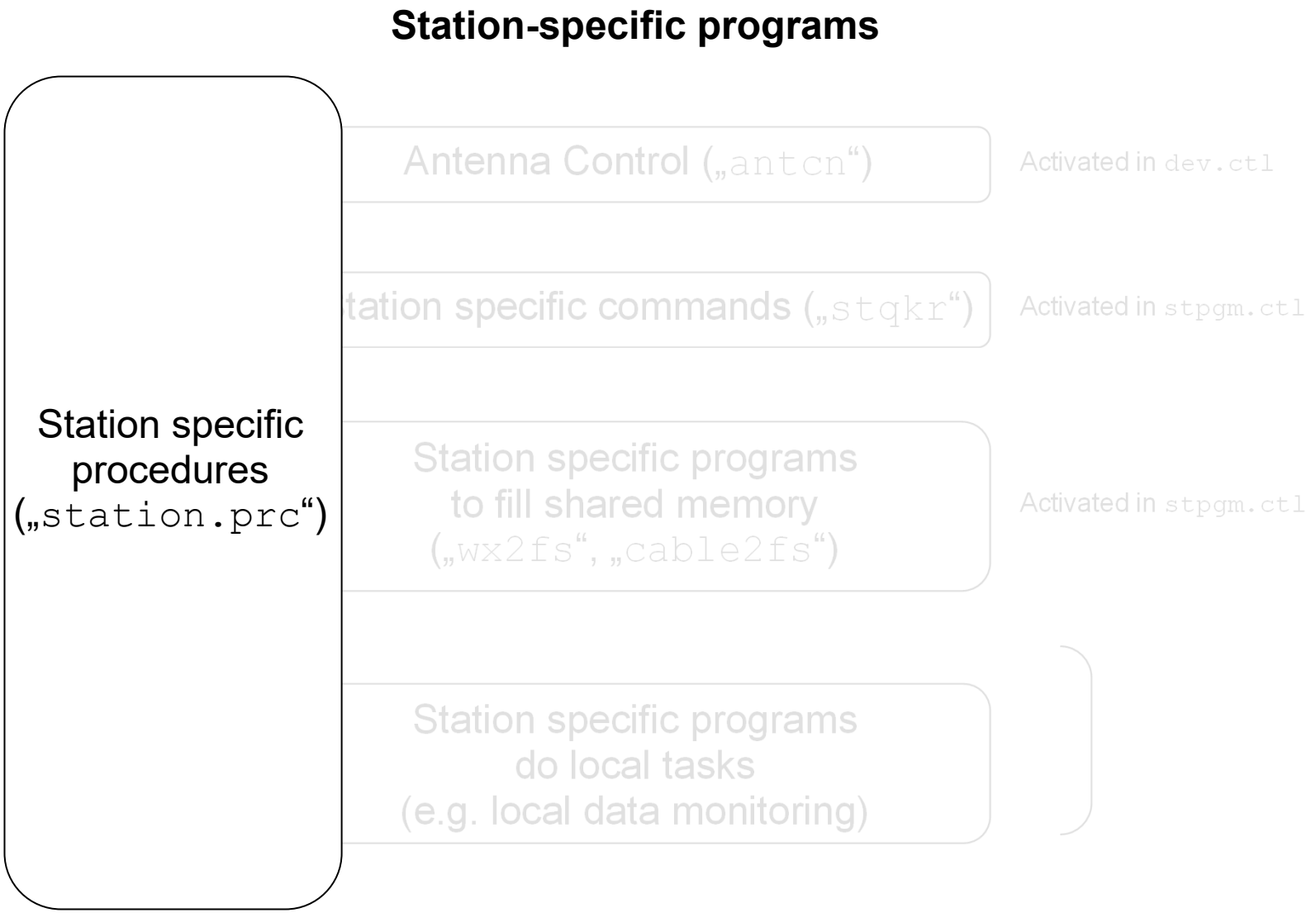


TOW2025 - Seminar

FS Station Code

What about FS?
What does a station has to offer to the FS?
How to control your antenna from FS?
How to control your equipment from FS?
How to fill data sets of the FS?
How to add functionality to the FS?
How to extend number of parallel devices?
How to combine functionalities to the FS?

How to combine functionalities to the FS?



How to combine functionalities to the FS?

Procedures in „station.prc“

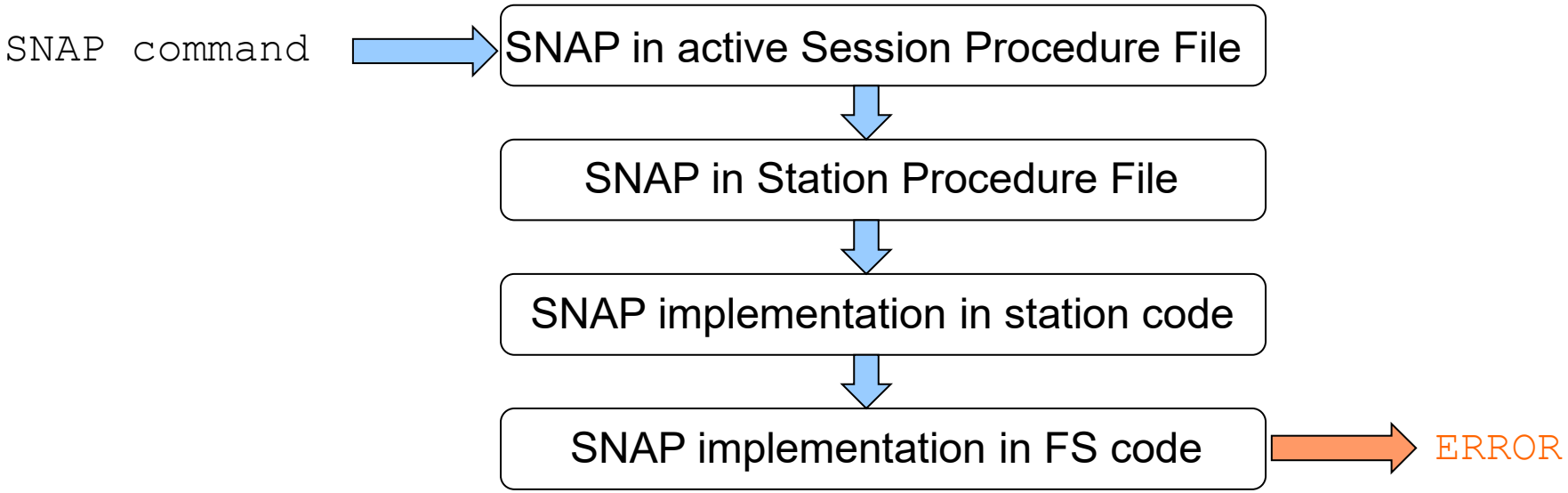
```
define  preob          23111184450x
if=cont_cal,,!*+4s
if=cont_cal,,caltsys_man
onsource
"caltsys
check=*
enddef
define  midob          23111184501x
onsource
antenna=status
wx
rx=dewar?
cable
ifa
ifb
ifc
ifd
bbc01
bbc05
bbc09
bbc13
mk5c_mode
!+1s
sy=run setcl adapt &
enddef
define  postob         23111184541x
dotmon
sy=lgput2 `lognm` &
enddef
```

```
define  casa           00000000000x
check=
source=casa,232324.8,+584859.,2000.0,null
enddef
define  safepos        00000000000x
antenna=safepos
enddef
```

```
calon
caloff
...
```

How to combine functionalities to the FS?

SNAP commands
(Standard Notation for Astronomical Procedures)



TOW2025 - Seminar

FS Station Code

Thank you …