fsNet: connection between the Field System and the Yebes Control System for the 40 and 13.2 m Radiotelescopes

L. Barbas, P. de Vicente, F. Beltrán, J. González, R. Bolaño

Informe Técnico IT-CDT 2020-10

Revision history

Version	Date	Author	Updates
1.0	28-04-2020	P. de Vicente	First version
1.1	06-05-2020	P. de Vicente	Minor changes
1.2	06-05-2020	J. González	Satellite section
1.3	06-05-2020	J. González & F. Beltrán	Minor changes
1.4	07-05-2020	P. de Vicente & J. González	Final version

Contents

1 Introduction

The Field System (FS hereafter) is the software used at many radio telescopes to monitor and control VLBI backends and manage VLBI observations at the stations. This software is composed of a suite of executables, libraries, and source code that is installed in a specialized PC running Linux/Debian. The FS provides some tools to allow the connection with the control system of each radiotelescope. This report briefly summarizes the connection developed at the Observatory of Yebes to send commands from the FS to the Yebes Radio Telescopes Control System (YRTCS) and to inject monitor variables back into the FS.

2 Some words on the FS customization

During its normal usage, the FS usually runs a script (the snap file) with repetitive commands which command sources to be observed, set the local oscillator frequency for each band, set the appropriate attenuation levels for each band at the VLBI backend, configure the sampling and digitizing scheme and monitor the status of the antenna (position and tracking), GPS versus Maser time difference and weather parameters.

One of the most versatile tools the FS has is a shared memory area which is divided in two blocks: FS and ST. The FS shared memory block is reserved for internal FS variables. The station shared block (ST) can be fully customized by the users to allow for new variables useful for the station. Both blocks of memory are reserved during the the computer boot process and can be used (read/write) by any code that connects to the shared memory area.

The code that implements the commands mentioned in the previous paragraph and which need to contact the local control system is stored in folder `/usr2/st`. Antenna commands are processed at directory `antcn` within executable `antcn` and any other commands related to equipment at directory `stqkr` within executable `stqkr`. Other directories in that folder host "include" files or additional auxiliary directories.

The key to making the connection between the FS and the external control system is to use the `antcn` and `stqkr` excutables to fill in some shared variables, that can be read by a third program at the FS computer which has the ability to directly communicate to the outside.

For example `antcn.c` has the following piece of code that gets executed when a new source is commanded (case 1) or when new offsets are commanded (case 2):

```
[....]
case 1:            /* source= command */
   ierr = 0;
   strcpy(buf,"Commanding to a new source/new offsets");
   logit(buf,0,NULL);
   fs->ionsor = 0;
   st->newsource = 1;
   sleep(RETARDO);
   break;
 case 2:            /* offsets         */
   ierr = 0;
   strcpy(buf,"Commanding new source/new offsets");
   logit(buf,0,NULL);
   fs->ionsor = 0;
   st->newoffsets = 1;
   sleep(RETARDO);
   break;
 [...]
```

In the example above we can see that, apart from some print and log commands, some variables get filled in each case:

- `fs->ionsor = 0;` This variable states that the antenna is off source because a new source has been commanded. The control system will set it back to 1 when the antena is on the source.
- `st->newsource = 1;` This variable acts as a semaphore and states that a new source has been commanded. The control system will set it back to 0 once the antenna starts moving towards this new source.
- `st->newoffsets = 1;` This variable acts as a semaphore to indicate that new offsets have been commanded. The control system will set it back to 0 once the antenna injects the new offsets.

Variables prepended with `fs->` belong to the FS shared memory, whereas those prepended with `st->` belong to the ST shared memory.

An executable located at `/usr2/st/fsNet` will read these variables and communicate with the external control system of the antenna. This short example shows the power of shared memory as a key tool for communication.

For a thorough and deep explanation on how the FS works please refer to TOG classes by Ed Himwich and manuals from Ed Himwich and Nancy Vanderberg.

3 ACS type legacy connection between the FS and YRTCS

Prior to 2015 the connection between the FS and the YCRTS was done using ACS components. An ACS component running at the FS computer and with access to its shared memory used the built in and native ACS connection between components. This solution is rather straightforward but has an important drawback: the FS computer is not detached from the rest of the YCRTS. In particular it needs to comply with all the hardware and software requirements of the YCRTS.

By the beginning of 2015 we decided to detach both parts and communicate them through sockets. This allowed independence and frequent OS updates at the FS computer, which traditionally required some special hardware interfaces for controlling the legacy VLBI Data Acquisition Rack (DAR). This is no longer necessary since the only specific hardware interfaces left by that time were GPIB cards which were replaced by Ethernet to GPIB converters which can be physically located close to the measuring devices.

4 New connection between the FS and YRTCS

Since 2015, the transfer of information between the FS and the YRCTS is done using a TCP socket connection and a structure which is passed forth and back every 100 milliseconds the fastest. When processing commands the structure will take much more time to be transferred because the communication is blocking (the communication channel will block until an acknowledegment is received within YRCTS once a command has been processed).

On the side of the YRCTS there is a C++ ACS component, called fsNetComp, which communicates with the rest of YCRTS components using some ACS tools: notification channel subscription and direct request. This component when executed for the first time creates a socket server ready to answer requests from any client. If it gets a new connection from the FS client it starts an infinite loop which pauses at the end of each cycle 50 milliseconds. The loop may be aborted if the connection from the client is closed.

Inside the component loop, the content of the structure received from the FS is copied in an internal structure called `m_shms_compInfo` which contains the same format as the transferred structure. The structure in the YCRTS fsNetComp component is then examined variable per variable looking for commands for the antenna, frontends, backends and other parts of the YCRTS which may be requested. The commands are considered as such when a given variable in the structure is set to TRUE. Once the command is passed to the rest of components, it waits for acknowledgement and when received that variable is set back to FALSE and the flow within the loop resumes.

In parallel to the loop, some variables of the structure in the fsNetComp component are constantly updated within some notification channels. For example, fsNetComp suscribes to the weather channel and updates all meteorological information on the structure with every new data received. The update frequency depends on each channel.

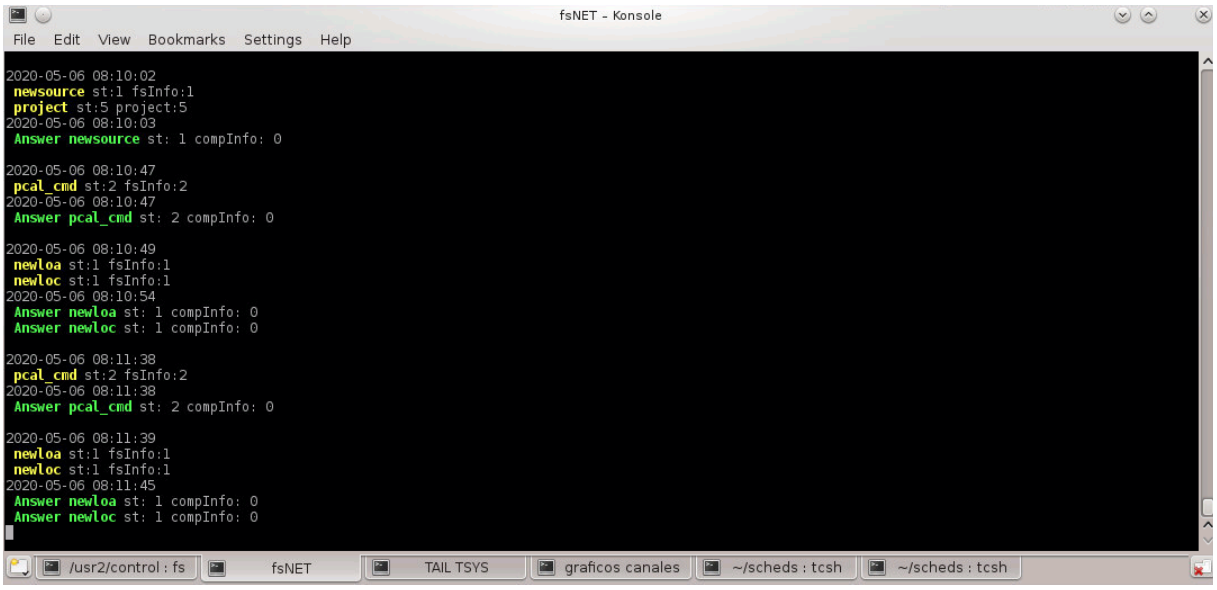
At the end of the loop the structure of the shared memory is sent back to the FS stating that the commands have been performed and with new updated information from several monitor parameters. The time required to complete the loop depends on the commands received. If no command has been sent from the FS, the execution of the loop will be very fast because it will only send monitored data.

On the FS side there is a C++, fsNet, which is started manually from the Linux shell. When it boots, it opens a TCP socket to the server and enters into an infinite loop which can be broken by a manual Control-C or by stopping the server at the YCRTS side. The loop fills in the structure with information from the shared memory (FS and ST blocks) and sends this structure to the server. At the end of the loop it reads back the structure, updates the shared memory variables which are used for monitoring and starts the loop again only changing those variables which are associated to commands.

The fsNet client allows to set a debugging level for printing messages tagged with a date stamp. This is very useful for debugging purposes and for monitoring the transfer process. The messages get displayed at the Linux console colorized to help a fast interpretation. Fig 1 shows a screenshot of the fsNet client running at the FS computer.

The allowed commands from the FS towards the YRTCS are described below. Each of them have a tag that indicates that they have been triggered by the FS and that the request has not been processed yet:

- New source to be tracked.
- New offsets with respect to the tracking source.
- Initiate antenna boot sequence.
- Send antenna to stow.



```
2020-05-06 08:10:02
 newsource st:1 fsInfo:1
 project st:5 project:5
2020-05-06 08:10:03
 Answer newsource st: 1 compInfo: 0

2020-05-06 08:10:47
 pcal_cmd st:2 fsInfo:2
2020-05-06 08:10:47
 Answer pcal_cmd st: 2 compInfo: 0

2020-05-06 08:10:49
 newloa st:1 fsInfo:1
 newloc st:1 fsInfo:1
2020-05-06 08:10:54
 Answer newloa st: 1 compInfo: 0
 Answer newloc st: 1 compInfo: 0

2020-05-06 08:11:38
 pcal_cmd st:2 fsInfo:2
2020-05-06 08:11:38
 Answer pcal_cmd st: 2 compInfo: 0

2020-05-06 08:11:39
 newloa st:1 fsInfo:1
 newloc st:1 fsInfo:1
2020-05-06 08:11:45
 Answer newloa st: 1 compInfo: 0
 Answer newloc st: 1 compInfo: 0
```

Figure 1: *fsNet client screenshot at the Field System computer. Yellow messages are commands towards the YCRTS. Green messages indicate responses from the server*

- Stop antenna.
- Set the antenna to standby.
- New configuration for frontend A, B, C and/or D
- Reset configuration for frontend A, B, C and/or D
- Set calibrarion noise diode to ON, OFF or CONTINOUS
- Switch ON or OFF the phase cal tone.
- Get active pointing model

The monitored information is classified as follows:

- Weather information.
- Position of the antenna.
- Tracking activated and on source: false or true.
- Opacity at the observed frequencies.
- Frequency and attenuation for the receivers.

Fig. 2 summarizes the communication process described above, and the execution sequence for the fsNet instance at the FS is shown in Figure 3. The structure with the information is thoroughly described in next section.

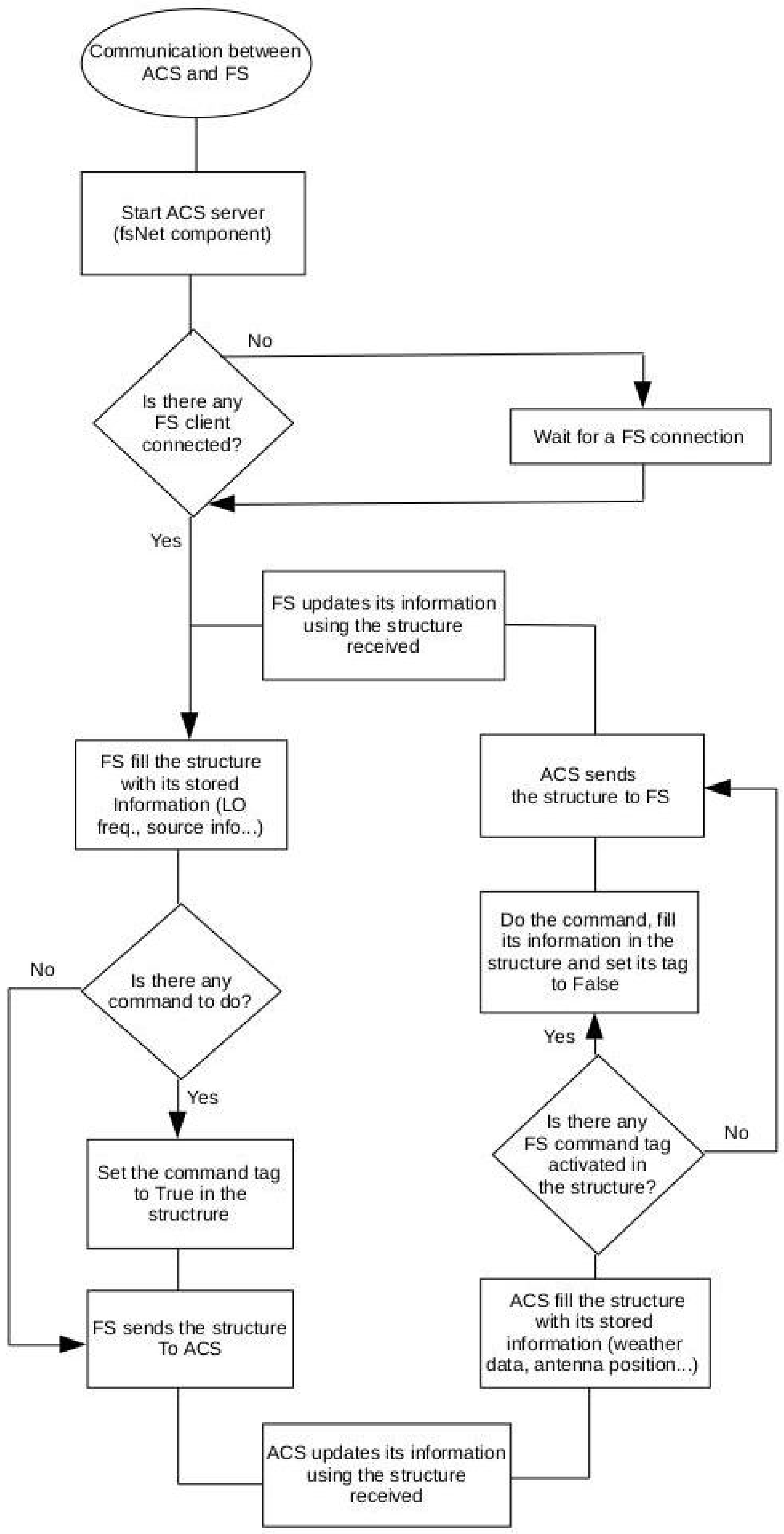

Communication between ACS and FS
Start ACS server (fsNet component)
Is there any FS client connected?
No
Wait for a FS connection
Yes
FS updates its information using the structure received
FS fill the structure with its stored Information (LO freq., source info...)
ACS sends the structure to FS
Do the command, fill its information in the structure and set its tag to False
No
Is there any command to do?
Yes
Yes
Is there any FS command tag activated in the structure?
No
Set the command tag to True in the structrure
FS sends the structure To ACS
ACS fill the structure with its stored information (weather data, antenna position...)
ACS updates its information using the structure received

Figure 2: *Communication flowchart between the FS and the YRTCS.*

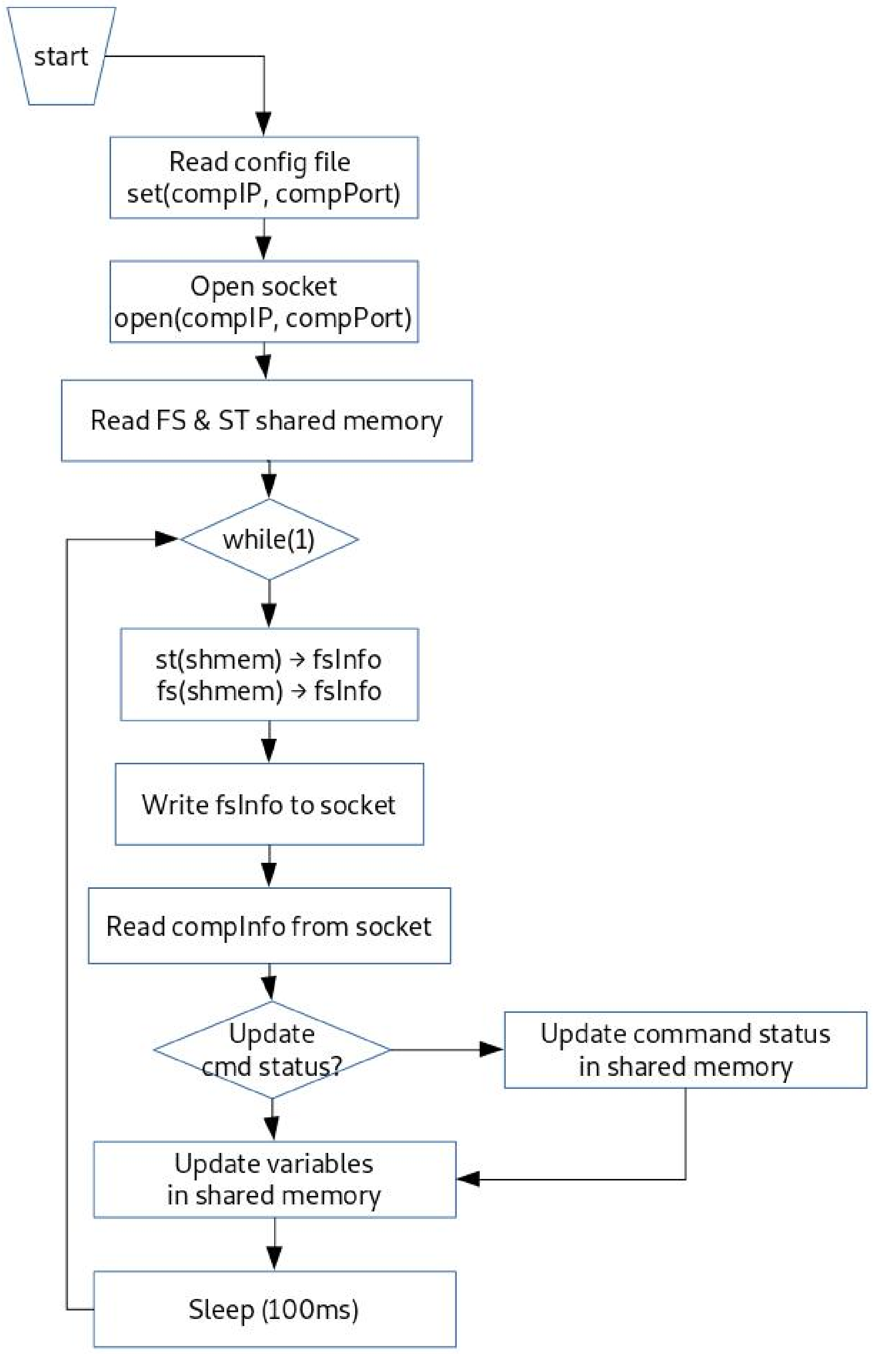


Figure 3: *Process flowchart at the FS side. A similar sequence is executed at the ACS side.*

5 Structure with exchanged information

The structure with information is described below:

```
struct shmSelection {
        unsigned int id;                 // 0, just read, 1,2,3,4,5 write in memory
        unsigned int caln_cmd;           // 0: do nothing, 1: noise diode off,
        // 2: noise diode on, 3: noise diode 80Hz
        unsigned int caln_sts;           // 1: noise diode off, 2: noise diode on, 3: noise diode 80Hz
        unsigned int pcal_cmd;           // 0: do nothing, 1 phase cal off, 2 phase cal on
        unsigned int pcal_sts;           // 1: phase cal off, 2 phase cal on
        unsigned int pmodel_cmd;         // 0: do nothing, 1 update pointing model values

        unsigned int newsource_cmd;      // 0: no new source commanded, 1: new source commanded
        unsigned int newoffsets_cmd;     // 0: no new offsets commanded, 1: new offsets commanded
        unsigned int project_cmd;        // 1:EVN, 2:GMVA, 3:IVS, 4:Radioastron, 5:tests

        unsigned int newloa;             // 0: no new loa commanded, 1: new loa
        unsigned int newlob;             // 0: no new lob commanded, 1: new lob
        unsigned int newloc;             // 0: no new loc commanded, 1: new loc
        unsigned int newlod;             // 0: no new lod commanded, 1: new lod
        unsigned int resetlo;            // 0: do nothing, 1: reset lo

        unsigned int boot_cmd;           // 0: no boot required, 1: new boot required
        unsigned int stow_cmd;           // 0: do nothing, 1: drive to stow, 2: unstow
        unsigned int standby_cmd;        // 0: do nothing, 1: standby, 2: activate
        unsigned int stop_cmd;           // 0: do nothing, 1: stop antenna
        unsigned int rx_reset_cmd;       // 0: do nothing. 1: reset list of receivers in use
        unsigned int m2mode_cmd;         // 0: do nothing, 1: m2 geo mode, 2: m2 astro mode

       //Component Variables
        double az;                      // Azimuth in degs
        double el;                      // Elevation in degs

        double pmodel[9];               // Pointing model parameters

        unsigned int ionsor;
        unsigned int point;              // 0: do nothing, 1: new pointing commanded
        unsigned int correctpoint;       // 0: do nothing, 1: apply last pointing corrections

        double gps;
        double opaca;                   // opacity of the atmosphere (neper)
        double opacb;                   // opacity of the atmosphere (neper)
        double opacc;                   // opacity of the atmosphere (neper)
        double opacd;                   // opacity of the atmosphere (neper)
        double wh2omm;                  // ammount of precipitable water in the atmosphere (mm)

        float averageWindDirection;
        float averageWindSpeed;
        float humidity;
        float pressure;
        float temperature;

        // FS variables
        char sourcename[11];
        double ra50;
        double dec50;
        float ep1950;
        float raoff;
        float decoff;
        float azoff;
        float eloff;
        float loa;
        float lob;
        float loc;
        float lod;
        int atta;                       // Attenuation IF A
        int attb;                       // Attenuation IF B
        int attc;                       // Attenuation IF C
        int attd;                       // Attenuation IF D
```

This structure is stored at the FS and transferred through the TCP socket connection. There is also a copy of such structure at the component (YCRTS server side) which contains the updated information from the telescope as explained in the previous section. Variables associated to commands get modified first at the client (FS) and later at the server (YCRTS).

Variables modified at the FS computer are listed below:

```
struct shmSelection {
        unsigned int id;                // 0, just read, 1,2,3,4,5 write in memory
        unsigned int caln_cmd;          // 0: do nothing, 1: noise diode off, 2: noise diode on,
        // 3: noise diode 80Hz
        unsigned int caln_sts;          // 1: noise diode off, 2: noise diode on, 3: noise diode 80Hz
        unsigned int pcal_cmd;          // 0: do nothing, 1 phase cal off, 2 phase cal on
        unsigned int pcal_sts;          // 1: phase cal off, 2 phase cal on
        unsigned int pmodel_cmd;        // 0: do nothing, 1 update pointing model values

        unsigned int newsource_cmd;     // 0: no new source commanded, 1: new source commanded
        unsigned int newoffsets_cmd;    // 0: no new offsets commanded, 1: new offsets commanded
        unsigned int project_cmd;       // 1:EVN, 2:GMVA, 3:IVS, 4:Radioastron, 5:tests

        unsigned int newloa;            // 0: no new loa commanded, 1: new loa
        unsigned int newlob;            // 0: no new lob commanded, 1: new lob
        unsigned int newloc;            // 0: no new loc commanded, 1: new loc
        unsigned int newlod;            // 0: no new lod commanded, 1: new lod
        unsigned int resetlo;           // 0: do nothing, 1: reset lo

        unsigned int boot_cmd;          // 0: no boot required, 1: new boot required
        unsigned int stow_cmd;          // 0: do nothing, 1: drive to stow, 2: unstow
        unsigned int standby_cmd;       // 0: do nothing, 1: standby, 2: activate
        unsigned int stop_cmd;          // 0: do nothing, 1: stop antenna
        unsigned int rx_reset_cmd;      // 0: do nothing. 1: reset list of receivers in use
        unsigned int m2mode_cmd;        // 0: do nothing, 1: m2 geo mode, 2: m2 astro mode

        unsigned int point;             // 0: do nothing, 1: new pointing commanded
        unsigned int correctpoint;      // 0: do nothing, 1: apply last pointing corrections

        // FS variables
        char sourcename[11];
        double ra50;
        double dec50;
        float ep1950;
        float raoff;
        float decoff;
        float azoff;
        float eloff;
        float loa;
        float lob;
        float loc;
        float lod;

        int atta;                       // Attenuation IF A
        int attb;                       // Attenuation IF B
        int attc;                       // Attenuation IF C
        int attd;                       // Attenuation IF D
```

Variables modified at the YCRTS computer are listed below:

```
struct shmSelection {
        unsigned int caln_cmd;           // 0: do nothing, 1: noise diode off, 2: noise diode on,
        // 3: noise diode 80Hz
        unsigned int caln_sts;           // 1: noise diode off, 2: noise diode on, 3: noise diode 80Hz
        unsigned int pcal_cmd;           // 0: do nothing, 1 phase cal off, 2 phase cal on
        unsigned int pcal_sts;           // 1: phase cal off, 2 phase cal on
        unsigned int pmodel_cmd;         // 0: do nothing, 1 update pointing model values

        unsigned int newsource_cmd;      // 0: no new source commanded, 1: new source commanded
        unsigned int newoffsets_cmd;     // 0: no new offsets commanded, 1: new offsets commanded

        unsigned int newloa;             // 0: no new loa commanded, 1: new loa
        unsigned int newlob;             // 0: no new lob commanded, 1: new lob
        unsigned int newloc;             // 0: no new loc commanded, 1: new loc
        unsigned int newlod;             // 0: no new lod commanded, 1: new lod
        unsigned int resetlo;            // 0: do nothing, 1: reset lo

        unsigned int boot_cmd;           // 0: no boot required, 1: new boot required
        unsigned int stow_cmd;           // 0: do nothing, 1: drive to stow, 2: unstow
        unsigned int standby_cmd;        // 0: do nothing, 1: standby, 2: activate
        unsigned int stop_cmd;           // 0: do nothing, 1: stop antenna
        unsigned int rx_reset_cmd;       // 0: do nothing. 1: reset list of receivers in use
        unsigned int m2mode_cmd;         // 0: do nothing, 1: m2 geo mode, 2: m2 astro mode

       //Component Variables
        double az;                      // Azimuth in degs
        double el;                      // Elevation in degs

        double pmodel[9];               // Pointing model parameters

        unsigned int ionsor;

        double gps;
        double opaca;                 // opacity of the atmosphere (neper)
        double opacb;                 // opacity of the atmosphere (neper)
        double opacc;                 // opacity of the atmosphere (neper)
        double opacd;                 // opacity of the atmosphere (neper)
        double wh2omm;                  // amount of precipitable water in the atmosphere (mm)

        float averageWindDirection;
        float averageWindSpeed;
        float humidity;
        float pressure;
        float temperature;
```

6 Commands

6.1 Basic antenna commands

[boot_cmd] Unsigned int variable. Initiate boot sequence. The boot sequence is composed of more than 10 individual commands that prepare the antenna for observation. For example it sets the correct time, loads the pointing model, sets the correct position of the subreflector, unstows the pins ....

- 0. Do not initiate boot sequence.
- 1. Initiate boot sequence.

[standby_cmd] Unsigned int variable. Sets the antenna to standby mode. Brakes are active and motors deactivated.

- 0. Do not set to standby.
- 1. Set the antenna to standby.

[stop_cmd] Unsigned int variable. Stop antenna but keep motors on.

- 0. Do not stop the antenna.
- 1. Stop the antenna.

[stow_cmd] Unsigned int variable. Send the antenna to the nearest stow position or unstow. During the stow process the pointing model is unloaded. Unstow does not load back the pointing model.

- 0. Do nothing.
- 1. Stow to the nearest position.
- 2. Unstow the antenna.

6.2 Command noise diode

[caln_cmd] Unsigned int variable. This variable can take 4 values:

- 0. Do not command the noise calibration diode and keep its current status.
- 1. Command the noise calibration diode and switch it OFF.
- 2. Command the noise calibration diode and switch it ON.
- 3. Command the noise calibration diode and switch it to a 80 Hz continuous operation.

Any other value will be ignored. Once this command is processed by the YCRTS, the variable is set to 0 in the internal component structure which will be sent back to the FS.

6.3 Phase calibration mode

[pcal_cmd] Unsigned int variable. This variable can take 3 values:

- 0. Do not command the phase calibrator and keep its current status.
- 1. Command the phase calibrator and switch it OFF.
- 2. Command the phase calibrator and switch it ON.

Any other value will be ignored. Once this command is processed by the YCRTS the variable is set to 0 in the internal component structure which will be sent back to the FS.

6.4 Pointing model update

`[pmodel_cmd]` Unsigned int variable. This variable can take 2 values:

- 0. Do not retrieve the current pointing model.
- 1. Retrieve the current pointing model.

Any other value will be ignored. Once this command is processed by the YCRTS the variable is set to 0 in the internal component structure which will be sent back to the FS.
Pointing model passed through
The goal is to display this pointing model in the FS log for logging purposes during the initial configuration phase. The pointing model is stored in a an array of 9 doubles. The meaning of each parameter is explained in de Vicente and Barcia (2007).

- `pmodel[9]`. Nine element double array. Nine parameters which include offsets in azimuth and elevation encoders, tilts of the elevation axis towards the north and east, collimation errors, lack of perpendicularity between azimuth and elevation axis and gravitational effects. See de Vicente and Barcia (2007).

6.5 New source to be tracked

`[newsource_cmd]` Unsigned int variable. This variable can take 2 values:

- 0. No new source commanded.
- 1. New source commanded.

Any other value will be ignored. Once this command is processed by the YCRTS the variable is set to 0 in the internal component structure which will be sent back to the FS.
Source coordinates to be passed

- `[sourcename]`. Char variable which allows a maximum of 11 characters. Name of the source. The YCRTS recognizes special names and allows for special tracking. For example a name that starts by "`RA_`" is understood by the YCRTS as a source whose tracking will be based on an azimuth and elevation table that will be passed independently to the control system. This table allows for tracking satellites or fast moving objects (see section 8). This can be customized for each case to allow for tracking non natural sources from the FS. If the name of the source is "AZEL", the coordinates `ra50` and `dec50` will be considered to be Azimuth and Elevation in radians and `ep1950` will be ignored.
- `ra50`. Double variable. Right ascention in radians referred to the equinox in variable `ep1950`.
- `dec50`. Double variable. Declination in radians referred to the equinox in variable `ep1950`.
- `ep1950`. Float variable. Equinox. It usually only takes two possible values: 1950 or 2000. YCRTS will correct for precession, nutation and aberration and compute the apparent coordinates for the current date.

6.6 New offsets to be applied to the source being tracked

`[pmodel_cmd]`. Unsigned int variable. This variable can take 2 values:

- 0. No new offsets commanded.
- 1. New offets commanded.

Any other value will be ignored. Once this command is processed by the YCRTS the variable is set to 0 in the internal component structure which will be sent back to the FS.
Offsets to be passed
These offsets are treated in a peculiar way, which can be adapted to any requirements. Offsets are applicable either in right ascention and declination or in azimuth and elevation, but not at the same time. All variables are float.

- `raoff`. Offsets to be applied in right ascention. Units at the FS: radians. These offsets are only applied if `azoff` and `eloff` are zero.
- `decoff`. Offsets to be applied in declination. Units at the FS: radians. These offsets are applied only if `azoff` and `eloff` are zero.
- `azoff`. Offsets to be applied in azimuth. Units at the FS: radians. These offsets are applied only if `raoff` and `decoff` are zero. This is the case for typical ONOFF scans
- `eloff`. Offsets to be applied in elevation. Units at the FS: radians. These offsets are applied only if `raoff` and `decoff` are zero. This is the case for typical ONOFF scans

6.7 Subreflector control

`[m2mode_cmd]`. Unsigned int variable. This variable allows to control the subreflector for the next observation. The subreflector has 5 or 6 degrees of freedom and its optimum position usually depends on the elevation of the antenna.

- 0. Do nothing and keep the current mode.
- 1. Geo mode. The subreflector is blocked at its optimum position and does not change with elevation.
- 2. Astro mode. The subreflector moves depending on elevation to have a maximum aperture efficiency at all elevations.

6.8 Project identifier

`[project_cmd]` Unsigned int variable. This variable can be adapted to the needs of the radioelescope. For the 40m RT it allows to distinguish the type of observations for further storage.

- 1. EVN observations.

- 2. GMVA observations.
- 3. IVS observations.
- 4. Radioastron observations.
- 5. Testing observations.

6.9 New frequency of observation

`[newloa]` Unsigned int variable. This command allows to set 4 different frequencies associated to IFs A, B, C and D and IF attenuations at the receiver. The variables may only take two values:

- `newloa/b/c/d`. 0. No new frequency for IF A/B/C/D commanded.
- `newloa/b/c/d`. 1. New frequency for IF A/B/C/D commanded.

Local oscillator frequency This command allows to set 4 different local oscillator frequencies for channels A, B, C and D (units: MHz).

- `loa`. Float variable. Local oscillator frequency for channel A. Only taken into account when `newloa` is commanded.
- `lob`. Float variable. Local oscillator frequency for channel B. Only taken into account when `newloa` is commanded.
- `loc`. Float variable. Local oscillator frequency for channel C. Only taken into account when `newloa` is commanded.
- `lod`. Float variable. Local oscillator frequency for channel D. Only taken into account when `newloa` is commanded.

Relative attenuation at IF channels This command allows to set 4 different relative attenuations to IFs A, B, C and D at the receiver level. These variables apply a negative or positive attenuation value (units: dBs) relative to the default attenuation levels.

- `attA`. int variable. Relative attenuation to be applied at the frontend. Only taken into account when `newloa` is commanded.
- `attB`. int variable. Relative attenuation to be applied at the frontend. Only taken into account when `newlob` is commanded.
- `attC`. int variable. Relative attenuation to be applied at the frontend. Only taken into account when `newloc` is commanded.
- `attD`. int variable. Relative attenuation to be applied at the frontend. Only taken into account when `newlod` is commanded.

6.10 Reset frequency of observation and attenuation

`[resetlo]`. Unsigned int variable. This command resets all local oscillator frequencies and IF attenuations associated to IFs A, B, C and D to a default value. The variable may only take two values:

- `resetlo`. 0. No reset.
- `resetlo`. 1. Reset to default values. All receivers are deselected.

7 Monitored variables

These variables are required by the FS to properly work and display its values at the logging window and to tag the recorded data as valid. They are modified at the server running at the YCRTS .

7.1 Antenna position and on source tracking

`ionsor` Unsigned int variable. It monitors if the antenna is tracking the source with a maximum error of 10% of the beam size. The value is updated every 200 ms.

- 0. The antena is not tracking the source, or it is off source.
- 1. The antenna is tracking the source.

Variables `az` and `el` monitor the position of the antenna

7.2 Monitor noise diode

`caln_sts` Unsigned int variable. This variable can take 3 values:

- 1. Calibration noise diode is OFF.
- 2. Calibration noise diode is ON.
- 3. Calibration noise diode is in 80 Hz continuous mode.

7.3 Monitor phase calibrator

`pcal_sts` Unsigned int variable. This variable can take 2 values:

- 1. Phase calibrator is OFF.
- 2. Phase calibrator is ON.

7.4 Weather parameters

Float variables. All these variables are updated every 2 seconds (wind speed and direction), and every 10 minutes (rest of parameters). The frequency update is selectable and depends on the weather station configuration.

- `averageWindDirection`. It monitors the wind direction (units: degrees from the north clockwise).
- `averageWindSpeed`. Monitors the ambient average wind speed (units: m/s) during the last 10 minutes.
- `humidity`. Monitors the local ambient relative humidity (units: %).
- `pressure`. Monitors the local ambient pressure (units: hPa)

7.5 Other parameters

Float variables. These variables are updated according to the refreshing period of the monitor devices and their configuration: GPS (every 10 minutes), wh2omm depend on weather parameters and opacity is calculated when the local oscillator frequency is changed.

- `gps`. Monitors the GPS-Maser PPS difference (units: microseconds)
- `opaca`. Opacity for the frequency of observation of channel A at the current antenna elevation.
- `opacb`. Opacity for the frequency of observation of channel B at the current antenna elevation.
- `opacc`. Opacity for the frequency of observation of channel C at the current antenna elevation.
- `opacd`. Opacity for the frequency of observation of channel D at the current antenna elevation.
- `wh2omm`. Precipitable water (units: mm).

8 Satellite tracking

As mentioned above, the fsNet component allows to track satellites using the Field System if an azimuth - elevation versus time table is provided. This option needs an additional tool, a Python script that parses the file with the satellite's position table: *sattrack.py*. This script, that should be available in the FS's user PATH takes two arguments; the experiment's snap file, and the azimut-elevation file, in the following format

```
2016-11-28 20:00:00  az = 048.760639  el =  13.101975
```

With this information, the script modifies the source commands in the snap file adapting the schedule to the available positions table and including a special prefix in the source name: “RA_”. Then, it transfers a modified version of the ASCII file with the positions table to the YRCTS directory hierarchy (/home/almamgr/aries21/Catalogs/), to make it available for the control system. Whenever a source with the special “RA_”" prefix is commanded by the Field System, the fsNet component at the YRTCS side will detect this condidition and issue the necessary orders to get the positions table loaded by the observing engine.

This option has been widely used for tracking Radioastron satellites and spatial probes in the solar system.

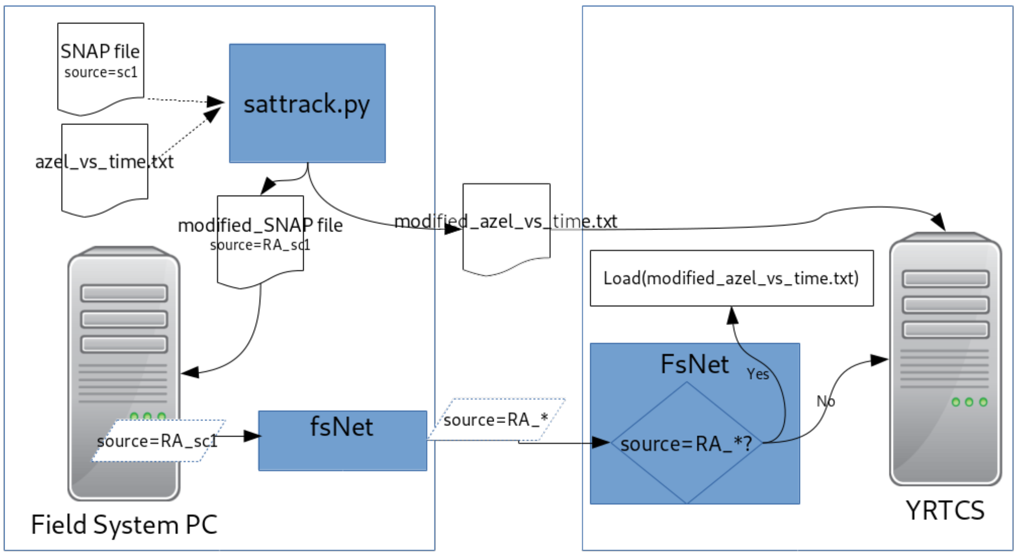


Figure 4: *fsNet with satellite tracking flowchart.*

References

[1] de Vicente, P. Barcia, A. Deconstructing a pointing model for the 40m RT. OAN Technical Report 2007-26.

[2] Himwich, E. Vandenberg, N. R. Mark IV Field System Documentation. NVI Inc./Goddard Space Flight Center 1997.