Giant Meterwave Radio Telescope

National Center for Radio Astrophysics

Tata Institute of Fundamental Research

ABCcom Software

of GMRT Telemetry System

Description and implementation

Author : Laurent Pommier

Project Supervisor : Prof. Pramesh Rao

Date : 05/01/06

Table of Contents

Illustration Index

1. Introduction

The Giant Meterwave Radio Telescope (GMRT) is a radio telescope located in Khodad, at 80 km from Pune (Maharastra, India). It is composed of an array of 30 antennas spread over distances up to 25 km. Each antenna is 45 m in diameter, and has been designed to operate at a range of frequencies from 50 to 1500 Mhz.

Modern radio telescopes are complex assemblies of electronic and electro-mechanical subunits. To allow a successful observation, all these subunits have to be set as per the user requirements. For example, antennas have to track the selected source, front-ends have to be tuned to the chosen frequency band, all the amplifiers along the signal path have to be set up to the appropriate value which would give the optimum signal to noise ratio, local oscillators have to be tuned to select the desired frequency, and the correlator has to be set up to do the appropriate fringe and delay tracking.

In an interferometer like the GMRT, this means that in a coordinated manner, one has to control systems which are several tens of kilometers separated from one another. In addition, systems must be periodically monitored, so that should any of them fail, the affected data can be flagged and remedial action can be taken to fix the faulty unit. Besides since it is not humanly possible to remember all the various safety limits of each subsystem, the telescope control system must also forbid any wrong operation. The Telemetry System is the one in charge of all these control and monitoring tasks in GMRT.

This report introduces ABCcom, a Linux PC software which aims at replacing and improving the ABC element of the Telemetry chain. ABC is the unit inside each antenna, communicating with the telescope control room on one side, and with antenna systems on the other side. A Teleset software is implemented to be a new control room program adapted to ABCcom, but antenna systems remain unchanged. Thus ABCcom is a part of a new generation software for GMRT Telemetry.

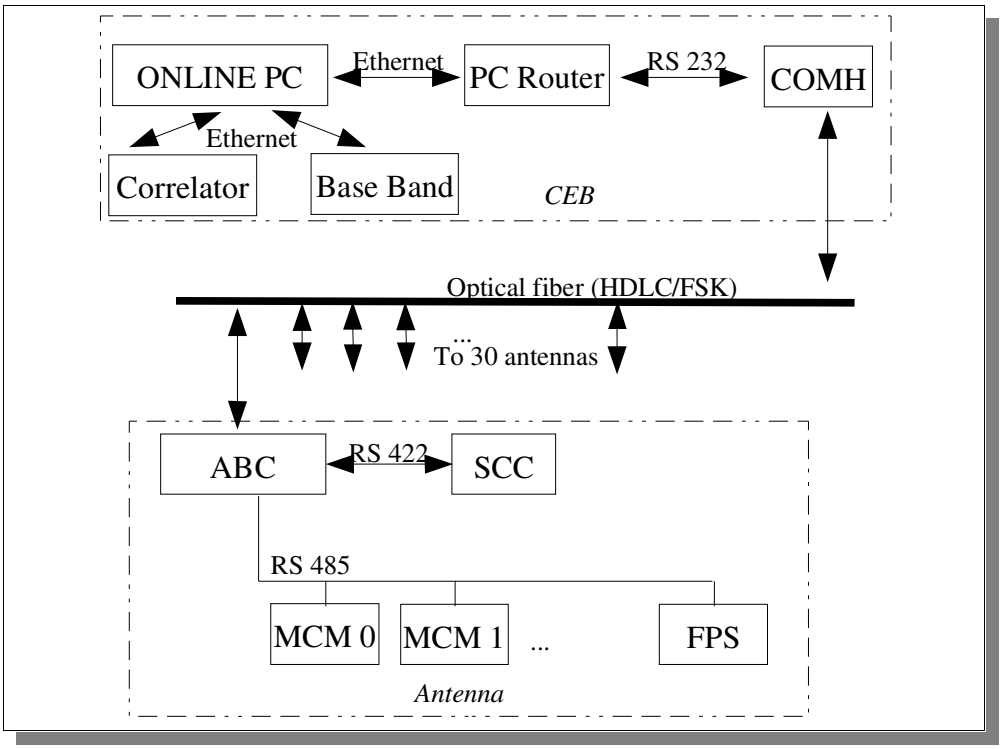
This document first gives an overview of this project context in parts 2 and 3. Chapters 4 and 5 detail the ABCcom configuration, functions and design. After a brief note on programming techniques in part 6, ABCcom implementation is described from high level layer (parts 7 to 10), down to low level communication layer (parts 11 and 12).

2. General presentation of the Telemetry System

The GMRT Telemetry is the system whose goal is to control other systems operations in a coordinated manner and to monitor their current state to check parameter values and detect any dysfunction. Its main tasks are:

- To rotate all the 30 antennas in azimuth and elevation, to track a celestial source
- To bring the required feed in the feed turret to the focus via the Feed Position System (FPS).
- To select the Front End system parameters like observing frequency band, noise calibration...
- To set the Intermediary Frequency (IF) and Local Oscillator (LO) systems including frequencies, IF bandwidths and attenuations, Automatic Level Controller (ALC) operation...
- To set the Base Band and Correlator parameters
- To monitor hundreds of systems parameters at all points along the signal flow path

This system is divided into many components based whether inside the CEB or in antennas shells:



Legend:

. MCM	*Monitor and Control Module*	*. ABC*	*Antenna Based Computer*
. FPS	*Feed Positioning System*	*(=ANTCOM)*	*(=Antenna Computer)*
. COMH	*Communication Handler*	*. SSC*	*Station Servo Computer*
. CEB	*Central Electronic Building*		

Illustration 1 : Components of the Control and Monitoring system

3. ABCcom project

3.1. ABC current state

An ABC (Antenna Base Computer, also called an ANTCOM) is located inside each antenna shell. All communication between the antenna and the CEB is routed through the ABC in that antenna. The ABC has 3 communication links:

(i) the main link between COMH and ABC which operates at 250 kbps,

(ii) an asynchronous 9.6 kbps RS-422 communication between ABC and the Station Servo Computer (SSC),

(ii) an asynchronous 9.6 kbps RS-485 communication link between ABC and up to 16 Monitor and Control Modules (MCMs).

The actual ABC is a Plug In Unit (PIU) composed of an electronic card including a 80C186 processsor, a 82510 USART, a 85C30 controller... It transmits all commands from Online operator PC to MCMs and SSC, and send back their respective answers. As a part of its implementation, ABC is also generating the tracking commands to SSC.

3.2. ABCcom and Teleset, a new Telemetry software chain

The ABCcom project aims at replacing the ABC PIU by an ABC Pentium PC. Under Linux OS, this PC runs ABCcom program presented in this document. This replacement offers the following advantages:

- better hardware stability, less maintenance and small risks of electronics failures.
- better performances and capacities (PC memory, processor, human interface...)
- more flexible and easy to evolve (C++ program)

With ABCcom, the heavy part of the telemetry intelligence is shifted from the Online PC, to the ABC PC. Indeed in the actual system, all data packets are generated from Online PC and ABC PIU just transmits them. The ABC PC is now generating all the commands, and only high level parameters are sent by Online PC.

For instance, monitoring the LO system implies to generate 30 commands to MCM 2 and 3. This task is now performed by ABCcom instead of Online, which then sends just one short LO monitor command. ABCcom can also generate this monitor operation automatically at regular time intervals if this mode is set by Online.

The communication between CEB and antennas is reduced and thus the information is transmitted faster. The Online PC load is also reduced, and new advanced sets of operation are implemented in ABCcom program (resetting automatically system parameters in case of power failure, human interface for local maintenance...).

Installing the ABCcom PC required to change as well Online program, since all the communication protocol between Online and ABC is changed. This new Online program is Teleset, running on a Linux PC. At this date it is completed as far as the communication with all antenna ABCcoms is concerned (with sub-array user processes, shared memory, and a display program Teledisp, cf. [6]).

4. ABCcom PC configuration

4.1. Serial links to other systems

The ABC is connected on one side with the CEB, via the optical fiber link up to Online PC, and on the other side to the various antenna systems. Antenna systems are connected to ABC as follow:

System	*Connected Components*	*Link*
Servo system (SSC)	Servo Station Computer	RS-422
Local Oscillator system (LO)	MCM 2, MCM 3	RS-485
Intermediary Frequency system (IF)	MCM 10	RS-485
Front End system (FE)	MCM 2, MCM 5	RS-485
Feed Positioning System (FPS)	MCM 14	Rs-485

Illustration 2 : Links between ABC and antenna systems

RS-232 is the standard for serial full duplex communication. RS-422 extents it to long distance communications. Finally RS-485 allows long distance multi-point communication, but restricts it to half duplex. Thus a master (ABC) and many slave components (MCMs) can be connected to a single link. A mcmdriver Linux module had been developed for this RS-485 link (cf. [5]).

The optical fiber link connection has first been replaced by a simple RS-232 link directly between ABCcom and Teleset PCs.

A PC is originally equipped with one or two serial ports, therefore a multi serial port card is needed. The ISI 4608 – PCI from Multi Tech Systems company offers 8 additional serial ports on a Linux PC. The mcmdriver program, which implements UART registers, is not compatible with this card.

4.2. Configuration file for ABCcom program

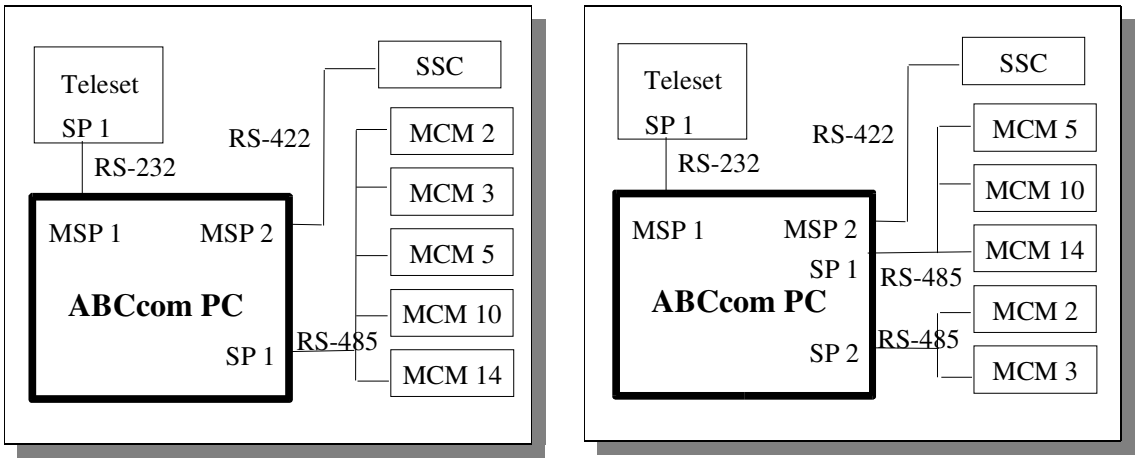
ABCcom program first configures the PC serial ports to communicate with all the systems according to a configuration file. It is named ***ConfigABC*** and is placed in a specific directory. By changing this file, the user can choose which systems he wants and where he connects them.

```
#####################################################################
# ConfigABC
#
# configuration file for ABCcom program
#
# Abc X, X = Abc address
#
# :N P for Serial Port number N with Priority P (=0 or 1)
# next line with unit addresses, ends with s
#
# unit address -1 for SSC, must be alone on the port (rs232, not rs485).
#
# unit address -2 for Teleset, must be alone on the port (rs232, not rs485).
#
# Possible MCM Addresses: 0 2 3 5 10 14.
# MCM3 and MCM5 require MCM2 previously written
# Pre-requirement: mcmcomi driver loaded on serial port i
#
# L.Pommier, 20/07/2005
#####################################################################

Abc 2
:2 0
-2 s
:1 0
-1 s
:0 0
2 3 5 10 14 s
```

Illustration 3 : ConfigABC file

The next diagrams illustrate examples of ABCcom PC configuration.



(a) *(b)*

. SP i *Serial Port number i*

. MSP i *Multi Serial Port card, port number i*

Illustration 4 : (a) and (b) Examples of ABCcom PC configuration

5. ABCcom general organization

5.1. List of ABCcom functionalities

To resume ABCcom functionalities, one must first resume each Telemetry task in respect to antenna systems, and then see how this task intelligence is divided between ABCcom and Teleset programs.

Teleset ***ABCcom***

- Servo system telemetry:
  - **operational commands (position, hold, stop, stow elevation / azimuth...)**
    send packet and read answer — by pass
  - **display commands (read angles, read motor currents...)**
    read answers — send packets every second
  - **set mode commands (set time, set high / low limit for elevation / azimuth...)**
    send packet and read answer — by pass
  - **load antenna parameters (angle offsets, latitude and longitude)**
    send parameters — store parameters
  - **track command: every 30 sec, send updated positions to follow a source**
    read answer — calculate angles and send command every 30 sec
  - **analyze events sent by SSC (motor speed high, stow released...)**
    read events — by-pass in priority

- FPS system telemetry:
  - **load turret angles for the feed selection**
    send parameters — store parameters
  - **move to the appropriate feed**
    send feed number — send move command with the proper angle
  - **monitor the turret angle to check the position**
    read angle — send monitor command at regular intervals
  - **set the FPS parameters (calibrate angle, maximum rotation speed...)**
    send packet and read answer — by pass

- LO, IF and FE systems telemetry:
  - **load antenna parameters**
    send parameters — store parameters
  - **set current parameters (frequencies, attenuations, bandwidth, feeds...)**
    send parameters — store parameters
    send all MCM commands to set parameters
  - **monitor the system state and signal errors**
    read error code — send all MCM commands to monitor at regular intervals
    decode and return a error code
  - **other specific operations (reboot, switch FE box off...)**
    send operation code — send respective MCM commands

Beside this, ABCcom has several additional functionalities:

- Local Teleset: During maintenance operations inside an antenna shell, a small local Teleset program can be launched from ABC PC to directly control this antenna systems without requesting the Control Room. After a command from the main Teleset, ABCcom also communicates with the local Teleset through a shared memory.

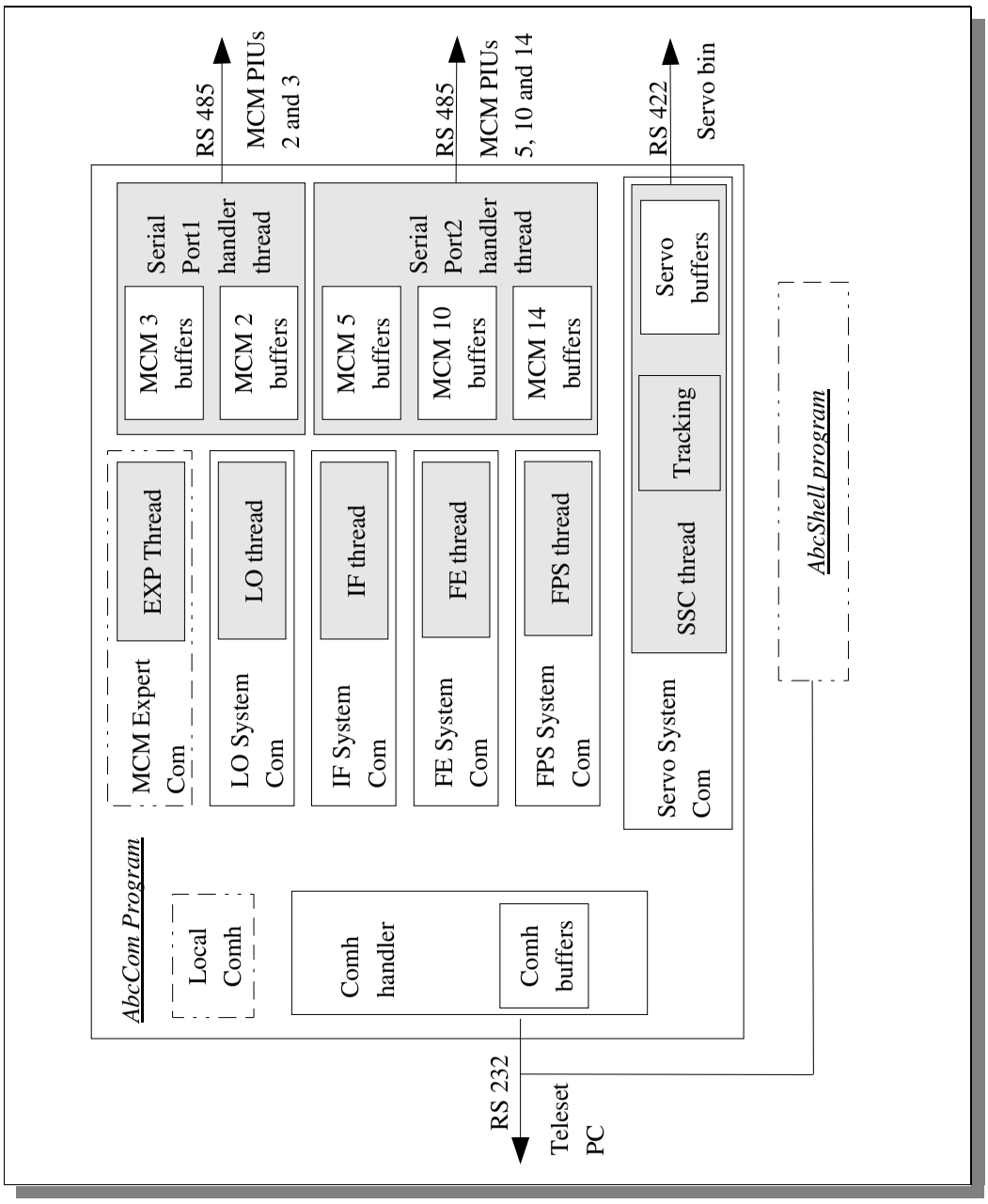
- MCM Expert: The Teleset operator chooses one MCM and communicates with it according to the MCM protocol (cf. [5]). This is useful for very specific operations.

- AbcShell: This mode allows to access ABC PC from Teleset PC for files transfer or Linux shell terminal commands. ABC is then no more communicating with antenna systems, the Telemetry is off. This is useful to get some ABCcom log files or put some modified ABCcom files before recompiling and restarting the Telemetry mode.

- LogFile: ABCcom writes in log files the history of its activity, organized by systems and periodically refreshed (cf. Appendix B).

5.2.ABCcom global block diagram

The next block diagram shows ABCcom global organization, for the configuration example (b) in Illustration 3.



. gray blocks *Child threads to the parent program process, cf. 6.2*
. dashed line blocks *Non permanent blocks (exist on Teleset command)*

Illustration 5 : ABCcom block diagram

Blocks in the diagram above correspond to C++ objects and are dedicated to specific functionalities. The next table briefly resumed them:

Blocks	*Class Names*	*Files*	*Description*
MCM buffers	ListMcm	common	chained lists storing command and answer packets for one MCM.
Servo buffers	baseFILO, DisplayFILO	common	circular queue storing command and answer packets for SSC.
Comh buffers	BaseFILO	common	circular queue storing command packets from Teleset.
Serial Port handler thread	PortMulti	mcmport	switch between MCM buffers blocks, write packets on the link and put back answers.
SSC thread	Servo	servo	perform the SSC system operations (Teleset direct commands, Display commands every sec., Tracking commands every 30 sec...)
LO, IF, FE, FPS threads	Losystm, Ifsystm, Fesystm, Fpssystm	losystm, ifsystm, fesystm, fpssystm	perform the system operations (reboot, set, monitor, analyze data...)
EXP thread	Expsystm	sysbase	created on Teleset command for direct communication with one MCM.
MCM Expert Com, LO Com, IF Com, FE Com, FPS Com, Servo Com	Comexp, Comlo, Comif, Comfe, Comfps, Comssc	comsys, combase	transmit Teleset commands to system threads and compose the corresponding answer.
Comh handler	Comh	comh	read Teleset commands from the link and write back their answer.
Local Comh	LocalComh	comh	created on Teleset command so ABCcom communicates also through a shared memory with a small Teleset running locally on ABC PC.
AbcShell	AbcShell	mainshell	for file exchanges and shell terminal commands. run alternatively with AbcCom

Illustration 6 : ABCcom blocks description

6. Programming techniques

6.1. POSIX multi threading

A thread is a like a light process, but easier and faster to manipulate. In ABCcom, the program runs as a parent process, which creates many child threads. Those threads run virtually in parallel to perform different tasks.
With Teleset, ABCcom aims at controlling and monitoring all the antenna systems: SSC, LO, FE, IF and FPS. It means that within very short time periods, ABC has to handle several various tasks. Multi-threading technique is used to achieve this with better performances. For instance, a thread sets the LO system frequencies, while another thread monitors the Servo motor currents.
In Linux, POSIX threads libraries are the standard base for C / C++ multi threading applications.

6.2. Kernel module, C, C++

ABCcom is a Linux program, which needs to mcmcom driver module to be loaded in the Kernel for serial ports used with MCM units (cf. [5]).
ABCcom is a program mainly coded in C++. Only the low level communication functions are coded in C for a faster access to the serial ports hardware (open, close, write, read / poll). For the rest of the program, I have used C++ language to take advantage of the object oriented features.
ABCcom is organized in distinct C++ objects, each of them implementing a specific ABC task. It is thus a far better way to understand, code, debug and later modify the program.

6.3.Inheritance and templates in C++

Many objects in the block diagram have common features. These common variables and functions are coded in some C++ base classes, which behave like library packages. For instance, Combase class provides the resources to read and write messages exchanged between Teleset and ABCcom.
Concrete classes are then derived from base classes to inherit all their properties. The C++ inheritance allows to write some reliable and reusable code, which can also be modified in derived class instead of duplicating code and rewriting all classes from scratch.
Templates provide parameterized types, that is the capacity of passing a type name as an argument to a recipe for building a class or a function. This C++ property is also very convenient to code parts of ABCcom program. Examples are given while describing the code in next sections.

7. ABCcom main objects

During the communication between Teleset and ABCcom, two different modes can exist alternatively. AbcPlus class is used for the ABC telemetry tasks and AbcShell is used for transferring files.
They are both parts of the ABCcom program, which shall not end otherwise the connection between the antenna and the control would be closed. Therefore exiting AbcPlus goes to AbcShell and vice versa.
ABCcom execution program must also be written in ***/etc/rc.local*** file so that the program starts and the link gets established automatically when ABC PC boots.

7.1. AbcShell class

AbcShell class object polls the serial port connected to Teleset, and decodes packets sent by the equivalent object in Teleset program (TeleShell). Packets exchanged in this mode contain a header and a command in ASCII according to a protocol described in ref. [7]. These commands can be:

- 'put', and a file name in argument. AbcShell creates then a file with the same name. Next packets from TeleShell transfer the file data.
- 'get', and a file name in argument. AbcShell opens then the file in ABC PC with the same name and transfer it to TeleShell.
- 'exit'. AbcShell returns and AbcCom starts.
- 'newabc'. TeleShell switches from an antenna to another.
- a linux shell command (ls, pwd...). AbcShell executes this command in a terminal and sends back the terminal display answer to TeleShell. The implementation uses ***shellout*** file to temporarily store shell display before transferring it to TeleShell, as it does after a 'get' command. 'cd' shell command is not working here.

During the file transfer, if any error occurs on one end, this last one transmits an interruption to the other end and the file transfer is canceled. AbcShell is coded in ***mainshell*** files.

7.2. AbcPlus class

AbcPlus class object is run in the parent ABCcom process in telemetry mode. It is implemented in ***abccom*** files and its activity is recorded in a *ABC* history file.

- Overall configuration with Config class:

At the beginning of the program, AbcPlus calls a Config object which reads ***ConfigABC*** file (Illustration 3) and creates accordingly all the telemetry blocks of the ABCcom diagram (Illustration 5). All those blocks are object variables of Config object. When LocalComh and MCM Expert Com are selected through a Teleset command, these two variables are also created in Config object.

- Main process loop:

In a start() function loop, Teleset commands are read from Comh object.
A command packet is divided in sub packets addressed to one system (cf. [7]). AbcPlus transmits sub packets one by one to the corresponding System Com object. This one decodes the command, makes the appropriated changes in its System Thread object, and composes the answer to Teleset. Once all sub packets are processed, AbcPlus returns the global answer to Teleset through Comh (or LocalComh).
The same operations are performed with LocalComh if the local mode is set. Then the loop restarts.

- Commands to ABC system:

Some commands are addressed to ABC itself (set time, local mode, exit...). To execute them, AbcPlus contains a Comabc object. It is treated like another System Com object, the difference is that it acts on Config and AbcPlus objects themselves instead of a System Thread object. ABC system has no loading parameters, and 2 setting parameters: local and expert flags.

7.3. ABCcom state diagram

The UML state diagram of ABCcom program is shown in the illustration below.

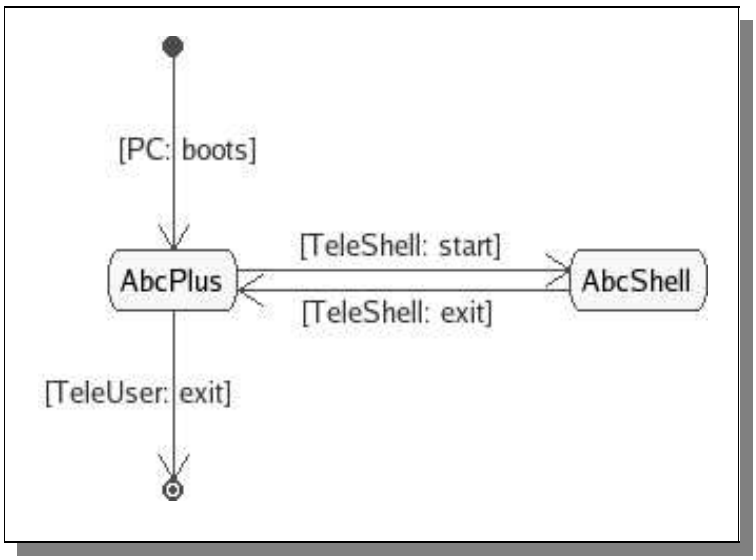


Illustration 7 : UML state diagram of ABCcom program

8. System Com classes

Each System Com object contains a System Thread object which runs in parallel to the main process (cf. Illustration 5). System Com is meant to make the interface between Teleset messages and a System Thread. It decodes commands from Teleset, makes the corresponding changes in the thread object, and composes the answer to Teleset.
One regular Teleset command is 'State' command. It asks for the current state of the system, which means that the answer composed by System Com will contain all parameters and current operations of the System Thread. Commands and answers between Teleset and ABCcom are more detailed in reference [7].

Com System classes are coded in ***combase*** and ***comsys*** files. Their activity is recorded in *ABC* history file. Their implementation is using C++ inheritance according to the following tree:

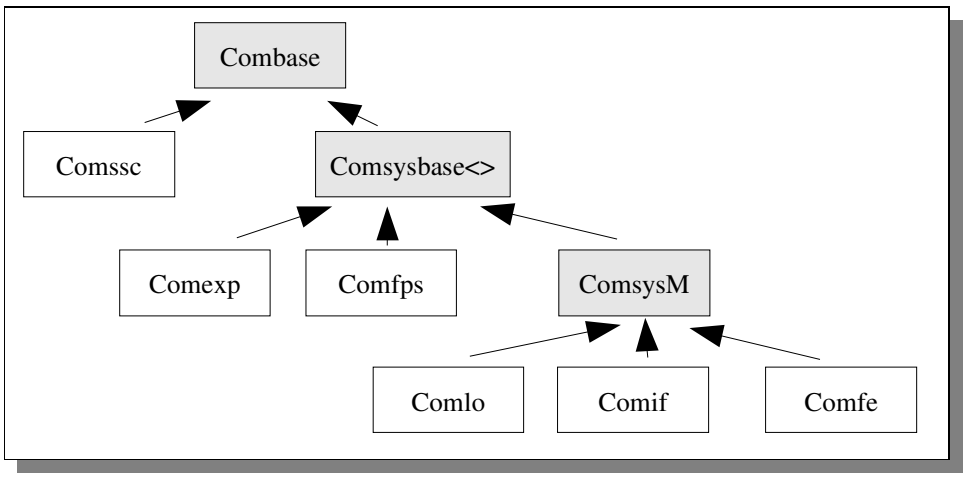


Illustration 8 : Inheritance tree of System Com classes

8.1. Abstract base Com classes

Abstract base Com classes are:

- Combase

This class provides the code to read and write packets exchanged between ABCcom and Teleset, according to the communication link protocol (cf. [7]). It is then very convenient to pop and push different types of data on the current message packet.

- Comsysbase<>

This class inherits Combase. It also provides functions to access loading and setting parameters of a system, to transmit direct MCM commands, and to return MCM time outs. This class is a template class (<class S>), so it can work with different System Threads.

- Comsys

This class inherits Comsysbase<SystM>. That means it contains a SystM System Thread object, described in the next section. Comsys functions decode commands common to SystM objects. They raise flags in SystM for operations like do settings, do monitoring, reboot... They also compose the 'State 'answer of the system which contains its current operations, parameters, eventual time-outs and error codes (cf. [7]).

8.2.Final System Com objects

System Com classes are:

- ✔ Comssc

Comssc contains a Servo Thread object, which itself contains a specific object for the tracking operations. In Cmdpacket() function, it decodes the Teleset command. This can be entering new parameters in the tracking object, transmitting a direct command to SSC, or returning 'State' packets of the Servo Thread object. These 'State' answer packets contain SSC event messages in priority, SSC last display answers (axis angles...), SSC answers to direct commands, eventual time outs and current tracking parameters.

- ✔ Comfps

Comfps contains a Fpssystm Thread object. It decodes commands to FPS system like loading feed count positions, moving to one feed, and direct Teleset commands. It also composes FPS 'State' answer containing its current Rpm and encoder position.

- ✔ Comexp

Comexp contains a Expsystm Thread object. It transmits direct Teleset commands to an MCM and returns the MCM answer faithfully along with eventual time-outs in the 'State' answer.

- ✔ Comlo

Comlo inherits Comsys class, its SystM Thread object is a Losystm (cf next section). Its personal functions are the ones decoding commands specific to LO system (modify synthesizer 2 frequency).

- ✔ Comfe

Comfe inherits Comsys class, its SystM Thread object is a Fesystm. Its personal functions are the ones decoding commands specific to FE system (set noise generation on, set common box off...).

- ✔ Comif.

Comif inherits Comsys class, its SystM Thread object is a Ifsystm. Its personal functions are the ones decoding commands specific to IF system (load or set pre-attenuation and post-gain parameters).

9. Servo System Thread class

System Thread classes are implementing the communication between ABCcom and antenna systems in a thread. There is one class per system: Servo, Expsystm, Fpssystm, Losystm, Ifsystm and Fesystm. Each class has variables to store the system parameters, and functions to generate operational commands and decode system answers.
Servo object is very specific and therefore independent of other MCM System classes. Its activity is recorded in a *SSCdata* history file for the Servo display answers, and in a *SSC* history files for other events or tracking operations.

9.1. Servo class

A Servo class object is a thread in charge of the communication with the SSC unit of the antenna. There is a full set of telemetry commands for SSC, basically divided into 3 categories: display commands, operational commands, and setting commands. All the communication protocol and the list of commands are detailed in ref. [1] and [2].

Packets exchanged with SSC are buffered in ABCcom program inside a PortSsc structure implemented in ***common*** files, rest of this class is coded in ***servo*** files.
PortSsc structure is a class containing several BaseFILO blocks, which are circular queues to store link packets:

- one BaseFILO for command packets to SSC
- one BaseFILO for answer packets from SSC, SSC Time-Outs and link errors.
- one BaseFILO for event packets from SSC (priority)
- one DisplayFILO, variation of BaseFILO adjusted to store last display answers from SSC

In the Start() function run by the thread, a Servo object performs the following tasks in a loop:

- ✔ Run a track operation if required (cf. 9.3)
- ✔ Send a display command (every 1 sec)
- ✔ Send a direct command from Teleset if required
- ✔ Receive responses from SSC
- ✔ Signal any time out of previous commands (AbcTimOut() function)

A Servo class contains 3 objects to help him perform this telemetry work: one of WriteCmd class, one of ReadResp class, and a bigger one of Track class. Next paragraphs describe in details those sub classes.
The Servo object is also opening and closing the Serial Port dedicated to communication with SSC, using ***ssclink*** file functions (cf. 12.2).

9.2. WriteCmd and ReadResp classes

WriteCmd is capable of writing whether packets from a BaseFILO block, or the next display command, to SSC on the Serial Port. It will then record whether the message has been acknowledged according to the SSC link protocol.

ReadResp is polling the Serial Port to read incoming messages from SSC. When it gets a message, it decodes the header to know which command this packet is answering to (operational / event / display). This allows to detect time outs and to store packets in different BaseFILO (or DisplayFILO) of PortSsc. This distinction is needed to give priorities between different kind of packets (Comssc takes events first).

Both classes are using writing and reading functions from ***ssclink*** file (cf. 12.2).

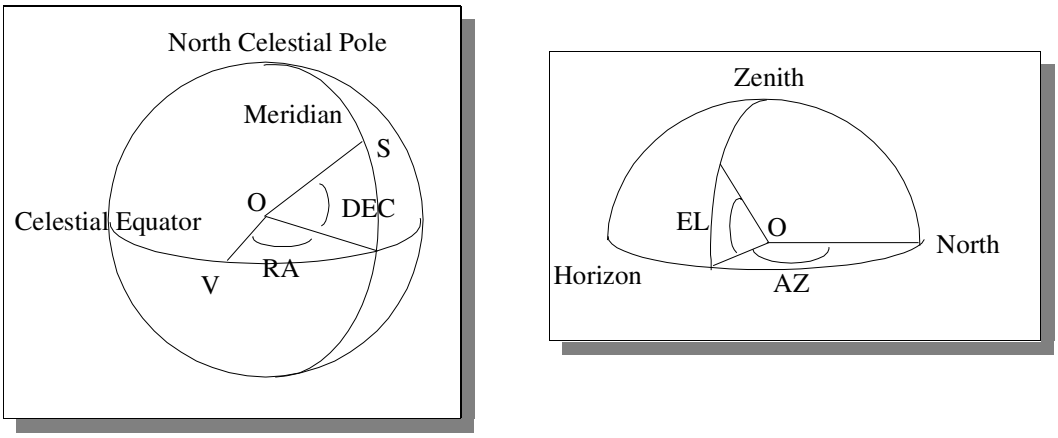
9.3.Track class

Servo Thread is having a special mode in which it will send regularly a Track command to SSC. This mode is coded in Track class.
When an antenna has to point at a specific source for some period, elevation and azimuth angles of the dish have to be actualized to compensate the earth rotation, and sometimes the source motion. Every 30 seconds, target angles are then recalculated and sent to SSC in a Track command.

9.3.1.Coordinate systems

A source in the sky can be located by different coordinate systems. The Horizon system is centered on the observer, and it is based on the horizon plan and the zenith axis. A source is located by the Elevation (or Altitude) and the Azimuth angles. The elevation is the vertical angle, and the azimuth is the angle in the horizon plan, with reference to the north. Those values vary as the earth is rotating.

Another coordinate system is the Equatorial system. It is centered at the observer and based on the celestial equator plan and the north celestial pole. A source is located by the Right Ascension and Declinaison angles. The declinaison is the vertical angle, and the right ascension is the angle in the equator plan between the star's meridian and the Vernal Equinox meridian. In this system, the celestial equator plan and the north celestial pole are fixed, so the coordinates of a source are constant.



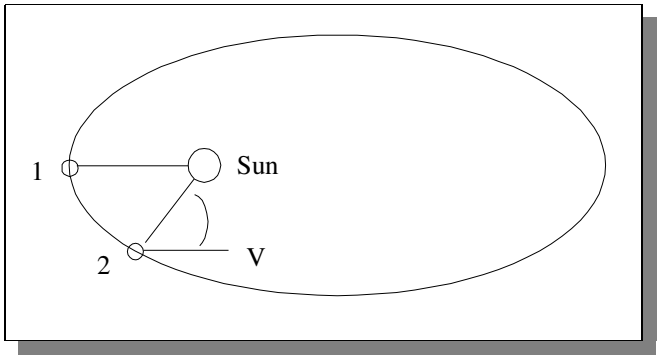
V : Vernal Equinox
RA : Right Ascension
DEC : Declinaison

AZ = Azimuth
EL = Elevation

Illustration 9 : Coordinates systems (left: equatorial, right: horizon)

9.3.2.Sidereal time

The current time system is called Universal Time, it gives 12:00 when the sun crosses the local meridian. The Sidereal Time is more accurate, it gives 24 hours for a 360 degree earth rotation. The Local Sidereal Time is the angle between the observer's meridian and the Vernal Equinox meridian.



Conversion:

24 hours = 360 deg
$24h_{ST} = 24h_{UT} - 360deg / 365$

V : Vernal Equinox, LST = 00:00:00 when the sun crosses the observer's meridian.

Illustration 10 : Sidereal Time

From there a method (or an equation) is giving the Local Sidereal Time (LST) of the observer from its current UT and its longitude. It is coded in Lmst() and MjdCal() functions of Track class.

9.3.3.Angles conversion

To calculate the new elevation (EL) and azimuth (AZ) angles of the antenna, the Track object needs to know the source declinaison (DEC) and right ascension (RA).
It first converts the UT time into Local Sidereal Time (LST) using the antenna longitude. The Hour Angle (HA) is the angle between the antenna's meridian and the star's meridian (GetHA() function). Tehn a linear formulas give EL and AZ from DEC, RA, HA and the antenna latitude (LAT). They are implemented in TrkCmd() function.

$$HA = RA - LST$$
$$EL = \arcsin(\sin(LAT).\sin(DEC) + \cos(LAT).\cos(DEC).\cos(HA))$$
$$AZ = \arctan\left(\frac{(\cos(DEC).\sin(HA))}{(\sin(LAT).\cos(DEC).\cos(HA) - \cos(LAT).\sin(DEC))}\right)$$

Illustration 11 : Angles conversion formulas

9.3.4.Tracking routine

Some additional parameters must be taken into account when implementing the tracking routine.

Azimuth outer / inner angle:
The Azimuth axis can go continuously from -270 to + 270 deg. The formula returns an angle in [-180; +180], when using an atan2() that takes into account the numerator and denominator signs. The conversion between GMRT antenna azimuth and astronomical azimuth is given in Illustration 11. When the formula gives an angle in [0;90] or in [270;360], one must then choose if the angle given to SSC will be in [-270;-90] or in [+90;+270]. This is set by a OutTrack parameter sent from Teleset.

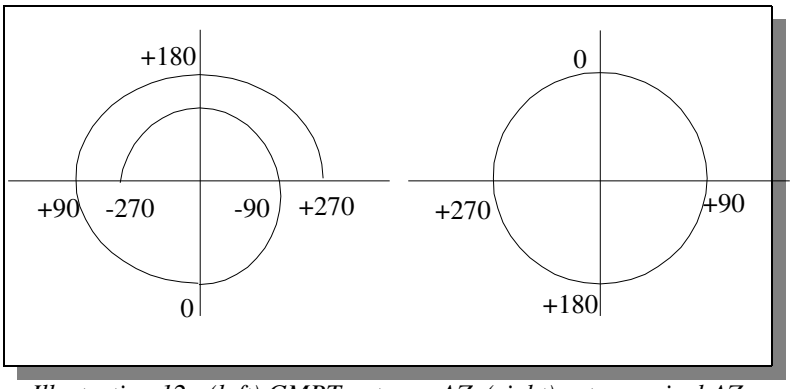


Illustration 12 : (left) GMRT antenna AZ, (right) astronomical AZ

The elevation antenna axis can go from +15 to +110 degrees. The formula answer gives an angle in [-90; +90]. Below +15 deg, the source is too low or below the horizon to be pointed. And when going beyond +90 deg, EL diminishes and AZ will be +/-180 deg by the formula. Some time is needed for this rotation and this zone is of lesser pointing accuracy.

SSC buffer for Track commands:
A Track command takes 3 parameters: EL, AZ, and time. SSC processes the command so that the AZ and EL are reached at the given time. But SSC also has a buffer for one tracking command. If a track command is received while another is in process, the SSC keeps it in the buffer, completes its current tracking and processes the buffered command just after.
So the tracking mode steps are:

- ✔ t_0 : send Track, t_0 + 30 secs, EL and AZ of target position at t_0+30secs
- ✔ $t_0$bis (just after t_0) : send Track, t_0+60 secs, EL and AZ of target position at t_0+60 secs
- ✔ $t_1 = t_0 + 30$ secs : send Track, t_1+60 secs, EL and AZ of target position at t_1+60 secs
- ✔ ...
- ✔ $t_n = t_{n-1} + 30$ secs : send Track, t_n+60 secs, EL and AZ of target position t_n+60 secs

This is implemented in Run() function of the Track class, with use of TrkCount variable and calls to NewTrkCmd() and TrkCmd()) functions.

The table below resumes parameters required for the Track object. They are sent by Teleset. Rate parameters are different from 0 when the source motion cannot be neglected.

	Parameter	***Description***
Loading parameters	AZ offset angle	Offset to adjust antenna AZ axis
	EL offset angle	Offset to adjust antenna EL axis
	Antenna Latitude	in degree
	Antenna Longitude	in degree
Setting parameters	TrackFlag	Flag to start / stop tracking mode
	OutTrack	Flag to choose [-270;-90] or [90;270]
	Right Ascension	source RA, in radian
	DEClinaison	source DEC, in radian
	RA rate	in deg/hour
	DEC rate	in deg/hour
	Time rate	Time reference for the rates, in hour

Illustration 13 : Tracking parameters

10.MCM System Thread classes

A MCM System Thread class is implementing the communication in a thread between ABCcom and an antenna system which has some MCM units. Those classes are Expsystm, Fpssystm, Losystm, Ifsystm and Fesystm respectively for EXP, FPS, LO, IF, and FE Systems. Each class has variables to store the system parameters, and functions to generate operational commands and decode system answers.
When MCM System Thread objects are created, pointers to corresponding unit buffers are passed to them so they can put in packets to write on the port, and get unit answers. For instance, Losystm has pointers to buffers of MCM 2 and MCM 3 to communicate with these units (cf. Illustration 5).
Each System Thread activity is recorded in a specific history file (*SysEXP*, *sysFPS*, *sysLO*, *sysFE*, and *sysIF*).

MCM System Thread classes are coded according to the following inheritance tree diagram:

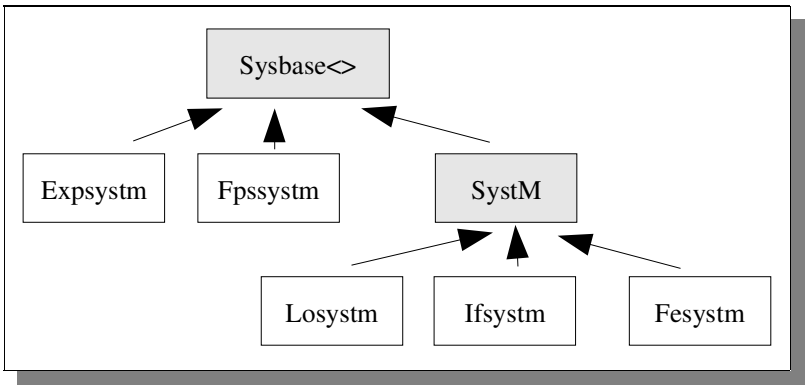


Illustration 14 : Inheritance tree of MCM System classes

10.1.LowMcm and LowFps file classes

Two protocols are used for MCM communication in GMRT systems. A standard one concerns MCM 2, 3, 5, and 10 and is detailed in ref. [5]. It is implemented in LowWrite and LowRead classes of ***lowmcm*** files.
A derived protocol is used for MCM 14 of the FPS system, and is detailed in ref. [3] and [4]. It is implemented in LowfWrite and LowfRead classes of ***lowfps*** files.
A packet message to MCM is built in two successive tasks. First the command code and its arguments are stored by Thread System functions in a CMD structure, named CurrentCMD. Then this structure is passed to a function of LowWrite (or LowfWrite) class, to form the final MSG message which is sent on the Serial Port.
A MSG structure is coded in C to be passed to write and read functions of link communication (cf. section 12). It is not possible to use a string library since data bytes can happen to be an ASCII delimiter. So the packet size is stored along the data bytes in MSG.
In the other direction, a function of LowRead (or LowfRead) class is used to decode MCM packets headers.

Such a transition is not required for SSC packets since their structure (in ASCII) is simpler and therefore packets can be formed and decoded in one step.

10.2.Sysbase<> abstract base class

Sysbase<> is a base class coded in ***sysbase*** files. It gathers arguments and functions that can be used for all derived classes. It has two template variables that are respectively either of LowWrite and LowRead classes, or LowWrite and LowRead classes. Its other main variables are:

- ✗ Setparam[] and Loadparam[] store the system parameters. All parameters are floats, the setting ones are for the current system values, and loading ones are constant for an antenna.
- ✗ Mcm1 and Mcm2 are pointers to system unit buffers.
- ✗ CurrentCmd is the structure for the current MCM command. Most of operations are a series of several commands, placed in Mcm1 or Mcm2.
- ✗ DataPlus is the structure for storing data (error codes...). It is transmitted to Teleset during the next 'State' answer.
- ✗ CmdCode contain flag bits for basic system operations.
- ✗ CmdState[i] contains the i^{eme} operation status:

Bit no	0	1	2	3	7
command	In process	failed	done	discarded	Syst not set

- ✗ MonInterval: number of seconds between automatic operations.

Sysbase<> main functions are:

- ➢ Start() is the main loop of the thread, calling successively:
  - ➢ AnalyseCom() generates commands to the system, implemented in derived objects.
  - ➢ AnalyseAns() decodes the system answers with Read() and PostRead() sub functions.
  - ➢ AutoOperation() starts AutoSys() every MonInterval seconds. AutoSys() is implemented in derived objects.
- ➢ HardRecordOn(), HardRecordOff() write system parameters in a file. When the PC reboots unexpectedly (power failure) and ABCcom restarts, Init() restores the parameters from this file and reset them with Reconfigure() (implemented in derived classes).
- ➢ DirectCmd() and Copy() are functions to write commands in an unit buffer. DirectCmd() is for a single command, Copy for a set of several commands. The buffer has flags to signal when commands can be sent and when answers can be read (FileMcm class).

10.3.SysM abstract base class

SysM class is coded in ***systm*** files. It is a base class providing variables and functions common to LO, IF and FE systems. Beside inheriting Sysbase<LowWrite, LowRead>, its main variables are:

- ✗ Monvolt[], to store system voltages decoded after a monitoring operation
- ✗ CmdSys is a byte to code system specific operations (when bit 4 of CmdCode is raised), and StateSys contains its status. The code used is the same than for other operations, but only one CmdSys can be in process at a time.
- ✗ thSys is a p_thread POSIX variable. Threads are normally implemented in C. Here a C helper function is used for every thread. This HelperSYS C function calls Start() for the SysM object passed in argument.

SysM main functions are:

- AnalyseCom() decodes CmdCode byte according to the next table.

Bit nb	0	1	2	3	4
command	Monitor()	Set()	Reboot()	Exit	SysCmd()

- Write() then starts the appropriate function (Set(), Monitor(), Reboot() or SysCmd(), implemented in derived objects) and update CmdState bytes.
- AutoSys() raises the monitoring operation bit of CmdCode ; Reconfigure() raises the setting operation bit of CmdCode.
- PostRead() starts the analyze of monitoring answer, and update CmdState bytes.
- Other functions are routines to convert data types.

10.4.Final System Threads classes

10.4.1.Expsystm

Expsystm is coded in ***sysbase*** files. It is the Thread class for the MCM Expert System and it inherits Sysbase<LowWrite, LowRead>.
Expsystm is meant only to transmit direct command (CmdCode = 0x01) to the MCM pointed by Mcm1 variable. Answers are placed faithfully in DataPlus variable and transmitted in the next 'State' answer, with eventual time-outs.

10.4.2.Fpssystm

Fpssystm is coded in ***fpssystm*** files. It inherits Sysbase<LowfWrite, LowfRead> and has the following specifications:

- Loading parameters: 4 encoder positions for the feed turret.
- Setting parameter: current feed selected.
- Operation functions and CmdCode :

CmdCode bit nb	0	1	2	3
function	InitFps()	MvFps()	DirectCmd()	Exit

- Decoding functions: DecodeList(), DecodeStatus().
- Decoding variables: EncCount, Rpm, AnsStatus, FpsStatus. DataPlus is filled with the second logical sub packet of FPS answer (cf. [3] and [4]).
- Reconfigure() calls MvFps().
- AutoSys() sends a FPS Null command with DirectCmd().

10.4.3.Losystm, Ifsystm and Fesystm

Losystm is coded in ***losystm*** files, Ifsystm in ***ifsystm*** files, and Fesystm in ***fesystm*** files. They all inherit SystM class which is described in a previous paragraph. Their files contain the implementation of functions specific to the system:

- Operation functions: Set(), Monitor(), Reboot(), SysCmd().
- Decoding functions: ExtractData(), CheckSettings(), UpdateStateSys().

Additional sub functions are mostly routines called by the above functions.

Class variables are resumed in the next table:

		Losystm	***Ifsystm***	***Fesystm***
loading parameters		--	- Pre-Attenuations Ch1, Ch2 - Post-Gains Ch1, Ch2 (all ant values: 8x11)	--
setting parameters		- LO frequency 1 - LO frequency 2	- Bandwidth Ch1, Ch2 - ALC On/Off Ch1, Ch2 - Frequency band 1, frequency band 2 (0 if not dual) - Pre-Attenuations Ch1, Ch2 - Post-Gains Ch1, Ch2	- Noise generation cycle - Walsh enable, Walsh group - Frequency band 1, frequency band 2 (0 if not dual) - Solar attenuation 1, solar attenuation 2 (0 if not dual) - Polarization swap - Noise calibration level
SysCmd		- 1: set aage - 2: set piche	- 1: set specific pre-attenuation and post-gain for Ch1 and 2	- 1: set MCM 5 on - 2: set MCM 5 off - 3: set NG on - 4: set NG off - 5: set NG 50% on - 6: set NG 25% on
SysState		- 0x01: aage set - 0x02: piche set	- 0x01: specific pre-attenuation and post-gain set for Ch1 and 2	- 1st bit (0x01): MCM 5 on/off - 2nd bit: NG on/off - 3rd bit: NG50 on/off - 4th bit (0x08): NG25 on/off
decoding variables	DataPlus	- monitoring error code - 2 monitored values relative to FE	- IF array of raw monitored data.	- monitoring error code
	Monvolt	- 2 LO lock voltages.	--	- reference voltages for FE box 1 and FE box 2 (4 floats)

Illustration 15 : SystM derived class parameters

After reading monitoring answers from a system, the Threads checks if the monitored data matches with the setting parameters. In case they do not, a monitoring error code is formed in DataPlus variable to be sent to Teleset with the next 'State' answer.

- LO error code: composed of 12 bytes, 6 for MCM2 and 6 for MCM3. Depending on the selected LO synthesizers, those bytes may not be relevant. A decoding stage writes values of error bytes, which are then 'and-ed' with a relevance mask to form the final error code.
  MCM 2 is for synthesizer 1, and MCM 3 for synthesizer 2. Their 6 error bytes follow the table below:

Bit nb	0	1	2	3	4	5	6	7
byte 0	N5	N6						
byte 1	M1	M2	M3	M4	N1	N2	N3	N4
byte 2	YigDac1	YigDac2	YigDac3	YigDac4	YigDac5	YigDac6	YigAt1	YigAt2
byte 3	Fstep1	Fstep5						
byte 4	D77cp1	D77cp2	D77cp3	D77cp4	D77cp5	D77cp6	D64cp1	D64cp2
byte 5	Vco1On	Vco2On	Vco3On	YigOn				

'Relevance' Masks:
mask 1: [0x03, 0xff, 0x00, 0x03, 0xff, 0x0f] ; mask 2: [0x00, 0x00, 0x00, 0x00, 0xc0, 0x00]
If synthesizer 1 is used (one LO < 600MHz), the error code is obtained by 'and-ing' mask1 to the first 6 bytes, else by 'and-ing' mask2. If synthesizer 2 is used (one LO > 600MHz), the error code is obtained directly by the last 6 bytes, else by 'and-ing' mask2 to them.

- FE error code: composed of 3 bytes. Except in dual frequency observation, only one FE box is used and the third byte is always null. The error bytes follow the table below:

Bit nb	0	1	2	3	4	***5***
byte 0: CB	Ch2 frequency	Ch2 solar attenuation	swap		Ch1 frequency	Ch2 solar attenuation
byte 1: FE box 1	Freq.	calibration noise level	Walsh on/off	RF on/off	noise generation on/off	
byte 2: FE box 2	Freq.	calibration noise level	Walsh on/off	RF on/off	noise generation on/off	

- IF error code: checking operation is not implemented yet, the IF system being currently changed in GMRT. Furthermore, the IF monitoring includes the LOR monitoring. All answers contain data that are crucial and must be transmitted to Teleset. The total monitoring results in a 16x30 raw data array, stored in DataPlus.

11.PortMulti and Comh classes

High level communication between ABCcom and antenna systems has been explained in previous sections. From now on the description starts to go down to lower communication layers, with objects closer to the PC Serial Ports.

11.1.PortMulti, FileMcm and ListMcm classes

With RS-485 link, many MCMs can be connected on one serial port, in opposition to SSC (RS-422) which has one dedicated port. This MCM feature imposes a specific software design.
From ***ConfigABC*** file, ABCcom program creates one PortMulti object per MCM-serial port. This object has a PortMcm variable, including one FileMcm variable per MCM unit connected, itself divided into two ListMcm (cf. next diagram). Its activity is recorded in a *MiPort* history file, with i index of the port.
PortMcm and its sub classes are implemented in ***common*** files.

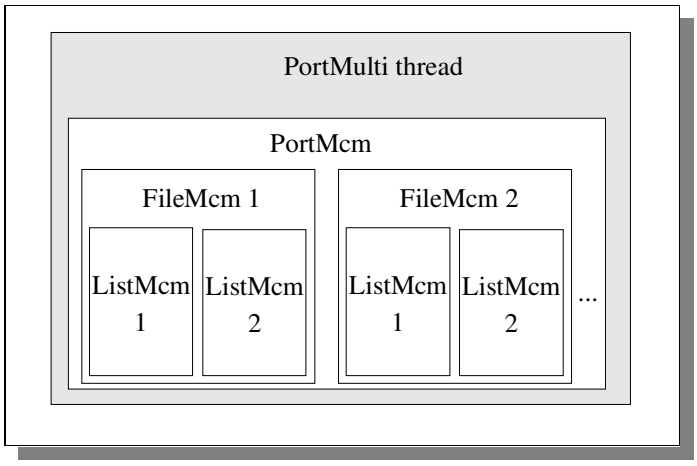


Illustration 16 : PortMcm structure

☞*Note: A PortSsc structure for the SSC-serial port has already been described as a part of Servo Thread (cf. 9.1).*

A ListMcm object is composed of a chained list of two MSG structures, one for commands and one for answer messages. A chained list corresponds to one series of commands (or answers), which corresponds to one operation. ListPrio and CurrentSys are integers used to synchronize the list access between different threads.

There is one FileMcm object for each MCM unit connected on the port. It is represented as a NCN unit buffer in Illustration 5. It stores its MCM address, the number of time-outs occurred with this unit, and two ListMcm objects. ListMcm 2 is treated by PortMulti in higher priority than ListMcm 1. Typically, System Threads which have pointers on FileMcm objects, place long series of monitoring commands in ListMcm 1, and other series of commands like reboot or set in ListMcm 2.

The PortMulti class is implemented in ***mcmport*** files. A PortMulti object handles all communication going on through one MCM-serial port. In a loop, its thread selects a FileMcm and one valid ListMcm, and sends one command on the serial port. Then it gets the MCM answer, places it in the ListMcm chain, and updates different variables (next list element, time-outs, synchronization variables). The loop restarts with the next valid FileMcm. For communication on the serial port, PortMulti uses C functions implemented in ***mcmdriver.c*** file (cf. 12.3).

11.2.Comh and LocalComh classes

Comh is the class of ABCcom program in charge of handling packets exchanged with Teleset. For the communication on the serial port, Comh uses C functions implemented in ***comhlink*** files (cf. 12.1). Packets from Teleset are stored in a PortComh FILO structure, and packets to Teleset are directly sent. Comh also filters incoming packets according to the ABC address.

LocalComh is the class equivalent to Comh, but adapted to the local Teleset communication. Instead of using a serial port, it communicates through a Shared Memory. On Teleset command, ABCcom can starts and stop a local mode (cf. 7.2). This is equivalent to creating or deleting a LocalComh object. This one creates and initializes a shared memory structure (ShareCom), which contains buffers for packets exchanged with Local Teleset. Thus any Local Teleset would fail attaching to the Shared Memory when ABCcom has its local mode off.

Comh and LocalComh are implemented in ***comh*** files. Their activities are recorded respectively in *Comh* and *LocalComh* history files.

12.Serial Port communications

A serial communication is asynchronous and implemented in Linux by providing the open, close, read and write functions for the PC serial ports. ABCcom has three different sorts of serial communication, each implemented in ***ssclink***, ***comhlink***, ***shelllink*** and ***mcmcom*** C files.

12.1.Teleset link communication

The ABC is connected on one side with the CEB, via the optical link up to Online PC. A complex hardware card (like CEBCOM PIU) has to be used to link ABC PC with GMRT optical link. For the implementation and testing phases of the ABCcom, I have used a simple RS-232 serial port communication connected to another serial port controlled by Teleset. Bytes are coded on 10 bits: 1 start bit, 8 data bits and 1 stop bits. The baud rate is chosen at 9.6 kbps.
Two different approaches have been implemented. In ***shelllink*** files, the 'read' function is in non-blocking mode with a signal interruption handler. When data arrive on the port, Linux signal SIGIO is raised and data are read by one handler function, and stored in a circular queue. A higher read function polls this queue. The ***comhlink*** files implement a simpler 'read' function without interruption handler, assuming that the higher read function will poll the port frequently enough. In ABCcom, Comh block is using comhlink functions. AbcShell is using shelllink functions.

12.2. Servo link communication

SSC unit is connected to ABC through a serial full-duplex RS-422 link. The RS-422 standard allows long cable distances thanks to its twisted pairs for Rx and Tx lines. Bytes are coded on 11 bits: 1 start bit, 8 data bits and 2 stop bits. The baud rate is fixed at 9.6 kbps.
A PC is originally equipped with one or two serial ports, on which the default communication follows the RS-232 standard. A conversion to RS-422 standard only requires a very small hardware converter, and no software change.
ABC sends messages to SSC which then answers, but SSC can also send an event message by itself. For this communication (cf. [1] and [2]), a protocol is used to provide acknowledgment, error detection, and retry mechanisms. It is character oriented using the ASCII standard format. The ***ssclink*** files implement the low level functions of this protocol, the 'read' function is in non-blocking mode with a signal interruption handler.

12.3.MCM link communication

RS-485 standard allows long distance multi-point communication, but restricts it to half duplex. Thus a master (ABC) and many slave components (MCMs) can be connected on a single link.
RS-485 communication with a PC requires a small hardware converter, and a 485 driver implementing a 9-bit protocol for the multi-point performance at 9.6 kbps. This program is ***mcmdriver***, a Kernel module for Linux written for MCM communications (cf. [5]). A better second version is used, it has to be previously loaded on appropriated ports.

13.Conclusion

ABCcom is a Linux PC software for GMRT Telemetry System. It is to be placed in an antenna shell to communicate on one side with antenna systems via serial port connections, and with Teleset control room software on the other side via optical link.

This software is mainly coded in C++ and thus allows an object oriented design. This is very convenient for general organization and future evolutions. Shall any modification be done in ABCcom code, the programmer must first understand the general block diagram to locate which object he wants to write in and how this can influence surrounding blocks.
ABCcom uses a multi-threading technique to optimize its efficiency. It also performs advanced telemetry tasks compared to the previous ABC unit, reducing the communication between GMRT control room and antennas.

ABCcom is setting and monitoring each antenna system (LO, FE, IF, FPS and Servo). It does not require any modification on those systems. On the control room side, the protocol is fully changed and Teleset program has to be installed to communicate with all antenna ABCcom PCs.

To conclude, ABCcom software was tested in GMRT laboratories all along its implementation phase. Running locally a first version of Teleset program, ABCcom software has been thoroughly and successfully tested in GMRT C9 antenna during the maintenance period of November 2005.

References and label index

References:

[1] BARC, *Servo software BARC: SSC interface details*, April 1991
[2] BARC, *Newsmu pascal program code*, 1997
[3] Mukund Gadgil, *Feed Positioning System Software*, April 1992, GMRT technical report.
[4] Mukund Gadgil, *FPS DOS program code*, 1992.
[5] Laurent Pommier, *Mcmcom program, a Linux PC / MCM communication*, September 2004, GMRT technical report.
[6] Laurent Pommier, *Teleset software of GMRT Telemetry System* , March 2006, GMRT technical report.
[7] Laurent Pommier, *Communication protocol between ABCcom and Teleset*, March 2006, GMRT technical report.

Index table:

Label	***Full Name***
ABC	Antenna Based Computer
ANTCOM	Antenna Computer
AZ	Azimuth
CEB	Central Electronic Building
COMH	Communication Handler
DEC	Declinaison
EL	Elevation
EXP	Expert system
FE	Front End system
FILO	First In Last Out list
FPS	Feed Positioning System
GMRT	Giant Metrewave Radio Telescope
IF	Intermediary Frequency system
LO	Local Oscillator system
MCM	Monitor and Control Module
PIU	Plug In Unit
POSIX	Portable Operating System Interface
RA	Right Ascension
RS	Research Standard
SSC	Station Servo Computer
UART	Universal Asynchronous Receiver Transmitter

Appendix A: Tables of ABCcom file contents

- C++ files for general and System Com classes:

Filename	*Included headers*	*Classes*	*Description*
mainshell (.cpp, .h)	common.h	AbcShell	Mode for Remote Linux shell and files transfer ABCcom main function
abccom (.cpp, .h)	common.h, mcmport.h, comh.h, comsys.h, comhigh.h	Config, Comabc, AbcPlus	Initial ABCcom configuration, Transfer of packets between Comh object and system objects
combase (.cpp, .h)	common.h, systm.h, servo.h	Combase, Comssc	Basic com. utilities and SSC specific com
comsys (.cpp, .h)	common.h, systm.h, fpssystm.h, comhigh.h	Comsysbase<>, Comsys, Comlo, Comif, Comfe, Comexp, Comfps	Com. utilities for MCM systems, Com. classes for LO, IF, FE, EXP and FPS
common (.cpp, .h)	none	MSG (C struct), CMD, LogFile, ListMcm, FileMcm. PortMcm, BaseFILO, DisplayFILO, PortSsc	Objects common to several files, Buffer structures, Log file class for the ABCcom history.

- C++ files for System Thread classes:

Filename	*Included headers*	*Classes*	*Description*
sysbase (.cpp, .h)	common.h, lowmcm.h	SysBase<>, Expsystm	Base object for MCM system thread thread for operations with EXP
systm (.cpp, .h)	common.h, lowmcm.h sysbase.h	SystM	Base object for LO / IF / FE systems
fesystm (.cpp, .h)	common.h, systm.h	Fesystm	Thread for operations with FE
fpssystm (.cpp, .h)	common.h, lowfps.h sysbase.h	Fpssystm	Thread for operations with FPS
ifsystm (.cpp, .h)	common.h, systm.h	Ifsystm	Thread for operations with IF
losystm (.cpp, .h)	common.h, systm.h	Losystm	Thread for operations with LO
servo (.cpp, .h)	common.h	WriteCmd, ReadResp, Track, Servo	Thread for operations with Servo

- C++ files for lower level classes:

Filename	*Included headers*	*Classes*	*Description*
lowfps (.cpp, .h)	common.h	LowfWrite, LowfRead	Make final FPS com. packets Decode FPS com. header
lowmcm (.cpp, .h)	common.h	LowWrite, LowRead	Make final MCM com. packets Decode MCM com. header
mcmport (.cpp, .h)	common.h	PortMulti	Handler thread for MCM com. on a serial port
comh (.cpp, .h)	common.h	Comh, ShareCom, LocalComh	Com. on serial port with Teleset Com. through shared memory with local Teleset

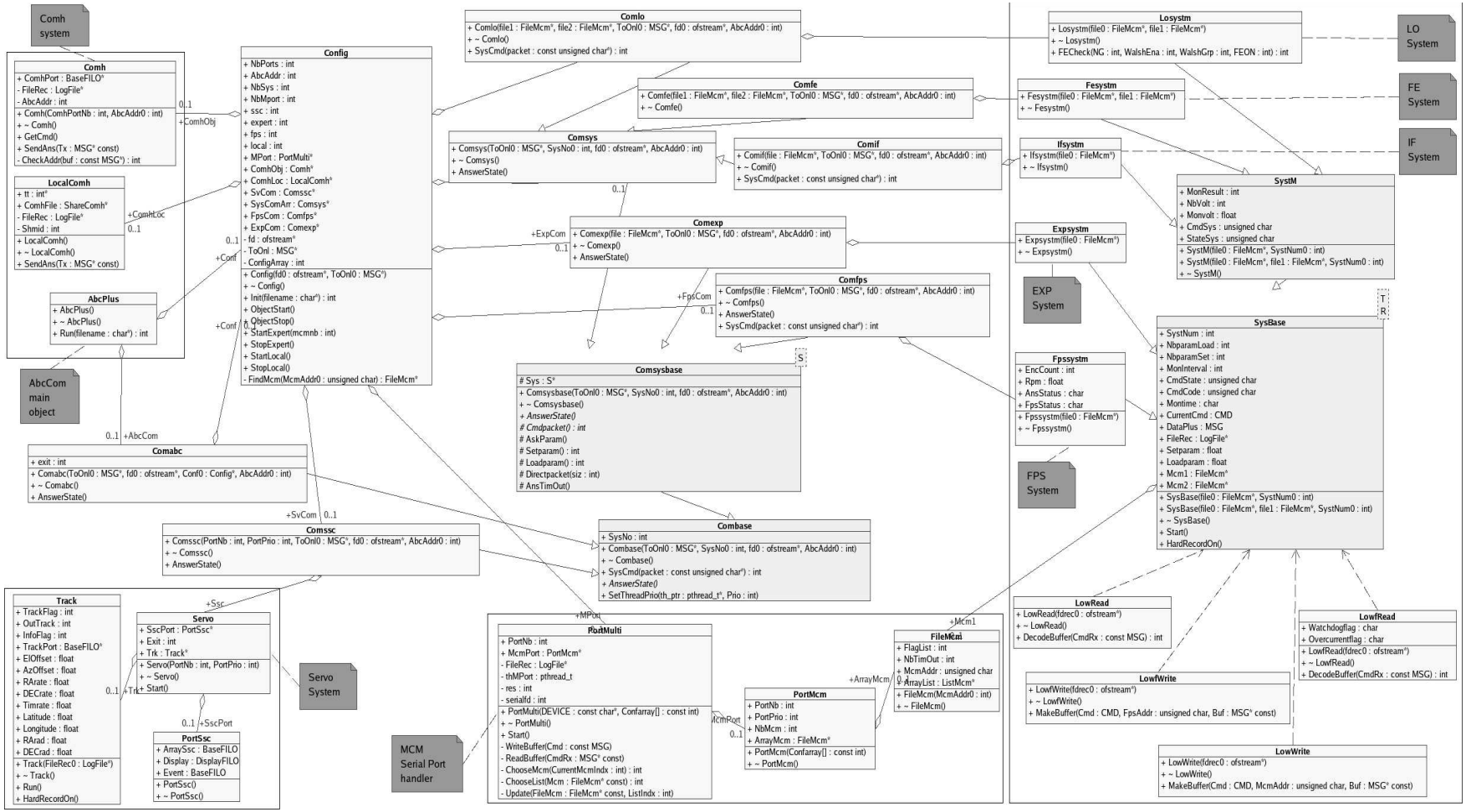
- C files for serial port communication:

Filename	*Included headers*	*Description*
comhlink (.c, .h)	none	Serial port communication with Teleset for ABC mode C open/write/read/close functions (RS-232, poll and master/slave)
shelllink (.c, .h)	none	Serial port communication with Teleset for shell mode C open/write/read/close functions (RS-232, interrupt based and full duplex))
ssclink (.c, .h)	none	Serial port communication with Servo Bin C open/write/read/close functions (RS-232, interrupt based and full duplex) Servo link layer protocol (acknowledgment and retry mechanisms)
mcmdriver (.c)	none	Kernel module driver for RS-485 communication between MCM units and Linux PC (9^{th} bit protocol for multi-point link). Must be loaded separately from ABCcom program.

Appendix B: ABCcom history file

- 29Dec14h_ABC_rec
-

Appendix C: ABCcom UML class diagrams



Appendix D: ABCcom source code

- ***Makefile*** for mcmdriver
- ***mcmdriver.c*** (last version)

- ***Makefile*** for ABCcom
- ABCcom code files:

abccom.cpp	***common.cpp***	***ifsystm.cpp***	***mainshell.cpp***	***ssclink.c***
abccom.h	***common.h***	***ifsystm.h***	***mainshell.h***	***ssclink.h***
combase.cpp	***comsys.cpp***	***losystm.cpp***	***mcmport.cpp***	***sysbase.cpp***
combase.h	***comsys.h***	***losystm.h***	***mcmport.h***	***sysbase.h***
comh.cpp	***fesystm.cpp***	***lowfps.cpp***	***servo.cpp***	***systm.cpp***
comh.h	***fesystm.h***	***lowfps.h***	***servo.h***	***systm.h***
comhlink.c	***fpssystm.cpp***	***lowmcm.cpp***	***shelllink.c***	
comhlink.h	***fpssystm.h***	***lowmcm.h***	***shelllink.h***	