GALILEO Project.
WORKSTATION SOFTWARE SYSTEM
Architecture Design Document

Technical Report #2

[1]A. Balestra, [1]P. Marcucci, [1,2]F. Pasian,
[1]M. Pucillo, [1]R. Smareglia, [1]C. Vuerli

[1] TNG Project, Astronomical Observatory of Trieste
[2] ST-ECF/ESO, Garching

December 1991

Pubbl. Osservatorio Astronomico di Trieste n. 1428

Abstract

This document contains the Architectural Design of the Workstation Software System, which, together with the Telescope Software System, is part of the GALILEO Telescope Control System. Special emphasys is put both on general issues like software standards, portability and modularity, and on more particular ones as remote control capabilities and fault tolerance. The document is divided into two main sections : the first describes the environment needed for the WSS to operate, including all off-line utilities dedicated to the housekeeping of the characteristics files. The other one concerns the structure of the on-line software, i.e. the monitoring processes and the user interface.

1 - INTRODUCTION

The general structure of TCS has been already described elsewhere [1]. It is sketched in fig. 1 with its main components put in clear evidence, in order to clarify the relationship between hardware and software structure of TCS, which will be described later on. The hardware structure is composed of a set of microprocessor systems based on the VME standard, containing the processing power and the hardware needed to operate the telescope subsystems and the instruments as well, and by a set of last generation, UNIX based, workstations. All these systems are connected by an Ethernet network with the TCP/IP protocol for commands and status exchange, while a different kind of connections, still to be defined, will guarantee the science data acquisition.

As already noted two software environments exist in the Galileo Telescope Control System (TCS) : the first, the Telescope Software System (TSS), will embody all the microprocessor systems, and has been thoroughly described in [2]. The second one, the Workstation Software System (WSS), will be given the task of monitoring the TSS activities and of interfacing the users with it. Moreover it will provide the TCS with some basic services, not directly related with the telescope control, such as science data archiving, observations data base consulting and off-line data processing.

2 - GENERAL CONCEPTS

The guidelines followed in the design of the general structure of WSS, have been carefully chosen in order to lead to a flexible and modular structure, having in mind, as far as it's possible, the future unavoidable changes and enhancements to the structure itself, both by the points of view of the hardware platforms and of the software tools used.

Three points have been dedicated the major part of the design effort. The first point has been the decision to design all the WSS software as a table-driven one. No data are hardwired within the code. All the data needed by WSS processes are fed to them at startup through disk resident tables. They are maintained with off-line dedicated editors which eventually produce sets of files in a format suitable for both WSS and TSS. Such tables will include the definitions of parameters, commands and any other quantity used by both systems.

Communications, both intranodes and internode, were the second point, for they are the backbone over which the whole system works. A suitable, fault tolerant communication system has been devised in order to guarantee secure TCS operations even in the case of a workstation failure. A similar mechanism is also provided by TSS.

A third point has also been considered worth an accurate design : the overall organization of the static (definitions) and dynamic (status information) data relevant to TCS operations into a data structure with homogeneous access methods, both direct and indexed, where all the WSS tasks will read or write the data of their interest during TCS operations. This structure can be considered as a simple Telescope Data Base (TDB), and will be created and maintained by a dedicated task.

In the following paragraphs a complete description of those concepts will be given, together with the architecture of the software of concern.

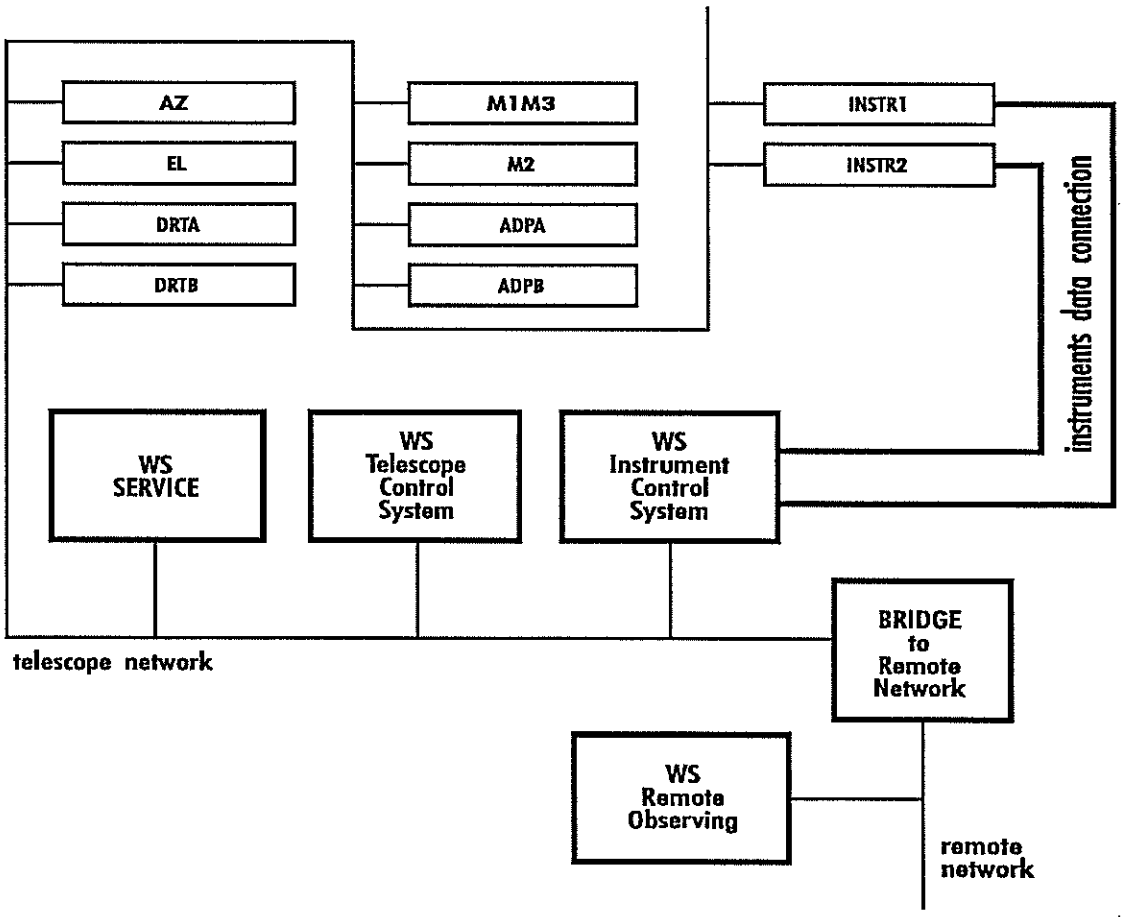


Figure 1: Telescope Control System

2.1 - TCS Structure and Naming Conventions

As can be seen in fig. 1, the telescope network will connect a set of nodes of two kinds: VME systems operating directly on the telescope subsystems and on the instruments, and workstation systems which are dedicated to interface the operators to the telescope and to monitor its activity and status.

Each VME system contains a set of tasks, or groups of logically correlated tasks, called units. Each unit controls a particular function of the subsystem through a set of parameters, and is in turn controlled through a set of commands. In the workstations case the same structure can be recognized substituting the concept of UNIX process to that of VME task.

In order to have the WSS working properly, a suitable naming convention has been defined, which maps directly to the TCS structure. The devised solution has been to introduce a hierarchical organization reproducing the physical structure of TCS, and to give each component a name built up by one to three fields, specifying the full path needed to reach it inside this structure. The first field will contain the name of the system, the second the name of the unit and the third the name of the item. The layout of the full name will thus be as follows :

<system>_<unit>_<item>

where:

<system> = four characters identifier of the TCS system to which the item belongs (e.g. VMAZ : azimuth control VME)

<unit> = three characters identifier of the the task (or group of tasks) enabled to act on the item (e.g. MPA : motors power amplifiers control task)

<item> = six characters self explaining acronym of the item itself (e.g. MT1CUR : motor 1 current)

The previous example leads to the following name :

VMAZ_MPA_MT1CUR

Obviously systems will have their name formed only by the first field, while units names will contain the first two fields. This approach allows the WSS to address correctly each item in the TCS, should it belong to the local system or to a remote one, and lends itself naturally to remote control and to remote observing.

The same convention has been used to build the names of the definition tables for the WSS components, as will be described in the following paragraph.

2.2 - Definition Files

As reported before, all the components of the WSS software are built as table driven modules. At startup each module will read its data either from the on-line TDB, built up during the initialization phase from the on-disk definition files, or directly from its private

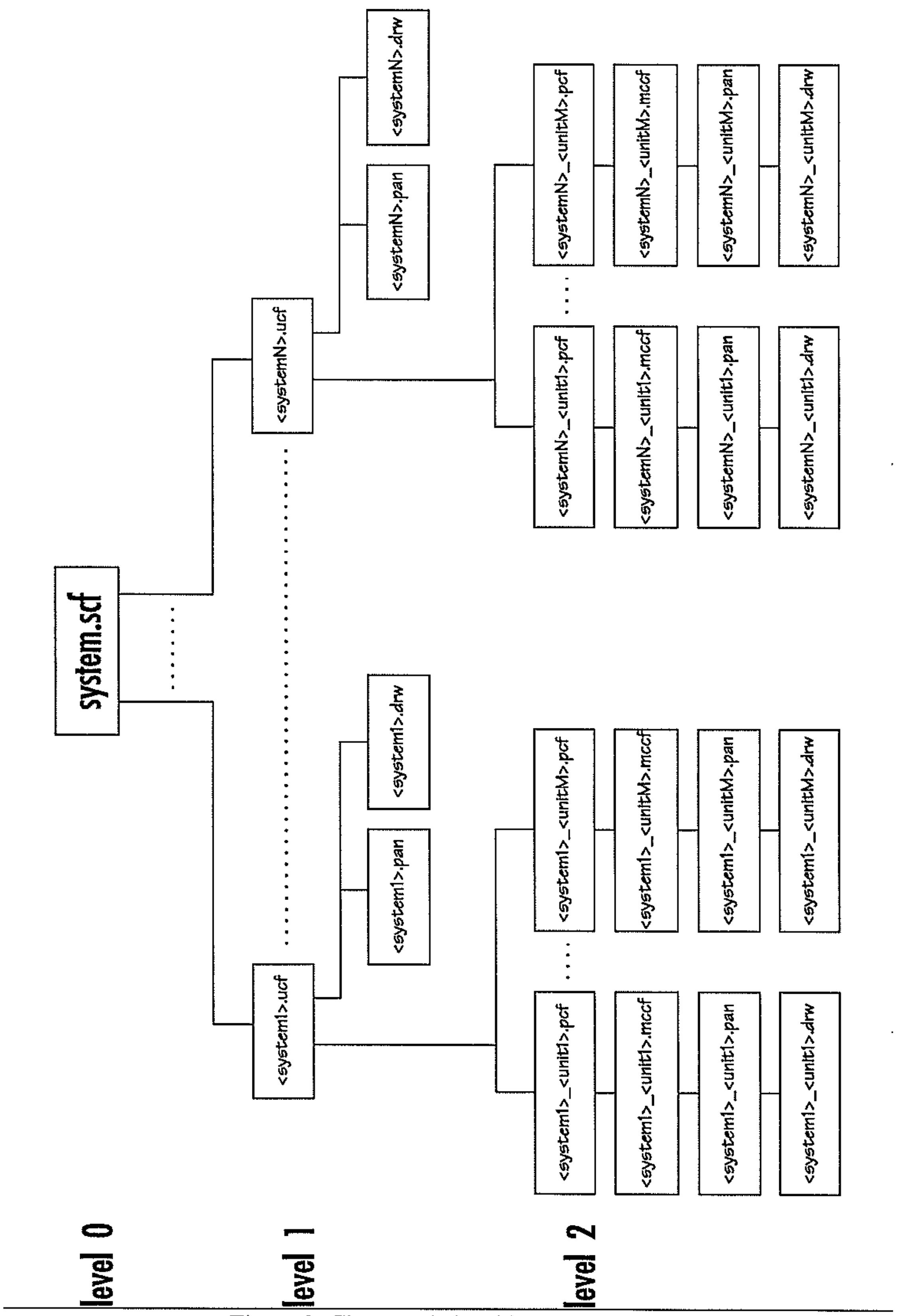


Figure 2: Characteristic tables organization

files. All those files are created and maintained by means of a set of off-line utilities which include a context-sensitive table editor to maintain the definition tables, an interactive panels editor for interaction panels and a graphic panels editor for graphic panels. Besides there are some programs which allow to convert definition tables from an old format to a new one, without the burden of having to rewrite them when a change of their structure is required by an upgrade in the software.

Tables, graphic panels and interaction panels contain all the information needed by WSS processes to interact with TSS and with the user. They are organized in a tree-like structure as reported in fig. 2. The names of the files follow the naming convention already discussed in par. 2.1, so that each file can be easily recovered through the names of the system, and the unit, to which it pertains.

The definitions tables tree is as follows:

root level : general structure of TCS, contains one file with the definitions for all the systems in the TNG network; the file name is defined in the environment : systems.scf

1st level : units definitions, contain the definitions of the units (tasks or processes) of each system; file names are derived from system names found in the previous file : <system>.ucf

2nd level : contains the definitions of the parameters used by each unit, and of the microcommands accepted by them; file names are derived combining the name of each system with the name of its units found in ".ucf" files: <system>_<unit>.pcf for parameters, <system>_<unit>.mccf for microcommands

Both at the 1st and 2nd level the definition files for the graphic and interaction panels may be included; their names follow the same rules described previously:

<system>.pan or <system>_<unit>.pan for interactive panel

<system>.drw or <system>_<unit>.drw for graphic panels

2.2.1 - Off-line Files Maintenance Tools

There are two classes of off-line tools: table editors and table converters. Table editors allow the user to create and edit definition tables and interactive or graphic panels, while table converters allow the conversion between different formats of table sets.

The TNG tables take their internal structure (the current one is described in paragraph 3.3) from a definition file called "dbms.src", written in a metalanguage that makes use of a subset of C structures.

Here is and example taken from the current definition file for the root level SCF (System Characteristic File) structure:

```
/* -------------------------------------- */
/* TNG - dbms.src                         */
/* Source file for data records definition */
/* Creation: PM-OAT 180991                */
/* -------------------------------------- */
/* enter "/lib/cpp dbms.src dbms.cfg" to  */
/* build the definition file.             */
/* -------------------------------------- */
#define ACRONYMLEN 16
#define MAXCOEFF 5
#define MAXOPER 5
#define DESCRLEN 44
#define PATHLEN 44
/* ---- Systems definition file ---- */
scf: {
     nodenum:          int;0;System id;
     nodename:         char[DESCRLEN];NULL;System description;
     acronym:          char[ACRONYMLEN];NULL;System acronym;
     dbcode:           long;0;NULL;
     arpa_node:        char[16];NULL;Internet address;
     byte_sex:         boolean;0;Swap bytes;
     send_data:        boolean;0;Send TM+Data;
     tm_period:        int;1;Telemetry refresh rate;
     ncode:            int;0;Number of code files to send;
     type:             char[4];TCS;System type (TCS,ICS,VAR);
     protection:       int;0;Protection level;
     havepanel:        boolean;0;Interaction panel;
     havedrawing:      boolean;0;Graphic panel;
     firstqueue:       int;0;VME's first queue index;
     }
```

As we can see, there are a "define" section and a "typedef" section. The syntax of the first is just like C, while the second one deserves a little description.

On the first line we find the name of the structure we are defining, followed by a colon and an open bracket. The other lines describe every field needed by the structure, divided in four parts:

```
NAME
|              TYPE
|              |   DEFAULT
|              |   | DESCRIPTION
|              |   | |
nodenum:       int;0;System id;
```

where :

NAME is the field variable name;

TYPE is the field variable type, choosen among a set of 11 types:

INT	four byte integer;
LONG	four byte long integer;

FLOAT	four byte float;
BOOLEAN	True or False;
CHAR	array of char;
INTARR	array of INT;
LONGARR	array of LONG;
FLOATARR	array of FLOAT;
CHARARR	array of CHAR;
BOOLARR	array of BOOLEAN;
INTDARR	double array of INT;
LONGDARR	double array of LONG;
FLOATDARR	double array of FLOAT;

DEFAULT is the default value; NULL means no default;

DESCRIPTION is the description of the field as will be displayed in the table editor; NULL means the item is for internal use.

A closing bracket ends the definition of the structure.

Using a utility called "makeinc", dbms.src is converted in dbms.cfg. At the same time all the defines are resolved and a set of include files is created.

Here is an example of the dbms.cfg structure and of the correspondent include file:

```
dbms.cfg
--------
scf:
     {
     nodenum:          int;0;System id;
     nodename:         char[44];NULL;System description;
     acronym:          char[16];NULL;System acronym;
     dbcode:           long;0;NULL;
     arpa_node:        char[16];NULL;Internet address;
     byte_sex:         boolean;0;Swap bytes;
     send_data:        boolean;0;Send TM+Data;
     tm_period:        int;1;Telemetry refresh rate;
     ncode:            int;0;Number of code files to send;
     type:             char[4];TCS;System type (TCS,ICS,VAR);
     protection:       int;0;Protection level;
     havepanel:        boolean;0;Interaction panel;
     havedrawing:      boolean;0;Graphic panel;
     firstqueue:       int;0;VME's first queue index;
     }
```

```
scf.h include file
------------------
/*
** Include file for SCF class type
** Creation: 06.12.91. */
```

```
typedef struct
     {
     int nodenum;                    /* System id */
     char      nodename[44];         /* System description */
     char      acronym[16];          /* System acronym */
     long      dbcode;               /* *** INTERNAL USE ONLY *** */
     char      arpa_node[16];        /* Internet address */
     int byte_sex;                   /* Swap bytes */
     int send_data;                  /* Send TM+Data */
     int tm_period;                  /* Telemetry refresh rate */
     int ncode;                      /* Number of code files to send */
     char      type[4];              /* System type (TCS,ICS,VAR) */
     int protection;                 /* Protection level */
     int havepanel;                  /* Interaction panel */
     int havedrawing;                /* Graphic panel */
     int firstqueue;                 /* VME's first queue index */
     } SCFREC;
```

The file dbms.cfg is used by table editors and by the WSINIT process to store and retrieve data from the definition tables. A dedicated library (dblib) is provided to allow programs to access transparently these capabilities.

There are three off-line table editors. One of them is the "Table Editor", a context-sensitive editor, which, starting by a file name, can detect its type and open an editing panel suited to the file type.

The other two editors are devoted to the interactive and graphic panels. They are WYSIWYG programs that allow the programmer to rapidly build an interactive or graphic panel making use of buttons, menus and other widgets.

2.3 - Communications

The communications mechanisms are based on two UNIX tools: sysV messages and BSD sockets. The former are restricted to the use on a single computer, while the latter use protocols (e.g. tcp/ip) to communicate on computer networks.

SysV messages are used for communications between processes in the same node. Each unit at startup calls an "init communications" function, which in turn opens his own message queue using its process identifier (PID) as an access key, and writes the resulting queue identifier into the TDB. When the unit needs to send a message, the "send" function will get the target queue id from TDB and, at the same time, delivers the signal matching the priority of the message. The "read" function is a signal handler that reads the message from its queue and performs the action specified by the user.

Communications between processes on different nodes are realized using BSD sockets with Internet protocols. Sockets are either of SOCK_DGRAM (udp protocol) or SOCK_STREAM (tcp protocol) type. Sockets using Internet are able to manage communications also on wide area networks, and will allow both remote control and remote observing as a natural and straightforward extension.

2.3.1 - Workstations Communication System

The communication system is based on the concept that each unit can reach every other unit without bothering about its location, either local or remote, depending on the system configuration. The sender unit must fill a message header specifying its acronym, the target acronym, the type of the message, its priority and its size. The library function used to send the message analyses the system part of the target acronym, determining if it is local or remote: if the address is local, sysV messages are used to reach the specified unit; else, still using sysV calls, the message is forwarded to WSCOMM process.

The WSCOMM process takes care of all the TCP/IP connections with other systems or units on remote workstations. When a message with a remote target address is received, WSCOMM finds the Internet address of the required remote system and performs the send operation. On the remote system, the peer WSCOMM will get the message; if the calling unit is allowed to perform the requested operation, WSCOMM will provide to forward it to the specified unit.

The TCP/IP connection system is totally dynamic. Each time a workstation enters the network its WSCOMM process starts a connection procedure with the other workstations already present, using UDP. First a presence message is broadcasted over the network. Then, after a handshake phase about the respective roles, the workstation establishes its TCP connection mode, either as a server or a client. Such a system is absolutely open and allows for new connections even at run time.

The priority of the messages, either alarms or normal messages, is linked to a signal management procedure: every time a message is sent, a signal is also delivered to the receiver unit; the type of the signal (SIGUSR1 or SIGUSR2) depends on the priority of the message. Alarm priority messages will always be read before normal priority ones. The user is free to specify the action to be performed when a message is received; the interface with sysV signals and messages is provided by a dedicated library.

A simple scheme of fault tolerance is also obtained from a careful use of the communication system. Each workstation broadcasts periodically over the network a "heartbeat" signal, using the UDP protocol. These signals are monitored by all other workstations and used to maintain a knowledge of the network current configuration. Should a workstation fail, it stops sending the signal. The other systems recognize the new situation, and start a transaction phase to choose the new workstation elegible to substitute the one which has failed. After this phase the chosen workstation takes over the identity of the failed one and tries to connect with the concerned VMEs. Then, if the mechanism is successful, operations can resume their normal behaviour.

2.3.2 - VME Communication System

The scheme of the communications between VMEs and workstations is slightly different from the inter-workstation one. Considerations about efficiency and reliability led to a separate environment, even if based on the same concepts. The protocol used is TCP/IP, and the VMEs are always servers, while workstations are always clients. This was decided in order to allow maximum flexibility in the system configuration, i.e.

should a workstation fail, it is up to the workstation system to decide alternate connections to VMEs.

VME failure detection instead is exploited on a time-out base. Like in the workstation case, a heartbeat signal is sent to each attached VME by its controlling workstation. The protocol used is again TCP/IP, owing to the fact that UDP is not available under the operating system chosen for the VME systems. In case of a workstation failure, the attached VMEs detect the failure and enter a safe state, waiting for a new workstation to connect with them.

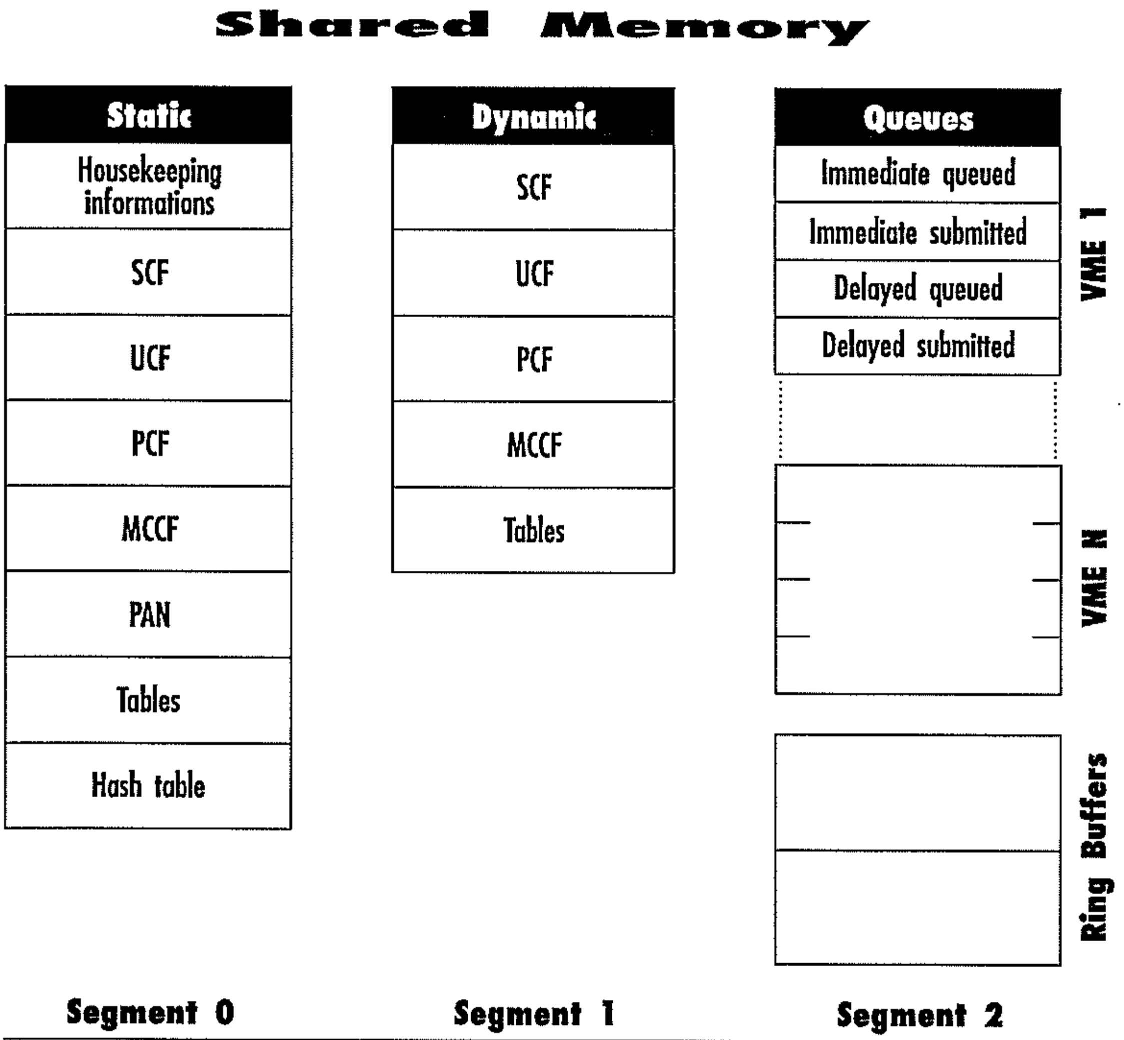


Figure 3: Telescope Data Base internal structure

2.4 - Data Base

The configuration tables of the whole TCS system, along with its dynamic status are contained in a dedicated memory area of WSS, typically a set of shared memory segments, accessible to all WSS processes. It is organized in a way suitable for both indexed access and fast direct access, as well as for fast memory/disk swapping, in order to increase efficiency of both on-line operations and off-line maintenance.

The internal structure of this area (the Telescope Data Base, TDB) is reported in fig. 3. It contains a root segment, where all the access control structures are located, followed by the static contents of the definition files read from the disk. A second segment contains the dynamic counterpart of the first segment, e.g. the dynamically changing status information related to the items defined in it. A third segment is allocated for some special internal data structures, such as the queues of the commands to be sent to the TSS, the time schedule table for the WSS watchdog process, and the telemetry log buffers.

The TDB is created by the initialization process during the TCS startup phase. It reads the definition files starting from the root level and following the tree-like structure already described. First of all an accurate computation of the size of all the three TDB segments is carried on. Then the segments are created and the access structures are prepared. These last contain some fixed information together with the address tables which allow WSS processes to access TDB items by their name. This capability is provided using a modified hashing algorithm, which has proved to be able to resolve the collisions among different names with very little overhead.

A library of dedicated routines is available to the WSS processes to make use of the addressing capabilities of TDB. Its data can be accessed by their name, as reported before or by their internal unique code. This solution allows each process to build internal lists of address codes, obtained through the item names, in order to make accesses as fast as possible, specially when the same items must be addressed repeatedly.

The detailed structure of the TDB is reported in appendix A as a set of C data structures.

3 - WSS STRUCTURE AND ORGANIZATION

3.1 - General Structure

Seven tasks are foreseen to be entrusted to the WSS. A single process is devoted to each of the main tasks. A special class of processes (ancillary processes, AP) which have no direct interaction with the external environment is also provided, and is used to support the main processes when table definitions are no more sufficient for their proper operations. This structure is a tradeoff between the characteristics of the UNIX operating system, and the needs of the WSS. The layout of this structure and the mutual interactions among processes are reported in fig. 4, together with the data and/or command paths to/from the TDB, the TSS and the internal message exchange facility.

3.2 - WSS Processes

The seven processes cover the following tasks : WSS initialization, user interface including TSS commands sending and alarms management, telemetry data verification

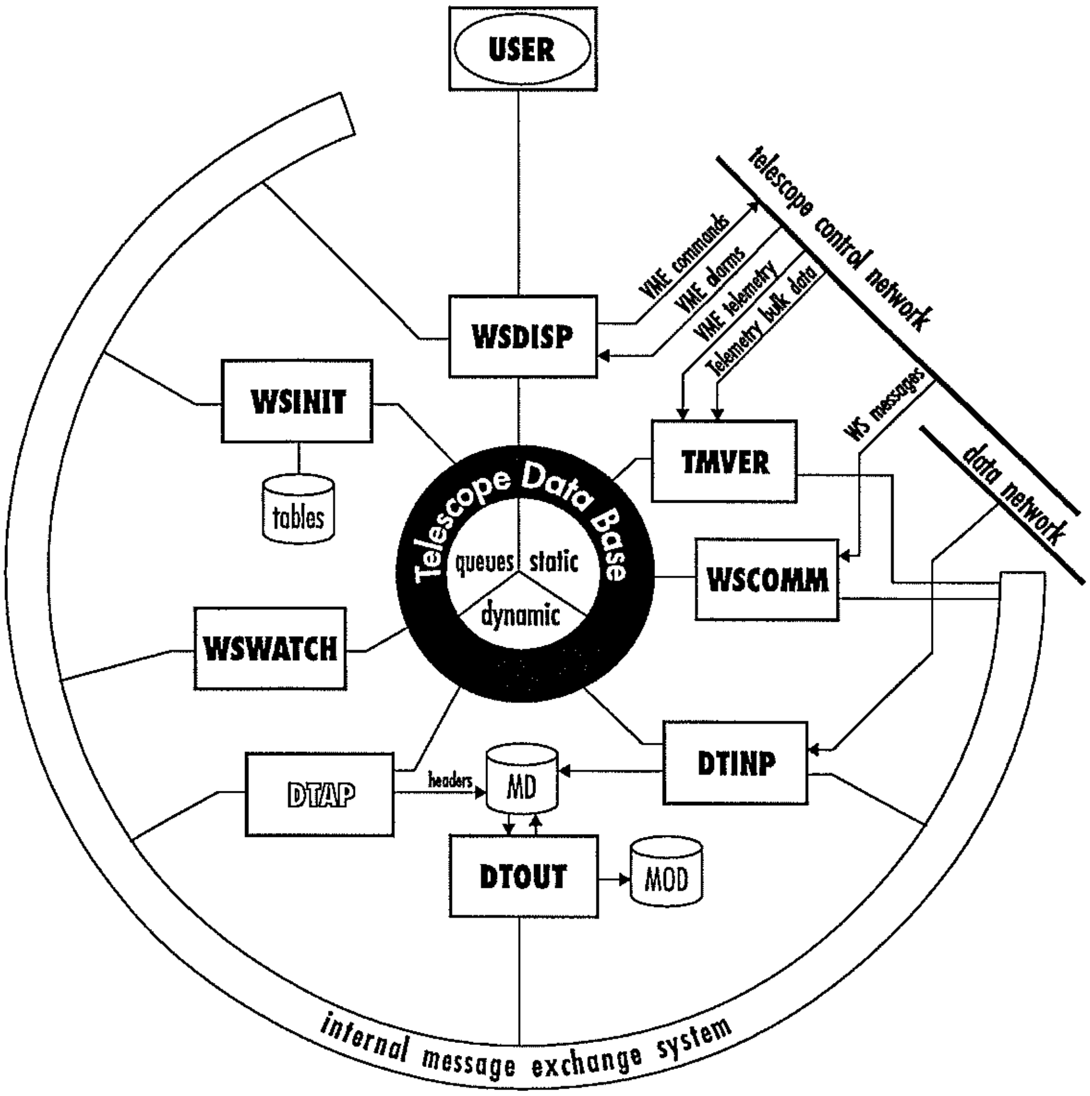


Figure 4: Workstation Software System layout

and TDB update, command execution verification, science data reception, science data archiving, communications among workstations.

3.2.1 - Initialization (WSINIT)

The WSINIT process is responsible for the initialization of the whole WSS system. As first step, WSINIT attaches the TDB if it already resides in the shared memory, left there by a previous run of the WSS, and if it is consistent with the current definition files. If this is not the case, WSINIT creates the TDB static section and loads it from the on-disk definition files; creates the dynamic TDB section and initializes it; creates the queues TDB section and initializes it; finally it starts all the local WSS processes as children processes.

The mechanism used to build the WSS processes is the standard UNIX "fork-exec" method: fork duplicates the calling (parent) process environment, then exec runs a new image (child) in it. Due to the characteristics of these UNIX system routines, parent and children are able to communicate with each other using any of the standard methods provided by UNIX.

After these operations, WSINIT puts itself in a dormant state, waiting for the death of any of the children processes, for a message from any of the children, or for a signal from the WSDISP process requesting an operator initiated shutdown.

The recovery of a fault is based on this capability : an error status being detected by WSINIT can suspend the WSS operations in a predictable way. This allows the system to restart with minimal operator intervention, or even with no intervention at all, provided that the other workstations and the TSS systems are also able to suspend their own operations, and to wait for a restart signal.

In case of a complete WSS shutdown, WSINIT stops all the children processes; then a flush on disk-resident files of the dynamic TDB section and the queues TDB section is attempted. At next WSINIT startup the contents of these files will be loaded in the correspondent TDB sections.

System shutdown can be considered a special case of an error, which leads the WSS to a complete stop.

3.2.2 - User Interface (WSDISP)

The task of supervising all the interactions between the user and the TCS is taken on by the WSDISP process. It is actually divided into four sections, each carrying on a well defined and independent activity:

i) monitoring user inputs

ii) parsing, validating and sending command to TSS components

iii) displaying default or user requested sets of telemetry data, either in numeric or graphic form

iv) managing alarms arriving from the TCS.

These tasks constitutes the User Interface (UIF), and will be described later in section 4 in greater detail.

3.2.3 - Telemetry Verification (TMVER)

The reception and verification, as long as the storage in the TDB of the telemetry data arriving from the TSS, is carried on by a dedicated task, TMVER, which connects to the TSS components of concern at initialization time. After TMVER has been started, all the connections, both for telemetry bulk data and for normal telemetry, with VME systems related to the current workstation system (or systems) are done. Telemetry data are received and written into TDB. If necessary, byte swap and data conversion are performed before writing. Commands submission and execution messages are also received on the telemetry channel. When a submission message is received, the command of concern is seeked on the "issued commands" queue and is moved to the "submitted commands" queue. An execution message causes the command to be removed from the "submitted commands" queue. The bulk data channel is used to receive telemetry data ring buffers, which are used to contain telemetry data sampled by VME systems at a rate higher than the normal, for special diagnostic purposes.

3.2.4 - Commands and Timeouts Verification (WSWATCH)

This process performs controls on submission and execution of commands sent to VME systems . It checks periodically the microcommands queues, looking for commands not yet submitted or executed in the due time. If such a condition is found, a warning or alarm message is sent to the user through the interface of the system from which the command has been sent.

3.2.5 - Communications among Workstations (WSCOMM)

All connections among workstations are managed by this process. The main tasks performed are:

i) initialization and monitoring of network connections

ii) recovery of network failures

iii) sending and receiving messages to and from remote workstations

iv) database writing operations on remote request

The communication mechanism has already been described in greater detail in section 2.3.1.

3.2.6 - Science Data Handling (DTINP, DTOUT)

Handling of the data from acquisition to storage on a temporary magnetic disk area is performed on the instrument workstation. Magnetic disk units of appropriate capacity (in the order of 2 Gbytes) must be attached to this machine, and it is recommended to provide mirroring (shadowing) of the magnetic disk areas, in order to allow fault-tolerant operations.

The production of the observer's tapes and archiving should be performed on a centralized unique data server, in order not to interfere with instrument operations. The data are moved from the instrument workstation to the data server over the workstations network; all data storage units needed to temporarily and permanently store the observations, and to produce the observers' tapes are to be connected to the data server. Such devices include magneto-optical disk (MOD) units for temporary storage, digital audio tape (DAT) units for data distribution to observers, and high density optical disk (OD) for the storage of archive data. Furthermore, appropriate magnetic disk space needs to be available for the storage and handling of the catalogue of observations.

3.2.6.1 - Conceptual Scheme

When an acquisition is completed, data coming from the instrument are read by the DTINP process and stored on magnetic disk in standard format (big endian integers).

The file containing the data is called xxxx_zz_tt.BLK, where xxxx_zz_tt is a name tied to the observing time, defined later in this section. At the same time the ancillary process for the specific instrument (DTAP) prepares a FITS header, reading from the TDB the parameters related to the specific observation. The FITS header, called xxxx_zz_tt.HDR, is also written on disk. When DTAP has completed this operation, a message is sent to the DTOUT process to initiate operations.

DTOUT reads xxxx_zz_tt.BLK and xxxx_zz_tt.HDR, creates a single FITS file, writes it to a magneto-optical disk standing on the MOD unit, and updates the xxxx_zz_ii.FIL table, containing all files created during an observing session. The xxxx_zz_ii part of the file name is tied to the session itself, and will be defined later in this section.

The observer is entitled to receive all files generated during his/her observing session. However, using the quick-look facility through the WSDISP process, he/she may reject a specific file, if deemed not to be of interest. In this case, the entry related to the file is flagged as "rejected" in the xxxx_zz_ii.FIL file.

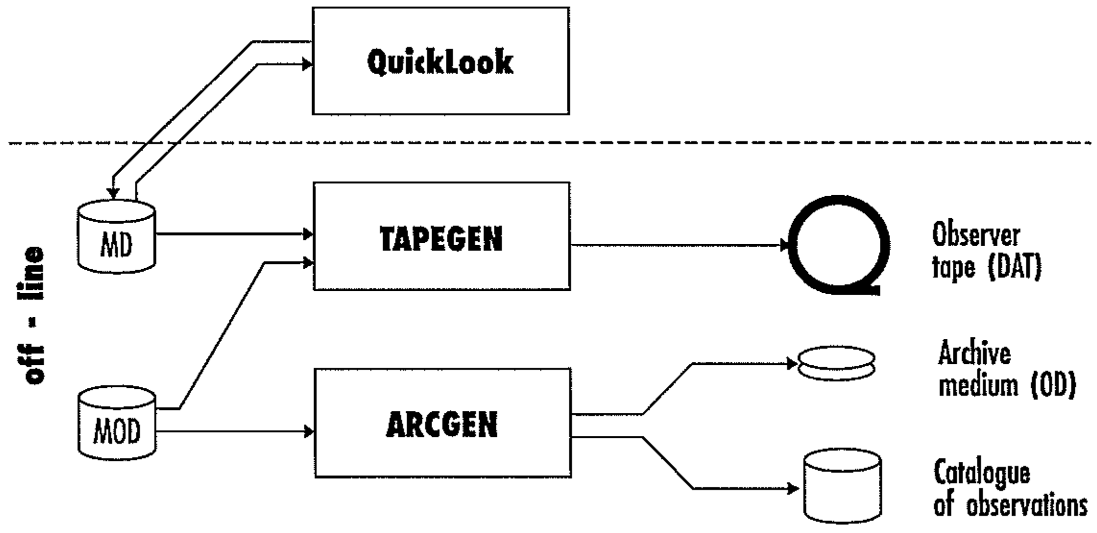


Figure 5: Off-line science data archiviation

At the end of the observing session, two off-line processes are initialized (see fig. 5). ARCGEN reads the xxxx_zz_ii.FIL file, and dumps all created files, at the moment stored on magneto-optical disks in the MOD unit, onto the archive media (optical disks?);

at the same time, it reads the headers of the FITS files and updates the catalogue of observations.

TAPEGEN prepares the output media for the observer (DATs): it reads the xxxx_zz_ii.FIL file, and dumps from the magneto-optical disks in the MOD unit to the digital audio tapes mounted on the DAT unit all files which have not been flagged by the observer as "rejected". The file containing all files created during an observing session, xxxx_zz_ii.FIL, is also dumped on the output medium in the form of a FITS table.

3.2.6.2 - Files Naming Conventions

Each file in the data handling chain is uniquely identified by a file name, which is built automatically from time, the system (instrument) name, and the session identifier.

The table containing all files created during an observing session is called xxxx_zz_ii.FIL, where xxxx is the system (instrument) name, zz is the current date (expressed as yymmdd), and ii a sequential number identifying the session itself. As an example, the table containing all files created on March 10th 1995 during the third session on instrument CAM will be called CAM_950310_03.FIL. When stored on the observer's tape, such a table will be in FITS format, and its name will be CAM_950310_03.FTAB.

The final result of an observation is stored on a FITS file called xxxx_zz_tt.FITS, and its temporary sections xxxx_zz_tt.BLK and xxxx_zz_tt.HDR. In this case, xxxx is again the system (instrument) name, zz is the current date (expressed as yymmdd), and tt is the time the observation was started, in the format hhmmss (GMT). Observation start time is truncated to the nearest second. As an example, a file created on March 10th 1995 by an observation starting at 2:36:21.57 on instrument CAM will be called CAM_950310_023622.FITS.

3.3 - TCS Definition Tables and Files

Inserting systems into WSS is a quite simple task, provided that systems characteristics and behaviour, as well as WSS interactions with them, have been carefully planned and defined. There are four steps which must be followed to make a system known to WSS.

1) Define the system characteristics, along with the units contained in it, and the parameters and commands which will be used by them. If applicable, define a graphic layout for the system and the units, and define the panels needed to interact with them.

2) Using the context sensitive table editor, create and fill in the tables containing the characteristics of each unit of the system.

3) Using the same editor create and fill in the files (<system>_<unit>.pcf and <system>_<unit>.mccf) containing the characteristics of the parameters and of the commands pertaining to each unit. If applicable create and fill in the graphic panels (<system>.drw or <system>_<unit>.drw) and the interactive

panels (<system>.pan or <system>_<unit>.pan) for the system and/or the units, using the graphic editor and the panel editor respectively.

4) Using the table editor add to the root file, system.scf, the definition and the characteristics of the system.

After this last step, the first time WSS will be rebooted the new system will be automatically included in the internal tables, and made available for subsequent TCS operations.

A detailed layout of definition tables is reported in Appendix B.

4 - USER INTERFACE

The User Interface (UIF) represents the only way by which the user can interact with the computer system. Being so, it is important that this doorway is built using software technologies that take in account both the efficiency of the system and the human ease of interaction.

The TNG Workstation Software System is based over a Unix software and a graphical workstation hardware platform. The use of a graphic device allows displaying lots of informations organized in a clear manner, by the use of windows, icons and text.

4.1 - Window System

The window system is a peculiar piece of software that runs in background controlling and managing all the interactions between the user and a series of virtual terminals called "windows". There are several window systems on the market but only one is supported by all of the major vendors: Xwindows release 11.x.

The choice of this window system if therefore mandatory, in order to provide a high grade of portability to the applications among all hardware systems that can be used in a long term project.

The window system provides a limited set of functions: graphic primitives, event handling and windowing. In order to give the user a higher degree of control over these functions we need a more sophisticated software called the "window manager". This software uses the functions of the window system to build a coherent and robust interface to the user. The market battle for window managers is more hot than that for the windows systems. There are two major contenders: the OFS/Motif 1.x from Open Software Foundation and Open Look from Sun Microsystems. The market numbers are similar due to the great penetration of Sun in the USA universities, but in fact Open Look is supported only by Sun and some clone-makers, while OSF/Motif is supported by all other vendors (a third party support for Sun is also available).

Moreover, OSF/Motif follows the guidelines given by IBM/CUA standards, giving to the user a consistent approach to the interface very close to that used in personal computers with Windows 3.0 or OS/2 2.0.

So, our choice for the Galileo project User Interface standards is:

window system - Xwindows release 11.4 window manager - OSF/Motif 1.1

4.2 - Display organization

The TNG Control System User Interface provides all the components needed to use interaction and graphic panels resembling the visual appearance and behavior of real-world control instruments. This means that instrument manufacturers can build interaction panels, using off-line WYSIWYG dedicated editors, completely by software, giving an added flexibility and ease of upgrading.

The drawback of this situation is that the screen becomes rapidily cluttered by panels, icons and graphical representations. But there are several solutions to this problem:

a) enlarge the physical screen;

b) squeeze panels;

c) add screens.

The better solution is to mix all the three above, using small, but meaningful, panels over a set of two or more large screens, either on the same workstation, or on different workstations or both.

In this way, a screen can be used to control the telescope system, while the others can be dedicated to instruments; mixing different instrument control panels, if they are simple, or giving full control over a screen to more sophisticated instruments.

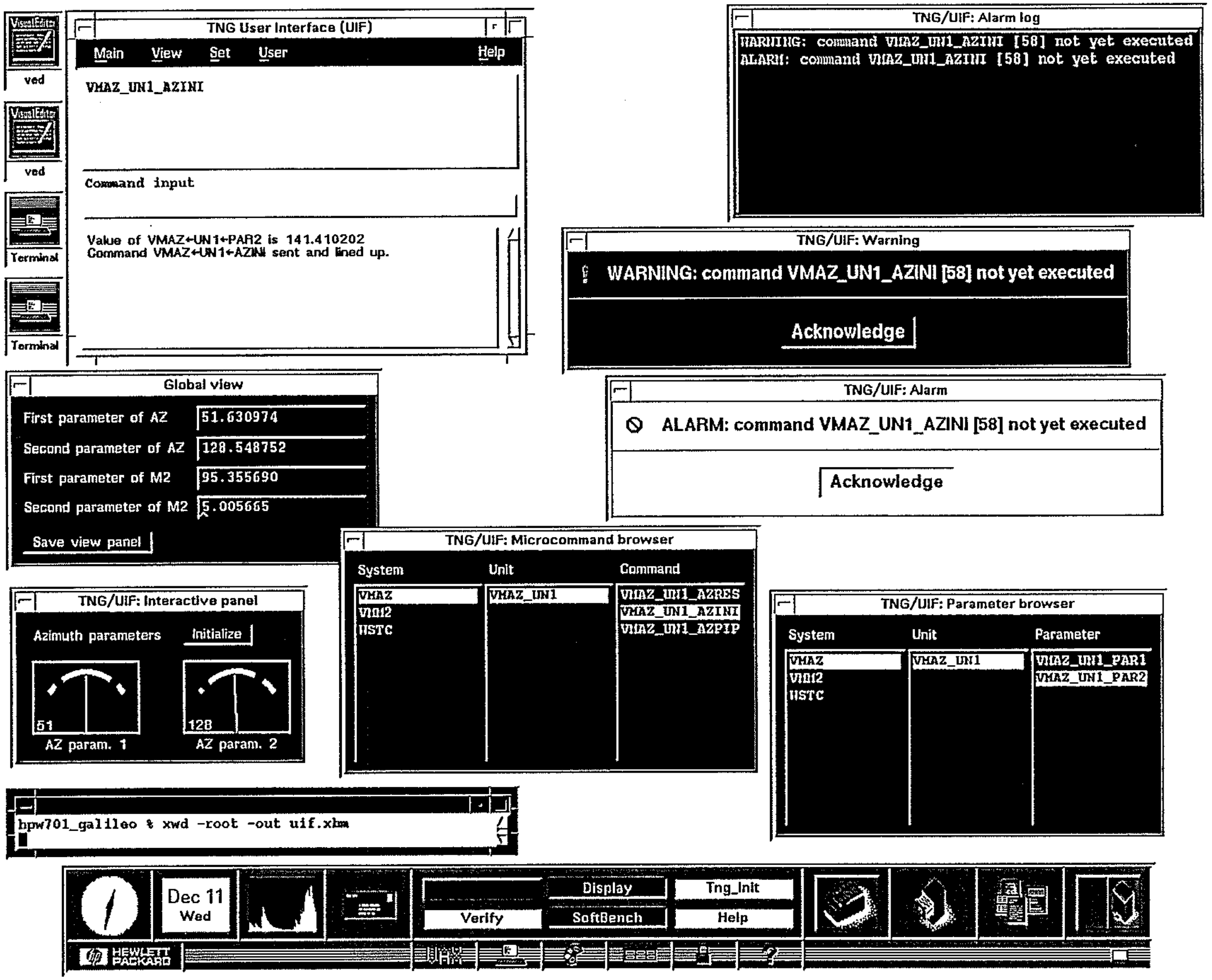
The main window of the UIF is displayed always on the screen devoted to the telescope control (see fig. 6), and is composed by three main parts:

a) the menu bar;

b) the command input line;

c) the system messages area.

Using the menu bar, the user can select system commands such as "Load an interactive panel" or "Set the default unit". Using the command line, the user can enter commands in a terminal-like environment with line editing facilities. A scrolling area, where all the entered commands are logged, allows stored lines to be retrieved and edited. The system message area displays the control system outputs and messages like "Command sent" or the like.



```
VMAZ_UN1_AZINI

Command input

Value of VMAZ+UN1+PAR2 is 141.410202
Command VMAZ+UN1+AZINI sent and lined up.
```

```
WARNING: command VMAZ_UN1_AZINI [58] not yet executed
ALARM: command VMAZ_UN1_AZINI [58] not yet executed
```

WARNING: command VMAZ_UN1_AZINI [58] not yet executed

ALARM: command VMAZ_UN1_AZINI [58] not yet executed

First parameter of AZ	51.630974
Second parameter of AZ	128.548752
First parameter of M2	95.355690
Second parameter of M2	5.005665

System	Unit	Command
VMAZ	VMAZ_UN1	VMAZ_UN1_AZRES
VMM2		VMAZ_UN1_AZINI
MSTC		VMAZ_UN1_AZPIP

System	Unit	Parameter
VMAZ	VMAZ_UN1	VMAZ_UN1_PAR1
VMM2		VMAZ_UN1_PAR2
MSTC		

```
hpw701_galileo % xwd -root -out uif.xbm
```

4.3 - User Interaction

The user can interact with the system in two ways: by using the command line provided in the main window or by using the interaction panels. These methods can be freely mixed to ensure maximum easiness of use.

For example, commands can be entered either by typing their acronym and operands on the command line, or by selecting the Command Browser dialog and navigating the system/units tree until the desired command is found. Then a click on it will perform the command.

Another way to interact with the system is by using graphic panels. They contain a graphical representation of the components of the telescope system, including instruments. If the user wants to select an instrument all he/she has to do is to click over the instrument icon: the interaction and graphic panels of the instrument will be displayed on the screen, ready for use.

Graphic panels also use animation to show changes in the system parameters. Temperature indicators change to red in an over-heating situation, or shutters icons move to stop the light path. This all happens on the screen with no user intervention, directly driven by changes in the internal database.

A description of the definition tables to be edited to build both the interaction panels and the graphic panels is given in Appendix C. It must be noted that, as reported in paragraph 2.2.1, these tables are generated interactively through the use of suitable interactive editors dedicated to this task.

The help system is invoked by pressing the Help item in the menu bar of the main window. It is an integrated system that makes use of hypertext-like links between informations and a scrolling browser that the user can access to display help screens in a non-linear way.

4.5 - Internal Structure

The UIF is an event driven process, this means that the main control over the operations flow is given to user actions instead of program statements. An event driven process must provide a set of functions that are asinchronously activated in response to user's actions like key or mouse buttons pressing. These functions are called 'event handlers' and are called ONLY if the user perform an action that requires them; the internal mechanism to poll the event ports and to dispatch events to handlers is provided by Xwindows.

UIF starts with an initialization part that builds the main window (displayed as described before) and the dialog boxes, and activates all the connections with the VMEs and the internal communication channel. Then the system will be in an idle status until a command is entered via the logical input channels, at this point the correct event handler will be called and the output of the command will be displayed via the logical output channels (see fig. 7).

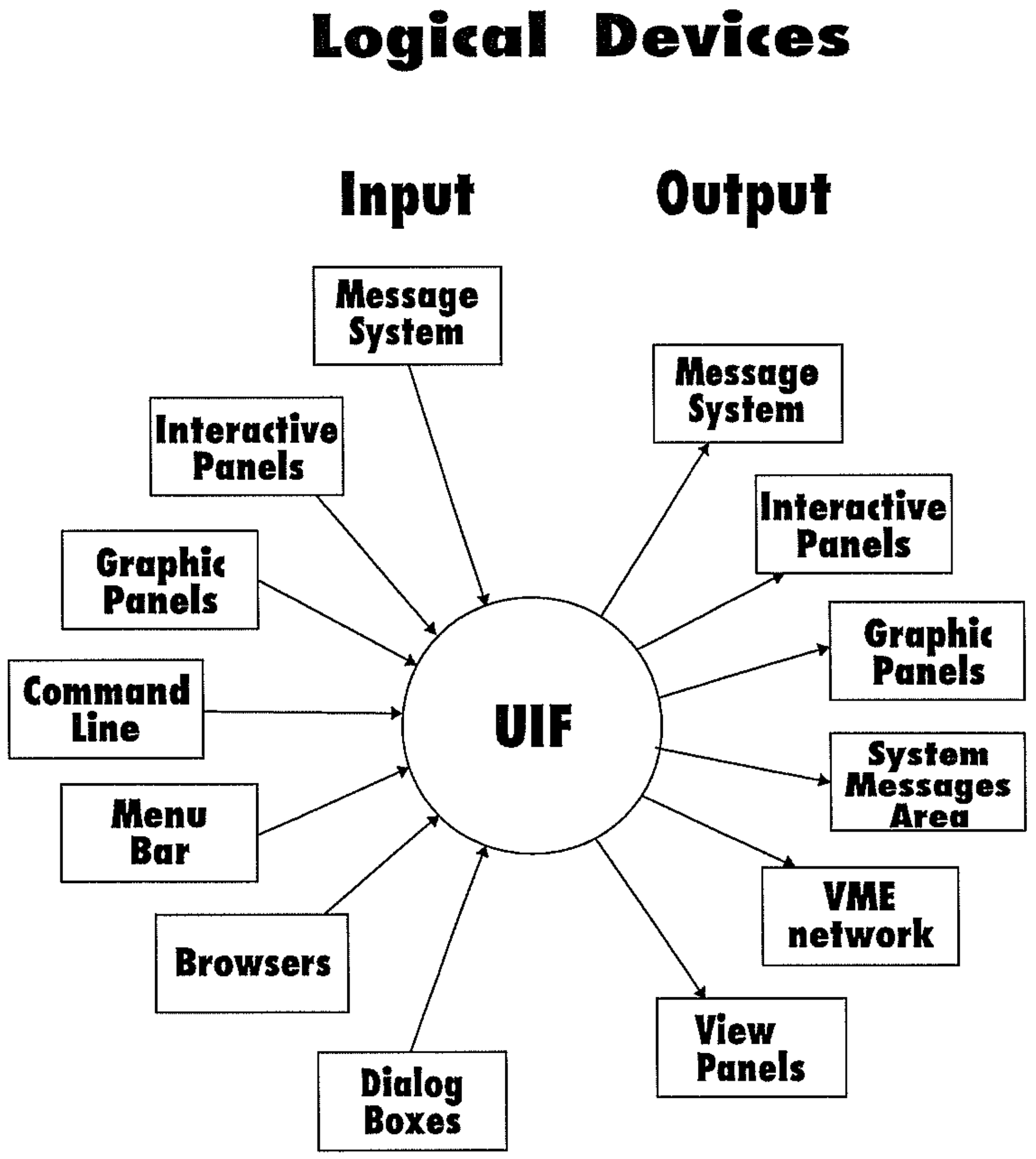


Figure 7: UIF logical input and output channels

User commands can be subdivided in two main sections: internal (or system) commands and external commands. The flow chart of commands input and parsing is reported in fig. 8.

4.5.1 - Internal (system) Commands

Internal commands are those that modify the UIF behavior. For example, a typical internal command is "Load a view panel", it is not a command that will be forwarded to the VMEs but act only at UIF level. A list of the internal commands is given below:

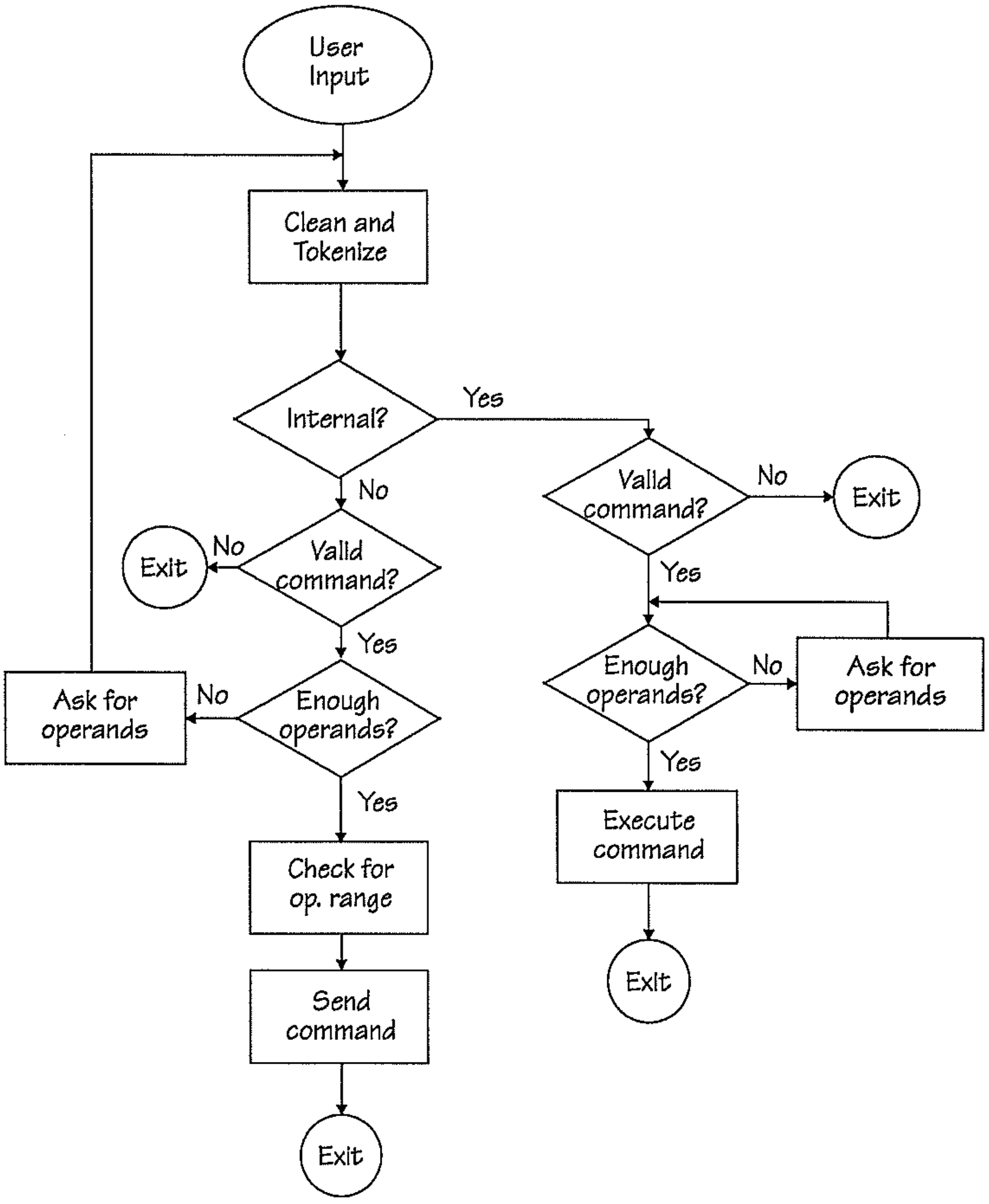


Figure 8: Command parsing flow-chart

SET

USER	Set a new user name.
SYSTEM	Set a default system.
UNIT	Set a default unit.

LOAD

INTERACTIVE	Load an interactive panel.
GRAPHIC	Load a graphic panel.
COMMANDS	Load the command browser.
PARAMETERS	Load the parameter browser.
VIEW	Load a view panel.
ALIAS	Load an alias file.
DESKTOP	Load a desktop configuration.

SAVE

VIEW	Save a view panel.
ALIAS	Save an alias file.
DESKTOP	Save a desktop configuration.

EXIT Exit the system.

4.5.2 - External commands

External commands are dispatched by UIF to the correct destination VME via the socket and network connections. An external command is issued by using any of the available logical input channels such as the command line or an interactive panel.

A peculiar logical input channel is the command browser, that displays the complete list of connected system. By clicking over one of them, the list of all the system's units are displayed, and via a subsequent click over a unit, a list of all the microcommands related to the unit. At this time, clicking over a microcommand acronym will execute the command, asking for operands, if needed.

4.5.3 - Telemetry Parameters Output

TM parameters (PCFs) are received in a continuous mode by the TMVER process that stores them in the TDB. As stated before, the TBD is an instant snapshot of the complete Telescope Control System (time resolution is of one second), and a process that access the TDB for a PCF, has immediately (in one second steps) the value of the requested PCF.

If the user wants to display the actual value of a PCF, all he has to do is to request the Parameters Browser and, in a manner similar to the Command Browser, select the PCF to view. Its value will appear in the system messages area.

To obtain a continuous view of a set of PCFs, the user can build a View Panel using an internal UIF interactive editor, or load a predefined one with the "Load View Panel" internal command.

PCF values output is also performed by interactive and graphic panels. The graphic panels can also display a PCF value using simple animation techniques.

5 - ACKNOWLEDGMENTS

Thanks are due to the colleagues of the TNG Project for many useful suggestion and discussions, and for their help during the development of the software project. A special acknowledgment goes to the colleagues of the ESO/TDE division for many useful discussions about the general concepts of a telescope control system, and for many fundamental suggestions about their practical implementation.

6 - REFERENCES

[1] GALILEO TELESCOPE - Phase A Report
M. Zambon ed., 1989
Astronomical Observatory of Padova

[2] A. Baruffolo, C. Bonoli, A. Ciani
"The TNG - command architecture"
TNG Technical Report n. 6, June 1991
Astronomical Observatory of Padova

Appendix A - Telescope Data Base Structure

TDB Structures - Static section

```
/* ==== TABLES HEADER ==== */

struct tabhead
     {
     int type;                     /* type of table */
     int n_rec;                    /* number of records */
     char acronym[NAMELEN];        /* acronym of table */
     char filename[81];            /* name of file */
     };

typedef struct tabhead TABHEAD;

/* ==== TABLES ==== */

struct db_tab
     {
     TABHEAD   header;      /* header */
     int       tabnum;      /* number of table */
     long      dbcode;      /* database code */
     int       first;       /* position of the first record */
     int       last;        /* position of the last record */
     };

typedef struct db_tab TABREC;

/* ==== STATIC TNG DATA BASE ==== */

struct db_static
     {
     int nrecscf;
     int nrecucf;
     int nrecpcf;
     int nrecmccf;
     int nrecpan;
     int ntables;
     char ws_sys[MAXSCF][TNG_LENSYS];
     SCFREC scf[MAXSCF];
     UCFREC ucf[MAXUCF];
     PCFREC pcf[MAXPCF];
     MCCFREC mccf[MAXMCCF];
     PANREC pan[MAXPAN];
     TABREC tables[MAXTAB];
     HASHREC hash_tab[HASHDIM];
     };

typedef struct db_static DBSTAT;
```

TDB Structures - Dynamic section

```
/* ==== SCF RECORDS ==== */

struct scfdyn
     {
     int ws_id;
     };

typedef struct scfdyn SCFDYN;

/* ==== UCF RECORDS ==== */

struct ucfdyn
     {
     int ws_id;
     int pid;
     int qid;
     };

typedef struct ucfdyn UCFDYN;


/* ==== PCF RECORDS ==== */

struct pcfdyn
     {
     float eng_value;
     float phis_value;
     int time;
     float set_value;
     int set_time;
     };

typedef struct pcfdyn PCFDYN;

/* ==== MCCF RECORDS ==== */

struct mccfdyn
     {
     int   nopers;
     float opers[MAXOPER];
     int   code;
     short vme_spec;
     short vme_err;
     int   time;
     char  counter;
     };

typedef struct mccfdyn MCCFDYN;

/* ==== TAB RECORDS ==== */

struct tabdyn
     {
```

```
     int active;
     };

typedef struct tabdyn TABDYN;

/* ==== DYNAMIC TNG DATA BASE ==== */

struct db_dyn
     {
     SCFDYN    scf[MAXSCF];
     UCFDYN    ucf[MAXUCF];
     PCFDYN    pcf[MAXPCF];
     MCCFDYN   ccf[MAXMCCF];
     TABDYN   tables[MAXTAB];
     };

typedef struct db_dyn DBDYN;
```

TDB Structures - Microcommands queues

```
/* ==== QUEUES ITEM ==== */

struct qitem
     {
     MCCFCOMM comm;         /* VME microcommand block */
     short cmd[2];          /* Command return value and status */
     long time;             /* Sampling time */
     int wf;                /* Warning flag (1 if warn. message sent) */
     int ef;                /* Error flag (1 if err. message sent) */
     int prev;              /* Previous queued microcommand */
     int next;              /* Next queued microcommand */
     char sys[TNG_LENSYS]; /* AB: sender system */
     };

typedef struct qitem QITEM;

/* ==== MICROCOMMANDS QUEUE ==== */

struct mc_queue
     {
     int free;              /* Beginning of free list */
     int first;             /* Beginning of queued microcommands list */
     int last;              /* End of queued microcommands list */
     QITEM queue[MAXMCQD]; /* Microcommands queue */
     };

typedef struct mc_queue MC_QUEUE;

/* ==== MICROCOMMAND QUEUE ==== */

struct db_queues
     {
     MC_QUEUE mcq[MAXVME*NQVME+1]; /* Segment of microcommands queues
*/
     };

typedef struct db_queues DBQUEUES;
```

Appendix B - Characteristics Files Layout

System Characteristics Table (system.scf)

int id;	System identifier : progressive record number, is the main key to retrieve records in the file
char nodename[44];	System description : system verbose description, for display purposes only
char acronym[16];	System acronym : four characters acronym, MUST begin with WS for workstations, VM for VME systems, followed by two characters which specify the system job (e.g. VMAZ = azimut control VME, WSTC = telescope control ws)
long dbcode;	Data Base code : filled in at run time by WSINIT process during initialization. Used by WSS processes to access data in TDB.
char arpa_node[16];	Internet address in dotted form
int byte_sex;	Swap bytes flag : true if data from system are in little endian form
int send_data;	Send TM+Data flag : true if system sends science data too
int tm_period;	Telemetry refresh rate : rate (in seconds) at which system must send changed telemetry data
int ncode;	Number of code files to send : number of system code file to download at initialization
char type[4];	System type : three characters code to define system grouping. TCS = telescope control system, Ixy = Instrument xy control system.
int protection;	Protection level : defines the access rights to the system internal structure
int havepanel;	Interaction panel flag : true if an interaction panel exists for the system
int havedrawing;	Graphic panel flag : true if a graphic panel exists for the system
int firstqueue;	VME's first queue index : filled in at run time by WSINIT process during initialization

Unit Characteristics Table

int id;	Unit identifier : progressive record number, is the main key to retrieve records in the file
char unitname[44];	Unit description : unit verbose description, for display purposes only
char acronym[16];	Unit acronym : three characters acronym to identify the unit (e.g. MPA in VMAZ identifies the Motor Power Amplifiers Unit in the Azimut control VME system)
char path[44];	Pathname of the program : used by WSINIT to fork the WSS processes. If empty no fork will be made
long dbcode;	Data Base code : filled in at run time by WSINIT process during initialization. Used by WSS processes to access data in TDB
int protection;	Protection level : defines the access rights to the unit internal structure
int havepanel;	Interaction panel flag : true if an interaction panel exists for the unit
int havedrawing;	Graphic panel flag : true if a graphic panel exists for the unit

Parameters Characteristic Table

int id;	Parameter identifier : progressive record number, is the main key to retrieve records in the file.
int pcode;	Parameter code, used as parameter identifier by VME software
char name[16];	Parameter name : parameter full name
char descr[44];	Parameter description : parameter verbose description, for display purposes only
char acronym[16];	Parameter acronym : six characters acronym, self descriptive (e.g. MT1CUR = motor 1 current in the Unit MPA in the Azimut control system)
long dbcode;	Data Base code : filled in at run time by WSINIT process during initialization. Used by WSS processes to access data in TDB.
int type;	Parameter type : code to define parameter type (e.g. 1 = Temperature, ...) T.B.D.
char format[5];	Format of data : four character data format tfxx, where t = type of item (D = digital value, A = analog value), f = format of item (I = integer, F = float, B = bit field, S = string, C = command) xx = number of bytes for I,F,S,C, of bits for B
char access[3];	Access mode : two characters access (e.g. RD = Read Only, WR = Write Only, RW = Read Write)
int vme_only;	VME only flag : true if the parameter is relevant to the VME system only
int tm_flag;	Telemetry flag : true if the parameter must be sent to the WSS via telemetry
int convert;	Convert to physical units : true if datum must be converted from engineering units to physical ones
long coeff[5];	Polynomial coefficients : array of coefficients for the transformation
int check_limits;	Check input limits : true if TMVER must check the parameter against the following limits
long intr_low_limit;	Lower input limit (eng.units)
long intr_high_limit;	Higher input limit (eng.units)
float low_alarm_thr;	Low threshold for ALARM

```
float    high_alarm_thr;   High threshold for ALARM

float    low_attn_thr;     Low threshold for ATTENTION

float    high_attn_thr;    High Threshold for ATTENTION

char     phy_unit[11];     Physical units
```

Microcommands Characteristics Table

int id;	Microcommand identifier : progressive record number, is the main key to retrieve records in the file.
int mcode;	Microcommand code, used as microcommand identifier by VME software
char name[16];	Microcommand name : microcommand full name
char vme[16];	Destination VME : acronym of the destination VME
char descr[44];	Microcommand description : microcommand verbose description, for display purposes only
char acronym[16];	Microcommand acronym : six characters acronym for the command
long dbcode;	Data Base code : filled in at run time by WSINIT process during initialization. Used by WSS processes to access data in TDB.
int queue;	Delayed queue : true if to be sent to delayed queue on VME
int waitflag;	Wait for execution : true if successive commands must wait for its completion
int task;	Destination VME's task : name of the VME task (unit) the command is sent to
int exec_verify;	Verify execution : true if TMVER must verify command execution (submission)
int compl_verify;	Verify completion : true if TMVER must verify command completion
long min_exec_time;	Min. time extimated for execution
long max_exec_time;	Max. time extimated for execution
int counter;	Number of command operands
int convert[5];	Convert to eng. units : true if operand[i] to be converted to engineering units
long coeff[5][5];	Interpolation matrix : conversion matrix for operand[i]
char opdescr[5][44];	Operand description : operand full description, display purpose only
long min_value[5];	Minimum value allowed for operand[i]

long max_value[5];	Maximum value allowed for operand[i]
long def_value[5];	Default value for operand[i]
int verify_flag;	Verify TM parameter : true if a parameter exists which must be checked to verify command completion
char tm[16];	TM parameter to verify : full acronym of the parameter to be checked (<system>_<unit>_<command>)
long tolerance;	Fractional tolerance in thousandths for the parameter to be checked

Appendix C - Panels Characteristic File Layout

Interactive Panels Characteristics Table

`int id;`	Element identifier : progressive record number, is the main key to retrieve records in the file.
`int type;`	Element type : type of widget to be used; 1 = label, 2 = text output, 3 = slider, 4 = led bar, 5 = push button, 6 = status (see mode), 7 = analog display, 8 = text input (operand for a command)
`int x;`	Start x position : upper left x position of widget
`int y;`	Start y position : upper left y position of widget
`int mode;`	Status mode : if type = 6, 0 = color changing led, 1 = changing text
`char acronym[16];`	Panel acronym : widget acronym (6 characters)
`long dbcode;`	Data Base code : filled in at run time by WSINIT process during initialization. Used by WSS processes to access data in TDB.
`float threshold;`	Color or text change threshold : refers to parameter value; status widget changes if this threshold is trespassed
`char text[44];`	Text to display : refers to labels
`char pcf[16];`	PCF to visualize : full acronym of parameter to be used with the widget (<system>_<unit>_<par>)
`char mccf[16];`	MCCF to activate : command to be activated if type is 3, 5 or 8
`char oper[20];`	Fixed operands : to be used with mccf (if applicable)
`char stat[2][44];`	Color or text to display : contains the two colors or the two strings to be used with a widget of type 6; if pcf value threshold the first element is used, otherwise the second one