THE GALILEO ANCILLARY PROCESS PROGRAMMER'S GUIDE

Version 2.0

Andrea Balestra, Paolo Marcucci, Mauro Pucillo, Claudio Vuerli
Astronomical Observatory of Trieste

October 1997

Astronomical Observatory of Trieste - Publication n. 1622

Abstract

In this document the Programmer's Guide to build a TNG Ancillary Process (AP) is described. APs are the sole way to implement complex control procedures, which cannot be realized using the definition tables provided within the WSS software. An AP acts as an intelligent interface between the UIF in the control workstations, and the low level software layer which operates directly on the TLP systems.

A brief description of the characteristics of an AP is given, followed by a complete description of the tools available to the TNG programmer to accomplish the task of integrating a process in the WSS environment. A template of a skeletal AP is also given, containing all the pieces of code needed to have it working correctly in the WSS.

Table of contents

Introduction

The baseline followed during the design and implementation phases of the TNG Software System, has been to develop a system where all components were completely integrated with each other, having a standard kernel providing all basic services (Telescope Data Base, User Interface, Communications, etc.), complemented by a set of processes developed by the builders of the instruments.

The kernel, actually the Workstation Software System (WSS), has been described in [1], while its components have been described in [2, 3, 4, 5, 6, 7]. Here we want to point out how to design an AP, and its relations and interconnections with the WSS environment and with the final user, either the system maintainer or the observing astronomer.

Controlling the TNG from an ancillary process

The TNG WSS (Workstation Software System) can be fully controlled by the user via the standard user interface (TUI) using the keyboard and the mouse installed at the astronomer's workstation. Commands can be sent to the local system or to other workstations and Telescope Local Processors (TLPs) in a transparent fashion, freeing the user from the need to remember where a command is to be executed and which are its operands.

The WSS can also be controlled in a programmatic way, using the so called Ancillary Processes (AP), which are launched from the **init** process and immediately go to sleep mode waiting for commands, this way they do not add too much load to the system where they run. Actually they sleep in the background and wake up only when their command queue is written in by some requestor, either the user - via the User Interface (UIF) - or another WSS process - via the internal message exchange system.

Designing an AP

An AP works as a layer between the UIF (or any other component of WSS) and the lower levels of interaction between workstations and TLPs. It is possible to control all the functionalities of a TLP sending microcommands and visualizing telemetry parameters, but it is much more feasible to use an intermediate layer that takes care of complex interactions and command parsing.

As an example, if we want to select an observing mode of an instrument, this must imply that operations not compatible with the selected mode have to be disabled. This behavior cannot be obtained simply by definition tables, and has to be implemented via a dedicated software. The AP is essentially a program built following some given guidelines and using a library of functions that provides all the possible method of interaction between itself and the WSS environment.

The program has to contain at least the following three parts:

Startup definitions

```
#define MAIN            /* this is an ancillary program, not a library */
#include <tng.h>        /* include all the TNG WSS standard include files */
```

Event handlers

Command handler

```
int cmd(from,acronym,txt,flags) /* command handler function */
 char from[],acronym[],txt[];
 int flags;
  {
  }
```

The command handler function will be called every time a command is received by the AP. To ensure this, the command originator (either the UIF or another AP) has to specify the complete command acronym built using the system name (eg. **`WSTC`** for Telescope Control Workstation), the unit name (in this case, the acronym of the AP, eg. **`CCD`** for a CCD controller) and the command acronym (it can be, for example, **`EXPOSE`** to start an exposition). The command built in this way (**`WSTC_CCD_EXPOSE`**), wherever generated, will be directed to this function, where a series of **strcmp** statements will be used to perform adequate actions. If the WSS system where the AP will run is not known at design time (for example, a system wide AP) the system name can be fetched with the internal function **`get_locsys()`** that returns the system name in a **`char*`** variable.

Example:

```
int cmd(from,acronym,txt,flags)  /* command handler function */
 char from[],acronym[],txt[];
 int flags;
  {
  if (!strcmp(tngAPGetCommand(acronym),"EXPOSE")) /* start exposition */
    {
    tngAPReadParameter ("wstc_ccd_extime/s",op);
    sprintf (s,"vmaz_ccd_expos %s",op);
    tngAPSendCommand (s,NULL,NULL);
    return (0);
    }
  }
```

The **tngAPGetCommand** function is an utility that strips down the command acronym to the last part, the command acronym itself, without system and unit information.

Example:

```
tngAPGetCommand("WSTC_CCD_EXPOSE") returns "EXPOSE"
```

Alarm handler

```
int alm(from,acronym,txt,flags) /* alarm handler function */
 char from[],acronym[],txt[];
 int flags;
  {
  }
```

Alarms are managed in the very same way as commands. They are sent by the UIF or other APs as microcommands. The main difference is that alarms are not buffered but are executed as soon as they arrive. This can be useful for **`ABORT`** operations or other non-timed commands.

To send an alarm, the originator calls the **tngAPSendCommand** function as for a normal command. The choice between a simple command or an alarm is made by the UIF's command parser, that looks for the Normal queue/execution field of the **`mccf`** file: if it is set, the "command" will be directed to the Command Handler, if it is not set, the command will be directed to the Alarm Handler.

Message handler

```
int msg(code)                   /* message handler function */
 int code;
  {
  }
```

Messages are managed in a way similar to callbacks. When a command is executed, it calls back this function with an execution code. See the **tngAPSendCommand** function for a detailed discussion about callbacks.

Timeout handler

```
int tout()                      /* timeout handler function */
  {
  }
```

Timeout events can be scheduled by setting the **`tmout`** operand in the **tngAPInit** call to a value different than NULL. The **`tmout`** operand is a `timeval struct` and has to be filled by the ancillary process' writer.

Descriptors event handler

```
int descrev(fds)                /* descriptor event handler function */
 fd_set *fds;
  {
  }
```

The descriptor event handler is invoked every time an event is generated on a custom socket port. Custom sockets are used by APs only in special cases; usually the socket-type communication with other processes and workstations on the telescope network are directly handled by the WSS, and only in particular cases, like managing external dedicated clients, these kind of sockets are to be used.

Startup function calls

```
main()
 {
/* assign the AP and UIF names, don't use custom sockets or timeouts */
  tngAPInit("WSTC_CCD","WSTC_UIF",NULL,NULL);
/* register event handlers */
  tngAPRegisterHandlers (cmd,alm,msg,tout,descrev);
/* enter the main loop */
  tngAPMainLoop();
 }
```

The first two event handlers have a fixed syntax: they are **int** functions that accept three strings (or **char***) and an **int** describing, respectively, the unit the event is coming from, the acronym of the command, the operands and the return code for the command itself. The **msg** handler has only one parameter (an **int**) that is the return code for the callback function handler (i.e. the **msg** function). See the **tngAPSendCommand** function for a discussion about callbacks. The **tout** handler is an **int** function without parameters. The **descrev** handler takes, as parameter, an array of custom socket descriptors.

Create the Unit

The design of an AP follows the logical definition of a unit as described in [1]. A unit is defined in the TNG WSS as a component of a system, being it a workstation or a TLP. A unit can be a process in a workstation or a set of related tasks implemented in a TLP. An AP can be used only in the first case (a WS process), to implement functionality that cannot be defined by tables.

To add a unit to a system, the user should add a new record to the system's **.ucf** file. i.e: for a system called **`WSSC`**, he should add a record to the **`wssc.ucf`** file. Moreover, he has to define the parameters, commands and interactive panels which will allow the WSS to interact with the AP.

These additions are made using the Table Editor.

Add the unit record

Open the .ucf file of the system (if the system is **`WSSC`**, then the ucf file will be called **`wssc.ucf`**)

Fill the unit record

Unit id	This field is automatically managed by the Table Editor program
Unit description	A brief description of the ancillary process
Unit acronym	The acronym of the ancillary process (max. 3 chars! Remember that this file is at the unit level)
Process name and options	The name of the AP, with no path information but with process switches as desired
Protection level	The protection level of the AP
Interaction panel id	If this field is checked, an interactive panel with the unit's name (ex. **`wssc_ccd.pan`**) is opened
Ancillary process	This field MUST be checked

Wait for signal	This field should normally be checked

Click on the **Add** menu item

Close the .ucf file

Add parameters

Create a new pcf file. To accomplish this task, select File|New|Parameters from the Table Editor menu. Then enter the pcf file name (if the system is **WSSC** and we are adding the **CCD** unit, the pcf file will be called **wssc_ccd.pcf**)

Open this file using the File|Open|Parameters menu entry on the Table Editor.

Fill the parameters record

TLP master descriptor	This field is automatically managed by the Table Editor program
Parameter name	The name of the parameter. This field has a length of 40 chars, but it is preferable to limit it to 20 chars. This in order to display it correctly on interactive panels.
Parameter acronym	The acronym of the parameter (max 6 chars!)
Parameter description	The description of the parameter
TLP parameter code	TLP internal code. This item is mandatory if the parameter refers to a TLP telemetry parameter. If the parameter refers to an ancillary process this item is not used.
Parameter type	Not currently implemented
Format of data	The internal format of data. It can have the following values: f - floating point, d - integer, s - character. In addition to that, a number can be added to indicate the number of elements in an array (for example *f* will mean a single floating point value, *f4* will mean an array of four [0..3] floating point values)
Access mode (RD,WR,RW)	Access mode. Can assume the following values: RO - Read Only, RW - Read Write, WR - Write Only
Decimal points	Number of decimal places to be used in display.
TLP only	If checked, the parameter is considered internal to the TLP and will not be involved in telemetry operations.
Telemetry flag	If checked, the parameter is added to the permanent telemetry list of the destination TLP. Other parameters can always be added to and removed from this list at runtime.
Convert to physical units	If checked, the parameter value (coming from TLPs in engineering units) is automatically converted to its physical value by using a polynomial transformation with the following coefficents.
Polynomial coefficients	The coefficents for the optional polynomial transformation.
Default value (for internal PCFs)	This item applies only to internal parameters. i.e. parameters that don't belong to TLPs. It is the default value as displayed on interactive panels.
Check input limits	If checked the parameter value is checked against the alarm and attention thresholds.
Lower input limit (eng.units)	Lower limit in engineering values.
Higher input limit (eng.units)	Higher limit in engineering values.

Low threshold for ALARM	In physical units.
High threshold for ALARM	In physical units.
Low threshold for ATTENTION	In physical units.
High threshold for ATTENTION	In physical units.
Physical units	A short string for unit description.
Telemetry rate	The transmission rate of the parameter coming from the TLP. This is a multiplier of the system's transmission rate. For example, if the system has a rate of two seconds and this item is set to three seconds, telemetry will be sent from the TLP every six seconds.

Click on the **Add** menu item

Repeat steps 3-4 until all parameters are entered

Close the .pcf file

Add commands

Create a new mccf file. To accomplish this task, select File|New|Microcommands from the Table Editor menu. Then enter the mccf file name (if the system is **WSSC** and we are adding the **CCD** unit, the mccf file will be called **wssc_ccd.mccf**)

Open this file using the File|Open|Microcommands menu entry on the Table Editor.

Fill the microcommands record

Microcommand code	This field is automatically managed by the Table Editor program
Microcommand name	The name of the microcommand. This field has a length of 40 chars.
Destination TLP/System	The destination system's acronym.
Microcommand description	The description of the microcommand.
Microcommand acronym	The acronym of the microcommand (max 6 chars)
TLP command code	TLP internal code. This item is mandatory if the command refers to a TLP command. If the command refers to an ancillary process this item is not used.
Immediate queue	If checked, the command will be sent to the immediate queue on the destination TLP. This item is not used in ancillary process' commands.
Wait for execution	If checked, the sending process will wait for the execution message from the destination TLP before to proceed with further processing.
Destination task/unit	The task of the destination TLP or the unit (process) of the destination workstation.
Verify execution	If checked, the sending process will wait for a VERIFIED message from the destination TLP/unit.
Verify completion	If checked, the sending process will wait for a COMPLETED message from the destination TLP/unit.
Min. time estimated for execution	The time (in telemetry periods) after which, is a command is still not executed, a EXWARN internal message is sent to the display process.
Max. time estimated for execution	The time (in telemetry periods) after which, is a command is still not executed, a EXALRM internal message is sent to the display process.

Number of operands	The number of operand that the command will accept.
Verify TM parameter	If checked, a verification is made on the given parameter.
TM parameter to verify	The acronym of the telemetry parameter to verify.
Error/1000 allowed in TM parameter	The allowed error margin. If the difference between the requested value (usually set in the first operand) and the telemetry parameter to verify is greater than the allowed error, an error message is generated.
Note: the following fields refer to the operand array	
Convert to eng. units	If checked, the operand, entered in physical units, will be converted to engineering units before being sent to the destination TLP/unit.
Interpolation matrix	An array of polynomial coefficients to convert operand from physical to engineering units.
Operand description	A description of the operand.
Operand type	The internal format of the operand. It can have the following values: f - floating point, d - integer, s - character. In addition to that, a number can be added to indicate the number of elements in an array (for example *f* will mean a single floating point value, *f4* will mean an array of four [0..3] floating point values)
Minimum value allowed	
Maximum value allowed	
Default value	The default value of the operand.

Click on the **`Add`** menu item

Repeat steps 3-4 until all microcommands are entered

Close the .mccf file

Add panels

Design a panel using the Graphic editor and save it with a unique filename.

Add this panel to the Panel Description File (pdf) of the just created unit.

Open the requested **`.pdf`** file using the File|Open|Panel definition menu entry of the Table Editor.

Fill the panel definition record

Panel identifier	This field is automatically managed by the Table Editor program
Panel acronym	The acronym of the panel (max 6 chars)
Default output screen	The preferred screen for output. This item is used only in multi-screen stations, otherwise, it defaults to screen :0.
Panel description	A brief description of the panel.
Panel file	The actual file name where the panel is stored (with no path information or the suffix, e.g. **`telcont`**)
Protection level	If the current user has a protection code higher than this, he cannot open this panel. This works to protect sensitive panels from accidental misusings.

Click on the **`Add`** menu item

Repeat steps 4-5 until all panel definitions are entered

Close the .pdf file

Function list

tngAPInit

This function executes the following steps:

1. Attaches the TDB
2. Initializes the signal system
3. Opens the message queue

It is MANDATORY to use this function as the first significant call in the AP.

Syntax:

```
int tngAPInit(acronym,uif,tmout,descarray)
 char acronym[],uif[];
 strcut timeval *tmout;
 fd_set *descarray;
```

Parameters:

acronym	The acronym of the AP (e.g. **WSTC_CCD**)
Uif	The acronym of the user interface where the output is redirected.
Descarray	The array of custom socket descriptors that will be used by the AP. If NULL, there are no custom sockets used by this process (this is the normal case)
Tmout	The value (defined by a timeval struct) in microseconds upon which the AP executes timed operations. If NULL, no timed operations are performed (this is the normal case).

Result:

If the **gdb_attach** function cannot attach to the TDB, the function exits the process.

Example:

```
.
.
.
struct timeval timeout;
fd_set descarray;

/* set timeout for select call to 10 usec */
timeout.tv_usec = 10;
timeout.tv_sec = 0;

/* set array of descriptors for 'select' call */
FD_ZERO (&descarray);   /* clear set */
```

```
FD_SET (first_socket_descriptor, &descarray);
FD_SET (second_socket_descriptor, &descarray);

tngAPInit (“WSTC_CCD”, “WSTC_UIF”, &timeout, &descarray);
.
.
.
```

tngAPExit

This function exits the AP. It calls **gdb_detach** for a clean exit. The function should be called during abnormal exits, because during normal exits the **init** process takes care of removing all the garbage left by APs on message queues and internal links.

Syntax:

```
int tngAPExit (code, where)
 int code;
 char *where;
```

Parameters:

code	the error code (or 0 for a normal exit)
where	a string describing the possible location of the error.

tngAPSendCommand

This function sends a command to another unit in the TNG environment. The command to be sent is included in the **cmd** string, and can be any of the commands defined in the configuration tables for the given unit. The AP can send:

internal commands	system-wide commands that control the functionality of the user interface or system processes, e.g **loadpanel, help, exit**
TLP commands	also known as microcommands (**V..._..._.....**)
process commands	commands directed to other APs (**W..._..._.....**)

Syntax:

```
int tngAPSendCommand (cmd,retmsg,flag)
 char cmd[],retmsg[];
 int flag;
```

Parameters:

cmd	the command to be sent, together with all the required operands.
retmsg	the result of the operation, if required.
flag	the type of command to be sent.

Notes:

flag can assume the following values:

AP_SIGWAIT (-1)	the function sends the command and waits for its completion; the **retmsg** parameter contains the return value for the operation

	requested. The content of **retmsg** is not predefined, and is up to the command executor to fill this variable in a consistent manner. As an example the **loadpanel** function returns the number of the opened panel.
AP_SIGNORM (0)	the command is simply sent, no checking about its execution is done.

any other positive value:	putting any other positive value in this field will activate a quasi-callback event to be fired at the completion of the command. The value (**int**) is passed as the first argument to the **msg** handler (see **tngAPRegisterHandlers**), where the ancillary process can use it to perform appropriate functions.
	Example:
	tngAPSendCommand ("WSIC_CCE_OSHUT",NULL,1234);
	will send the command to the correct destination task and will go on with the processing of following statements. When the command will be executed by the destination task, a message is sent back to the AP, with the 1234 code in the first (**int**) field.
	The msg handler function will look like this:
	`int msg(code)`
	`  int code;`
	`  {`
	`  if (code == 1234)`
	`   {`
	`    /* the command WSIC_CCE_OSHUT is completed.`
	`       Perform the`
	`       appropriate actions. */`
	`   }`
	`  }`
	It is up to the AP programmer to ensure a direct relation between commands and return flags, i.e. to identify a specific command with a specific code. The **msg** function must be registered with the **tngAPRegisterHandlers** call.

tngAPReadParameter

This function reads the TDB and returns to the AP the value of an internal parameter.

Syntax:

```
int tngAPReadParameter (acronym,value)
 char acronym[];
 char value[80];
```

Parameters:

acronym	the acronym of the parameter to be read. It can have the **/Snn, /Cnn** suffixes to specify set or current values and the index inside an array. Examples: **WSTC_CCD_EXTIME** will return the current exposure time **WSTC_CCD_EXTIME/S** will return the set exposure time **VMAZ_CCD_TEMP/C02** will return the second current value in an array of temperatures
value	the read value converted to a string. It is up to the programmer to

	convert the result to a suitable type.

tngAPSetDescriptors

This function sets the custom socket descriptors handler array.

Syntax:

```
int tngAPSetParameter (descarray)
 fd_set *descarray;
```

Parameters:

descarray	The array of descriptors to set

tngAPSetParameter

This function sets a value in the TDB.

Syntax:

```
int tngAPSetParameter (acronym,value)
 char acronym[];
 char value[80];
```

Parameters:

acronym	the acronym of the parameter to be set. No suffixes are required, the function will use **/S** by default to put the value in the "set" section.
Value	the value to set. There are no checks on the validity of this value, so the programmer should be careful and perform suitable integrity tests before calling this function.

tngAPShowInfo

This function displays a informational message box.

Syntax:

```
int tngAPShowInfo (txt)
 char txt[];
```

Parameters:

txt	the text to be displayed.

tngAPShowWarn

This function displays a warning message box.

Syntax:

```
int tngAPShowWarn (txt)
 char txt[];
```

Parameters:

txt	the text to be displayed.

tngAPShowAlarm

This function displays an alarm message box.

Syntax:

```
int tngAPShowAlarm (txt)
 char txt[];
```

Parameters:

txt	the text to be displayed.

tngAPRegisterHandlers

This function MUST be called immediately after the **`tngAPInit`** function to set up the pointers to the command and alarm handler functions. These functions MUST be provided by the programmer to handle the interpretation of commands, alarms and messages coming through the internal message system.

Syntax:

```
void tngAPRegisterHandlers (command,alarm,message,timeout,descrev)
 int (*command)();
 int (*alarm)();
 int (*message)();
 int (*timeout)();
 int (*descrev)();
```

Parameters:

Command	the pointer to the command handler function
Alarm	the pointer to the alarm handler function
Message	the pointer to the message handler function
Timeout	the pointer to the timeout handler function
Descrev	the pointer to the descriptor events handler function

Notes:

The handler functions should have the following layouts (the names can be different, just be sure to register them correctly to **`tngAPRegisterHandlers`**):

Command handler

```
int cmd(from,acronym,txt,flags)
 char from[],acronym[],txt[];
 long flags;

{
}
```

Alarm handler

```
int alm(from,acronym,txt,flags)
 char from[],acronym[],txt[];
 long flags;

{
}
```

Message handler

```
int msg(code)
 int code;

{
}
```

Timeout handler

```
int tout()

{
}
```

Descriptor events handler

```
int descrev(fds)
 fd_set *fds;

{
}
```

tngAPMainLoop

This function performs all the tasks needed to dispatch incoming commands and alarms. It also sends to the **`init`** process the signal that all initializations have been made

Syntax:

```
void tngAPMainLoop()
```

Note:

This HAS to be the last statement in the main section of the AP. When the AP enters this function, it will never exit. Other statements located after this statement will be ignored.

tngAPGetCommand

This is an utility function that extracts the last part (item) of a command acronym (e.g. **`tngAPGetCommand("WSTC_CCD_EXPOSE") -> "EXPOSE"`**)

Syntax:

```
char *tngAPGetCommand (acronym)
 char acronym[];
```

Parameters:

acronym	the acronym to be analyzed

Examples

This is the listing of a typical AP:

```
#define MAIN
#include <tng.h>

int cmd(from,acronym,txt)
 char from[],acronym[],txt[];

 {
 char s[80], op[80];
 char retmsg[80];
 int i;

/* ------------------------------------------------------------ */
/* use the tngAPGetCommand to find out if the EXPOSE command   */
/* was sent to the AP                                          */
/* ------------------------------------------------------------ */
 if (!strcmp(tngAPGetCommand(acronym),"EXPOSE"))

  {
/* ------------------------------------------------------------ */
/* read the set value of the wstc_ccd_extime parameter and     */
/* save it into the op string                                  */
/* ------------------------------------------------------------ */
  tngAPReadParameter ("wstc_ccd_extime/s",op);

/* ------------------------------------------------------------ */
/* prepare a string containing the vmaz_ccd_expos microcommand */
/* and the op value (e.g. if op = 20, then the resulting string*/
/* s would be vmaz_ccd_expos 20)                                */
/* ------------------------------------------------------------ */
  sprintf (s,"vmaz_ccd_expos %s",op);

/* ------------------------------------------------------------ */
/* send the string s, containing the microcommand, to the main */
/* command parser, NULL means that we aren't expecting any     */
/* return value and AP_SIGNORM means that the command will be  */
/* processed asynchronously without callbacks.                 */
/* ------------------------------------------------------------ */
  tngAPSendCommand (s,NULL,AP_SIGNORM);
  return (0);
  }

/* ------------------------------------------------------------ */
/* use the tngAPGetCommand to find out if the LOADP command    */
/* was sent to the AP                                          */
/* ------------------------------------------------------------ */
 if (!strcmp(tngAPGetCommand(acronym),"LOADP"))
```

```
  {

/* ------------------------------------------------------------ */
/* send a command to the user interface telling it to load and */
/* display the wstc_uif_vmaz interactive panel. Retmsg is the  */
/* index of the panel and AP_SIGWAIT tells the AP to wait      */
/* until the panel is fully loaded.                            */
/* ------------------------------------------------------------ */
  i = tngAPSendCommand ("loadpanel wstc_uif_vmaz",retmsg,AP_SIGWAIT);

/* ------------------------------------------------------------ */
/* create a string s containing a command (for the UIF) that   */
/* will disable the B widget group on the just opened panel.   */
/* Note that we use the retmsg value to specify the panel.     */
/* ------------------------------------------------------------ */
  sprintf (s,"disablegroup %s B",retmsg);

/* ------------------------------------------------------------ */
/* send the string s, containing the microcommand, to the main */
/* command parser, NULL means that we aren't expecting any     */
/* return value and AP_SIGNORM means that the command will be  */
/* processed asynchronously without callbacks.                 */
/* ------------------------------------------------------------ */
  tngAPSendCommand (s,NULL,AP_SIGNORM);
  }
 }

main()
 {
/* ------------------------------------------------------------ */
/* initialize the AP WSTC_CCD and set its default UIF on WSTC. */
/* No custom socket descriptors and timeouts are required for  */
/* this ancillary process.                                     */
/* ------------------------------------------------------------ */
 tngAPInit("WSTC_CCD","WSTC_UIF",NULL,NULL);

/* ------------------------------------------------------------ */
/* register only the command handler to the cmd function. All  */
/* other event handlers (alarm, message, socket descriptors    */
/* and timeout) are ignored.                                   */
/* ------------------------------------------------------------ */
 tngAPRegisterHandlers (cmd,NULL,NULL,NULL,NULL);

/* ------------------------------------------------------------ */
/* enter the ancillary process main loop.                      */
/* ------------------------------------------------------------ */
 tngAPMainLoop();
 }
```

Appendix A: Definition tables

Definition table for TNG systems (.scf)

Nodenum	long	Main code assigned by Table Editor
Nodename	char[40]	Full name
Acronym	char[24]	Acronym; used by the system to compose the name of an item in the Tdb – MANDATORY
Dbcode	long	Tdb internal code, assigned by software
Arpa_node	char[16]	Full Internet address – MANDATORY
Byte_sex	long	TRUE if the system supports the little endian byte ordering
Send_data	long	TRUE if the system sends scientific data besides telemetry (e.g. a TLP connected to an instrument)
Tm_period	long	Base period for the telemetry; must be defined following the kind of data sent, must be a multiple of one second (TLP only)
Ncode	long	Number of code files to be sent to the system at boot-strap (TLP only)
Type	char[8]	Acronym of the WS to which the TLP is connected; for the WS it is their own acronym
Protection	long	Protection level, reserved for future expansion
Havepanel	long	TRUE if the system has a dedicated control panel
Firstqueue	long	Pointer to the command queue in the Tdb, assigned by the software
Nscreens	long	Number of monitor screens connected to the system
Screendescr	char[4,8]	Screens descriptions

Definition table for TNG units (.ucf)

Unitnum	long	Main code, assigned by Table Editor
Unitname	char[40]	Description
Acronym	char[24]	Acronym; used by system to compose the name of Tdb items; must be used together to the name of the system to which the unit belongs to access to it - MANDATORY
Name	char[40]	Full pathname and possible options for the process associated to the unit in the workstation; for workstations only, must be empty for TLP
Dbcode	long	Tdb internal code, assigned by software
Protection	long	Protection level; reserved for future expansion
Havepanel	long	TRUE if the unit has a dedicated interactive panel
Ancillary	long	TRUE if the process defined in name is an ancillary process
Waitsignal	long	TRUE if the process defined in name must wait for a start signal from INIT

Definition table for TNG parameters (.pcf)

Tag	long	Main code, assigned by Table Editor
Name	char[24]	Full name
Acronym	char[6]	Acronym; used by system to compose the complete acronym

		of an element; is used together with the names of the system and the unit to which the parameter belongs to access the parameter itself in the Tdb - MANDATORY
Descr	char[40]	Description
Vmecode	long	TLP internal code – MANDATORY if the parameter is referred to a TLP
Dbcode	long	Tdb internal code, assigned by the software
Type	long	Type; reserved for future expansion
Format	char[4]	Format, can assume the following values [f|d|s[nn]] - f floating point, d integer, s char - nn gives the number of elements if the parameter is an array
Access	char[4]	Access mode; can assume the following values: RO=read-only, WR=write-only, RW=read-write
Decpoints	long	Number of decimal places; used for the display
Class	long	Class, assigned by the software; reserved for internal use
Vme_only	long	TRUE if the parameter is internal to the TLP; parameters with this flag set to TRUE are reserved to the internal functioning of the TLP and are not involved in the telemetry operations
Tm_flag	long	TRUE if the parameter must be sent with the telemetry; parameters with this flag set to TRUE at boot-strap time are considered stable components of telemetry operations and cannot be removed from the telemetry; it is anyway possible to add and remove other parameters to/from the telemetry list at run-time
Convert	long	TRUE if the value of the parameter must be converted to physical units
Coeff	double[5]	Polynomial coefficients for the conversion from engineering units to physical units; The polynome used is of the type: $ax^4 + bx^3 + cx^2 + dx + e$
Def_value	double	Default value; used by WSS to assign a default value to parameters of variable type; used by WS only
Check_limits	long	TRUE if the value of the parameter must be verified against the limits of attention and alarm
Intr_low_limit	long	Lower limit in engineering units
Intr_high_limit	long	Upper limit in engineering units
Low_alarm_thr	double	Lower ALARM limit, in physical units
High_alarm_thr	double	Upper ALARM limit, in physical units
Low_attn_thr	double	Lower ATTENTION limit, in physical units
High_attn_thr	double	Upper ATTENTION limit, in physical units
Phy_unit	char[12]	Physical units
Tm_rate	long	Period for telemetry, computed in units of tm_period as defined for the system to which the parameter belongs;

Definition table for TNG microcommands (.mccf)

Opcode	long	Main code, assigned by Table Editor
Name	char[24]	Full name
Vme	char[24]	Acronym of the destination system (TLP or WS) - MANDATORY
Descr	char[40]	Description

Acronym	char[24]	Acronym - MANDATORY
Vmecode	long	Private TLP code, as defined in the destination system (MANDATORY for TPLs)
Dbcode	long	Tdb internal code, assigned by software
Queue	long	TRUE if the destination queue is the immediate one
Waitflag	long	TRUE if the microcommand must be completed before successive microcommands are executed
Task	char[24]	Acronym of the destination unit
Exec_verify	Long	TRUE if the microcommand submission for execution must be reported through telemetry
Compl_verify	Long	TRUE if the microcommand execution must be reported through telemetry
min_exec_time	Long	Minimum execution time
max_exec_time	Long	Maximum execution time (must be referred to the value of tm_period)
Counter	Long	Number of operands (max. 10)
Convert	Long	TRUE if operands must be converted to engineering units
Coeff	double[10,5]	Table containing the polynomial conversion coefficients
Opdescr	char[10,40]	Operands description
Optype	char[10,4]	Operands type: f=float, d=long, sn=char[n]
Min_value	double[10]	Minimum value accepted for operands
Max_value	double[10]	Maximum value accepted for operands
Def_value	double[10]	Default value for operands
Verify_flag	long	TRUE if the microcommand execution must be verified through a telemetry parameter
Tm	char[24]	Acronym of the parameter to be used for the microcommand verification (current value versus preset value)
Tolerance	long	Tolerance in thousandths allowed in the verification
Slave_code	long	Slave CPU code on TLPs

Definition table for TNG messages (.msg)

Id	long	main code, assigned by Table Editor
Acronym	char[24]	message acronym – MANDATORY
Dbcode	long	Tdb internal code, assigned by software
Vmecode	long	TLP internal code
Typ	long	message type, allowed values are: 0=system, 1=info, 2=warning, 3=alarm
Descr	char[80]	message description
Txt	char[80]	message text, can contain any kind of data transferred with a casting operation; sender and receiver tasks must obviously agree on the content and meaning of the message

Definition table for TNG color palettes (.ctb)

Id	long	main code, assigned by Table Editor
Acronym	char[24]	acronym of the palette – MANDATORY
Dbcode	long	Tdb internal code, assigned by software
Descr	char[80]	palette description

Value	char[16,40]	table containing palette color names (max. 16)

Definition table for TNG interactive panels (.pdf)

Id	long	main code, assigned by Table Editor
Acronym	char[24]	panel acronym – MANDATORY
Dbcode	long	Tdb internal code, assigned by software
Screen	long	default screen (0-2), if the system has less screens, the panel is opened on screen 0
Descr	char[80]	panel description
Fname	char[80]	name of the file containing the panel definition (see the appropriate table)

Definition table for TNG interactive panel items (.pan)

id	long	main code, assigned by Graphic Editor
acronym	char[24]	item acronym – MANDATORY
group	char[8]	codes (up to 7) identifying the groups to which the item belongs; group codes are single alphabetic characters, and can be used for simultaneous operations on more items
type	long	item type, assigned by Graphic Editor; ranges between 1 and 15 (see Table 1)
color	long	item color, ranges between 0 and 23 (see Table 2)
style	long	item style; allowed values are: 0=solid, 1=dashed
thickness	long	item thickness; ranges between 0 and 7
x1	long	x coordinate of item
y1	long	y coordinate of item
x2	long	width/radius
y2	long	heigth/radius
font	long	font for labels, ranges between 0 and 9 (see Table 3)
mode	long	operating mode for STATUS type items; values range from 0 to 4 (see Table 4)
dbcode	long	Tdb internal code, assigned by software
sensible	long	TRUE if the item is sensible to operator's actions; can be used at run-time to enable or disable items or groups of items (see group)
threshold	double	threshold level at which items of type STATUS change
text	char[40]	text string to be shown in the items of type TEXT
pcf	char[28]	acronym of the Tdb parameter to be shown or to be used to compute the trasformation of a dynamic item; the acronym may end with an optional character string with the following format: [/[S|E|C][nn]] /S means that the operator defined value must be shown - /E means that the engineering value must be shown - /C means that the current value must be shown - no option means that the current value of element 0 must be shown - nn is the index of the element to be shown in case of items of type array
mccf	char[40]	acronym, followed by operands, of the command to be activated following an action on the item; applies to input items only; operands can be acronyms of other Tdb parameters, which will be substituted with their current value

stat	char[2,40]	name of two color elements, or two bitmaps, or two character strings to be assigned to the two stati of an item of type STATUS; *threshold* is the value at which the item switches from one status to the other
dynamic	long	TRUE if the item is a dynamic one; a dynamic item can undergo a transformation of its position, rotation angle or color following the value assumed by the parameter specified in pcf; the algorithm makes use of the minimum and maximum values allowed for the parameter, as stored in Tdb, and of the range defined below in this table to scale the output values correctly; if range=0 (or ctname=0) then no transformation takes place
xr	long	x coordinate variation range
yr	long	y coordinate variation range
angle	double	initial reference angle
angler	double	rotation angle variation range
xrot	long	x coordinate of rotation center
yrot	long	y coordinate of rotation center
ctname	char[24]	acronym of the color palette to be used for the transformation
ctmin	long	first color element to be used in the color palette
ctmax	long	last color element to be used in the color palette
radiogroup	long	index of the group for mutually exclusive buttons; it is equivalent to the concept of radiobutton

Table 1 : interactive panel graphical elements

N.	Name	I/O	Text	Dyn	Grph	Description
1	LINE	O	n	y	y	line segment
2	RECTANGLE	O	n	y	y	rectangle
3	BOX	O	n	y	y	filled rectangle
4	CIRCLE	O	n	y	y	circle
5	FILLCIRCLE	O	n	y	y	filled circle
6	LABEL	-	y	n	n	label
7	OUTPUT	O	y	y	n	output text field
8	SLIDER	I	n	n	n	slider with apply button
9	LEDBAR	O	n	n	n	color led bar
10	BUTTON	I	n	n	n	press button
11	STATUS	O	y	n	n	two status output field (color, text or bitmap)
12	ANALOG	O	n	n	n	analogue gauge
13	SETPAR	I	y	n	n	editable text field
14	CLOCK	O	n	n	n	bar indicator with number (scalable)
15	CHECKBUTT	I	n	n	n	check button (on/off)

I/O shows Input and Output items;
Text y means that the element contains text;
Dyn y means that the element is a dynamic one;

Grph y means that the element supports graphic transformations, besides color transformations

Table 2 : character fonts

Code	Font
0	tng_standard
1	tng_fixed
2	tng_fixed_bold
3	tng_fixed_large
4	tng_large
5	tng_large_bold
6	tng_large_italic
7	tng_large_extra
8	tng_symbol
9	tng_symbol_large

Table 3 : operating modes for elements of type STATUS

Code	Mode	Description
0	lamp	bicolor led, off=color[0], on=color[1]
1	text	off=tex[0] on dark background, on=text[1] on green background
2	fault	off=tex[0] on dark background, on=text[1] on red background
3	bitmap	off=bitmap[0], on=bitmap[1]

References

1. Galileo Project: Workstation Software System
A. Balestra, P. Marcucci, F. Pasian, M. Pucillo, R. Smareglia, C. Vuerli
TNG Technical Report n. 9

2. The Galileo User Interface User's Guide
Paolo Marcucci, Mauro Pucillo
May 1994 - Pubblicazione Osservatorio Astronomico di Trieste n. 1454

3. The Galileo User Interface Programmer's Guide
Paolo Marcucci, Mauro Pucillo
May 1994 - Pubblicazione Osservatorio Astronomico di Trieste n. 1455 DRAFT

4. The Galileo Help System v1.0
Paolo Marcucci
November 1992 - Pubblicazione Osservatorio Astronomico di Trieste n. 1456

5. The Galileo Table Editor
Paolo Marcucci, Mauro Pucillo

September 1992 - Pubblicazione Osservatorio Astronomico di Trieste n. 1450

6. The Galileo Interactive Panel Editor
Paolo Marcucci, Mauro Pucillo
May 1994 - Pubblicazione Osservatorio Astronomico di Trieste n. 1451 DRAFT

7. DBlib - Disk I/O handling library
Paolo Marcucci
April 1992 - Pubblicazione Osservatorio Astronomico di Trieste n. 1440