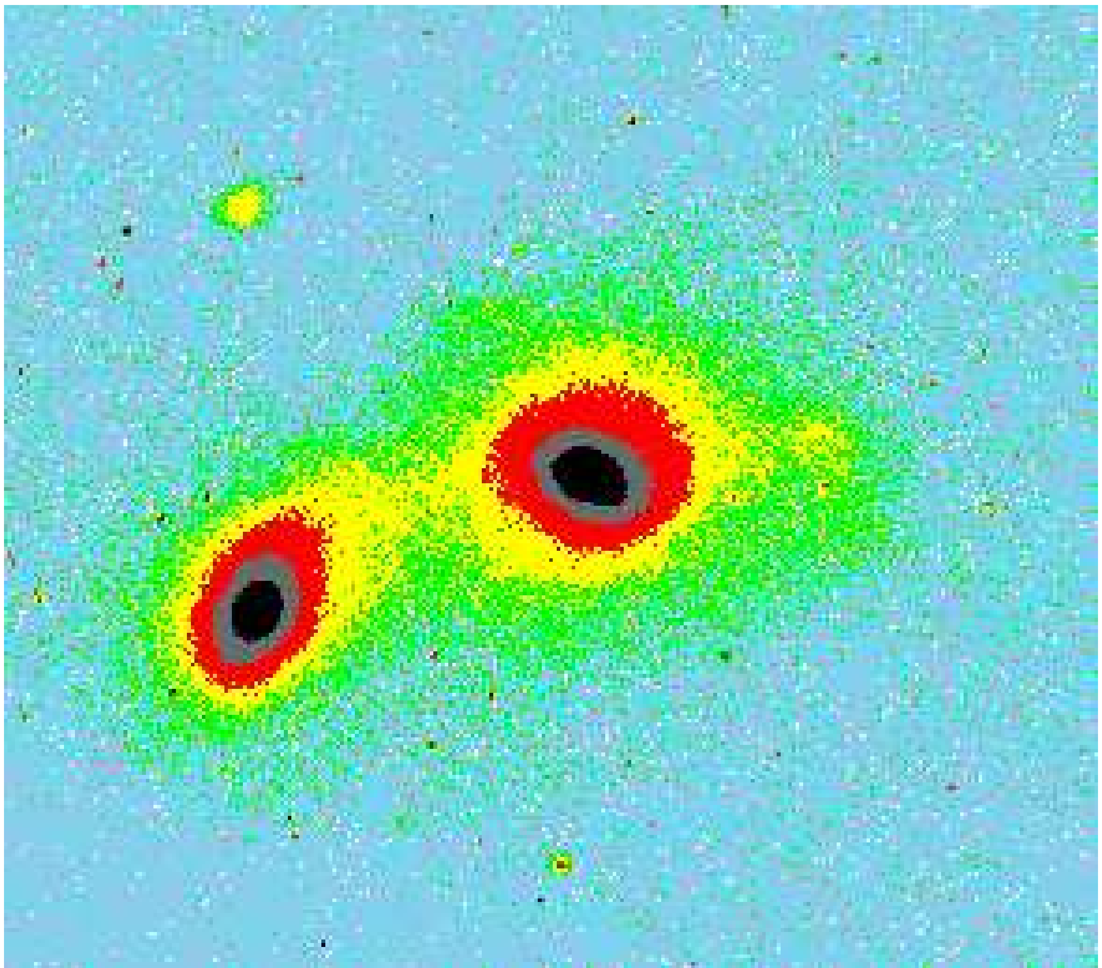
XNICS

Programmer's Guide

Version 1.0

Vladimir Gavryusev

Preprint n.9/2000



CAISMI CNR,
Largo E.Fermi 5, 50125, Firenze (Italia)

September 3, 2000

Acknowledgments

The Xnics software package and this manual have been developed by Vladimir Gavryusev at the Centro per l'Astronomia Infrarossa e lo Studio del Mezzo Interstellare del CNR and Osservatorio Astrofisico di Arcetri in the close contact with F. Lisi, C. Baffa, S. Gennari, E. Oliva, M. Sozzi, E. Giani and A. Checcucci.

Many implemented features are the result of their suggestions given during the NICS development.

Contents

Release notes **4**
Version 1.00 4

Introduction **5**

1 Xnics **7**
1.1 General organization 8
1.2 Interprocess Communication 9
1.3 Initialization and Acquisition Processors 10
1.4 Common Elements of the Widgets 13
1.5 Xnics Resources 15
1.6 Xnics Startup Arguments 23

2 Nicsgate **24**
2.1 General organization 24
2.2 Communication with Telescope 25
2.3 Communication with Transnix 26
2.4 Communication with Sensors 27
2.5 Communication with Motor Controller 28
2.6 Hardware Initialization Processor 28
2.7 Telescope Processor 30
2.8 Sensors Processor 31
2.9 Motors Processor 31
2.9.1 Motors Initialization 32
2.9.2 Motors in Debug Mode 34
2.10 Acquisition Processor 35
2.11 Internal Data Handling 36
2.12 Internal Image Viewer 37
2.13 Debug Mode 39
2.14 Nicsgate Resources 39

Bibliography **49**

List of Figures

1.1 Widget with Main menu 13
1.2 Widget with an Example of the Message 15

2.1 The menu for the Manual Control of the Motors 34
2.2 The Widget with the Acquired Image 38
2.3 The NICSgate Debug Monitor Widget 40

Release notes

Version 1.00

This is a description of the programming solutions used in the first production version of the **Xnics**, released at 20.07.2000 for the commissioning stage of the **NICS** (Near Infrared Camera-Spectrometer) at **TNG** (Telescopio Nazionale Galileo), La Palma, Spain. The software supports all basic measurement tasks as well as contains the functionality intended for the instruments development and their tuning. The user interface is X11 widget based interface (created upon ATHENA widget set). The software is created as a network distributed software, while basically is executed on the PC dedicated for the measurements.

The manual contains the basic information necessary for the understanding and, may be, the farther development of the software. The information how to use it in the real measurements one can find in *XNICS. User's Guide*[3]. The other technical details concerning the software and file system organization and procedures of software maintenance are described in *XNICS. Software Maintenance Guide*[4].

Introduction

This manual describes the general programming solutions implemented in the software **Xnics** (**X**11 **nics**, which was created for the real time control of the measurements using the Arcetri two-dimensional infrared instrument **NICS** (Near Infrared Camera-Spectrometer) [1,2]

This software is written using C-language and is splitted into two parts — user interface *xnics* and hardware dependent processes *nicsgate*. Both parts are organized as the standard widget–based X11 client programs and can be executed under any Unix X11 environment. This choice grants the mobility of the software.

The interprocess communication uses the standard stream sockets on the top of TCP/IP protocol. The Xnics is an intrinsically distributed software and can be executed across the Internet as well as on the stand alone computer, dedicated to the measurements.

The observer starts the user interface, *xnics*. All other necessary processes are started by *xnics* itself. The hardware control process *NICSgate* always has to run at the computer, directly connected to the **NICS** electronics, while *xnics* can be in principle executed at any other computer in the local network or even in the Internet. Currently both processes run at the same PC dedicated to the **NICS** control.

This PC is controlled by the Linux operating system.

The compilation process is supported by the standard programming tool **make**. All the software can be compiled by a single run of the **make**. Source texts for both parts of the software are placed into two corresponding sub-directories (`xnics` and `nicsgate`) of the top source directory (`src`). The `src` directory has a single file `Makefile`. The sub-directories contain the source files of the corresponding parts of the software. The files generally contain the programm material for the support of one certain task or the small number connected tasks and have indicative names. As usually, there are some miscellaneous routines which are typically in the file with the name `xutil.c`. Each part of the software can be compiled independently using **make**. For the compilation is used free distributed broadly available GNU `gcc` compiler. After the compilation both modules, *xnics* and *NICSgate* are moved to the `bin` directory.

The *xnics* uses the standard `C` and `X11R6` libraries, including `3D` library

of Athena Widget Set. Obviously, the standard version of this library can be used, influencing only a little bit the widgets look.

The *nicsgate* uses additionally support for fits files maintenance, provided by `libfitscio.a` [5]. There are available free sources of this library which can be easily installed at any unix system.

The software development is organized on the top of `rcs` system, providing the flexible way to follow the changes in software in multiple versions environment. This permits to return back if some taken solution is proved inappropriate. Also it is suitable when the group of programmers is working on the same project.

The routines contain a lot of comments, the names of the variables are usually self-explanatory, as far as it is possible to achieve. The text is written in a “structured” way, i.e. the blocks of the text are evidenced by the appropriate indents.

Chapter 1

Xnics

The choice of X11 client as an user interface is an obvious solution in the measurements controlled by a computer. It is hardware (computer) independent, flexible, well supported and covers all programmer's needs. `Xlib` and `Xtoolkit` provide all necessary low-level routines. There are also several sets of general purpose widgets, most of them commercial. This choice is not very appropriate for the scientific environment and we decided to use the free *Athena Widget Set*, creating on its base our own widgets. This widget set is developed enough to facilitate the programmer's work, is distributed freely with standard X11 distribution and has some development support from the independent programmers. For example, there is now available its `3D` version with audio support.

As any X11 client, *xnics* first of all initializes the application resources, creates the main menu widget, realizes it, starts the hardware controlling *NICSgate* process, writes all necessary *log* messages and leaves the control to the standard X11 event handler, the routine `XtAppMainLoop`. This handler supports all standard X11 client events, coming from a keyboard, screen and mouse through the Xserver.

The specific need of the *xnics* of the control of other types of events is supported by the routine `XtAppAddInput`.

The *xnics* as well as *nicsgate* refer to many specific files. All these files are the necessary part of the software. To make the software flexible, the certain file structure is used which is based on two **root** directories referenced as internal variables. All other directories and files are referenced *relatively* these two *RootDirectory* and *HardwareRootDirectory*. In a local installation of the software both variables refer to the **same** real directory in the file structure. These variables are defined as the X11 resources (Section 1.5).

1.1 General organization

All variables of the global use are placed into the application *header file* `xnics.h`. There is the structure, named `Wm`, with the widgets which need a special maintenance, additional to the standard of Xt. Then follows the structure with the instrument and session parameters, the flags with states of widgets, the instruments and internal “processors”. In this structure are also included some widget labels, which can be changed through X11 resource base. This structure, named `xnir_res`, is used as an application resource structure, thus `xnics.h` contains the section with resource name definition and the section `XtResource` with their initialization.

There are all necessary global variables for the support of the interprocess communication (using sockets). There are also defined some slave variables and arrays which are used in many routines temporarily or for the transmission of the information between routines described in different files.

For the use in a similar way of some callback functions there is a description of a `CallbackInput` structure. One such a structure, `XnirMessage`, used frequently in many places, is defined here as a global.

Finally, there are the global variables with display properties, some translation tables and `xnir_context` is defined as an application context structure (`XtAppContext`).

This header file, `xnics.h`, as well as the file `xfuncs.h`, where the prototypes of all globally used routines are described, are included in **all** files, containing the routines texts.

There are two major tasks of the hardware control (trough the communication with the *nicsgate*), which are organized as internal “processors”. Each “processor” is a set of the routines aimed on the solution of a single task, maintaining the common internal state, which in combination with coming events permits to select the appropriate action. These tasks and, correspondingly, the “processors” are the **hardware initialization** and the **data acquisition**.

The file `xnics.c` contains the main program with application initialization code and the main menu widget code.

The `hwinit.c` file contains the code starting the *nicsgate*, the “processor” for the communication with *nicsgate* during hardware initialization and its control, and the corresponding widget code.

All callback functions, invoked by command buttons from main menu reside in `xfuncs.c` file. They start the major acquisition tasks as well as submenus, etc. There are also the routines controlling the setting of observation parameters.

The file `hwacq.c` contains the routines for the control of the completeness of the observation task parameters, the acquisition “processor”, the corresponding widget code and some internal callback functions.

The routines for the support of the acquisition controlled from a job file ("automode"), including the *Object Pointing/Job Confirmation* widget code are in the `automode.c` file.

The routines to write the session log files (`xnics.log` and `sensors.log`), together with the routines making and retrieving a snap-shot of the *xnics* session state into/from the file `xnics.ini` are in the `environment.c` file.

The codes of the widgets with *Setup*-menus are in the `setup.c` file.

The file `nir_sock.c` contains the code of a *ServerSocket* routine, which realizes the socket connection from the server side in the server-client model of the communication. There are also two routines used from the client side, one for non-blocked type of connection, another for the blocked type. There is also the file `nir_sock.h`, where the prototypes of the mentioned above routines are described.

Several miscellaneous routines, for example *Message* widget, the support for the *List* widgets, etc. are in the `xutil.c` file.

1.2 Interprocess Communication

The communication between two parts of the software, *xnics* and *NICSgate* is implemented in client-server model, using stream sockets on top of TCP/IP protocol. Of course, when the software is executed on the same computer, the internal sockets are used.

In this communication *xnics* plays , in general, the part of a server, while *NICSgate* works as a client.

To create the robust links *xnics* uses the *ServerSocket* routine (`nir_sock.c`) which first of all creates the base socket, defines an environment (network or local), binds the socket and starts the listing on it. Then the *NICSsgate* process is started, provided in the argument line with the port for the connection, the display to connect and special initial string with necessary session parameters. The *NICSgate* immediately uses the *ClientSocket* routine to establish the connection. This routine create two "in" sockets, intended for the receiving of the messages from *xnics* and one "out" socket to send the messages. After the setting of the connection environment it asks server for the connections. The server is waiting the connection requests (up to 6, for robustness) using the routine *select* with predefined time-outs. If there is no request the session is aborted with corresponding diagnostic. In a normal case 3 socket connections are established, the checking initial transmissions executed and both programs can proceed.

The sockets are used in a different way. There are "in" and "out" sockets to permit asincronius communication in both directions. The message sent from any side should be acknowledged. If there is no acknowledgment received in due time, the sending side tries to transmit the pending message

up to 500 times, when the attempts are stopped and the corresponding diagnostic is issued. The acknowledgment could be a simple "ok" or an asked information.

The messages are the plain ASCII strings, readable immediately, thus avoiding any control character incidental appearance in the link. Each message has a message stamp field and the message body separated by at least one space. The message body could be empty.

The length of the message could be up to 256 bytes, more then enough for all needs.

Two sockets ("in" and "out") are used for the "routine" communication. From the *NICSgate* side the communications can be initiated at any appropriate place. The third, "out" for *xnics* and "in" for *NICSgate*, is used for a **priority** messages from *xnics*. Among them are the commands starting the acquisition tasks, the *NICSgate Debug Monitor*, the requests to show image or start the image viewer and the **STOP/CONTINUE** messages. This permits to break a task in progress even if the "routine" communication is blocked.

The sockets are controlled for the functionality and in case of the problems the diagnostic is issued.

The socket events are added to the main loop of the X11 events. The routines, which read the row messages coming through a socket, retrieve the message stamp and the body into two strings and transmit them for consequent elaboration to the top levels of the corresponding processors.

1.3 Initialization and Acquisition Processors

The tasks solved by the "processors" of *xnics* are relatively simple. They receive the message and take a corresponding action, usually edit the parameters in the internal data base, publish the obtained information, send the necessary response and change the sensitivity of the menu buttons. The stream of the messages from *NICSgate* first of all comes to the top logic routine *ReadMessageGate* (`hwinit.c`). This routine filters the messages of a top level (such as the sensors values, etc.). In the other case the corresponding processor is activated accordingly to the `xnir_res.state` flag.

Below there is a list of all the messages supported currently by the initialization processor and short description of the corresponding actions.

- *ACQ_MES3* : the error message; the *Message* widget with it is popped-up. The message is written into session log file.

- *INITXHW* : the messages about the current stage of the initialization process. They are published in the most frequently renewed label.

- *TELESCOPE, TELESCOPE_OFF, TELESCOPE_ON* : the telescope informative messages. They are published in the corresponding label.
- *TNX* : the messages about the stage of `transnix` initialization. They are published in the corresponding label.
- *DSP* : the messages about the stage of the `transnix` **DSP** sub-system initialization. They are published in the corresponding label.
- *DSPBAR* : the messages with the percentage of the execution of the current sub-task of the `transnix` initialization. They are shown in the corresponding scrollbar.
- *MOTORenvironment* : the request of the current motor environment settings. The information is send to *NICSgate*.
- *MOTOR* : the messages about the stage of the motors initialization. They are published in the corresponding label.
- *MOTOR_ON* : the message comes when the motor initialization is finished.
- *MOTORBAR* : the messages with the percentage of the execution of the current sub-task of the motor initialization. They are shown in the corresponding scrollbar.
- *ENDXHW* : received when the initialization is finished. Processor changes the sensitivity of the command buttons in *Hardware Initialization* widget and of the buttons in main menu. The corresponding information is published and the request to the user to enter his/her name is popped-up.

All the messages supported currently by the acquisition processor follow:

- *ACQenvironment* : the request of the current observation mode and session environment settings. The information is send to *NICSgate*.
- *OBSERVER* : the request of the observer name. The information is send to *NICSgate*.
- *MOTORenvironment* : the request of the current motor environment settings. The information is send to *NICSgate*.
- *ASKSOURCE* : the request of the source name to use when it is not available from the telescope data base. The information is send to *NICSgate*.

- *SOURCEFILE* : the message with the current source name received from the telescope data base. It is published in the corresponding label of the main menu.

- *MOSAICFILE* : the request of the mosaic file name to use. The information is send to *NICSgate*.

- *ACQUISBAR* : the messages with the percentage of the execution of the current sub-task of the data acquisition. They are shown in the corresponding scrollbar.

- *ACQ_MES1* : the message about the current stage of the acquisition process. They are published in the most frequently renewed label.

- *ACQ_MES2* : the messages about the current stage of the acquisition process and destinated for longer life to permit the user to consult them. They are published in the less frequently renewed label.

- *ACQ_MES3* : the error message; the *Message* widget with it is popped-up. If the session have been in *automode*, the current job section is marked as pending. Normally the acquisition task is interrupted. The message is written into the session log file.

- *MOSAICNUM* : the message with the current telescope position according the mosaic type of the acquisitions. It is published in the corresponding label.

- *SOURCESKY* : the message with the current telescope position according Source-Sky type of the acquisitions. It is published in the corresponding label.

- *AIRMASS* : the airmass received from the telescope data base. It is saved in the internal data base.

- *FOCUS* : the telescope focus value received from the telescope data base. It is saved in the internal data base.

- *MEASNUM* : this message is received when a data file is written. The current session state snap-shot is saved.

- *FILENAME* : this message is received when a data file is written. The special message is written into the session log file.

- *READY* : this message is received when the *NICSgate* state may be ambiguous from logical point of view of the *xnics*.

- *STOPXACQ* : received when the current acquisition task is interrupted (after the user request or because of any problem. Processor changes the sensitivity of the command buttons in *Acquisition* widget and of the buttons in the main menu. If the session have been in *automode*, the current job section is marked as pending. The corresponding information is published. The message is written into session log file.

- *ENDXACQ* : received when the current acquisition task is completed. Processor changes the sensitivity of the command buttons in *Acquisition* widget and of the buttons in the main menu. The current session state snap-shot is saved. The corresponding information is published.

1.4 Common Elements of the Widgets

Below we discuss the common elements, appearing in the different widgets created by *xnics*. They are generally similar in the organization and the treatment.

The Fig. 1.1 presents the main menu, created by *xncis*. It contains many of these common widgets.

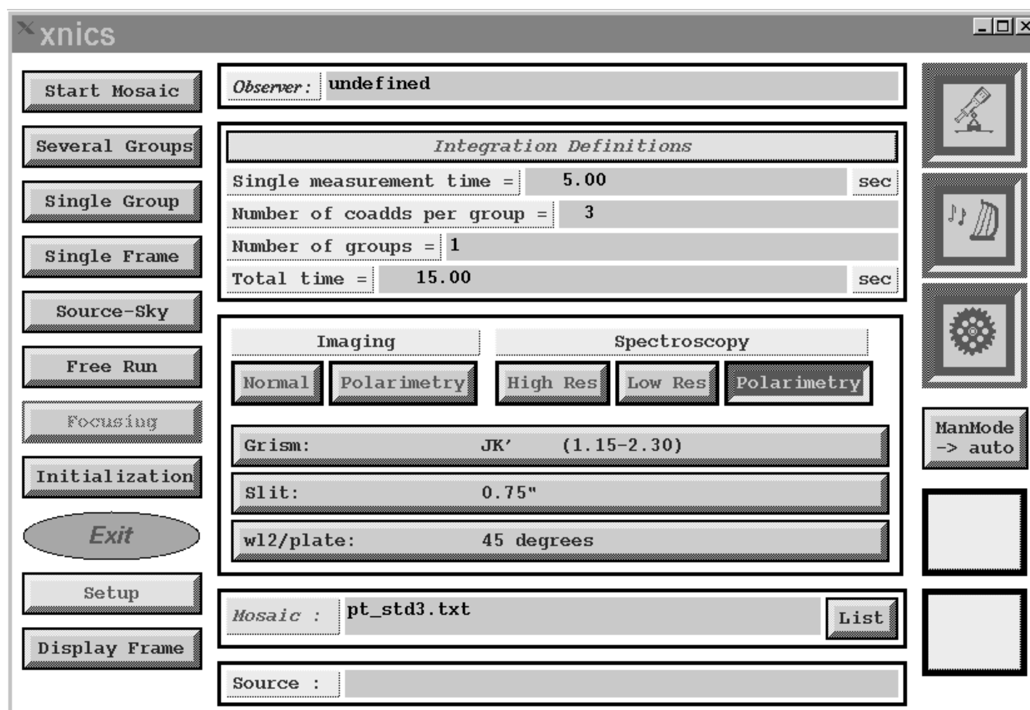


Figure 1.1: Widget with Main menu

The generic window, appearing during the work, can contain several common elements, such as:

- *Button* : The buttons can be of two types. The first one invokes the associated action (starts the jobs, pops-up the sub-menus, etc.). The button border is highlighted when the pointer is placed on it, to indicate that the button is ready for the selection. To invoke the corresponding action, it is enough to **click** on it the left mouse button. The selected button changes its color for the short time. The second type of the buttons usually belongs to a group of several buttons, from which a single one can be selected. In this case the selected button changes its color till the other button from the same group is selected. Sometimes the button can have several "positions", changing its label after each selection.

- *Label Sub-window* : This window usually contains short text, explaining the meaning of the other accompanied window. This type of the window is used also for the display of the regular messages following the execution of the current task.

- *Text Sub-window* : The main purpose of this type of the window is to permit the user to set the desirable value for some job parameters. There are displayed one or more lines of the text. The window can be or not be editable by the user. In the first case a cursor appears when the pointer is inside the window. Editable mode lets the user place the cursor anywhere in the text and modify the text at that position. The text written by the user is automatically stored by the program when the user press the *Return* key or moves the pointer outside the text window. It is achieved by the use of an appropriate *TextFieldTranslation* table. **If the cursor doesn't appear when the pointer is inside the window, it means that this window is not-editable and its behavior is like a label window**. But the parameter value shown in such a window still can be changed by the choice of the suitable item from the *List* window, which in this case can be popped-up using the nearly placed button **List**.

- *Scrollbar* : It is composed of a slider (with vertical or horizontal orientation) with a thumb inside. The scrollbar can be *informative* and in this case the thumb inside the scrollbar can not be moved by the user. Usually it is used to reflect roughly the performed percent of the task under the execution. In the other situations the *controlled* scrollbars provide the possibility to scan the contents of the accompanied window in the different directions. To move the thumb down (right) the user has to **click** the left mouse button in it. To move the thumb in the opposite directions the user has to use the right mouse button. The middle mouse button can be used to move the thumb in both directions while the button is pressed on it and not released (**dragged**).

- *List Sub-window* : This window displays several lines of the text (usually list of the files, list of the options, etc.) and one of the items can be chosen by the **click** of the left mouse button on it. If the list of the items do not fit into the window size, the controlled scrollbars are provided. To close the *List*-menu one should click on the **Dismiss** button. To make a choice and close the *List*-menu in a single shot one should make a **double click** on the *List*-menu item.
- *Compound List Widget* : This widget is popped-up by use of the button **List**. It contains two buttons, **DISMISS** and **DEFAULT**, and the list sub-window. The list sub-window is used to choose the desirable item from the list. The **DEFAULT** button can be used to return back the default item. The button **DISMISS** destroys this widget.
- *Compound Message Widget* : This widget is popped-up by the program when the observer should be informed about something what can influence the session. There is, evidently, the text window with the information and one button, **DISMISS**, which should be used to destroy this widget. The Fig. 1.2 shows an example of such a message widget.

Figure 1.2: Widget with an Example of the Message

All the windows, which can appear during the session, are the **standard X11 windows** and can be treated using the **standard Xserver actions**. But some internal logic can be destroyed in this way sometimes. Anyway, in the abnormal situation the Xserver actions can sometimes help.

1.5 Xnics Resources

Below are presented all X11 resources of *xnics* and the structure, where they are initialized, receiving the default values. In this structure one can see the correspondence between the resource name and the internal variable name.

```
#define XniButtonVertDistance      "buttonVertDistance"
#define XnCButtonVertDistance      "ButtonVertDistance"

#define XniObserverLabel           "observerLabel"
#define XnCObserverLabel           "ObserverLabel"
#define XniObserverName            "observerName"
#define XnCObserverName            "ObserverName"

#define XniIntLabel                "integrationLabel"
#define XnCIntLabel                "IntegrationLabel"
#define XniIntTimeLabel            "integrationTimeLabel"
#define XnCIntTimeLabel            "IntegrationTimeLabel"
#define XniIntTimeValue            "integrationTimeValue"
#define XnCIntTimeValue            "IntegrationTimeValue"
#define XniIntTimeMode             "singleTimeMode"
#define XnCIntTimeMode             "SingleTimeMode"
#define XniIntTimeUnit             "integrationTimeUnit"
#define XnCIntTimeUnit             "IntegrationTimeUnit"
#define XniIntNumberLabel          "integrationNumberLabel"
#define XnCIntNumberLabel          "IntegrationNumberLabel"
#define XniIntNumberValue          "integrationNumberValue"
#define XnCIntNumberValue          "IntegrationNumberValue"
#define XniIntGroupsLabel          "integrationGroupsLabel"
#define XnCIntGroupsLabel          "IntegrationGroupsLabel"
#define XniIntGroupsValue          "integrationGroupsValue"
#define XnCIntGroupsValue          "IntegrationGroupsValue"
#define XniIntTotalTimeLabel       "integrationTotalTimeLabel"
#define XnCIntTotalTimeLabel       "IntegrationTotalTimeLabel"

#define XniFrameLabel              "frameLabel"
#define XnCFrameLabel              "FrameLabel"
#define XniFrameName               "frameName"
#define XnCFrameName               "FrameName"

#define XniMsLabel                 "mosaicLabel"
#define XnCMsLabel                 "MosaicLabel"
#define XniMsName                  "mosaicName"
#define XnCMsName                  "MosaicName"

#define XniPolarLabel              "polarLabel"
#define XnCPolarLabel              "PolarLabel"
#define XniPolarInsert             "polarInsert"
#define XnCPolarInsert             "PolarInsert"
```

```
#define XniPolarLinea              "polarLinea"
#define XnCPolarLinea              "PolarLinea"

#define XniTimerDelay              "timerDelay"
#define XnCTimerDelay              "TimerDelay"

#define XniDispLabel               "displayFileLabel"
#define XnCDispLabel               "DisplayFileLabel"
#define XniDispPath                "displayPath"
#define XnCDispPath                "DisplayPath"

#define XniTelescopeName           "telescopeName"
#define XnCTelescopeName           "TelescopeName"
#define XniTelescopeAddress        "telescopeAddress"
#define XnCTelescopeAddress        "TelescopeAddress"
#define XniTelescopePort           "telescopePort"
#define XnCTelescopePort           "TelescopePort"

#define XniHostsList               "hostsList"
#define XnCHostsList               "HostsList"
#define XniRootDir                 "rootDirectory"
#define XnCRootDir                 "RootDirectory"
#define XniHardHost                "hardwareHost"
#define XnCHardHost                "HardwareHost"
#define XniHRootDir                "hardwareRootDirectory"
#define XnCHRootDir                "HardwareRootDirectory"

#define XniiAccount                "internalAccount"
#define XnCiAccount                "InternalAccount"
#define XniPassword                "passwordSoftir"
#define XnCPassword                "PasswordSoftir"

#define XniNomotors                "noMotors"
#define XnCNomotors                "NoMotors"
#define XniNoreset                 "noReset"
#define XnCNoreset                 "NoReset"
#define XniHardreset               "hardReset"
#define XnCHardreset               "HardReset"

#define XniNosensors               "noSensors"
#define XnCNosensors               "NoSensors"

#define XniBitTeles               "bitmapTelescope"
```

```
#define XnCBitTeles                "BitmapTelescope"
#define XniBitInstr                "bitmapInstrument"
#define XnCBitInstr                "BitmapInstrument"
#define XniBitMotor                "bitmapMotor"
#define XnCBitMotor                "BitmapMotor"

#define XniSockBufsize               "socketBufferSize"
#define XnCSockBufsize               "SocketBufferSize"

#define XniIniExp                    "iniexpires"
#define XnCIniExp                    "Iniexpires"

#define XniFocStep                   "focusStep"
#define XnCFocStep                   "FocusStep"
#define XniFocLevel                  "focusLevel"
#define XnCFocLevel                  "FocusLevel"
#define XniFocSum                    "focusSum"
#define XnCFocSum                    "FocusSum"

#define XniIntBad                    "badPixelLevel"
#define XnCIntBad                    "BadPixelLevel"
#define XniIntSat                    "saturationLevel"
#define XnCIntSat                    "SaturationLevel"
#define XniSatWarn                   "saturationWarning"
#define XnCSatWarn                   "SaturationWarning"


static XtResource  xnir_resources[] = {
  {
     XniButtonVertDistance,XnCButtonVertDistance,
     XtRInt,sizeof(int),XtOffsetOf(XnirAppData, button_vert_distance),
     XtRImmediate, (XtPointer) 10
  },
  {
     XniObserverLabel,XnCObserverLabel,
     XtRString,sizeof(String),XtOffsetOf(XnirAppData, observer_label),
     XtRImmediate, "Observer : "
  },
  {
     XniObserverName,XnCObserverName,
     XtRString,sizeof(String),XtOffsetOf(XnirAppData, observer_name),
     XtRImmediate, "undefined"
  },
```

```
{
   XniIntLabel,XnCIntLabel,
   XtRString,sizeof(String),XtOffsetOf(XnirAppData, int_label),
   XtRImmediate, "Integration Definitions"
},
{
   XniIntTimeLabel,XnCIntTimeLabel,
   XtRString,sizeof(String),XtOffsetOf(XnirAppData, int_time_label),
   XtRImmediate, "Single measurement time ="
},
{
   XniIntTimeValue,XnCIntTimeValue,
   XtRFloat,sizeof(float),XtOffsetOf(XnirAppData, int_time_value),
   XtRString, "1."
},
{
   XniIntTimeUnit,XnCIntTimeUnit,
   XtRString,sizeof(String),XtOffsetOf(XnirAppData, int_time_unit),
   XtRImmediate, "sec"
},
{
   XniIntTimeMode,XnCIntTimeMode,
   XtRString,sizeof(String),XtOffsetOf(XnirAppData, singleTimeMode),
   XtRImmediate, "SetByHands  "
},
{
   XniIntNumberLabel,XnCIntNumberLabel,
   XtRString,sizeof(String),XtOffsetOf(XnirAppData, int_number_label),
   XtRImmediate, "Number of coadds per group ="
},
{
   XniIntNumberValue,XnCIntNumberValue,
   XtRInt,sizeof(int),XtOffsetOf(XnirAppData, int_number_value),
   XtRImmediate, (XtPointer) 5
},
{
   XniIntGroupsLabel,XnCIntGroupsLabel,
   XtRString,sizeof(String),XtOffsetOf(XnirAppData, int_groups_label),
   XtRImmediate, "Number of groups ="
},
{
   XniIntGroupsValue,XnCIntGroupsValue,
   XtRInt,sizeof(int),XtOffsetOf(XnirAppData, int_groups_value),
```

```
    XtRImmediate, (XtPointer) 10
  },
  {
    XniIntTotalTimeLabel,XnCIntTotalTimeLabel,
    XtRString,sizeof(String),XtOffsetOf(XnirAppData, int_total_time_label),
    XtRImmediate, "Total time ="
  },
  {
    XniTelescopeName,XnCTelescopeName,
    XtRString,sizeof(String),XtOffsetOf(XnirAppData, TelescopeName),
    XtRImmediate, "TNG"
  },
  {
    XniTelescopeAddress,XnCTelescopeAddress,
    XtRString,sizeof(String),XtOffsetOf(XnirAppData, TelescopeAddress),
    XtRImmediate, "/dev/ttyS0"
  },
  {
    XniTelescopePort,XnCTelescopePort,
    XtRInt,sizeof(int),XtOffsetOf(XnirAppData, TelescopePort),
    XtRImmediate, (XtPointer) 1040
  },
  {
    XniFrameLabel,XnCFrameLabel,
    XtRString,sizeof(String),XtOffsetOf(XnirAppData, frame_label),
    XtRImmediate, "Source : "
  },
  {
    XniMsLabel,XnCMsLabel,
    XtRString,sizeof(String),XtOffsetOf(XnirAppData, mosaic_label),
    XtRImmediate, "Mosaic : "
  },
  {
    XniMsName,XnCMsName,
    XtRString,sizeof(String),XtOffsetOf(XnirAppData, mosaic_name_default),
    XtRImmediate, "pt_std3.txt"
  },
  {
    XniTimerDelay,XnCTimerDelay,
    XtRInt,sizeof(int),XtOffsetOf(XnirAppData, timer_delay),
    XtRImmediate, (XtPointer) 30
  },
  {
```

```
    XniRootDir,XnCRootDir,
    XtRString,sizeof(String),XtOffsetOf(XnirAppData, root_dir),
    XtRImmediate, "/home/nics/nics"
  },
  {
    XniHardHost,XnCHardHost,
    XtRString,sizeof(String),XtOffsetOf(XnirAppData, hardware_host),
    XtRImmediate, "local"
  },
  {
    XniHRootDir,XnCHRootDir,
    XtRString,sizeof(String),XtOffsetOf(XnirAppData, hardwareRoot_dir),
    XtRImmediate, "/home/nics/nics"
  },
  {
    XniiAccount,XnCiAccount,
    XtRString,sizeof(String),XtOffsetOf(XnirAppData, iaccount),
    XtRImmediate, "nics"
  },
  {
    XniNomotors,XnCNomotors,
    XtRBoolean,sizeof(Boolean),XtOffsetOf(XnirAppData, nomotors),
    XtRImmediate, (XtPointer)False
  },
  {
    XniNosensors,XnCNosensors,
    XtRBoolean,sizeof(Boolean),XtOffsetOf(XnirAppData, nosensors),
    XtRImmediate, (XtPointer)False
  },
  {
    XniNoreset,XnCNoreset,
    XtRBoolean,sizeof(Boolean),XtOffsetOf(XnirAppData, noreset),
    XtRImmediate, (XtPointer)False
  },
  {
    XniHardreset,XnCHardreset,
    XtRBoolean,sizeof(Boolean),XtOffsetOf(XnirAppData, hardreset),
    XtRImmediate, (XtPointer)False
  },
  {
    XniBitTeles,XnCBitTeles,
    XtRString,sizeof(String),XtOffsetOf(XnirAppData,bitTelescope ),
    XtRImmediate, "telescop.bit"
```

```
  },
  {
     XniBitInstr,XnCBitInstr,
     XtRString,sizeof(String),XtOffsetOf(XnirAppData,bitInstrument),
     XtRImmediate, "instrum.bit"
  },
  {
     XniBitMotor,XnCBitMotor,
     XtRString,sizeof(String),XtOffsetOf(XnirAppData,bitMotor),
     XtRImmediate, "motor.bit"
  },
  {
     XniSockBufsize,XnCSockBufsize,
     XtRInt,sizeof(long),XtOffsetOf(XnirAppData,sock_buffer_size),
     XtRImmediate, (XtPointer) 1040
  },
  {
     XniIniExp,XnCIniExp,
     XtRInt,sizeof(int),XtOffsetOf(XnirAppData,iniexpired),
     XtRImmediate, (XtPointer) 20
  },
  {
     XniFocLevel,XnCFocLevel,
     XtRFloat,sizeof(float),XtOffsetOf(XnirAppData, focus_level),
     XtRString, ".5"
  },
  {
     XniFocStep,XnCFocStep,
     XtRInt,sizeof(int),XtOffsetOf(XnirAppData,focus_step),
     XtRImmediate, (XtPointer) 10
  },
  {
     XniFocSum,XnCFocSum,
     XtRInt,sizeof(int),XtOffsetOf(XnirAppData,focus_sumNumb),
     XtRImmediate, (XtPointer) 4
  },
  {
     XniIntBad,XnCIntBad,
     XtRInt,sizeof(long),XtOffsetOf(XnirAppData,int_bad),
     XtRImmediate, (XtPointer) 15000
  },
  {
     XniIntSat,XnCIntSat,
```

```
    XtRInt,sizeof(long),XtOffsetOf(XnirAppData,int_sat),
    XtRImmediate, (XtPointer) 12000
  },
  {
    XniSatWarn,XnCSatWarn,
    XtRInt,sizeof(int),XtOffsetOf(XnirAppData,sat_warn),
    XtRImmediate, (XtPointer) 257
  },
};
```

1.6 Xnics Startup Arguments

To start the program the user has to issue the command

xnics [-[option] -[option] ...]

The most simple way is to omit all the options, using those defined by default or in the resource data base.

Currently there are the following options, relevant to the measurements, available:

- **-har***[dreset]* to start a session **with** an unconditional resetting of all the motors, controlling filters, grism, etc.
- **-nor***[eset]* to start a session **without** automatic resetting of all the motors.
- **-nom***[otors]* to start a session **without** any control of the motors. Evidently this option is of a little use for an observer. It is intended for the abnormal situations.
- **-nos***[ensors]* to start a session **without** the control of the temperature and pressure sensors. Has **no** impact on the measurements. Just the corresponding information is lost.

Obviously, **only one** of the options controlling motor behaviour should be issued. The detailed description of the influence of the chosen options on the measurement session is provided in the corresponding sections of this manual (2.9.1).

The *xnics* program is an X11 client and as a such can be provided by the standard X11 options. One important example is the **-display** parameter. If this parameter is provided, all the windows will appear on the referenced screen, given that all the permissions are provided.

Chapter 2

Nicsgate

The purpose of the *nicsgate* is to control the instruments. It should be done in real time, with the minimum delays, especially during the acquisition of the data. When there is more then a single instrument to control, the mixed stream of many events of the different origin should be treated simultaneously. The X11 has very elaborated tools to support such a situation. Thus the *nicsgate* is also organized as X11 client. For the reasons of the effectiveness, it also includes some elements of the user interface — namely the tools for data visualization and analysis and the *debug* menu, intended for the support of the instrument development and tuning.

The compiled module of *nicsgate*, ready for the execution, is named *NICSgate*.

After the establishing of the colloquium with *xnics*, the *NICSgate* makes an initialization of its X11 resource data base and the session environment. All necessary parameters of the session are transmitted in an initial string from *xnics*. Then it does not realizes any widgets but simply gives a control to the `XtAppMainLoop`, including into the controlled stream of the events the established socket connections. The only widget created at this stage is a **topLevel** widget, supporting an application structure.

2.1 General organization

As in the case of *xnics*, all variables of the global use are placed into the application *header file* `nicsgate.h`. Again there is the structure, named `Wm`, containing currently a single widget **topLevel**. The structure, also named `xnir_res`, is used as an application resource structure. It contains also the instrument and session parameters, the flags with the states of the internal "processors" dedicated for the control of the instruments . Then follow the section with resource name definitions and the section `XtResource` with their initialization.

Also there are all necessary global variables for the support of the inter-process communication (using sockets) and for the support of the communication with instruments. There are also defined some slave variables and arrays which are used in many routines temporarily or for the transmission of the information between routines described in different files.

This header file, `nicsgate.h`, and the file `nicsfun.h`, where the prototypes of all globally used routines are described, are included in **all** files, containing the routines texts.

The main program with an application initialization code is in the file `nicsgate.c`. It contains also the top logic routine, responsible for the colloquium with the *xnics*.

The file `socket.c` contains the low level routines supporting the *socket* communication with the *xnics*.

All low level functions, and the event controlling "processor" for *transnix* communication support reside in `tnixcom.c` file.

The *telescope* supporting "processor" routines are in the file `xtelescope.c`.

The routines of the *motors* "processor" are in the file `xmotors.c`, the *sensors* "processor" — in the file `sensors.c`.

The file `fits.c` contains all necessary support for data input/output from/into fits files.

The `xhwinit.c` file contains the code of the instrument *initialization* "processor" (its top logic and some task dedicated routines).

The routines of the *acquisition* "processor" reside in the `xacquis.c` file.

There are also the elements of the user interface — the tools for data visualization and analysis and the *debug* menu, intended for the support of the instrument development and tuning.

The file `ximage.c` contains the routines of the internal data image viewer and the corresponding widget codes.

The widgets and routines, used to invoke the external auxiliary utilities and image viewer are in the file `xdisplay.c`.

In the file `xmenu.c` there are the routines of the *debug monitor*, used for the **low-level** control of some instruments.

Several miscellaneous routines, for example *Message* widget, the support for the *List* widgets, etc. are in the `xutil.c` file.

2.2 Communication with Telescope

The *nicsgate* communicates with a dedicated auxiliary program of TNG software through the usual serial port. From the TNG side the port is doubled thus, in fact providing two serial connections, one for the control of the telescope movements, and another for the interrogation of TNG data base. There is a software switch by which the *nicsgate* controls with which of two

auxiliary programs it is communicating. This working mode is already used by *Xnir* software controlling **ARNICA** instrument.

Any time the writing or the reading is executed on the telescope serial port, its functionality is checked.

In the case of data base interrogation the communication is handled in a simple blocked mode — request and waiting of the response.

For the telescope movements at low level the protocol is the same, just, because the telescope movements take a time, the response is only the confirmation of the command receiving.

The control, that the telescope movement is really finished, is done by the interrogation of the current position of the telescope through the TNG data base.

2.3 Communication with Transnix

The data acquisition by the **NICS** instrument is controlled by the transputer based electronics, where should run the *transnix* program [1,2]. The low-level protocol of the communication between *xnics* (in fact, *NICSgate*) and *transnix* is described in [6].

The electronics board is accessible by the software through the special link. First of all the link status is checked by a program *ispy*. The response is analyzed and, if the link is not used by the other program and the transputer network connected to it is the correct one, the work can proceed.

The transputer electronics has initially only a *boot loader*, which permits to boot in it through the link the compiled module of the *transnix*. This module is read from a file in a binary mode and written into the link. The binary image file name is referenced by an internal variable and can be set as a resource, or dynamically, using the *Debug* menu, if necessary. Its standard name is *transnix.btl*. The routine making the initial boot is named `boot_TNX` and resides in the file `tnixcom.c`, as the most number of other relevant to the *transnix* routines. The internal global pointer to the *transnix* link is named `tnxid`.

From this point the normal communication with *transnix* can be started. The low-level communication protocol defines the structure of the messages sent/received to/from *transnix*. To make a connection controllable in any real situation, the high-level protocol is implemented.

The *transnix* should acknowledge any received message immediately, while from the *NICSgate* side it is not obligatory. The reason is that from the *transnix* may come many different informative, warning or error messages and it is not always capable to receive after each sending. So the acknowledge from the *NICSgate* side is send only in well defined cases, when it is necessary. The communication "processor" assigns to any message to send

to the *transnix* the **pending status**. It can have three levels:

1. **Send pending**. Means the command (message) should be send to the *transnix* in the appropriate time.

2. **Acknowledgment pending**. Means the command is sent but not yet acknowledged.

3. **Execution pending**. The command is received and is acknowledged by the *transnix*. It should produce some delayed result marked by a corresponding message from *transnix*.

The transnix communication processor maintains the status of all the pending commands. The link basic functionality is controlled by the `ioctl` system routine. The special routine *WriteToTNX* controls all writing activity to the transnix link, while the routine *ReadFromTNX* controls all incoming activity from it. Both routines are **time-out** controlled. There is implemented the set of the different time-outs for the different situations, providing thus the robust control on all levels of the command pending.

One of the transputer nodes sends always with a predefined repetition delay a special message, meaning "I'm a live". It permits to control the *transnix* functionality even when there is no production activity.

The incoming message is analyzed and in a correspondence with its contents and the pending commands the appropriate action is taken. It could be an action of the *transnix communication* processor, or the control can be passed to the other processor, but always the *transnix communication* processor is leaved in the **watching** state. Normally it is active only for very short time, expecting the time-outs.

There is one particular case, when this processor takes a control for relatively long time. It happens when the acquired data frame comes through the link. It is a huge amount of a data, hence, to minimize the measurement time and to ensure the integrity of the image, the processor does not leave control at all till the complete frame is received (the frame consists of many messages).

2.4 Communication with Sensors

The communication with sensors is executed by transnix communication processor in a *transparent* mode — the command goes through the transnix link to the serial port of the NICS electronics and vice versa. The messages from sensors are passed to the *sensor* processor.

2.5 Communication with Motor Controller

As the telescope, the motor controller is connected directly to the computer serial port. There are seven motors controlled simultaneously by this controller [7]. The given motor can be addressed by explicit request in the sent command. All motors together can be addressed also.

The functionality of the motors serial port is checked at all communications. When the command is send to the motor controller it responds by echo, which confirms the command receiving. The protocol of the communication supports the possible delayed response, when the controller is not able to report its state immediately (some movements need minutes). In this case the special event processor is used for the control of this serial port. This routine is controlled by the time-outs and finishes its work or with normally coming response or with expectation time expiring. The time-outs are set to provide the effective functionality of all instruments together and the granted execution of the requested movement.

Thus, sometimes the *motors* processor is in a "watching" mode, expecting the motors controller messages. The most of time it is sleeping, especially when **no** activity is permitted on the motors (during the acquisition).

2.6 Hardware Initialization Processor

The execution of the hardware initialization processor is controlled by the parameter **action**, provided in a calling statement and by the internal processor **state** variables, which reflect the current environment. The parameter **action** permits to provide this processor with the information about the special situation which was established by the other processor. The **state** variables reflect the achieved progress in the initialization process.

The **action** parameter (int) can have the next values:

- **0** : start the initialization process;
- **1** : continue TRANSNIX initialization;
- **2** : continue TELESCOPE initialization;
- **3** : continue MOTORS initialization;
- **4** : continue SENSORS initialization;
- **-1** : initialization is stopped by an user, do not proceed, keep `tnx_state`;
- **-2** : continue previously stopped initialization process;
- **-3** : break the current initialization part with an error;

- **-4** : break the current initialization part with an error.

There are several `state` variables (all of the integer type).
The general processor state variable `state` can have the next values:

- **0** : the processor is idle, not yet initialized;
- **1** : the initialization is in progress;
- **-1** : the processor is idle, the initialization is done;
- **-2** : the processor is idle, the initialization failed.

There is also the `stop` general processor variable which can have the next values:

- **0** : normal initialization process;
- **1** : the initialization is stopped by the user, do not proceed, keep state;
- **-1** : an error during the current part of the initialization.

There is also one state variable for each instrument to be initialized.
The telescope variable `tel_state` can have the next values:

- **0** : telescope is not yet initialized;
- **1** : the initialization is in progress;
- **-1** : the initialization is done successfully;
- **-2** : the initialization failed.

The transnix variable `tnx_state` can have the next values:

- **0** : transnix is not yet initialized;
- **1** : booting the transnix;
- **2** : watch transnix for the first message;
- **3** : DSP initialization start/continue;
- **4** : expecting the NICS board identifier;
- **5** : expecting the LED to be toggled;
- **6-12** : loading the $\widehat{\text{W}}$aveForms number 10-70;

- **13-18** : setting the biases;
- **19** : setting of the CCD area parameters;
- **20** : setting of the CCD box parameters;
- **-1** : the initialization is done successfully;
- **-2** : the initialization failed.

The motors variable `mot_state` and the sensor variable `sens_state` can have the next values:

- **0** : not yet initialized;
- **1** : the initialization is in progress;
- **-1** : the initialization is done successfully;
- **-2** : the initialization failed.

When it is necessary to initialize all the instruments from the very beginning, the initialization processor is called with an action value 0. First of all the telescope processor is called with action=0, then the transnix program is booted and the transnix initialization is continued in a sequence defined by the `tnx_state` variable.

Only if the transnix is initialized correctly the motors initialization is started. It is executed by the *motors processor* and is described in the Section 2.9.

When the motor initialization is finished, the *sensors processor* (2.8) is called. It simply sends to the sensors the contents of the file `sensors.iniseq`, residing in a `resource` directory. This finishes the whole process of the initialization.

2.7 Telescope Processor

The telescope processor is rather simple. It is controlled only by an **action** input parameter.

When called with `action` equal to 0, the telescope processor opens and controls the telescope serial port.

With `action` equal to 1, it does the normal processing, depending on the global flag `xnir_res.acq_state`. The telescope processor asks the necessary information from the telescope data base, and, if necessary, moves the telescope to the next position according the mosaic acquisition definitions. This definitions are taken from the *mosaic* file, which name is provided by *xnics*. The rules to which this file should confirm are described in [3].

When the mosaic acquisition is finished or interrupted by the user, the additional movement of the telescope is executed — it is moved to the origin, the position **before** the acquisition was started.

To break the mosaic execution the telescope processor should be called with `action`=-1.

2.8 Sensors Processor

This processor is also very simple. It is invoked during the initialization, if the option *-nosensors* was not used at the start-up of the *xnics*. The sensors processor uses for the communication with NICS sensors the transnix communication processor, because the sensors are connected to the serial port of the NICS electronics. The contents of the file `sensors.iniseq` is sent during the initialization and the processor continues its work. With a predefined delay it is awaked and sends the series of the sensors interrogation commands. When the response comes it is transmitted to the *xnics* and placed into internal data base. *Xnics* also writes the received information into the `sensors.log` file. If there is no response after a predefined time-out, the corresponding sensor interrogation is disabled. Normally the sensors processor asks the pressure, the temperature from LS330 sensor and 3 temperatures from LS208 sensor.

2.9 Motors Processor

The motors processor is in some sense the most complicated processor of this software. The reason is that there are seven motors to control, and several of them have very peculiar behaviour. The timings of all the motors are different. Some motors — number 1 (camera), 4 (filter), 5 (grism), 6 (aperture) — control the wheels, thus permitting the movements in both directions. The motors 2 (array), 3 (lyot) and 7 (aperture sector paddle) control slides, thus their movements are limited from both ends. While most of the motors have well defined home positions and can be resetted simply enough, just there should be chosen correct direction, velocity and acceleration of the movements, the 7th motor needs very special treatment due to the technical reasons. The motors positioning should be finished always approaching from the same side, because the motor controller can loose the counts.

There is also necessity turn on the current of the motors for the movements, and turn it off during the acquisition, because it have been proved that the motors under current produce additional, not negligible noise in CCD detector.

Finally, this processor should be active only when the motors positioning is executed.

The motors processor provides three types of the services — the complete initialization of the motors, their positioning to reflect the settings for the current acquisition task and the free positioning of any motor for the development and tuning purpose.

As far as the normal (for the current acquisition) positioning of the motors is the last part of the initialization we do not describe it separately.

As all other processors the actions of this one depend on the input parameter **action** and on the maintained internal state. There is also additional input parameter (string), the **command** to execute when asked in *debug* mode.

The following actions are permitted:

- **0** : the initialization;
- **1** : the regular processing controlled by the state values;
- **2** : the processing with indirect call when the response from the motor controller comes and support for the *debug* mode.

The internal state of the motors processor is much more complicated than in the case of the others.

There is a variable of the processing `state` (int), which can have the values from 0 to 8. The chosen action depends not only on this variable value but also on the values of the motors state variables, stored in an array `mot_state`, one for each motor. We will describe the state values setting below.

NICSgate also maintains the positions of all motors in two global arrays `xnir_res.MotPreviousPosition` and `xnir_res.MotPosition`.

2.9.1 Motors Initialization

The complete initialization of the motors consists in their resetting (move to the home position, where the position values are set to zero) and their positioning into the situation saved in `xnics.ini` file.

Because the motor controller maintains their positions (if everything is ok), *xnics* implements as a normal behaviour so called “soft reset”.

First of all, all the motors are interrogated for their positions.

If the presumed position of the motor and the responded value are the same we accept it and do not reset. Otherwise, or if there is no saved information about motors position, the motor is forced to move to the home. The position is checked again and, if it is correct, the motor moves to the position defined by observation mode setting.

If *xnics* is started with option `-noreset`, the reported positions are taken as valid without any comparison.

The opposite happens when *xnics* is started with option `-hardreset`. In this case all the motors are forced to the home position.

If *xnics* is started with option `-nomotors`, the motors processor does not execute any control of the motors, just skipping any action on them.

The *NICSgate* obtains the starting option from *xnics* in the initial string and establishes the corresponding global flags.

At the beginning of the initialization the motors processor is called with the action value 0 and the *command* parameter equal to "0". It asks the *xnics* about the current environment setting, possibly restored from `xnics.ini`, sets for all *controlled* motors `mot_state` to 0 and for all motors *excluded from the control* to -1. Then it reads from the file `motors.iniseq` residing in the directory `resource` all commands for the initial setting. Then the commands are executed (sent to motor controller) one by one. The `state` is equal to 0. When all the initial commands are sent, the `state` is changed to 1 and the *AskPosition* sub-processor is activated. This sub-processor interrogates all the motors for the current position. The `mot_state` of the corresponding motor is changed to 1. The requests are sent to all the motors simultaneously. When the answers come, the motors processor in called with the action value 2. The *SendReset* sub-processor is activated. This sub-processor compare the returned and expected positions for each motor and activates the reset procedure in accordance with the global flags set. If the `xnir_res.hardreset` is set to 1, the reset is done for all motors in any situation. If the `xnir_res.noreset` is set to 1, the reset is **not done** for all motors in any situation, the reported position is accepted as valid and set as a current position into `xnir_res.MotPosition`. In a normal processing the reset is done only for those motors, for which the reported position does not correspond to the expected one. When all resets are issued, the *SendReset* sub-processor reports "nojob", the `state` is changed to 5 and the *AskPosition* sub-processor is activated again. We skip the reported positions at this point, they are asked only to ensure that the reset movements are finished. Now we interrogate the indexer status by the *AskIndexerStatus* sub-processor with `state` set to 6. The responses are controlled by *CheckIndexerStatus* routine, if all is correct, we force the motor controller to set the current positions to 0. The `xnir_res.hardreset` flag, if set, is turned out (to 0). We do the hard reset only once. The `state` is changed to 4 and the *AskPosition* sub-processor is activated again. When all position checking is executed, the `state` is changed to 2, the `xnir_res.noreset` flag, if set, is turned out (to 0), permitting the soft reset in the farther work.

The first stage of the initialization is finished. The positions of the motors are known. Then, we can proceed with the current observation mode settings. This work is normally done by the motors processor in the beginning of all the acquisition tasks. The information is asked again from *xnics*. If there is some job to do, i.e. the asked positions for some motors are different from

the previous ones, the `state` is set to 3 and the *SetPosition* sub-processor is activated. It sends the corresponding requests to all influenced motors and then send the command to execute the requests. The `state` is set to 4, thus activating again the pair of *AskPosition/CheckPosition* sub-processors. When all the controls are executed and all is in the correct situation the current of the motors is turned off and the motors processor becomes inactive. The idle value for the `state` variable is 8.

2.9.2 Motors in Debug Mode

For the development of the instrument and the tuning of its functionalities it is necessary to have the possibility to move any motor to any possible position and, probably, to make the data acquisition in this non-standard environment. For this purpose the motors processor provides the possibility to set any position of the chosen motor, make its reset in positive and negative directions. Also any valid motor controller command can be send to it by the direct definition. The menu supporting the manual control is accessible trough the *NICSgate Debug Monitor* (Fig. 2.3) using the button Set Motors and is shown in the Fig. 2.1.

Figure 2.1: The menu for the Manual Control of the Motors

Obviously, the standard working mode, when the motors can be moved only to a certain well defined set of the positions should be temporarily disabled when the manual control of the motors is executed. All the sub-processors have slightly changed behaviour to permit the manual control. The motors processor pops-up the message widget with results of the command execution and with received response of the motor controller, when it was interrogated. The motors processor tries to maintain always the correct internal state of the motors, including their positions. Because of this the certain level of the attention is needed. While the user can think that he/she has issued a single command, when the issued command changes the motor position, the additional command, asking the resulting position is send by the motors processor. The execution time could be long (even minutes), thus the user can send another and another command. This can create some break in the internal protocol, because only finite (and small) number of

such, practically incorrect, commands can be supported effectively. It is not fatal, but may be misleading.

2.10 Acquisition Processor

The acquisition processor executes all acquisition tasks asked by the user. It is controlled by an `action` parameter, and by three state variables — `stop`, `state` and the global flag `xnir_res.acq_state`.

The parameter `action` can have the next values:

- **0** : start the acquisition procedure;
- **1** : continue the acquisition procedure;
- **-1** : stop the acquisition procedure;
- **-2** : break the acquisition procedure.

The variable `stop` is necessary because the acquisition process should be stopped not at any moment but at certain appropriate moments. Thus the asked action is not immediate but delayed. When `stop`=0 the acquisition process proceeds normally. When it is set to 1 it stops the task execution at nearest suitable moment.

The variable `state` follows the stages of the task execution and can have the next values:

- **0** : idle;
- **1** : motors processing to be finished;
- **2** : send command “set sampling” to transnix;
- **3** : send command “set data return” to transnix;
- **4** : telescope processing to be finished;
- **5** : clean memory;
- **6** : send the command to acquire one frame;
- **7** : the frame data are received;
- **8** : send command “set data idle” to transnix.

The direct influence on the acquisition processor work has also the global variable `xnir_res.acq_state`, which defines the type of the current acquisition task. It can have the next relevant values:

- **3** : mosaic;
- **4** : several groups;
- **5** : single group;
- **6** : free run;
- **8** : single frame;
- **9** : source-sky special mosaic.

Called with `action`=0, the acquisition processor gets all necessary information concerning the task definition, sets all relevant internal variables, clears the arrays for the data storage and activates the *motors* processor, which completes its part of the preparation.

When the *observation mode* environment is ready, two commands are sent to the transnix — set the needed sampling mode and to activate the sending of the data frame after its completion.

Then the telescope processor executes all necessary movements and receives the relevant information from the telescope. In fact, at the beginning there is no real movement, because initially the object should be pointed by the telescope operator. The movements are executed only in mosaic type acquisition, starting usually from the second mosaic position. But it depends on the information found in the corresponding mosaic file.

The telescope processor returns “READY” after completion of its part of work and the command for the acquisition of an image is sent to the transnix.

When the data frame is received from transnix, the measurements are continued until the group is finished. The resulting averaged image is saved into the fits file.

Then the acquisition processor makes the next group at the same telescope position, if asked. After that it changes the telescope position, activating again the telescope processor and so on, until the telescope processor returns “MOSAICFINISHED”.

At this point the current acquisition task is finished. The processor sends to the transnix the command “do not transmit data to me” and passes the “idle” state to the *transnix* processor.

During all this stages from many different points to *xnics* may be send the informative or even error messages. In the case of an error the acquisition processor breaks its task at the achieved stage.

2.11 Internal Data Handling

The scheme of the data handling in *nicsgate* is rather simple. There are two 2D arrays of the preset in resources dimensions, one long integer `map_s` and

another short integer `map`.

All internal manipulations are done in `map_s` thus preserving an accuracy of the image. The image resulting from the averaging through the group of the measurements is placed into `map` and then is written into the output fits file.

In a special case of source–sky measurements this type of data handling permits to have immediately in the memory the subtracted image.

The acquired data reside in a memory until the new acquisition task is started. It permits to analyze the image by an internal image viewer so long as the observer likes.

When the acquisition task creates multiple frames, the image control can be done by an external viewer.

2.12 Internal Image Viewer

The widget with internal viewer (Fig. 2.2) is described in [3]. Here we give some details how it is organized in a program.

Both arrays, where the data are stored, can be shown in the *Image* widget. The choice depends on the value of the global variable `xnir_res.image_source`. When it is set to 0 (default), the `map` array, i.e. the *production* image is visualized. If this variable is set to 1, the *current* average of stored till the current moment image in `map_s` is visualized. This variable can be set in the resource data base.

When the *image* widget is popped-up, the properties of the display are checked, and, if the model supports the pseudo-color colormaps, the possibility to change them is provided. The number of the color indexes is set to 128. If the display does not support the dynamic change of the colormap, the number of the color indexes is defined in a correspondence with the image depth used by the Xserver.

The chosen interval of the intensity from the data array is mapped into the color indexes accordingly to the chosen procedure. By default it is linear mapping, while the square and square root mappings can be used too.

The image array (1024x1024 pixels) is very large to view it in an original size even on a relatively large, `17"` displays. So we use the view-port widget to demonstrate an image. The default size of the view-port is 256x256 pixels. The scrollbars are provided to scroll the complete image. There is no problem to change the current view-port size by the standard X11 facilities dynamically.

The x,y coordinates of the shown image are chosen to correspond to the data presentation by the *SAOimage* tool. The image seen by this internal viewer and by the *SAOimage* are placed on the screen in the same manner.

Part of the image, the square centered at the point marked by the *left*

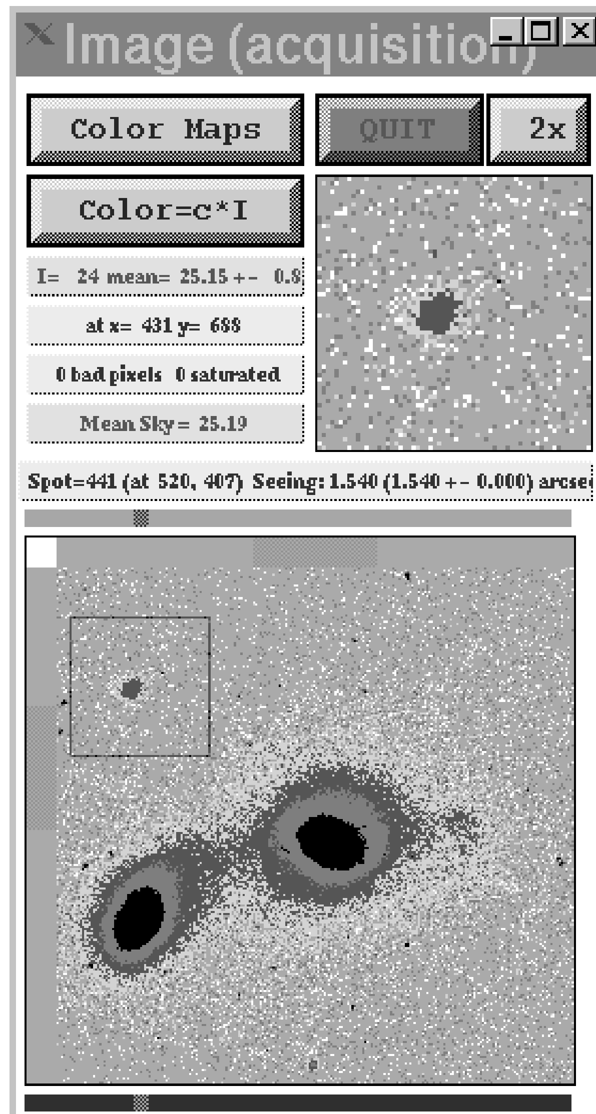


Figure 2.2: The Widget with the Acquired Image

mouse button click (or defined through the resources), is shown in "zoomed" mode. There is 128x128 zoomed image window. Thus the single pixel in the original image can be presented as the 2x2, 4x4 and 8x8 square of the same color in the zoomed image leaving it still representative. Correspondingly the original area reduces in the chain 64x64, 32x32, 16x16 pixels. The original area is shown in the core image by the rectangle.

We calculate the mean "sky" intensity averaging the means over several 10x10 areas far away from the source position (which is usually near the array center). The intensity interval mapping into the color indexes is "centered"

at this value. It means that when the minimum and maximum limits of the intensity interval to be mapped (they can be tuned independently one from the other), the first one can be not larger than a sky value, and the second one can not be lower than it. Thus a minimum 1 intensity interval near this value is granted for the visualization. The sky can be suppressed by the setting of the thumb of the *minimum* intensity scrollbar to its maximum (the most right position), and the "source" can be suppressed by the setting of the thumb of the *maximum* intensity scrollbar to its minimum (the most left position).

Choosing the scrollbars positions the user can evidence those features of the image he/she likes.

2.13 Debug Mode

To provide the instrument developers by the means to investigate the difficult situations and to tune the different parameters of the instrument, there is a special *debug mode* of the *nicsgate*.

Internally it is marked by the global variable **DEBUGMENU** set to 1, when this mode is in effect. The debug mode is activated when the *NICSgate Debug Monitor* (2.3) is started.

The user can boot into the NICS electronics the chosen binary image of the program, reset link to the electronics in case of the problems with its driver, interrogate or change the different internal parts of the transputer network, load the desirable waveform, set the different parameters of the *transnix*, change the behaviour of the *xnics* and *NICSgate* and control the motors using the sub-menu (Fig. 2.1).

More details on these options are written in [4].

2.14 Nicsgate Resources

Here we present all X11 resources of the *NICSgate* and the structure, where they are initialized, receiving the default values. In this structure one can see the correspondence between the resource name and the internal variable name. Some of these resources are the same as those of the *xnics*, other are unique for the *NICSgate*.

```
#define XniButtonVertDistance       "buttonVertDistance"
#define XnCButtonVertDistance       "ButtonVertDistance"

#define XniSi                      "singleIntegration"
#define XnCSi                      "SingleIntegration"
```

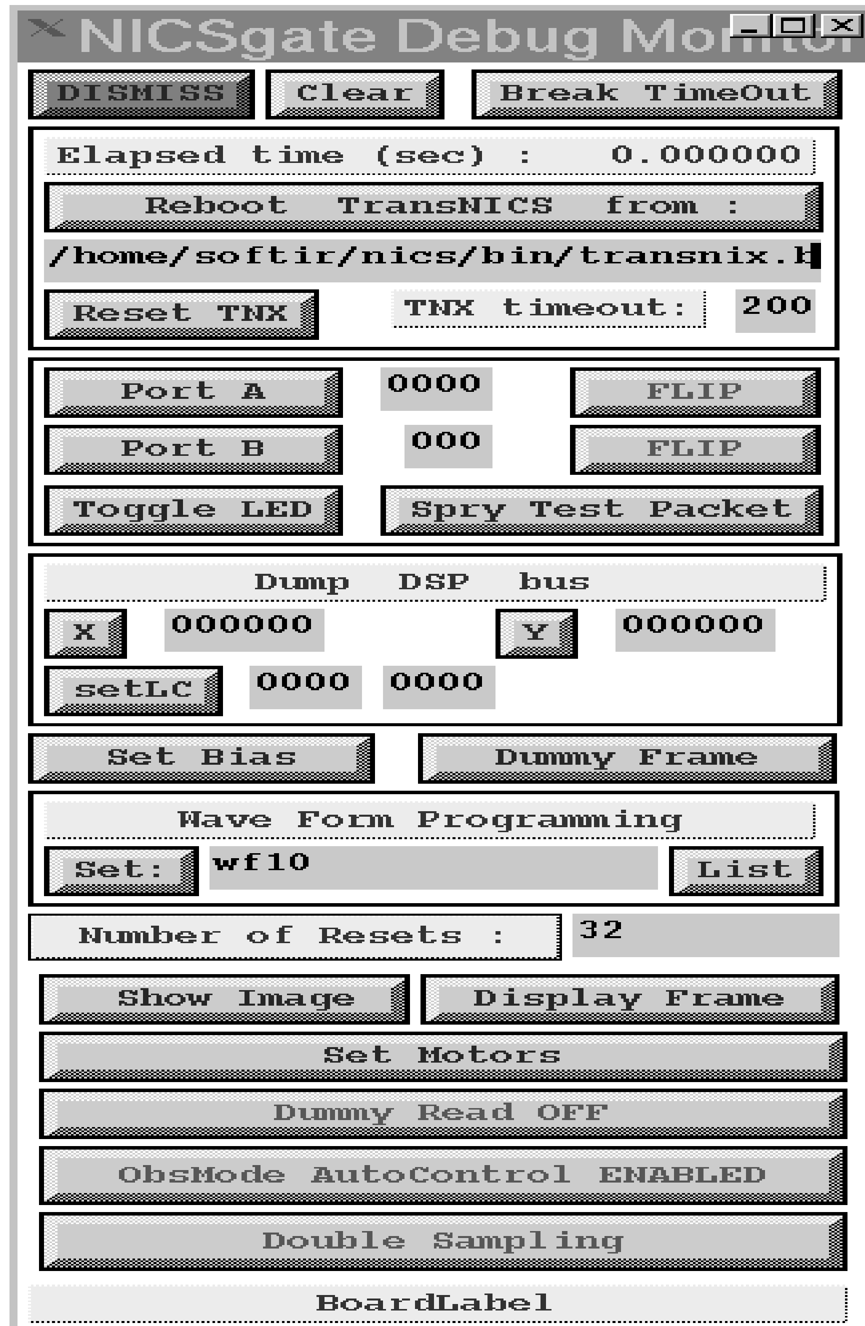


Figure 2.3: The NICSgate Debug Monitor Widget

```
#define XniCMap                        "colorMap"
#define XnCCMap                        "ColorMap"

#define XniMotAdr                      "motorsAddress"
```

```
#define XnCMotAdr                    "MotorsAddress"

#define XniImageDim                  "imageViewSize"
#define XnCImageDim                  "ImageViewSize"
#define XniImageMin                  "imageMin"
#define XnCImageMin                  "ImageMin"
#define XniImageMax                  "imageMax"
#define XnCImageMax                  "ImageMax"
#define XniImageSource               "imageSource"
#define XnCImageSource               "ImageSource"
#define XniImageTransf               "imageTransf"
#define XnCImageTransf               "ImageTransf"

#define XniSourceX                   "sourceX"
#define XnCSourceX                   "SourceX"
#define XniSourceY                   "sourceY"
#define XnCSourceY                   "SourceY"

#define XniFocStep                   "focusStep"
#define XnCFocStep                   "FocusStep"
#define XniFocLevel                  "focusLevel"
#define XnCFocLevel                  "FocusLevel"
#define XniFocSum                    "focusSum"
#define XnCFocSum                    "FocusSum"

#define XniPixelarcs                 "pixel2arcsf"
#define XnCPixelarcs                 "Pixel2arcsf"
#define XniPixelarcl                 "pixel2arclf"
#define XnCPixelarcl                 "Pixel2arclf"

#define XniIntBad                    "badPixelLevel"
#define XnCIntBad                    "BadPixelLevel"
#define XniIntSat                    "saturationLevel"
#define XnCIntSat                    "SaturationLevel"
#define XniSatWarn                   "saturationWarning"
#define XnCSatWarn                   "SaturationWarning"

#define XniScanTime                  "scanTime"
#define XnCScanTime                  "ScanTime"

#define XniTelTimeOut                "telTimeOut"
#define XnCTelTimeOut                "TelTimeOut"
#define XniTelDelay                  "telDelay"
```

```
#define XnCTelDelay                    "TelDelay"

#define XniTimerDelay                  "tnxDelay"
#define XnCTimerDelay                  "TnxDelay"

#define XniTimeOutW                    "timeOutW"
#define XnCTimeOutW                    "TimeOutW"

#define XniTimeOutR                    "timeOutR"
#define XnCTimeOutR                    "TimeOutR"

#define XniTimeOutA                    "timeOutA"
#define XnCTimeOutA                    "TimeOutA"

#define XniTimeOutE                    "timeOutE"
#define XnCTimeOutE                    "TimeOutE"

#define XniRootDir                     "rootDirectory"
#define XnCRootDir                     "RootDirectory"
#define XniBootFile                    "bootFile"
#define XnCBootFile                    "BootFile"

#define XniBiasQ1                      "biasQ1"
#define XnCBiasQ1                      "BiasQ1"
#define XniBiasQ2                      "biasQ2"
#define XnCBiasQ2                      "BiasQ2"
#define XniBiasQ3                      "biasQ3"
#define XnCBiasQ3                      "BiasQ3"
#define XniBiasQ4                      "biasQ4"
#define XnCBiasQ4                      "BiasQ4"

#define XniPolTen1                     "polTension1"
#define XnCPolTen1                     "PolTension1"
#define XniPolTen2                     "polTension2"
#define XnCPolTen2                     "PolTension2"

#define XniAreaX                       "areaSizeX"
#define XnCAreaX                       "AreaSizeX"
#define XniAreaY                       "areaSizeY"
#define XnCAreaY                       "AreaSizeY"
#define XniBoxX0                       "boxSizeX0"
#define XnCBoxX0                       "BoxSizeX0"
#define XniBoxY0                       "boxSizeY0"
```

```
#define XnCBoxY0                    "BoxSizeY0"
#define XniBoxX                     "boxSizeX"
#define XnCBoxX                     "BoxSizeX"
#define XniBoxY                     "boxSizeY"
#define XnCBoxY                     "BoxSizeY"

#define XniNresets                  "nresets"
#define XnCNresets                  "Nresets"
#define XniNcampions                "ncampions"
#define XnCNcampions                "Ncampions"


static XtResource  xnir_resources[] = {
  {
        XniButtonVertDistance,XnCButtonVertDistance,
        XtRInt,sizeof(int),XtOffsetOf(XnirAppData, button_vert_distance),
        XtRImmediate, (XtPointer) 10
  },
  {
        XniTimerDelay,XnCTimerDelay,
        XtRInt,sizeof(unsigned long),XtOffsetOf(XnirAppData, tnx_delay),
        XtRImmediate, (XtPointer) 200
  },
  {
        XniTimeOutW,XnCTimeOutW,
        XtRInt,sizeof(unsigned long),XtOffsetOf(XnirAppData, tnx_timeoutw),
        XtRImmediate, (XtPointer) 2000
  },
  {
        XniTimeOutR,XnCTimeOutR,
        XtRInt,sizeof(unsigned long),XtOffsetOf(XnirAppData, tnx_timeoutr),
        XtRImmediate, (XtPointer) 5000
  },
  {
        XniTimeOutA,XnCTimeOutA,
        XtRInt,sizeof(unsigned long),XtOffsetOf(XnirAppData, tnx_timeouta),
        XtRImmediate, (XtPointer) 2000
  },
  {
        XniTelTimeOut,XnCTelTimeOut,
        XtRInt,sizeof(unsigned int),XtOffsetOf(XnirAppData, tel_timeout),
        XtRImmediate, (XtPointer) 20
  },
```

```
{
      XniTelDelay,XnCTelDelay,
      XtRInt,sizeof(unsigned int),XtOffsetOf(XnirAppData, tel_delay),
      XtRImmediate, (XtPointer) 1
},
{
      XniScanTime,XnCScanTime,
      XtRInt,sizeof(unsigned long),XtOffsetOf(XnirAppData, scan_time),
      XtRImmediate, (XtPointer) 928
},
{
      XniCMap,XnCCMap,
      XtRString,sizeof(String),XtOffsetOf(XnirAppData, cmap),
      XtRImmediate, "gray scale"
},
{
      XniMotAdr,XnCMotAdr,
      XtRString,sizeof(String),XtOffsetOf(XnirAppData, MotorsAddress),
      XtRImmediate, "/dev/ttyS1"
},
{
      XniImageDim,XnCImageDim,
      XtRInt,sizeof(int),XtOffsetOf(XnirAppData, image_dim),
      XtRImmediate, (XtPointer) 256
},
{
      XniImageMin,XnCImageMin,
      XtRFloat,sizeof(float),XtOffsetOf(XnirAppData, imbar_min),
      XtRString, ".5"
},
{
      XniImageMax,XnCImageMax,
      XtRFloat,sizeof(float),XtOffsetOf(XnirAppData, imbar_max),
      XtRString, ".5"
},
{
      XniPixelarcl,XnCPixelarcl,
      XtRFloat,sizeof(float),XtOffsetOf(XnirAppData, pixelarclf),
      XtRString, ".25"
},
{
      XniPixelarcs,XnCPixelarcs,
      XtRFloat,sizeof(float),XtOffsetOf(XnirAppData, pixelarcsf),
```

```
        XtRString, ".13"
  },
  {
        XniFocLevel,XnCFocLevel,
        XtRFloat,sizeof(float),XtOffsetOf(XnirAppData, focus_level),
        XtRString, ".5"
  },
  {
        XniFocStep,XnCFocStep,
        XtRInt,sizeof(int),XtOffsetOf(XnirAppData, focus_step),
        XtRImmediate, (XtPointer) 10
  },
  {
        XniFocSum,XnCFocSum,
        XtRInt,sizeof(int),XtOffsetOf(XnirAppData, focus_sumNumb),
        XtRImmediate, (XtPointer) 4
  },
  {
        XniImageSource,XnCImageSource,
        XtRInt,sizeof(int),XtOffsetOf(XnirAppData, image_source),
        XtRImmediate, (XtPointer) 0
  },
  {
        XniSi,XnCSi,
        XtRInt,sizeof(int),XtOffsetOf(XnirAppData, singleframe),
        XtRImmediate, (XtPointer) 1
  },
  {
        XniSourceX,XnCSourceX,
        XtRInt,sizeof(int),XtOffsetOf(XnirAppData, source_x),
        XtRImmediate, (XtPointer) 480
  },
  {
        XniSourceY,XnCSourceY,
        XtRInt,sizeof(int),XtOffsetOf(XnirAppData, source_y),
        XtRImmediate, (XtPointer) 540
  },
  {
        XniImageTransf,XnCImageTransf,
        XtRInt,sizeof(int),XtOffsetOf(XnirAppData, image_transf),
        XtRImmediate, (XtPointer) 0
  },
  {
```

```
        XniIntBad,XnCIntBad,
        XtRInt,sizeof(long),XtOffsetOf(XnirAppData, int_bad),
        XtRImmediate, (XtPointer) 15000
  },
  {
        XniIntSat,XnCIntSat,
        XtRInt,sizeof(long),XtOffsetOf(XnirAppData, int_sat),
        XtRImmediate, (XtPointer) 12000
  },
  {
        XniSatWarn,XnCSatWarn,
        XtRInt,sizeof(int),XtOffsetOf(XnirAppData, sat_warn),
        XtRImmediate, (XtPointer) 257
  },
  {
        XniRootDir,XnCRootDir,
        XtRString,sizeof(String),XtOffsetOf(XnirAppData, root_dir),
        XtRImmediate, "/home/nics/nics"
  },
  {
        XniBootFile,XnCBootFile,
        XtRString,sizeof(String),XtOffsetOf(XnirAppData, tnx_boot),
        XtRImmediate, "/home/nics/nics/bin/transnix.btl"
  },
  {
        XniBoxX0,XnCBoxX0,
        XtRInt,sizeof(int),XtOffsetOf(XnirAppData, box_sizex0),
        XtRImmediate, (XtPointer) 0
  },
  {
        XniBoxX,XnCBoxX,
        XtRInt,sizeof(int),XtOffsetOf(XnirAppData, box_sizex),
        XtRImmediate, (XtPointer) 512
  },
  {
        XniBoxY0,XnCBoxY0,
        XtRInt,sizeof(int),XtOffsetOf(XnirAppData, box_sizey0),
        XtRImmediate, (XtPointer) 0
  },
  {
        XniBoxY,XnCBoxY,
        XtRInt,sizeof(int),XtOffsetOf(XnirAppData, box_sizey),
        XtRImmediate, (XtPointer) 512
```

```
},
{
      XniAreaX,XnCAreaX,
      XtRInt,sizeof(int),XtOffsetOf(XnirAppData, area_sizex),
      XtRImmediate, (XtPointer) 512
},
{
      XniAreaY,XnCAreaY,
      XtRInt,sizeof(int),XtOffsetOf(XnirAppData, area_sizey),
      XtRImmediate, (XtPointer) 512
},
{
      XniBiasQ1,XnCBiasQ1,
      XtRInt,sizeof(int),XtOffsetOf(XnirAppData, bias_q1),
      XtRImmediate, (XtPointer) 3590
},
{
      XniBiasQ2,XnCBiasQ2,
      XtRInt,sizeof(int),XtOffsetOf(XnirAppData, bias_q2),
      XtRImmediate, (XtPointer) 4095
},
{
      XniBiasQ3,XnCBiasQ3,
      XtRInt,sizeof(int),XtOffsetOf(XnirAppData, bias_q3),
      XtRImmediate, (XtPointer) 2850
},
{
      XniBiasQ4,XnCBiasQ4,
      XtRInt,sizeof(int),XtOffsetOf(XnirAppData, bias_q4),
      XtRImmediate, (XtPointer) 1665
},
{
      XniPolTen1,XnCPolTen1,
      XtRInt,sizeof(int),XtOffsetOf(XnirAppData, pol_tension1),
      XtRImmediate, (XtPointer) 3500
},
{
      XniPolTen2,XnCPolTen2,
      XtRInt,sizeof(int),XtOffsetOf(XnirAppData, pol_tension2),
      XtRImmediate, (XtPointer) 2050
},
{
      XniNresets,XnCNresets,
```

```
        XtRInt,sizeof(int),XtOffsetOf(XnirAppData, nresets),
        XtRImmediate, (XtPointer) 32
  },
  {
        XniNcampions,XnCNcampions,
        XtRInt,sizeof(int),XtOffsetOf(XnirAppData, ncampions),
        XtRImmediate, (XtPointer) 2
  },
};
```

Bibliography

[1] Lisi, F., Baffa, C., Gennari, S., Oliva, E., 1999, "**Nics**, the near IR imager/spectrograph of the TNG", International Meeting on Astronomical Technologies, S.Agata, Memorie della Societá Astronomica Italiana, in press.

[2] Comoretto, G., Baffa, C., Gavryusev, V., Lisi, F., Sozzi, M., 1999, "The Data Acquisition System for Nics — Hardware Solutions", International Meeting on Astronomical Technologies, S.Agata, Memorie della Societá Astronomica Italiana, in press.

[3] Gavryusev, V., 2000, "XNICS. User's Guide", Arcetri Astrophysical Preprint, n.8/2000.

[4] Gavryusev, V., 2000, "XNICS. Software Maintenance Guide", Arcetri Astrophysical Preprint, n.10/2000.

[5] Pence W.D., 1997, "CFITSIO User's Guide. An Interface to FITS Format Files for C Programmesrs", version 1.2, HEASARC, code 662, Goddard Space Flight Center, Greenbelt, MD 20771, USA

[6] Baffa, C., 2000, "Il Protocollo Transnix-Xnics", Arcetri Technical Report, n.2/2000.

[7] OEM Series Software Reference Guide, 1993, Compumotor Division of Parker Hannifin Corporation, p/n 88-013785-01 A.