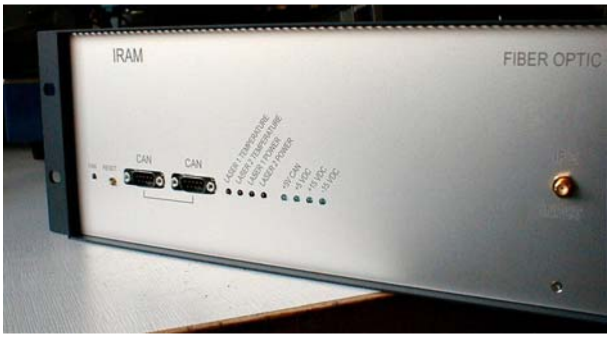
IRAM-COMP-034

Revision: 0
2006-09-15

Contact Author

Institut de RadioAstronomie Millimétrique

Fiber Optic
Software Documentation



Owner Sebastien Blanchet

Keywords: fiber optic, software

Approved by:	Date:	Signature:
A.Perrigouard	Sept 2006	

Change Record

REVISION	DATE	AUTHOR	SECTION/PAGE AFFECTED	REMARKS

Content

1 Introduction 4

2 Presentation 4

2.1 FO_TX 4
2.2 FO_RX 4
2.3 Device location 5

3 Requirements 5

3.1 Hardware requirement 5
3.2 Software requirements 5
3.3 Network environment 6

4 General instructions 6

4.1 Installation 6
4.2 API documentation 7

5 CanManager and drivers 7

6 Software architecture 8

6.1 Files and directories 8
6.2 CAN Threads 8
6.3 CAN device 9

7 Server software 10

7.1 foRxServer.py 10
7.1.1 Data structures 10
7.1.2 Protocol 11
7.1.3 Performances 12
7.1.4 Implementation 12

8 User software 12

8.1 foRxGui.py 12
8.2 foTxGui.py 15
8.3 foRxClientTest 18
8.3.1 Compilation on HP-UX 18
8.3.2 Compilation on LINUX 19
8.3.3 Usage 19
8.3.4 Libraries 21

9 Maintenance command 21

10 Troubleshooting 21

10.1 Modules 21
10.2 Processes 21

11 Simulation....................22

11.1 RX Simulator....................22
11.2 TX Simulator....................22

1 Introduction

In October 2006, IRAM has begun to install a new transmission system based on fiber optic to connect the new generation receiver (in each antenna cabin) to the correlator (in the computer room). This document is the documentation reference for fiber optic software: installation, technical documentation, daily usage and troubleshooting.

You need also the hardware documentation of the Fiber Optic Processor from Philippe Chavatte: *FO_TX.pdf* and *FO_RX.pdf*

2 Presentation

2 devices composes the fiber optic transmission system:

- An optical tranceiver, named FO_TX
- An optical receiver, named FO_RX

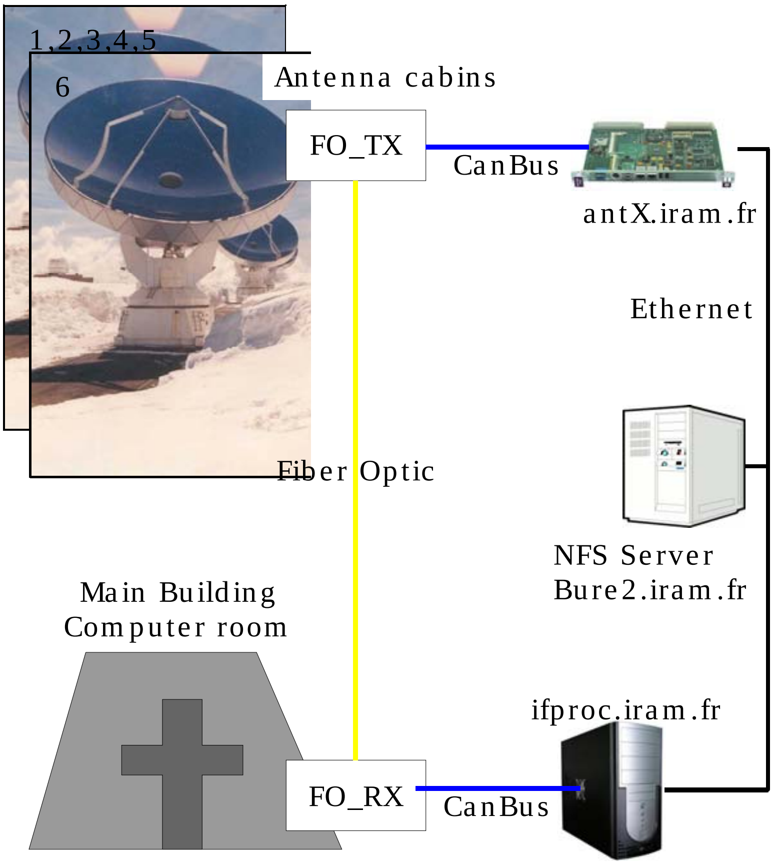
2.1 FO_TX

There are 6 FO_TX, one per antenna. They are located in the antenna cabin.
These devices are autonomous, but to allow remote maintenance, a monitoring software can be run from antX.iram.fr

2.2 FO_RX

There is 1 FO_RX. It is located in the computer room. All the fiber are connected to FO_RX
This device is initialized by a computer via a CAN bus. The controlling software runs on *ifproc.iram.fr*. *Ifproc.iram.fr* runs also the IF Processor software. (see the *IF Processor Software Documentation)*

2.3 Device location



3 Requirements

3.1 Hardware requirement

To run the Fiber Optic monitoring software, you need at least:

- A standard PC, CPU 1GHz, RAM 512 Mo

In this situation, you can run the software in simulation mode.

Although, for a real usage you need also:

- A PXE-compatible network card. For the moment only the 3com 3C905C and the Intel Pro/100+ are supported.
- A TPMC816 CAN controller from Tews Technologies. This card uses the PMC format so a PCI carrier is required.

3.2 Software requirements

The software has been developed for Linux Fedora Core 4, 32 bits edition.
The following packages are required:

- python 2.4
- Tkinter

Note: I have found a bug in Tkinter files.

- A trivial bug in ToolTip.py. I have submitted a patch to http://python.org/dev/, the modification will be include in the Python official release.

3.3 Network environment

For the real exploitation, the Fiber Optic software will run on a disk-less computer. So an NFS server is required to export the filesystem. For better security, the main part of the filesystem will be exported as read-only.

4 General instructions

4.1 Installation

The software runs on *ifproc.iram.*fr. This computer is located in the computer room in the main building. This machine runs also the *IF processor interfacing software.*

Here after, the different installation steps:
Get the sources:

```
$export BUILDDIR=/home/blanchet/build
$ export CVSROOT=:pserver:blanchet@netsrv1.iram.fr:/CVS/PdB
$ mkdir  $BUILDDIR
$ cd $BUILDDIR
$ cvs login
Logging in to :pserver:blanchet@netsrv1.iram.fr:2401/CVS/PdB
CVS password:
$ cvs co –r FC4-branch LINUX
```

Install the drivers:
Warning: for the following section, you need the write privilege on the whole filesystem. Therefore it will fail, it the NFS root is exported in read-only mode.

```
$ cd $BUILDDIR/LINUX/drivers/tpmc816
$ make
$ su –c 'make install'
```

Install the controlling software:

```
$ cd $BUILDDIR /LINUX/fiberOptic
$ su –c 'make install'
```

Install the initialization script:

```
$ cd $BUILDDIR /LINUX/
$ su –c 'make install'
```

4.2 API documentation

The API documentation can be automatically extracted with pydoc. There are two possibilities:

- The dynamic documentation with the pydoc http embedded server
- The static documentation with static HTML files

The quickest and most convenient method is the interactive documentation

Generate the dynamic documentation (the quickest method)

```
$ make doc_dynamic
```

Then open http://localhost:8080 to browse the documentation. The HTML files are generated on-the-fly by pydoc.

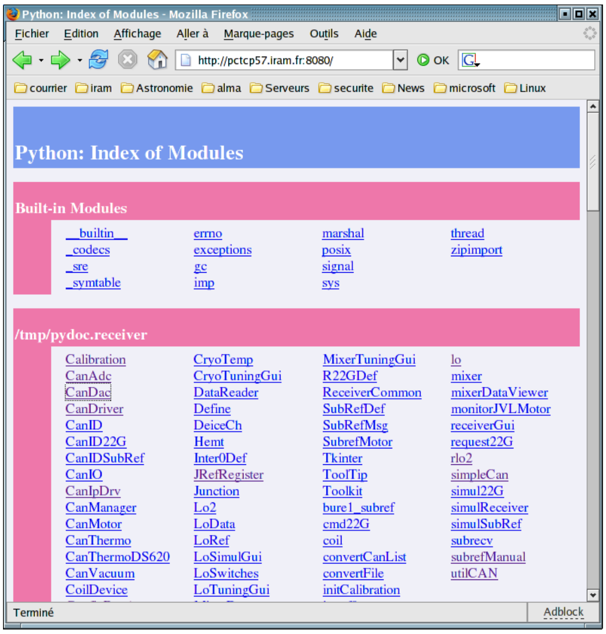


Figure 1: Index of Modules, generated by pydoc

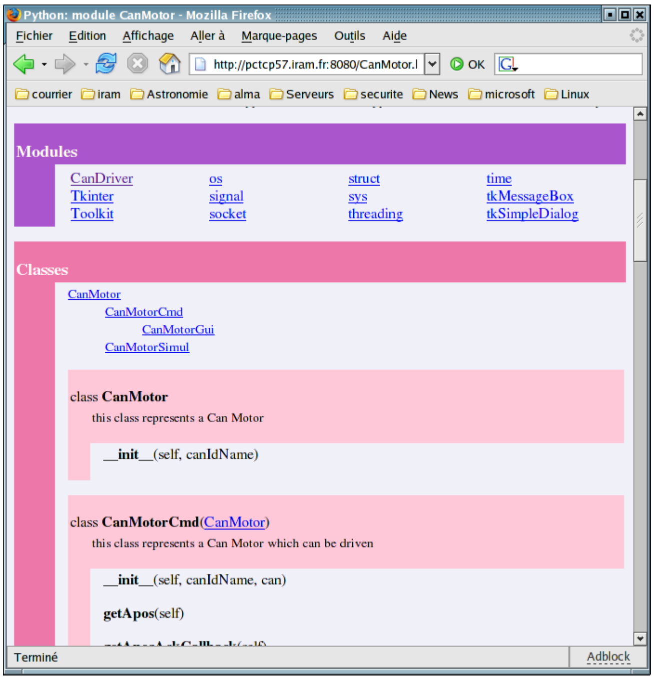


Figure 2: Details of a module

Generate the static documentation (the slowest method)

```
$ make doc_static
```

then the */tmp/pydoc.ifProcessor* directory now contents the API documentation, in HTML format. This time, pydoc writes all the HTML files to the *doc* directory.

5 CanManager and drivers

To drive the CAN interface, we reuse the *CanManager* which has been originally developed for the receiver software. It is basically a python server + a CAN over UDP protocol.

Some advantages of this solution:

- *Simple and fast*: it is just a small python program
- *Transparent for applications*: the clients do not know that they use CanManager. They only use *read* and *write* functions. Therefore the client software is independent from CanManager

- *Universal*: compatible with every program languages (current bindings exists for Python and C)
- *Flexible*: it allows to plug multiple virtual CAN device, like simulators or traffic analyzers on the CAN bus.
- *Validated*: it runs in lab since April 2005, it runs in Bure 24h/24h since November 2005
- *Powerful acknowledgement management*: user-defined ACK and NACK callback function are available for every CAN message.
- *Advanced error management*: a client cannot crash the CanManager. If a client crashes, just restart it, if the CanManager crashes, just restart it, and the client reconnect automatically.

6 Software architecture

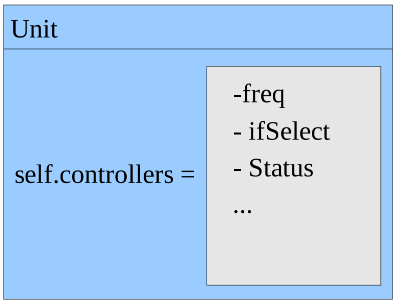
6.1 Files and directories

The CVSROOT for the fiber optic files is /CVS/PdB/Linux/fiberOptic
This is a description of the directories:

Name	Description
python	Software to monitor FO_RX and FO_TX
src	Client software example for HP-UX and LINUX (foRxClientTest)
include	Headers files

6.2 CAN Threads

Fundamentally, all the IF Processors programs are similar: they hold a "*Unit*" object with a "*controllers*" attribute, in which you put all the CAN objects you want to manage:



For example to build the main windows of ifpGui, you select the following components:

```
self.controllers = {
    'freq':        CanIfpFreqGui(…)    ,
    'ifSelect':    CanIfpIfSelectGui(…),
    'psuVoltage':CanIfpPsuGui(…),
    'status':      CanIfpStatusGui(…),
    'init':        CanIfpInitGui(…)
}
```

Then the programs will execute 2 additional threads on *self.controllers*.

ListenCan
For every CAN message, this thread calls the *unit.listenCan* method to dispatches the CAN message.

```
    def listenCan(self, msg):
```

```
        for k,controller in self.controllers.iteritems():
            controller.listenCan(msg)
```

RefreshInternalValues
Every second, this thread calls the *unit.refreshInternalValues* method, then the CAN component will send CAN messages to request the receiver actual values. *ListenCan* will receive the answers.

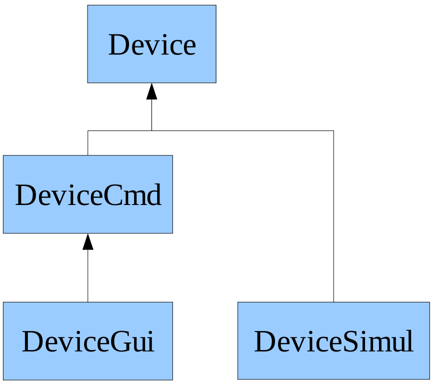
```
    def refreshInternalValues(self):
        for k,controller in self.controllers.iteritems():
            controller.refreshInternalValues()
```

Note: Contrary to the receiver software, *RefreshInternalValues* is disabled by default, because the monitoring is done only on demand.

Flexibility is the main advantage of this architecture: CAN components can be added/removed with a few code modification.

6.3 CAN device

In this section we will show a preview of CAN device. All the CAN device are represented with the same class hierarchy



Device
This class defines mainly the device's CAN identifiers.

DeviceCmd
This class extends *Device* to be used in a command line program.

DeviceGui
This class provides the methods to drive the CAN device from a graphical interface. It adds widgets to extend *DeviceCmd*.

DeviceSimul
This class provides a device simulator. It can be included in a simulator graphical interface.

This a list of the actual CAN devices:

Name	Description
CanFoFan	Fan monitoring
CanFoInit	Init and reset command
CanFoRxLaser	Laser RX monitoring
CanFoRxPsu	Power supply RX monitoring

CanFoRxRecord	Laser RX history EEPROM downloading
CanFoRxSelect	Laser RX output selection (noise/receiver)
CanFoStatus	Device status (module ID, firmware version, etc.)
CanFoTime	Time elapsed since power on
CanFoTxLaser	Laser TX monitoring
CanFoTxPsu	Power supply TX monitoring
CanFoTxRecord	Laser TX history EEPROM downloading
CanFoVoltage	Voltage monitoring (mother class for CanFoRxPsu, CanFoTxPsu and CanFoTxLaser

7 Server software

Unlike FO_TX, FO_RX need some commands from the interferometer main computer. (FO_TX support only monitoring, FO_RX supports monitoring and remote commands)
All programs described in this section must be always active; otherwise the FO_RX does not work.
Note: they are executed automatically when the computer starts.

7.1 foRxServer.py

This program receives commands from *bure1.iram.fr*, (the interferometer main computer), and then generate the needed CAN commands to the FO_RX device.
Syntax:
foRxServer.py

7.1.1 Data structures

To exchange data with bure1, we use two special data structures: `FoRxRequest_t` and `FoRxStatus_t`. This data structures are described in *foRxClient.h*

7.1.1.1 FoRxRequest_t

This data structure is received from bure1

```
typedef enum{
    FORX_GET,
    FORX_SELECT_OUTPUT,
    FORX_GET_TEMPERATURE,
    FORX_INIT_IO,
    FORX_RESET_CPU,
    FORX_ACTION_LAST_INDEX,          /* use for iteration over enum */
} FoRxAction_t;

typedef struct {
    FoRxAction_t action    /* Enum: FORX_GET, … */;
    Int outputSelect ; /* 0 == Noise Source Output, 1 == Receiver Output */
} FoRxCommand_t;
```

7.1.1.2 FoRxStatus_t

This data structure is sent back to bure1

```
enum {
      LASER_H_IDX,
      LASER_V_IDX,
      MAX_LASER_IDX
};

#define MAX_PHOTONIC_VOLTAGE 6

typedef struct  {
      float temperature;      /*degree Celcius, internal temperature*/
      time_t temperatureTimeStamp;       /*date when 'temperature'  was
                                          unit: in seconds since Epoch */
      float laser[MAX_LASER_IDX][MAX_PHOTONIC_VOLTAGE];   /* Volt,
photonic voltage */
      float PS_CAN_5V;           /* Volt, Can Bus 5V power supply voltage */
      float PS_SWITCH_5V;      /* Volt,Switches 5V power supply voltage */
      float PS12_0D;          /* Volt, 12.0V digital power supply voltage */
      float PS15_0D;          /* Volt, 15.0V digital power supply voltage */

      int outputSelected    /* 0=> Noise source , 1=> receiver output */
      int canError;             /* number of CAN error on the bus */
} FoRxStatus_t;
```

7.1.2 Protocol

This section explains the network protocol between bure1 and foRxServer.py

foRxServer.py receives on port TCP:1082 a foRxRequest_t buffer. For this section, we assume that *foRxServer.py* has stored this request in the *req* variable.

According to the *action* value, *foRxServer.py* process the request.

If *action* is equal to:
- **FORX_GET**: read the FO_RX device status
- **FORX_SELECT_OUTPUT**: it sets the FO_RX output according to *req.outputSelect* value.
- **FORX_GET_TEMPERATURE**: gets the internal temperature This command is very CPU consuming for the FO_RX processor microcontroller, so it **must never** be requested during the observations. When *temperature* is updated, *temperatureTimeStamp* is set with the current time (in seconds since Epoch)
- **FORX_INIT_IO**: it initialises the I/O with their default values.
- **FORX_RESET_CPU:** it resets the FO_RX CPU

When foRxServer.py has completed the request processing, it returns a foRxStatus_t buffer to bure1.

foRxServer.py issues the CAN monitoring messages immediately after the CAN control messages, but it does not wait that the monitoring value are available. Instead it sends the current Status values. **Therefore the status buffer reflects the FO_RX processor status at the previous request time.**

Note 1: If you send an unknown command, it is discarded and no status buffer are sent back.

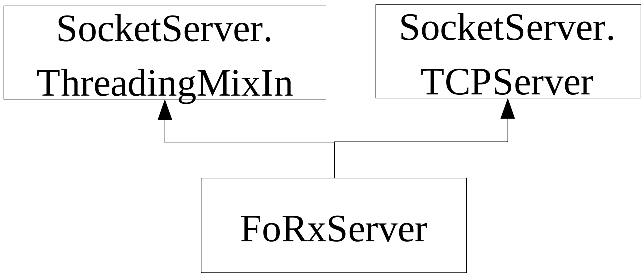
Note 2: Temperature reading is very time consuming, so a *FORX_GET_TEMPERATURE* request is required to monitor the temperature, otherwise we do not update it. Therefore *temperatureTimeStamp* is updated when *'temperature'* is measured to know the *temperature* field age.

7.1.3 Performances

The total processing time (request processing + status sending back) is slightly lower than 50 milliseconds. So we have very comfortable performances.

7.1.4 Implementation

foRxServer.py is implemented in a very simple and "elegant" way:
It derivates two standard classes to create servers: *SocketServer.ThreadingMixIn, SocketServer.TCPServer*
It means that we want a TCPServer which creates a new thread for each request. In fact we just have to define the request handler.



Like all server CAN programs, *foRxServer.py* has a *listenCan* thread, and a *refreshInternalValues* thread.

8 User software

All the software are installed in */home/introot/fiberOptic*
The python software are installed in */home/introot/fiberOptic/python*

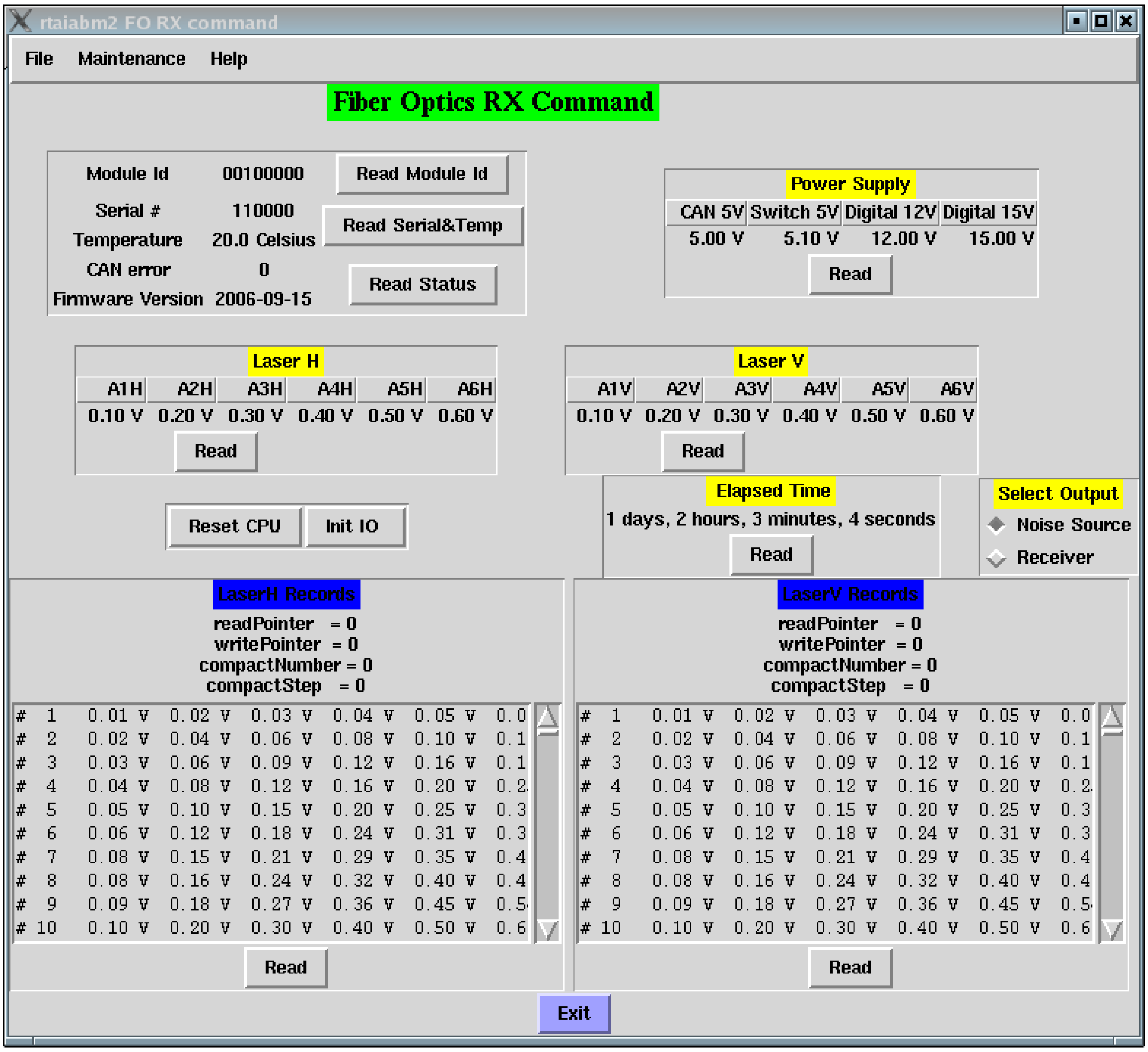
8.1 foRxGui.py

foRxGui.py is a basic graphical software to send command to the FO_RX device.

Syntax:
foRxGui.py

Procedure to log in ifproc.iram.fr, and execute the program

```
$ ssh –Y backend@ifproc.iram.fr
backend@ifproc.iram.fr password:
$ foRxGui.py
```



With this software, you can

- read all the FO_RX internal values. If the value is out of range, the widget background becomes red.
- select the wanted output (Noise Source or Receiver)
- reset the FO_RX microcontroller
- download the Laser EEPROM data, which hold the history.

You can save the current state in a file.
Open menu "*File | Save Report*" and specify a file name.

Example:

```
#######################################################################
# Fiber Optic RX - Automatic Report
# Date: 2006-09-15T16:13:44.178774
# Generated on rtaiabm2
#######################################################################


#######################################################################
#######################################################################
```

```
Status
moduleId = 100000
serial = 110000
Firmware Version = 2006-09-15
temperature = 20.00 deg Celcius
Can Error = 0

######################################################################
Selection:
Output Selected = Noise Source

######################################################################
Elapsed Time: 1 days, 2 hours, 3 minutes, 4 seconds

######################################################################
Power Supply
CAN 5V = 5.00
Switch 5V = 5.10
Digital 12V = 12.00
Digital 15V = 15.00

######################################################################
Laser H
A1H = 0.10
[...]
A6H = 0.60

######################################################################
Laser V
A1V = 0.10
[...]
A6V = 0.60

######################################################################
Name: forx_downloadLaserH
readPointer   = 0
writePointer  = 0
compactNumber = 0
compactStep   = 0
#  1   0.01 V  0.02 V  0.03 V  0.04 V  0.05 V  0.07 V
#  2   0.02 V  0.04 V  0.06 V  0.08 V  0.10 V  0.13 V
[...]
#255   0.00 V  2.55 V  2.53 V  2.53 V  2.51 V  2.50 V


######################################################################
Name: forx_downloadLaserV
readPointer   = 0
writePointer  = 0
compactNumber = 0
compactStep   = 0
#  1   0.01 V  0.02 V  0.03 V  0.04 V  0.05 V  0.07 V
#  2   0.02 V  0.04 V  0.06 V  0.08 V  0.10 V  0.13 V
[...]
#255   0.00 V  2.55 V  2.53 V  2.53 V  2.51 V  2.50 V
```

Note 1:
By default, the software runs in manual mode, i.e. you must click on the *Read* buttons to get the FO_RX values. Nevertheless, the automatic refreshing can be enabled by opening the menu *"Maintenance | refresh internal values"*

Note 2:
You have to click two times on *Read,* to display the Laser Records
The reason is simple: we cannot know before how many data will be sent by the device. It depends on the EEPROM content, so it is difficult to know when the widget should be updated.

8.2 foTxGui.py

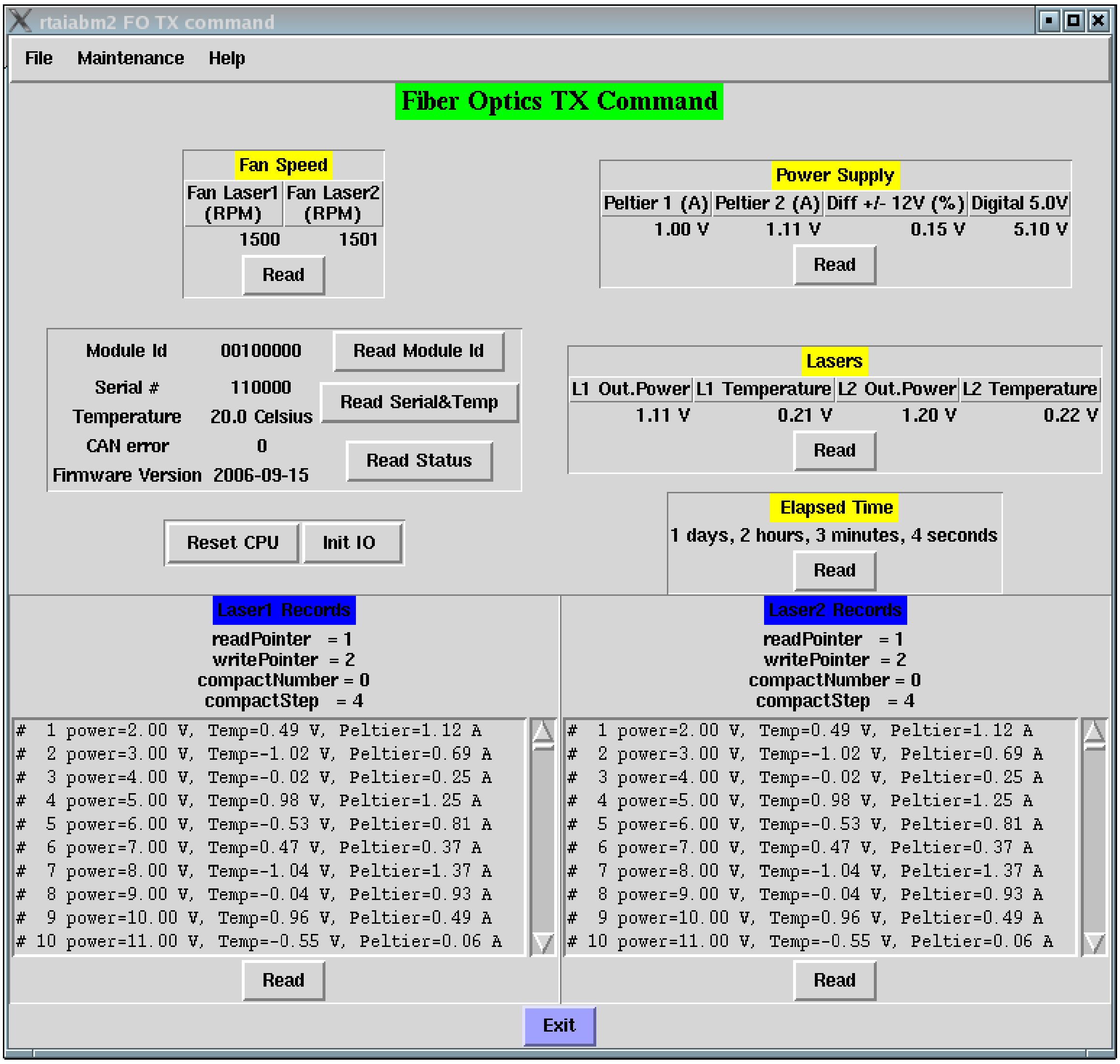
foTxGui.py is a basic graphical software to send command to the FO_TX device.

Syntax:
foTxGui.py

The software run on antX.iram.fr (X=[1..6]

Procedure to log in antX.iram.fr, and execute the program

> *$ ssh –Y backend@ant62.iram.fr*
> *backend@ant62.iram.fr password:*
> *$ foTxGui.py*



With this software, you can

- read all the FO_TX internal values. If the value is out of range, the widget background becomes red.
- reset the FO_TX microcontroller
- download the Laser EEPROM data, which hold the history.

You can save the current state in a file.
Open menu “*File* | *Save Report*” and specify a file name.

Example:

```
########################################################################
# Fiber Optic TX - Automatic Report
# Date: 2006-09-15T16:50:43.117403
# Generated on rtaiabm2
########################################################################



########################################################################
########################################################################
```

```
Status
moduleId = 100000
serial = 110000
Firmware Version = 2006-09-15
temperature = 20.00 deg Celcius
Can Error = 0

######################################################################
Elapsed Time: 1 days, 2 hours, 3 minutes, 4 seconds

######################################################################
Power Supply
Peltier 1 (A) = 1.00
Peltier 2 (A) = 1.11
Diff +/- 12V (%) = 0.15
Digital 5.0V = 5.10

######################################################################
Fan:
Fan Laser 1 (RPM): 1500.00
Fan Laser 2 (RPM): 1501.00

######################################################################
Lasers
L1 Out.Power = 1.11
L1 Temperature = 0.21
L2 Out.Power = 1.20
L2 Temperature = 0.22

######################################################################
Name: fotx_downloadLaser1
readPointer = 1
writePointer = 2
compactNumber = 0
compactStep = 4

#  1 power=2.00 V, Temp=0.49 V, Peltier=1.12 A
#  2 power=3.00 V, Temp=-1.02 V, Peltier=0.69 A
[...]
#255 power=96.16 V, Temp=1.00 V, Peltier=0.56 A



######################################################################
Name: fotx_downloadLaser2
readPointer = 1
writePointer = 2
compactNumber = 0
compactStep = 4

#  1 power=2.00 V, Temp=0.49 V, Peltier=1.12 A
#  2 power=3.00 V, Temp=-1.02 V, Peltier=0.69 A
 [...]
#255 power=96.16 V, Temp=1.00 V, Peltier=0.56 A
```

Note 1:
By default, the software runs in manual mode, i.e. you must click on the *Read* buttons to get the FO_TX values. Nevertheless, the automatic refreshing can be enabled by opening the menu *"Maintenance | refresh internal values"*

Note 2:
You have to click two times on *Read*, to display the Laser Records
The reason is simple: we cannot know before how many data will be sent by the device. It depends on the EEPROM content, so it is difficult to know when the widget should be updated.

8.3 foRxClientTest

foRxClientTest is a small C program to test foRx*Server.py*. It runs either on HP-UX or on LINUX. This program has also been designed to illustrate how to dialog with foRx*Server.py*. It is just an example !.
With this software you can also benchmark *foRxServer.py*, because the processing time is displayed.

8.3.1 Compilation on HP-UX

The software is already installed on iraux4.iram.fr
The software and all the needed components to rebuild it are available in *iraux4:/urs/local/fiberOptic*. The executable name is *foRxClientTest..HPUX*

To rebuild it, copy all the files in */usr/local/fiberOptic* in your home directory

```
$ cp –r /usr/local/fiberOptic ~/
$ cd ~/fiberOptic
$ gmake –f Makefile.hpux.example
```

For your information, this is the installation step to build the software on HP-UX (iraux4.iram.fr) from the CVS repository.

Get the sources:

```
$export BUILDDIR=/home/blanchet/build
$ export CVSROOT=:pserver:blanchet@netsrv1.iram.fr:/CVS/PdB
$ mkdir  $BUILDDIR
$ cd $BUILDDIR
$ cvs login
Logging in to :pserver:blanchet@netsrv1.iram.fr:2401/CVS/PdB
CVS password:
$ cvs co –r FC4-branch LINUX
```

Build and install the software:

```
$ cd $BUILDIR/LINUX/utils
$ gmake –f  Makefile.hpux
$ cd ../fiberOptic/src
```

```
$ gmake –f Makefile.hpux
$ su
Password:
# /usr/local/gnu/bin/gmake –f Makefile.hpux install
```

8.3.2 Compilation on LINUX

Obvisouly, *foRxClientTest* has been written to compile also on Linux

This is the installation step on LINUX, to build the software from the CVS repository.

Get the sources:

```
$export BUILDDIR=/home/blanchet/build
$ export CVSROOT=:pserver:blanchet@netsrv1.iram.fr:/CVS/PdB
$ mkdir  $BUILDDIR
$ cd $BUILDDIR
$ cvs login
Logging in to :pserver:blanchet@netsrv1.iram.fr:2401/CVS/PdB
CVS password:
$ cvs co –r FC4-branch LINUX
```

Build and install the software:

```
$ cd $BUILDIR/LINUX/utils
$ make –f  Makefile
$ cd ../fiberOptic/src
$ make –f Makefile
```

The executable file is *../bin/foRxClientTest*

8.3.3 Usage

Syntax:
foRxClientTest [serverHostname]

Note: on iraux4, you can found foRxClientTest in /usr/local/fiberOptic/foRxClientTest.HPUX

If *serverHostname* is not specified, the program tries to connect to *ifp-gnb.iram.fr*, (the lab computer for the IF processor). s*erverHostname* is typically used to connect to another computer running an IF Processor Simulator instance).

Use Control-C to quit the program. It is interactive software, so you have to redirect the standard input if you want to use a file instead of using the keyboard to enter command.

Command (not case sensitive)	**Description**
?	Display online help
#	Comment (the other characters are ignored.
D	Display the command buffer (but do not send it)
U *n*	Define command buffer value, n = 0 => noise source ; n = 1 => receiver output

S *cmd*	Send the command buffer with action=*cmd* The possible values for command are: GET, SELECT_OUTPUT, GET_TEMPERATURE, INIT_IO, RESET_CPU
Q	Exit the software.

For example:
$ /usr/local/fiberOptic/foRxClientTest.HPUX   ifp-gnb

```
$Id: foRxClientTest.c,v 1.1.2.1 2006/09/12 12:19:24 blanchet Exp $
Connecting to ifp-gnb:1082 ... OK
Use '?' + Enter to display the Help
foRxClient > ?
foRxClientTest Help Commands:
Syntax:
      cmd args

  ?   - display this help
  #   - comment
  D  - display the command buffer, without sending it
  Q  - quit
  U n  -
     where 'n==0' to select the NOISE source output
     where 'n==1'  to select the receiver output
  S cmd  - send buffer command cmd= GET | SELECT_OUTPUT |
GET_TEMPERATURE | INIT_IO | RESET_CPU
foRxClient > U 1
foRxClient > D
Command Buffer: action=GET ; output = RECEIVER

foRxClient > S SELECT_OUTPUT
Command Buffer: action=SELECT_OUTPUT ; output = RECEIVER

Status=
temperature = 0.00
temperatureTimeStamp = Thu Jan  1 01:00:00 1970

laserH= 0.10 0.20 0.30 0.40 0.50 0.60
laserV= 0.10 0.20 0.30 0.40 0.50 0.60
PS_CAN_5V = 5.00 , PS_SWITCH_5V = 5.10
PS12_0D   = 12.00 , PS15_0D      = 15.00
canError = 0, outputSelected = 0
time delay: 288.0 ms
foRxClient > Q
```

8.3.4 Libraries

If you want to write your own software to dialog with foRxServer.py, you need only two files:

- foRxClient.o: to handle easily the command and status buffer
- TcpClient.o to create and use the TCP socket

A minimal Makefile is available on iraux4 in /usr/local/fiberOptic/Makefile.hpux.example.

9 Maintenance command

This section describes the maintenance commands. In normal operation mode, you need not these programs.

10 Troubleshooting

This section explains how to solve some trouble on the *IF processor interfacing PC,* for example, *ifp-gnb.iram.fr* in the backend lab.

10.1 Modules

You can check if the device drivers are loaded:

```
$ /sbin/lsmod | grep tpmc
Module                  Size  Used by
tpmc816drv             10704  4
```

If this module is missing, it means that there was a problem during the computer initialization. You should restart the computer with the 'reboot' command.

10.2 Processes

You can list the current processes and check that all the required processes are present:

Check for foRxServer.py,

```
$ ps –edf
```

Look for the following processes:

```
oper      3123     1  0 14:43 ?        00:00:00 python2 ./CanManager.py /dev/tpmc816_1 2500
root      3165     1  0 14:44 ?        00:00:00 su oper -c python/foRxServer.py
oper      3168  3165  0 14:44 ?        00:00:00 python2 python/foRxServer.py
```

If one of these processes is missing, it means that it has crashed. These programs must never crash, otherwise there is a bug. If it happens, please contact the IRAM computer group. And then you can reboot the computer.

If one of the CanManager processes crashes or hangs, you can restart them with the following command:

```
$  /home/introot/fiberOptic/initFoCan.sh
```

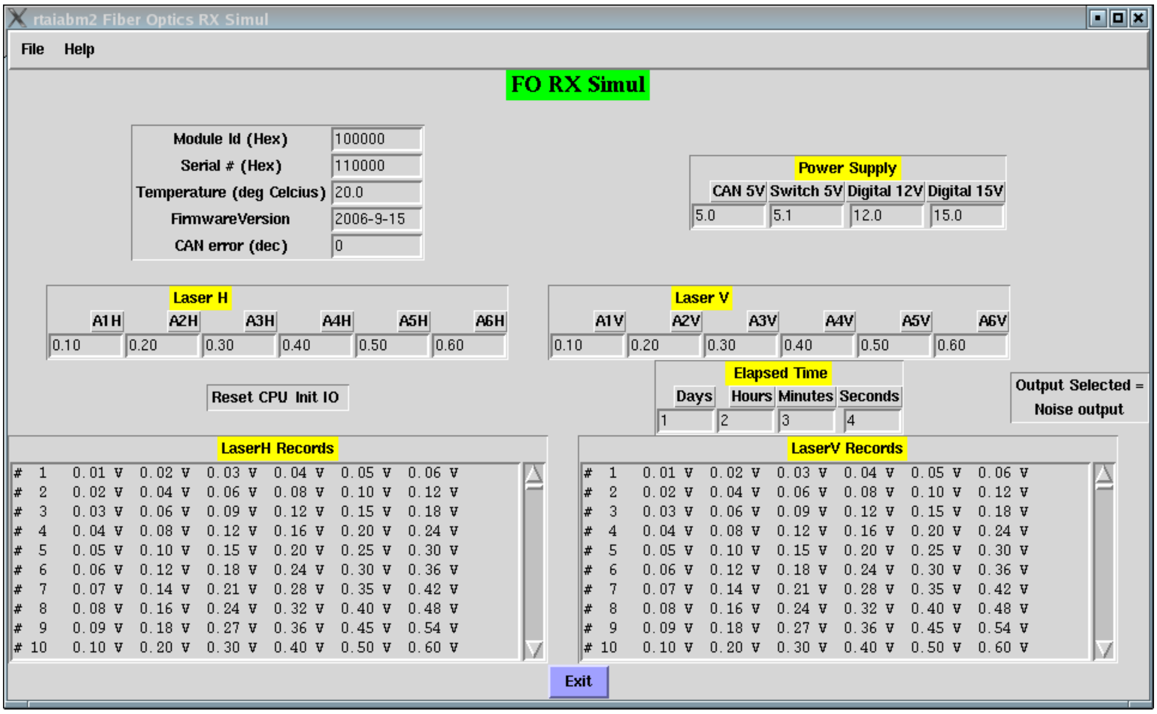
Warning: ifpServer (IF Processor) and foRxServer share the same CanManager. So if you restart the CanManager it may disturb ifpServer.

11 Simulation

11.1 RX Simulator

foRxSimul.py is the IF processor simulator

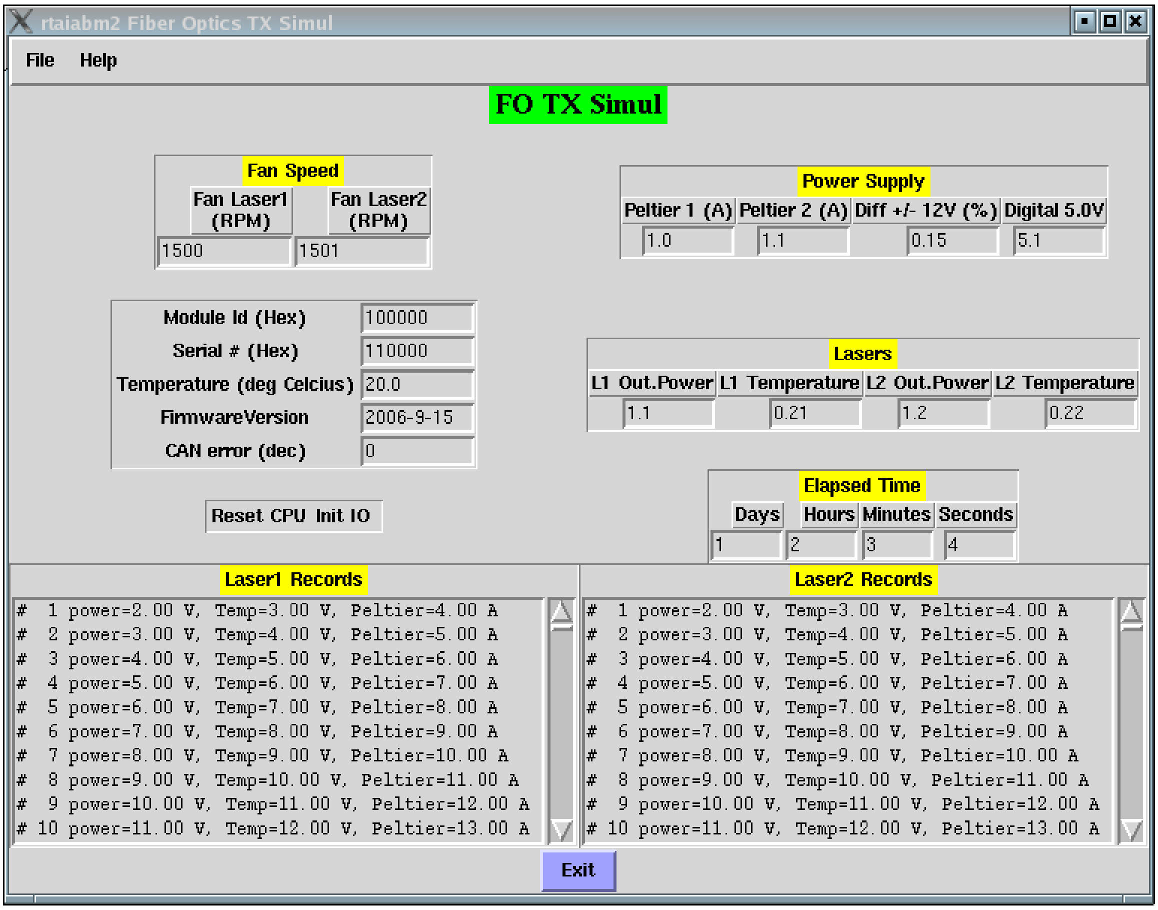
syntax:
foRxSimul.py



11.2 TX Simulator

foTxSimul.py is the IF processor simulator

syntax:
foTxSimul.py


rtaiabm2 Fiber Optics TX Simul
File
Help
FO TX Simul
Fan Speed
Fan Laser1 (RPM)
Fan Laser2 (RPM)
1500
1501
Power Supply
Peltier 1 (A)
Peltier 2 (A)
Diff +/- 12V (%)
Digital 5.0V
1.0
1.1
0.15
5.1
Module Id (Hex)
100000
Serial # (Hex)
110000
Temperature (deg Celcius)
20.0
FirmwareVersion
2006-9-15
CAN error (dec)
0
Lasers
L1 Out.Power
L1 Temperature
L2 Out.Power
L2 Temperature
1.1
0.21
1.2
0.22
Elapsed Time
Days
Hours
Minutes
Seconds
1
2
3
4
Reset CPU Init IO
Laser1 Records
1 power=2.00 V, Temp=3.00 V, Peltier=4.00 A
2 power=3.00 V, Temp=4.00 V, Peltier=5.00 A
3 power=4.00 V, Temp=5.00 V, Peltier=6.00 A
4 power=5.00 V, Temp=6.00 V, Peltier=7.00 A
5 power=6.00 V, Temp=7.00 V, Peltier=8.00 A
6 power=7.00 V, Temp=8.00 V, Peltier=9.00 A
7 power=8.00 V, Temp=9.00 V, Peltier=10.00 A
8 power=9.00 V, Temp=10.00 V, Peltier=11.00 A
9 power=10.00 V, Temp=11.00 V, Peltier=12.00 A
10 power=11.00 V, Temp=12.00 V, Peltier=13.00 A
Laser2 Records
1 power=2.00 V, Temp=3.00 V, Peltier=4.00 A
2 power=3.00 V, Temp=4.00 V, Peltier=5.00 A
3 power=4.00 V, Temp=5.00 V, Peltier=6.00 A
4 power=5.00 V, Temp=6.00 V, Peltier=7.00 A
5 power=6.00 V, Temp=7.00 V, Peltier=8.00 A
6 power=7.00 V, Temp=8.00 V, Peltier=9.00 A
7 power=8.00 V, Temp=9.00 V, Peltier=10.00 A
8 power=9.00 V, Temp=10.00 V, Peltier=11.00 A
9 power=10.00 V, Temp=11.00 V, Peltier=12.00 A
10 power=11.00 V, Temp=12.00 V, Peltier=13.00 A
Exit