IRAM-COMP-075

Revision: 0
2013-02-05

Contact Author

Institut de RadioAstronomie Millimétrique

PdB Receiver Software

Owner Sebastien Blanchet

Keywords: receiver, software

Approved by: A.Perrigouard	Date: September 2010	Signature:

Change Record

REVISION	DATE	AUTHOR	AFFECTED ;ECTION/PAGE	REMARKS
IRAM-COMP-33	Oct 2006			Previous version (python software)

Content

1 Introduction

This document is the user manual for the IRAM receivers of the Plateau de Bure Interferometer. It concerns only the software that runs on the cabin computer, except for the last few pages that provides documentation about the software that run on the interferometer main computer.

2 History

This section explains the historical evolution of the receiver controlling software

2.1 1985-1995

It was the first version of the controlling software. There was no local computer in the antenna cabin, but only CAMAC crates. The receiver controlling software run on the interferometer main computer in the control room and drove directly the CAMAC crates through high-speed serial links.
The receiver controlling software was written in Fortran.

2.2 1995-2005

The VME bus replaced the CAMAC bus, and there was a local computer in the antenna cabin to control the receiver. The computer was an Eltec Eurocom16 VME board (Motorola 68030 @ 25 MHz with 8 MB of memory) that run the OS-9 operating system.
Compared to the previous version, this solution offered a lot of new commodities:

- Unix-like operating system
- C compiler
- Ethernet and TCP/IP

The receiver controlling software was written in C.

2.3 2005-2010

For this new design, IRAM decided to use a multiplexed buses between the controlling computer and the receiver to reduce the number of wires. The main bus is the CAN bus (Controller Area Network). Nevertheless the VME bus and the I2C were still used for legacy reasons..

The controlling computer was a true VME PC (a VMIVME 7700 with an Intel Celeron @ 650 MHz and 512 MB of memory). It ran a standard Linux operating system.
The receiver controlling software was written in python/tk.

2.4 2010-today

5 years later, the receiver is upgraded with a new local oscillator based on YIG tuned oscillators. So it is an opportunity to modernize the controlling software.

Improvement list:

- Programs are rewritten in C++/Qt, so they run 5-time faster than the equivalent program in Python/Tk.
- A SQL database replaces the text configuration files.
- The standard XML-RPC protocol replaces the IRAM binary network protocol for communication between the receiver and the interferometer controlling room.

3 Requirements

3.1 Network requirements

Since the cabin computers are diskless, a server is required:

- to boot with the DHCP / PXE protocols
- to export the filesystem with the NFS protocol.

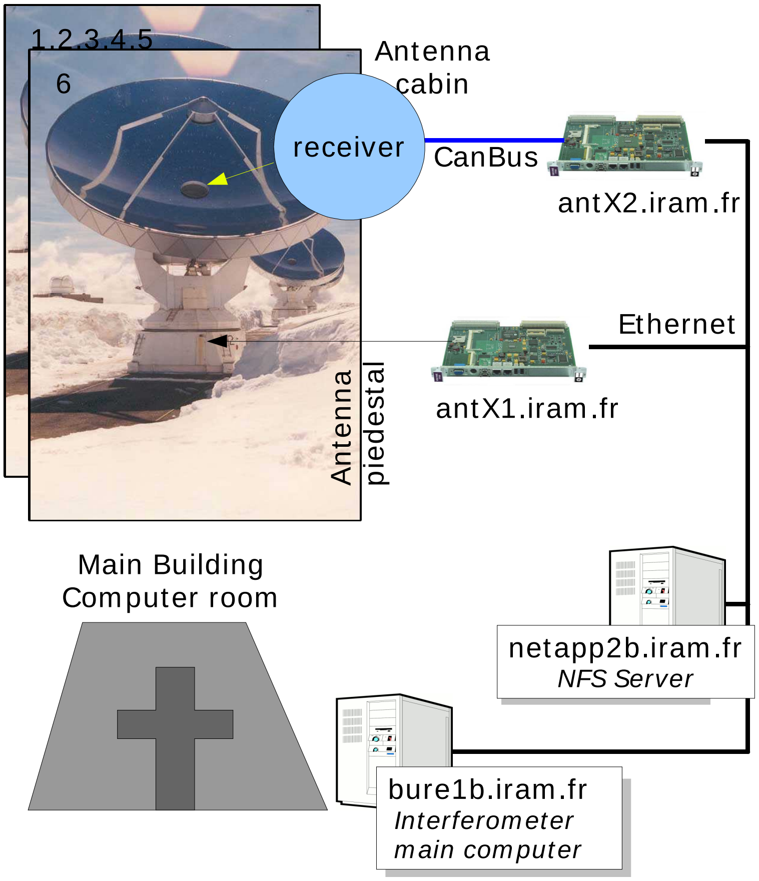


Figure 1: Network topology

For a full description of the installation see the document IRAM-COMP-035 *NFS for diskless x86 computers on the PdbI*

3.2 Hardware requirement

In the production environment, the control software runs on a VMIVME- 7700 with a Tews TPMC816 PMC card.

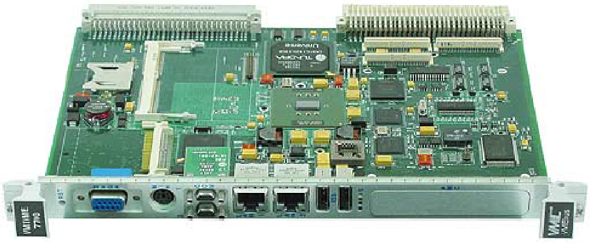


Figure 2: VMIVME-7700 from GEFanuc Automation

But for the test environment, any PC can be used, because a CAN bus simulator is provided

3.3 Software requirements

The software targets Linux Fedora Core 4 i386 as main platform, but it should run on any computer with Linux 2.6.x

For example, the software has been tested successfully tested on different systems:

- Debian 4.0 and 5.0 for i386 / x86_64
- Fedora Core 4 and 6 for i386 / x86_64

The following software are required to build and run the receiver programs:

Name	Description	Version	Download
g++	GNU C++ compiler	>= 4.0.2	http://gcc.gnu.org
Qt	C++ framework	>= 4.4.3	http:/qt.nokia.com
Sqlite	Embedded SQL database	>= 3.3.6	http://sqlite.org
Xmlrpc-c	XML-RPC for C and C++.	>= 1.06	http://xmlrpc-c.sourceforge.net
Gnuplot	Graphing utility	>= 4.2.5	http://www.gnuplot.info/
subversion	Version control system	>= 1.5	http://subversion.tigris.org
doxygen	Automatic documentation system	>= 1.6.1	http://doxygen.org
Cxxtest	A unit test framework for C++	>= 3.10.1	http://cxxtest.tigris.org

All these software are very common, and have ready-to-install packages for your favorite Linux distribution.

4 Installation

4.1 Build receiver software

Extract code from the repository

```
$ git clone git@git.iram.fr:comp/noema noema-dev
$ cd noema-dev/recng
$ qmake-qt4
$ make get_deps
$ ./build.sh
```

Optional, you can build the API documentation. The documentation will be created in the *doxydoc* subdirectory.

```
$ make doc
```

Build also the CAN driver (not required if you use only the simulation)

```
$ cd ~/develSVN/tdrv011
$ su -c "make install"
```

To create devices (and to load the driver)

```
$ su -c "./create_devices.sh"
```

The `create_devices.sh` script creates nodes in `/etc/udev/devices`.

Install the CAN tools

```
$ su
# cd ./CanIp/
# ./install.sh
# exit
```

To install the software **and the default data** in `/home/introot/recng`
Note: the shared libraries are installed in `/home/introot/lib`

```
$ su -c "./install.sh all"
```

Warning:
If you wish to update only the software **without reinstalling the default data**, use `./install.sh(`without the *all*). It is equivalent to run `./install.sh programs`

If you are not sure, it is safer to backup `/home/introot/recng` first

```
$ tar cvfz ~/recng-backup-`date --iso`.tar.gz /home/introot/{recng,lib}
```

4.2 Configure environment

You should add `/home/introot/recng/bin` to your PATH

If you wish to use CanLogger, you should also define CAN_DB_FILE as follow

```
export DEVICE_NAME=recng
export CAN_DB_FILE=/home/introot/recng/data/${DEVICE_NAME}.db
```

4.3 Initial test

For this initial test, we run

1. The CanManagers to enable the Can Over IP protocol
2. The simulator
3. The autotuning lo program

First we remove the `/dev/tpmc816*` devices, so the CanManager will run in simulation mode

```
$ su –c "rm –f /dev/tpmc816_*"
$ recng-init-can.sh
$ xterm -e recng-simul &
```

(click on LO button to activate the LO simulation)

Now run the LO autotuning program:

```
$ recng-lo -b=1 -f=90.0
```

4.4 Start application automatically

Run `/home/introot/recng/bin/recng-init-receiver.sh` at the startup.

5 Internal programs

This section describes internal programs that drive the receiver. These programs are executed automatically, therefore normal users are note expected to run them.

5.1 CanManager

For a complete description see the document *can-ip.pdf*

```
CanManager - Bridge between the CAN bus and the CAN/IP protocol
Usage: CanManager [options]
Options:
  -d=/dev/devname     CAN controller device name to use. If missing,
                        the application runs in simulation mode
  -p=udpPort          Listen to UDP port udpPort
  -t1=delay1_us       Delay in microseconds between two CAN 1.0 messages.
                          Default=0
  -t2=delay2_us       Delay in microseconds between two CAN 2.0 messages.
                          Default=0

  -v             Display version information
  -h, -?         Display help

Example: CanManager -d=/dev/tpmc816_0 -p=2500 -d1=100000
```

For the recng software, CanManager must listen on the following ports:

UDP port	Device	Description
2500	`/dev/tpmc816_0`	CAN bus for the calibration system (JVL motors)
2501	`/dev/tpmc816_1`	CAN bus for the receiver
2502	*None*	Virtual CAN bus for the subreflector

Commands

```
$ CanManager –d=/dev/tpmc816_1 –p=2500 –d1=50000
$ CanManager –d=/dev/tpmc816_1 –p=2501 –d1=50000
$ CanManager –p=2502
```

5.2 recng-server

recng-server is an XML-RPC server to remotely control the receiver.

What is XML RPC ?
XML-RPC is a remote procedure call protocol that uses XML to encode its calls and HTTP as a transport mechanism. For a detailed introduction to XML RPC see http://en.wikipedia.org/wiki/XML-RPC
This protocol is very simple to use, and can be used from any programming language (many opensource libraries are available)

recng-server listens for XML-RPC calls on http://localhost:1080/RPC2

Note: The path */RPC2* is the default path for a XML-RPC server. Therefore, it can be sometimes omitted (it depends on the implementation library)

This server supports:

- introspection http://xmlrpc-c.sourceforge.net/introspection.html
- multicalls

5.2.1 Syntax

```
$ recng-server -h
Recng Server - XML-RPC Server
Listen on port 1080
Usage: recng-server [options]
Options:
  -v        Display version information
  -h, -?    Display help
```

5.2.2 API Description

Since the XML-RPC server support introspection, we can retrieve the API with a simple program

```
$ recng-ListMethods.py
# Connect to http://localhost:1080
antenna.resetDeicing
antenna.setCentralHub
antenna.setDeicing
deviceList
deviceProperties
getProperty
receiver.getStatus
receiver.parkLo
receiver.resetLo1Ref
receiver.setAttenuator
receiver.setCalibration
receiver.setFrequency
receiver.setLoSwitch
receiver.setObserving
setProperty
system.listMethods
system.methodHelp
system.methodSignature
system.multicall
system.shutdown
```

```
$ recng-Introspection.py
# Connect to http://localhost:1080
----------------------------------------------------------------------
Name       : antenna.resetDeicing(  )
Return Type: int
Description: Reset Deicing.
Return code: always 0

Syntax:  resetDeicing( string positionName )

----------------------------------------------------------------------
Name       : antenna.setCentralHub( string )
Return Type: int
Description: open/close central hub
Return code: always zero.
Syntax:  setCentralHub( string command )
- 'command' must be in {close, open}

----------------------------------------------------------------------
Name       : antenna.setDeicing( string )
Return Type: int
Description: Set deicing.
```

```
Return code: always 0

Syntax:  setDeicing( string a_deicingMode )

Valid deicing modes are (insensitive case):
    OFF, 3SECTORS, 6SECTORS

----------------------------------------------------------------------
Name       : deviceList(  )
Return Type: array
Description: Device list
Return list of devices (array of strings)
Syntax:  deviceList()

----------------------------------------------------------------------
Name       : deviceProperties( string )
Return Type: array
Description: Get properties list of a device
Return array of string
Syntax:  deviceProperties(string deviceName)

----------------------------------------------------------------------
Name       : getProperty( string, string )
Return Type: string
Description: Set any property
Return value:   string: OK
        an empty string means error
Syntax:  getProperty(string deviceName, string propertyName)

----------------------------------------------------------------------
Name       : receiver.getStatus(  )
Return Type: struct
Description: Returns the receiver status

----------------------------------------------------------------------
Name       : receiver.parkLo( int )
Return Type: int
Description: Send LO to a parking position that does not interfere with the
others LOs.
Return code is always zero.
Syntax: receiver.parkLo(int bandNum)
 - bandNum must be in range [1, 4]

----------------------------------------------------------------------
Name       : receiver.resetLo1Ref( string )
Return Type: int
Description: Reset Lo1Ref
Syntax: resetLo1Ref( string channelList)
- 'channelList' must be a comma-separated list of channels.   valid channel are
in {'A', 'B'}

Example for channeList:
        channelList = "A"
        channelList = "A,B"

Return code is
        - 0: resetting is OK
        - 1: a problem occurs

----------------------------------------------------------------------
Name       : receiver.setAttenuator( int, string, double )
Return Type: int
Description: Set attenuator. Return code is always zero.
```

```
Syntax:  setAttenuator(int bandNum, string attenuatorName, int
attenuationDecibel)
- 'bandNum' must be in range [1, 4]
- 'attenuatorName' must be in {V,H}
- 'attenuationDecibel' must be in range [0,31]

----------------------------------------------------------------------
Name       : receiver.setCalibration( string )
Return Type: int
Description: Set Calibration.
Return code:
    - 0 : OK
    - 1 : Error
Syntax:  setCalibration( string positionName )

Valid position names are:
01_amb, 01_cold, 01_qwplate, 01_sky, 02_amb, 02_cold, 02_qwplate, 02_sky,
03_amb, 03_cold, 03_qwplate, 03_sky, 04_amb, 04_cold, 04_qwplate, 04_sky

----------------------------------------------------------------------
Name       : receiver.setFrequency( double, int, int, int )
Return Type: int
Description: Set LO frequency.
Return code is always zero.
Syntax: SetFrequency( double flo, int bandNum, int sideband, int deltaF)
 - flo is the LO frequency in GHz
 - bandNum must be in range [1, 3]
 - sideband: 0 -> LSB, 1 -> USB
 - deltaF: 0 -> -, 1 -> +

----------------------------------------------------------------------
Name       : receiver.setLoSwitch( int, string, int )
Return Type: int
Description: Set a LO switch.
Return code is always zero.
Syntax: setLoSwitch(int bandNum, string switchName, int switchValue)
 - bandNum must be in range [1, 3]
 - switchName must be in: { gunn, deltaf, loop, sweep }
 - switchValue must be 0 or 1

----------------------------------------------------------------------
Name       : receiver.setObserving( double )
Return Type: int
Description: Set Observing ON/OFF.
When Observing is ON, some graphical user interfaces are disabled
to avoid mistakes.

Return code is always zero.
Syntax: SetObserving( int mode )
 - mode: 0 -> Off, 1 -> ON

----------------------------------------------------------------------
Name       : setProperty( string, string, string )
Return Type: int
Description: Set any property
Return code:    1: OK
        0: Error
Syntax:  setProperty(string deviceName, string propertyName, string value)

----------------------------------------------------------------------
Name       : system.listMethods(  )
Return Type: array
Description: Return an array of all available XML-RPC methods on this server.
```

```
----------------------------------------------------------------------
Name       : system.methodHelp( string )
Return Type: string
Description: Given the name of a method, return a help string.

----------------------------------------------------------------------
Name       : system.methodSignature( string )
Return Type: array
Description: Given the name of a method, return an array of legal signatures.
Each signature is an array of strings.  The first item of each signature is the
return type, and any others items are parameter types.

----------------------------------------------------------------------
Name       : system.multicall( array )
Return Type: array
Description: Process an array of calls, and return an array of results.  Calls
should be structs of the form {'methodName': string, 'params': array}. Each
result will either be a single-item array containg the result value, or a
struct of the form {'faultCode': int, 'faultString': string}.  This is useful
when you need to make lots of small calls without lots of round trips.

----------------------------------------------------------------------
Name       : system.shutdown( string )
Return Type: int
Description: Shut down the server.  Return code is always zero.

----------------------------------------------------------------------
```

5.3 XML RPC client examples

5.3.1 Minimal python example

XML RPC is very easy and pleasant to use in python.

For example to call method *receiver.setCalibration* on the server, you need only the following lines.

```
import xmlrpclib
server = xmlrpclib.ServerProxy("http://localhost:1080" )
print server.receiver.setCalibration("01_amb" )
```

5.3.2 Python examples

See directory `ReceiverNG/scripts/xmlrpc` for other XML-RPC python examples.

5.3.3 C++ client example

I have written a small C++ client example, the source code is available in:
`ReceiverNG/apps/GetStatus`

5.4 recng-id

Print the receiver ID.
Antennas have receiver ID between 1 and 6. Receivers in the laboratory have receiver ID greater than 100.

5.4.1 Example

```
$ recng-id
receiverId=5
```

5.5 recng-jvl-list

Print the list of the JVL motors names.
There are two motors manufactured by JVL (http://www.jvl.dk) for the calibration. They are used to switch between amb, cold, sky and vlbi position

Calibration position per motor:

Motor name	Bands	Defined positions
calA	1 2	01_amb, 01_cold, 01_sky, 01_vlbi 02_amb, 02_cold, 02_sky, 02_vlbi
calB	3 4	03_amb, 03_cold, 03_sky, 03_vlbi 04_amb, 04_cold, 04_sky, 04_vlbi

5.5.1 Syntax

```
$ recng-jvl-list -h
Jvl List - Print the list of Jvl motor names
Usage: recng-jvl-list [options]
Options:
  -v            Display version information
  -h, -?        Display help
```

5.5.2 Example

```
$ recng-jvl-list
calA
calB
```

5.6 recng-jvl-init

This programs:

- Write the settings into the motor controller.
- Read back the settings to be sure they are correctly written.
- Start the 0-reference searching procedure (So the motor slowly rotates backward until it find the mechanical stop that marks the 0 position)

5.6.1 Syntax

```
$ recng-jvl-init -h
Jvl Init - Initialize Jvl motor reference
Usage: recng-jvl-init [options]
Options:
  -m=MotorName      Specify motor to initialize

  -v                Display version information
  -h, -?            Display help
```

5.6.2 Example

```
$ recng-jvl-init -m=calA
Setting for this tuning
        MotorName = calA

Initializing motor calA
Check motor settings
OK
calA: reference is now initialized
```

5.7 recng-jvl-check

This program

- Reads back the JVL motor controller settings
- Compares the read settings with the expected settings that are stored in the SQL database.
- Prints an error message if the settings do not match.

5.7.1 Syntax

```
$ recng-jvl-check -h
Jvl Init - Check Jvl motor settings
Usage: recng-jvl-check [options]
Options:
  -m=MotorName  Specify motor to check

  -v            Display version information
  -h, -?        Display help
```

5.7.2 Example

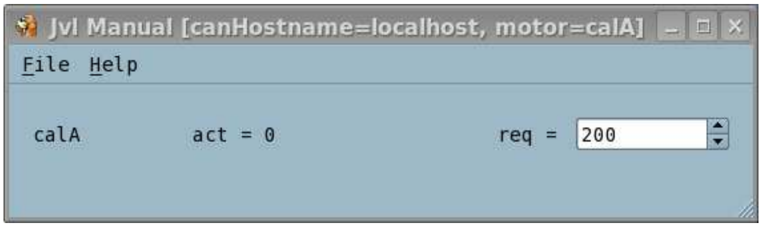
Example when a setting is incorrect

```
$ recng-jvl-check -m=calA
Setting for this tuning
        MotorName = calA

Checking motor calA
Parameter KVOUT(13) has not been set correctly. Expected: 0x00001b00   Found:
0x00000a00
Settings are wrong
Aborted
```

Example when all settings are correct

```
$ recng-jvl-check -m=calA
Setting for this tuning
        MotorName = calA

Checking motor calA
OK
```

5.8 recng-jvl-manual



calA act = 0 req = 200

This program allows moving a JVL motor to any position. The positions are specified in motor unit. Obviously, there are software limits to avoid out-of-range positions.

This program is typically used to find the best motor positions for each calibration step (amb, cold). Then the user have to update `/home/introot/recng/data/conf-calibration.sql` with the found motor positions, and then he runs `recng-update-db.sh` to update the database.

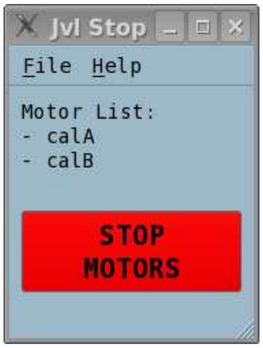
5.8.1 Syntax

```
$ recng-jvl-manual -h
Jvl Manual - Control manually Jvl motor
Usage: recng-jvl-manual [options]
Options:
  -m=name              Motor name to drive (Mandatory)

  -v                   Display version information
  -h, -?               Display help
```

5.8.2 Example

```
$ recng-jvl-manual -m=calA
```

5.9 recng-jvl-stop



Motor List:
- calA
- calB

This program is used to stop all the JVL motors displayed on the list.
The window is always visible and cannot be hidden by other windows.

5.9.1 Syntax

```
$ recng-jvl-stop -h
Jvl Stop - Emergency Stop button for Jvl motors
Usage: recng-jvl-stop [options]
Options:
  -v                   Display version information
  -h, -?               Display help
```

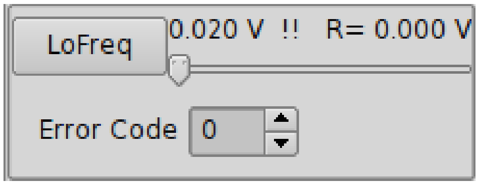
5.10 Simulator

The goal of this program is to simulate the receiver, so that the other software can be written before the receiver hardware is ready.

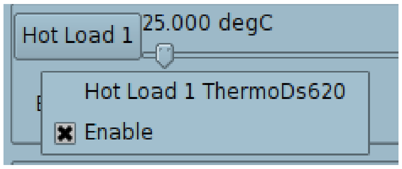
5.10.1 Syntax

```
$ recng-simul
```

5.10.2 General notes

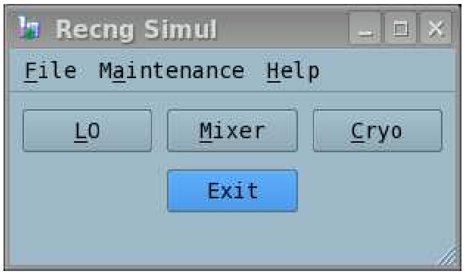


Almost items have an *Error Code* spinbox, that is used to set the device error code.



Almost items have enable/disable menu to simulate the device unplugging.
Right-click on the widget to display the enable/disable menu

5.10.3 Main window



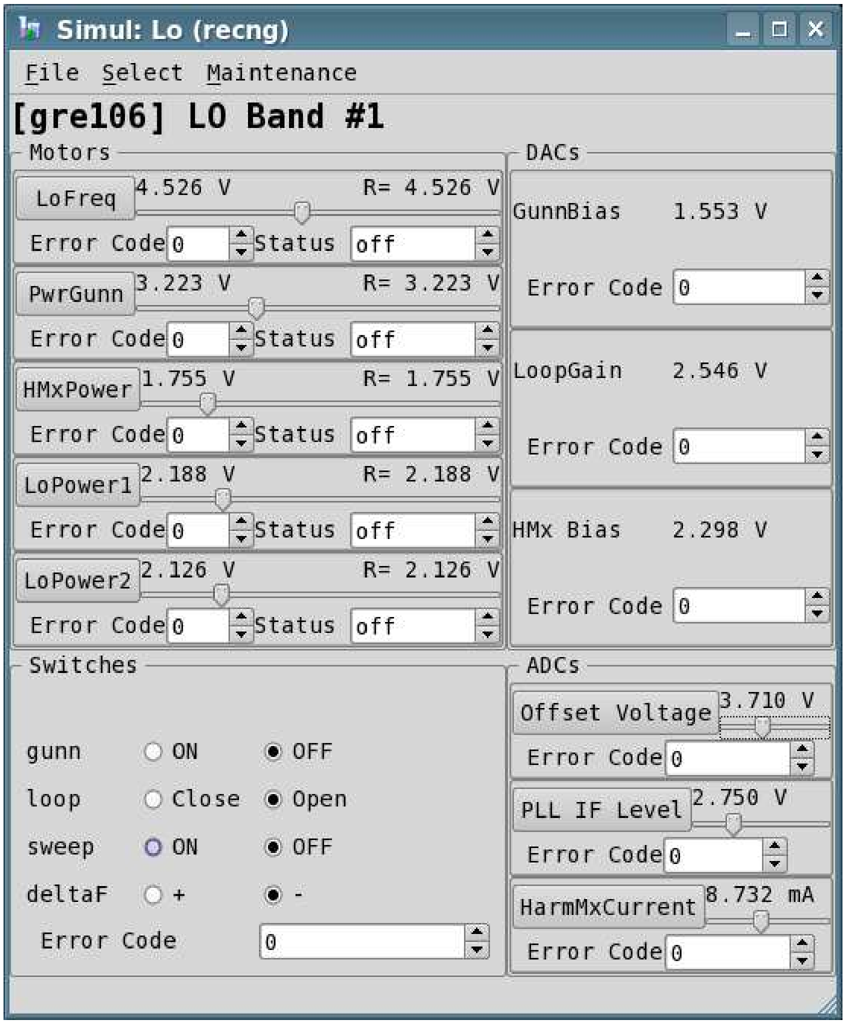
The main window has three buttons, click on them to display/hide simulation window for LO, Mixer or Cryo.

The simulation occurs only when the associated subwindow is opened.

For example, if you want to simulate only the LO, open only the LO window.

5.10.4 LO Simulation Window

The LO window display the simulator for the LO.
The window can display only one band, but the 4 LO bands are simulated together. You can display the other bands with the *Select* menu.



Simul: Lo (recng)

File Select Maintenance

[gre106] LO Band #1

Motors

LoFreq 4.526 V R= 4.526 V
Error Code 0 Status off

PwrGunn 3.223 V R= 3.223 V
Error Code 0 Status off

HMxPower 1.755 V R= 1.755 V
Error Code 0 Status off

LoPower1 2.188 V R= 2.188 V
Error Code 0 Status off

LoPower2 2.126 V R= 2.126 V
Error Code 0 Status off

DACs

GunnBias 1.553 V
Error Code 0

LoopGain 2.546 V
Error Code 0

HMx Bias 2.298 V
Error Code 0

Switches

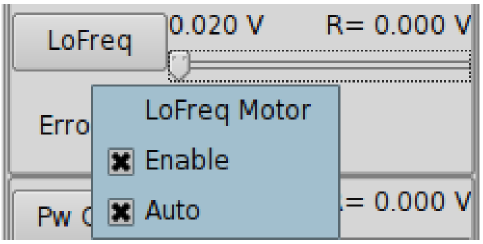
gunn ○ ON ● OFF
loop ○ Close ● Open
sweep ○ ON ● OFF
deltaF ○ + ● -
Error Code 0

ADCs

Offset Voltage 3.710 V
Error Code 0

PLL IF Level 2.750 V
Error Code 0

HarmMxCurrent 8.732 mA
Error Code 0

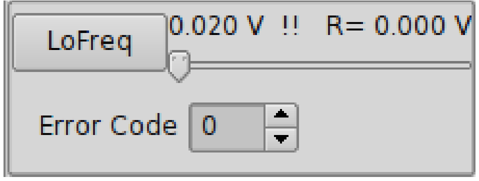


LoFreq 0.020 V R= 0.000 V

LoFreq Motor
☒ Enable
☒ Auto

Motor simulator:
It simulates a CAN motor. It moves when position requests arrive.
You can simulate the following failure:
The motor returns an error code: enter the error code in the Error Code spinbox.
The motor is missing: right-click, and unselect *"Enable"*
The motor answers, but does not move: right-click and unselect *"Auto"*

When *"Auto"* is unselected, a *!!* symbol appears.



LoFreq 0.020 V !! R= 0.000 V
Error Code 0

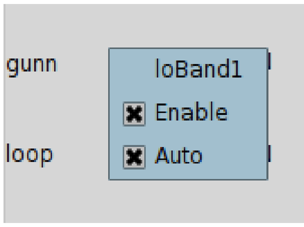
Adc simulator
It simulates a CAN ADC.
You can simulate a device missing error: right-click and unselect *"Enable"*

Dac simulator
It simulates a CAN DAC.

You can simulate a device missing error: right-click and unselect *"Enable"*

Lo Switches

It simulates the Lo switches.

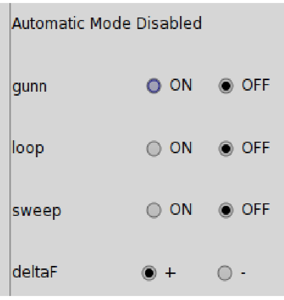


☒ Enable

☒ Auto

You can simulate the following errors:
A device is missing error: right-click and unselect *"Enable"*
The device answers, but values do not switch: right-click and unselect *"Auto"*

When *"Auto"* is unselected, a warning appears in the widget:*"Automatic Mode Disabled"*



Automatic Mode Disabled

gunn ○ ON ● OFF

loop ○ ON ● OFF

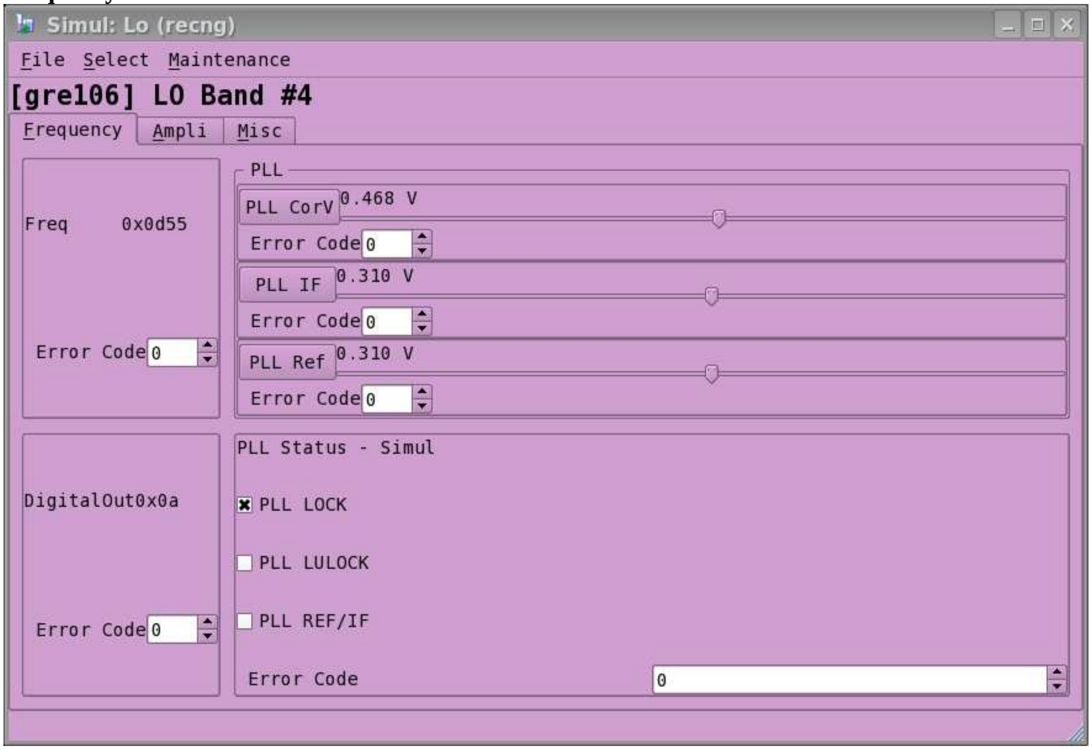
sweep ○ ON ● OFF

deltaF ● + ○ -

5.10.5 Lo YIG simulation window

For band #4, the simulation window is different because the band #4 use a YIG tune oscillator instead of a gunn oscillator.

Frequency tab



[gre106] LO Band #4

Freq 0x0d55

Error Code 0

PLL

PLL CorV 0.468 V

Error Code 0

PLL IF 0.310 V

Error Code 0

PLL Ref 0.310 V

Error Code 0

DigitalOut0x0a

Error Code 0

PLL Status - Simul

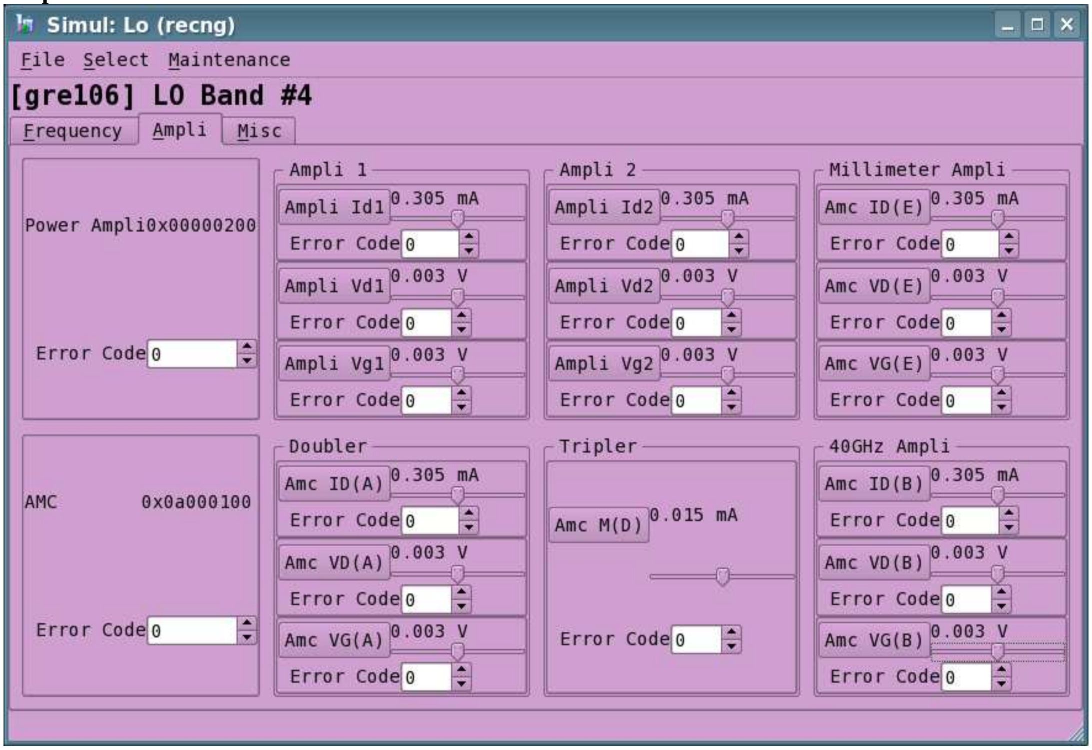
☒ PLL LOCK

☐ PLL LULOCK

☐ PLL REF/IF

Error Code 0

Ampli tab



[gre106] LO Band #4

Power Ampli 0x00000200
Error Code 0

AMC 0x0a000100
Error Code 0

Ampli 1
Ampli Id1 0.305 mA
Error Code 0
Ampli Vd1 0.003 V
Error Code 0
Ampli Vg1 0.003 V
Error Code 0

Ampli 2
Ampli Id2 0.305 mA
Error Code 0
Ampli Vd2 0.003 V
Error Code 0
Ampli Vg2 0.003 V
Error Code 0

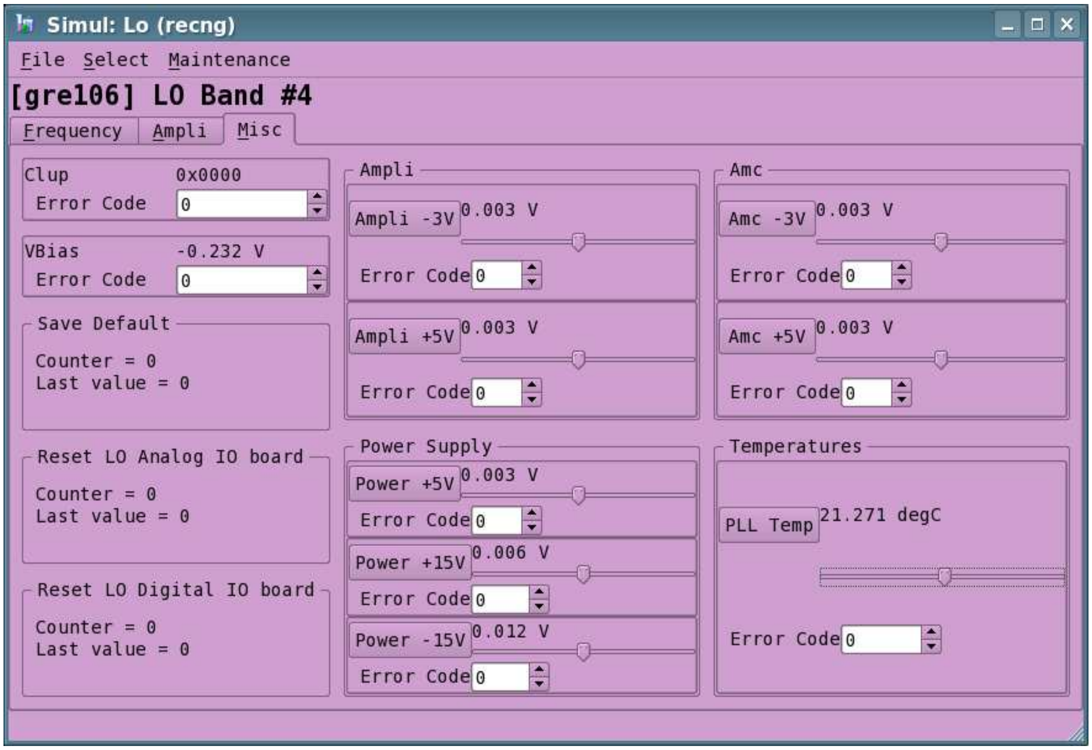
Millimeter Ampli
Amc ID(E) 0.305 mA
Error Code 0
Amc VD(E) 0.003 V
Error Code 0
Amc VG(E) 0.003 V
Error Code 0

Doubler
Amc ID(A) 0.305 mA
Error Code 0
Amc VD(A) 0.003 V
Error Code 0
Amc VG(A) 0.003 V
Error Code 0

Tripler
Amc M(D) 0.015 mA
Error Code 0

40GHz Ampli
Amc ID(B) 0.305 mA
Error Code 0
Amc VD(B) 0.003 V
Error Code 0
Amc VG(B) 0.003 V
Error Code 0

Misc tab



[gre106] LO Band #4

Clup 0x0000
Error Code 0

VBias -0.232 V
Error Code 0

Save Default
Counter = 0
Last value = 0

Reset LO Analog IO board
Counter = 0
Last value = 0

Reset LO Digital IO board
Counter = 0
Last value = 0

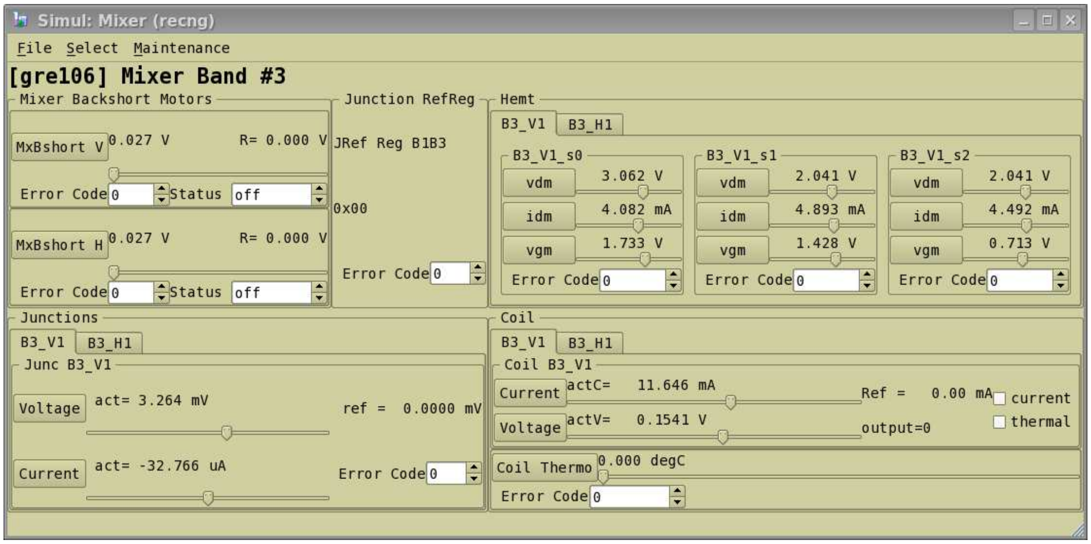
Ampli
Ampli -3V 0.003 V
Error Code 0
Ampli +5V 0.003 V
Error Code 0

Amc
Amc -3V 0.003 V
Error Code 0
Amc +5V 0.003 V
Error Code 0

Power Supply
Power +5V 0.003 V
Error Code 0
Power +15V 0.006 V
Error Code 0
Power -15V 0.012 V
Error Code 0

Temperatures
PLL Temp 21.271 degC
Error Code 0

5.10.6 Mixer Simulation Window



[gre106] Mixer Band #3

MxBshort V 0.027 V R= 0.000 V
Error Code 0 Status off
MxBshort H 0.027 V R= 0.000 V
Error Code 0 Status off

JRef Reg B1B3
0x00
Error Code 0

B3_V1_s0: vdm 3.062 V; idm 4.082 mA; vgm 1.733 V; Error Code 0
B3_V1_s1: vdm 2.041 V; idm 4.893 mA; vgm 1.428 V; Error Code 0
B3_V1_s2: vdm 2.041 V; idm 4.492 mA; vgm 0.713 V; Error Code 0

Junc B3_V1
Voltage act= 3.264 mV ref = 0.0000 mV
Current act= -32.766 uA Error Code 0

Coil B3_V1
Current actC= 11.646 mA Ref = 0.00 mA □ current
Voltage actV= 0.1541 V output=0 □ thermal
Coil Thermo 0.000 degC
Error Code 0

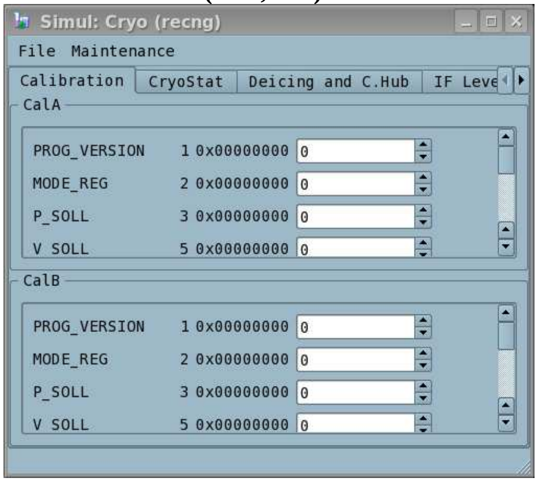
This window simulates:

1. Mixer backshort motors
2. Attenuators
3. ADC IF levels
4. Junctions
5. Hemt
6. Coils

5.10.7 Cryo simulation window

There are 8 tab widget in this window.

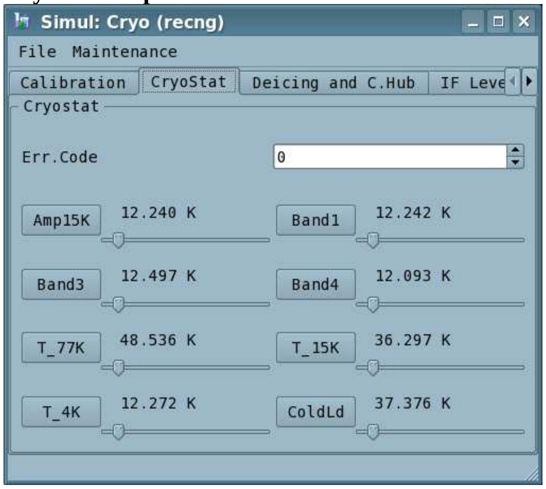
Calibration motor (calA,calB)



CalA
PROG_VERSION 1 0x00000000 0
MODE_REG 2 0x00000000 0
P_SOLL 3 0x00000000 0
V_SOLL 5 0x00000000 0

CalB
PROG_VERSION 1 0x00000000 0
MODE_REG 2 0x00000000 0
P_SOLL 3 0x00000000 0
V_SOLL 5 0x00000000 0

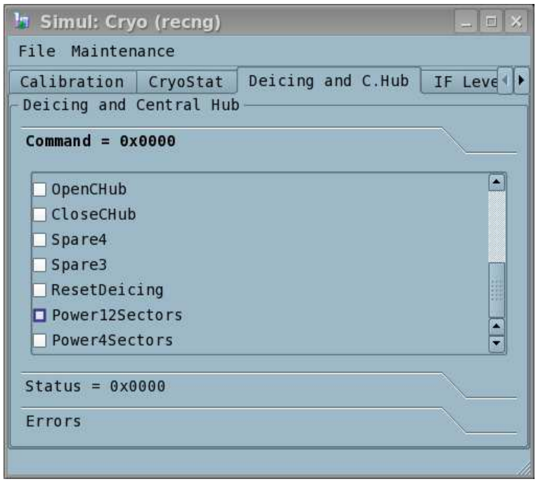
Cryostat temperature



Cryostat
Err.Code 0
Amp15K 12.240 K Band1 12.242 K
Band3 12.497 K Band4 12.093 K
T_77K 48.536 K T_15K 36.297 K
T_4K 12.272 K ColdLd 37.376 K

Deicing and central hub

If levels



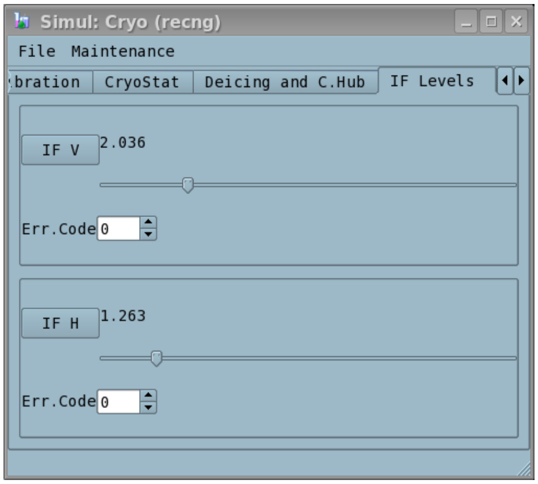
Deicing and Central Hub

Command = 0x0000

- □ OpenCHub
- □ CloseCHub
- □ Spare4
- □ Spare3
- □ ResetDeicing
- □ Power12Sectors
- □ Power4Sectors

Status = 0x0000

Errors



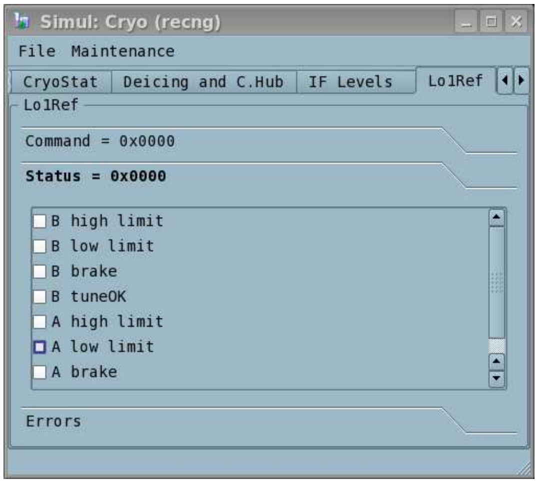
IF V 2.036

Err.Code 0

IF H 1.263

Err.Code 0

Lo1Ref



Lo1Ref

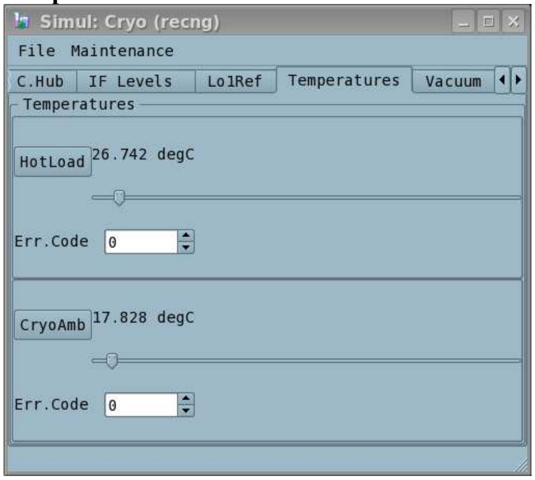
Command = 0x0000

Status = 0x0000

- □ B high limit
- □ B low limit
- □ B brake
- □ B tuneOK
- □ A high limit
- □ A low limit
- □ A brake

Errors

Temperatures



Temperatures

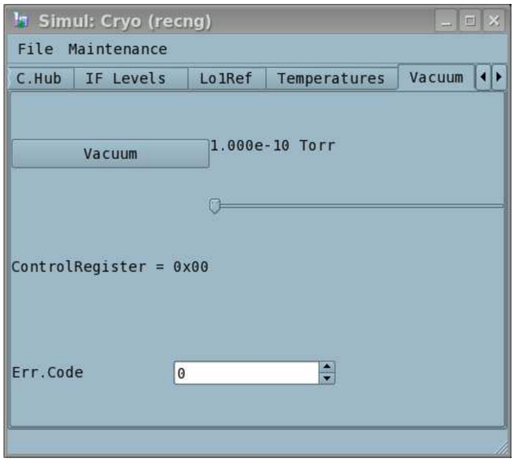
HotLoad 26.742 degC

Err.Code 0

CryoAmb 17.828 degC

Err.Code 0

Vacuum

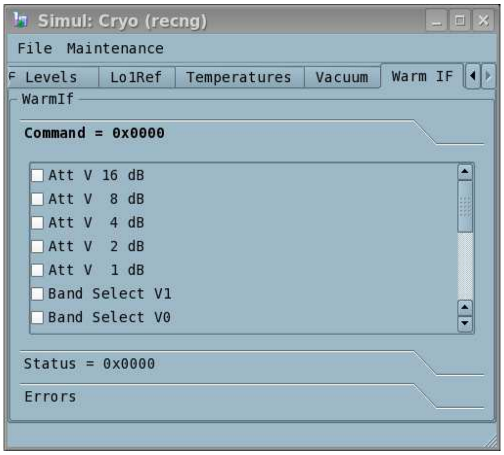


Vacuum 1.000e-10 Torr

ControlRegister = 0x00

Err.Code 0

Warm IF



WarmIf

Command = 0x0000

- □ Att V 16 dB
- □ Att V 8 dB
- □ Att V 4 dB
- □ Att V 2 dB
- □ Att V 1 dB
- □ Band Select V1
- □ Band Select V0

Status = 0x0000

Errors

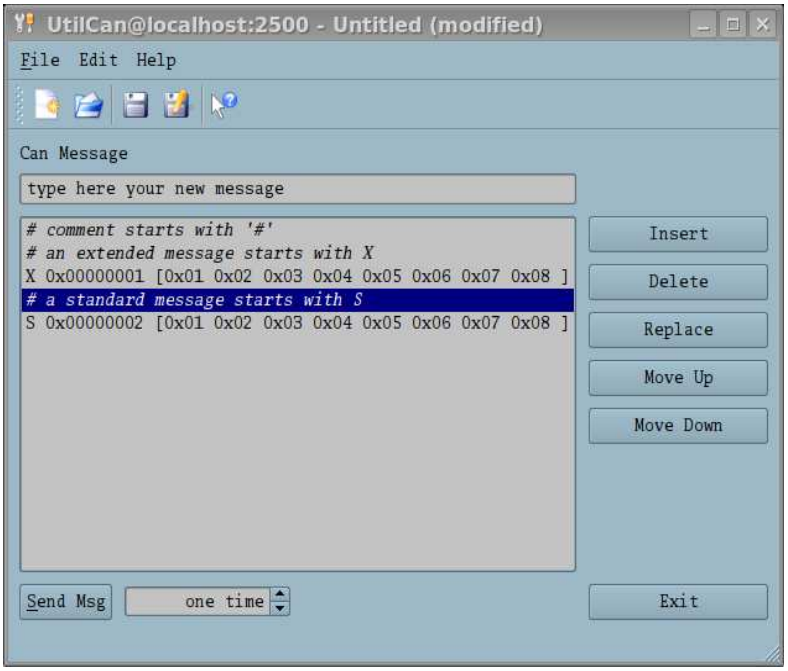
5.11 UtilCan

UtilCan is an utility program to send arbitrary CAN messages.
For a complete description, see document *can-ip.pdf*

5.11.1 Syntax

```
$ UtilCan -h
UtilCan - CAN utility to read/write CAN messages
Usage: UtilCan [options]
Options:
  -p=N                  UDP port to contact (mandatory)
  -f=filename           File to load (optional)

  -v                    Display version information
  -h, -?                Display help

Example: UtilCan -p=2500 -f=myfile.txt
```

5.11.2 Screenshot



Can Message

type here your new message

```
# comment starts with '#'
# an extended message starts with X
X 0x00000001 [0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 ]
# a standard message starts with S
S 0x00000002 [0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 ]
```

one time

5.12 CanLogger

It is a CAN monitor tool, that can optionally decode the CanID into symbolic names.
For a complete description, see document can-ip.pdf

5.12.1 Syntax

```
$ CanLogger -h
CanLogger - CAN logger
Usage: CanLogger [options]
Options:
  -p=N                  UDP port to contact (mandatory)

  -v                    Display version information
  -h, -?                Display help

Example: CanLogger -p=2500
```

```
Note: If the environment variable CAN_DB_FILE is set, the program loads the
database to decode CanID into symbolic names.
```

5.12.2 Example

```
CanLogger -p=2501
Settings:
Port= 2501
DatabaseFile= /home/introot/recng/data/recng.db

Load data from /home/introot/recng/data/recng.db
2010-09-07T16:45:50.852: msg 1 : yig_B4_freq_get : X 0x05000110 []
2010-09-07T16:45:50.852: msg 2 : yig_B4_powerAmpli_get : X 0x05000130 []
2010-09-07T16:45:50.852: msg 3 : yig_B4_amc_get : X 0x05000150 []
2010-09-07T16:45:50.852: msg 4 : adc_B4_yigPllCorV : X 0x05040108 []
```

6 Automatic benchmarks

To automate measures, several automatic benchmarks have been developed.

6.1 Protocols and buses

The measurement instruments are controlled through the GPIP bus, therefore the gpib8065 server is required. See document IRAM-COMP-076 for more details about this GPIB bridge.
The other devices are controlled with the CAN bus.

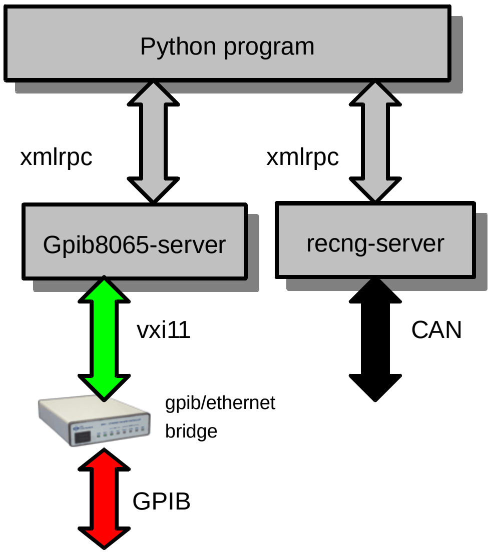
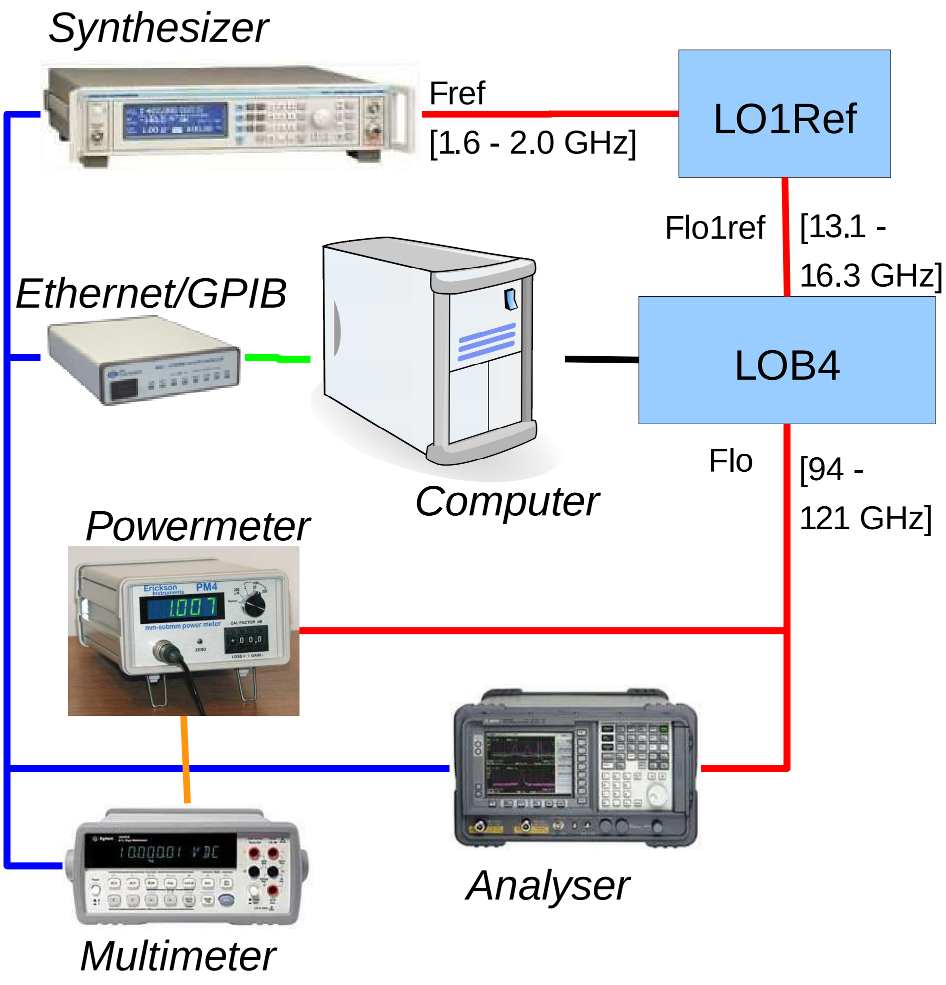


Figure 3 Automatic benchmarks: Software architecture and buses

6.1.1 Bench overview



Caption	
GPIB	🟦
Waveguide	🟥
CAN	⬛
Ethernet/IP	🟩
Analog (copper cable)	🟧

The computer sets fref [1.6-2 GHz] on the synthesizer.
Then the Lo1Ref device generates flo1Ref (13.1- 16.3 GHz) from fref.
The LOB4 device generates flo [94-121 GHz]
The analyzer measures the signal-to-noise ratio of flo
The powermeter measures the output power. Note: our powermeter (Erickson PM4) has not GPIB interface but only an analog output that is read from the multimeter (Agilent 34401A)

Hardware

- Synthesizer: MARCONI INSTRUMENTS,2024
- Analyser: Agilent, E4407B
- Powermeter: Erickson Instruments PM4
- Multimeter: Agilent,34401A

Software

The following software must running

- gpib8065-server. It must listen on localhost:8065 (default port)
- recng-server

6.2 LO band 4 signal/noise

The goal of this benchmark is to measure the signal-to-noise ratio of the band #4 local oscillator for each frequency.

6.2.1 Syntax

```
$ recng-yig-b4-signal-noise.py -h
usage: recng-yig-b4-signal-noise.py [options]

Benchmark LO YIG band 4 signal/noise

options:
  -h, --help            show this help message and exit
  -b FREQ_BEGIN, --begin=FREQ_BEGIN
                        Beginning frequency lo mixer in GHz. Default=283.0 GHz
  -e FREQ_END, --end=FREQ_END
                        Ending frequency lo mixer in GHz. Default=365.0 GHz
  -s FREQ_STEP, --step=FREQ_STEP
                        Step frequency lo mixer in GHz. Default=1.0 GHz
  -o OUTPUT, --output=OUTPUT
                        Output filename. Default=<automatically_generated>
  -v VD, --vd=VD        power ampli vd1, vd2 values, in volt
```

6.2.2 Output example

```
# recng-yig-b4-signal-noise.py
# 2010-10-07T15:53:07.942734
# freq_begin = 283.0 , freq_end = 365.0, freq_step = 1.0
# s/n measure 20E+3 /carrier IF 100E+6
# Hardware:
#    - synthesizer: MARCONI INSTRUMENTS,2024,112261/009,44533/446/04.07
#    - analyzer: Hewlett-Packard, E4407B, MY45109239, A.14.01
#    - multimeter: HEWLETT-PACKARD,34401A,0,4-1-1
# vd1, vd2= 3.400000
# Analyser settings
#     frequency:center = 100E+6
#     frequency:span = 100E+3
#     bandwidth:resolution = 1E+3
#     bandwidth:video = 100
#     :display:window:trace:y:rlevel = 10
#
#
#    F3;        F1;    s/n;  Power;   flo1Ref;      fRef;  Harm;  lock;  vbias
#   GHz;       GHz;    dBc;     mW;       GHz;       GHz;      ;      ;   Volt
  283.0;   94.2333;  -79.5;  36.82;   15.7056;    1.9632;     6;     1;  -0.60
  284.0;   94.5667;  -76.9;  35.61;   15.7611;    1.9701;     6;     1;  -0.60
[…]
  363.0;  120.9000;  -77.0;   8.13;   15.1125;    1.8891;     8;     1;  -0.40
  364.0;  121.2333;  -76.7;   7.93;   15.1542;    1.8943;     8;     1;  -0.40
  365.0;  121.5667;  -78.3;   6.77;   15.1958;    1.8995;     8;     1;  -0.40
# End of measure
# 2010-10-07T16:03:01.793055
# Total duration = 593 seconds
# End of file
```

6.3 LO band 4 output power

The goal of this benchmark is to measure output power of the band #4 local oscillator at a specified frequency for all values of vd1/vd2.

6.3.1 Syntax

```
usage: recng-yig-b4-power-vd.py [options]

Benchmark LO YIG band 4 output power

options:
  -h, --help            show this help message and exit
  -b VD_BEGIN, --begin=VD_BEGIN
                        Beginning vd in volt. Default=0.4 V
  -e VD_END, --end=VD_END
                        Ending vd in volt. Default=3.0 V
  -s VD_STEP, --step=VD_STEP
                        Step vd in volt. Default=0.1 V
  -o OUTPUT, --output=OUTPUT
                        Output filename. Default=<automatically_generated>
  -f FLOMIX, --floMix=FLOMIX
                        flo mixer frequency in GHz
```

6.3.2 Output example

```
# recng-yig-b4-power-vd.py
# 2010-10-08T11:50:21.683310
# vd_begin = 0.3 , vd_end = 3.0, vd_step = 0.1
# flo mixer frequency = 283.0 GHz
# Hardware:
#    - synthesizer: MARCONI INSTRUMENTS,2024,112261/009,44533/446/04.07
#    - multimeter: HEWLETT-PACKARD,34401A,0,4-1-1
#
# lock = 1
# vbias = -0.599976
#
#    Vd1;    Vd2;    power;    delta
#      V;      V;       mW;       mW
     0.3;    0.3;     0.08;     0.08
     0.4;    0.4;     0.36;     0.27
 [...]
     2.9;    2.9;    28.30;     1.56
     3.0;    3.0;    30.18;     1.88


#    Vd1;    Vd2;    power;    delta
#      V;      V;       mW;       mW
     0.3;    0.0;     0.01;     0.01
     0.4;    0.0;     0.08;     0.07
[…]
     3.0;    0.0;     7.75;     0.44
# End of measure
# 2010-10-08T11:52:20.975415
# Total duration = 119 seconds
# End of file
```

7 User programs

This section describes the user programs. These programs are installed in `/home/introot/recng/bin`.
For convenience, all program names starts with the prefix `recng-`
For the software developers, there are debug versions of these programs: just add the suffix `.Debug` to the program name.
The data files are installed in `/home/introot/recng/data`

7.1 recng-coil

This program tunes coils to remove the Josephson effect in junctions. It also draws graphs to check that the automatic setting is correct. `recng-coil` can be applied only on band 3 and 4.
By default it tunes all the channels of a band, but it is possible to tune separately channels. The tuning may be better if channels are sequentially tuned, instead of simultaneously.
The graphic files are generated in the subdirectory B1, B2, B3, B4 according to the band number.

7.1.1 Syntax

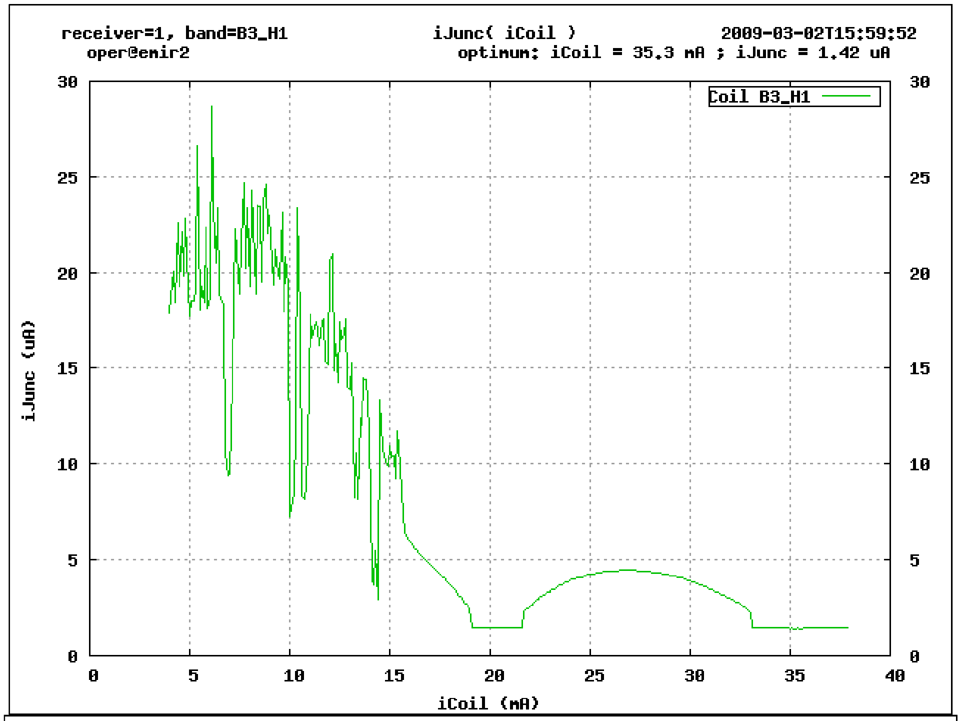
```
Recng Coil - Tune coils to remove the Josephson effect in junctions
Usage:  recng-coil [options]
Options:
        -b=bandNum    Specify band to tune [3,4]
        -c=a,b,c      Specify channel names to tune. If empty, tune all coils

        -v            Display version information
        -h, -?        Display help

Channel names:
        Band 3: B3_V1, B3_H1
        Band 4: B4_V1, B4_V2, B4_H1, B4_H2

Example: recng-coil -b=4 -c=B4_V1,B4_H1
```

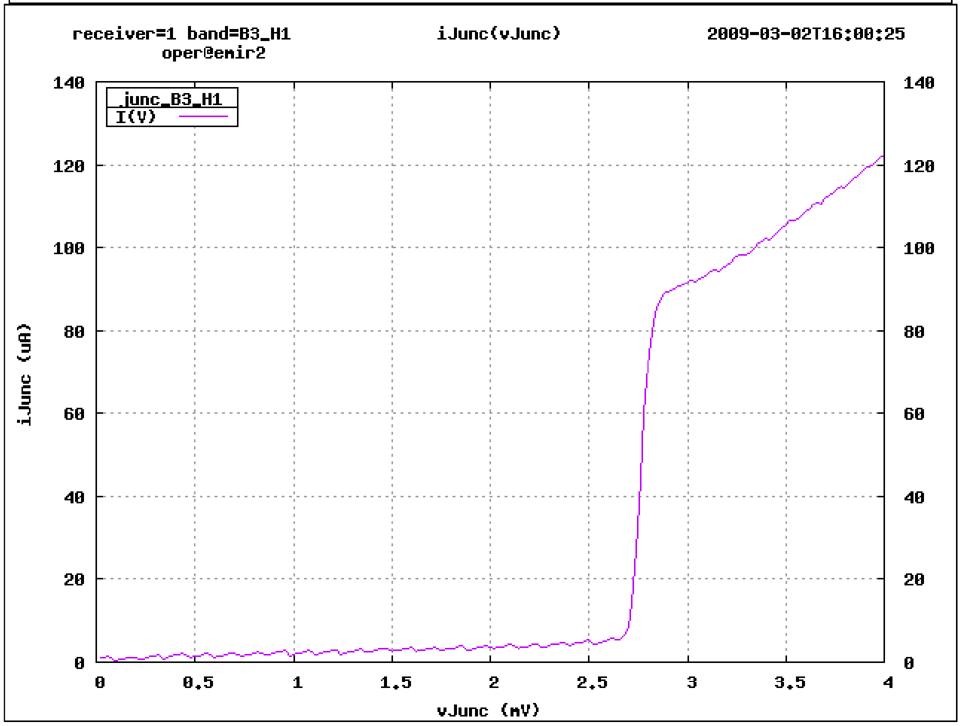
7.1.2 Example

```
$ recng-coil -b=3 -c=B3_V1
Setting for this tuning
        BandNum = 3
        ChannelList = B3_V1

Step 0: Decrease loPower2
Starts individuals threads
[coil_B3_V1] Load parameters from database:
[coil_B3_V1]         iJuncMargin          =  0.10
[coil_B3_V1]         juncAcqRef           =  0.20
[coil_B3_V1]         juncStdRef           =  2.30
[coil_B3_V1]         juncZeroRef          =  0.01
[coil_B3_V1]         minCoilRef           =  4.00
[coil_B3_V1]         minFlatWidth         =  2.00
[coil_B3_V1]         maxCoilRef           = 40.00
[coil_B3_V1]         maxDerived           =  0.15
[coil_B3_V1]         maxJosephsonCurrent  =  1.50
[coil_B3_V1]         iCoilNegExploration  = -1.00
[coil_B3_V1]         iCoilPosExploration  =  2.00

[coil_B3_V1] Thread starts
```

```
[coil_B3_V1] Step 1: Polarize junction with ref= 0.2 mV

[coil_B3_V1] Step 2: Start Coils and acquire iJunc(iCoil)
[coil_B3_V1]     Clear memory effect
[coil_B3_V1]     Acquiring iJunc(iCoil) on [4 mA, 40 mA]

[coil_B3_V1] Step 3: Apply magnetic field
[coil_B3_V1]     Searching for a flat level range inside [  4.0 , 21.9 ]
[coil_B3_V1]         Searching parameters:
[coil_B3_V1]                   minCoilRef =  4.00
[coil_B3_V1]                   maxCoilRef = 21.90
[coil_B3_V1]                   maxDerived =  0.15
[coil_B3_V1]                  iJuncMargin =  0.10
[coil_B3_V1]                 minFlatWidth =  2.00
[coil_B3_V1]                       minMin =  -inf
[coil_B3_V1]         Found minimum ( iCoil = 16.4 , iJunc = -20.2214 )
[coil_B3_V1]         Flat level range candidate: [ 14.7 , 18.4 ]
[coil_B3_V1]         OK: the flat level range is accepted (width = 3.7)

[coil_B3_V1]     Searching for a flat level range inside [ 22.1 , 40.0 ]
[coil_B3_V1]         Searching parameters:
[coil_B3_V1]                   minCoilRef = 22.10
[coil_B3_V1]                   maxCoilRef = 40.00
[coil_B3_V1]                   maxDerived =  0.15
[coil_B3_V1]                  iJuncMargin =  0.10
[coil_B3_V1]                 minFlatWidth =  2.00
[coil_B3_V1]                       minMin =  -inf
[coil_B3_V1]         Found minimum ( iCoil = 30.2 , iJunc = -20.2336 )
[coil_B3_V1]         Flat level range candidate: [ 27.2 , 30.2 ]
[coil_B3_V1]         OK: the flat level range is accepted (width = 3)

[coil_B3_V1]     Build list of candidates
[coil_B3_V1]     Applying optimal iCoil
[coil_B3_V1]     Optimizing Josephson current
[coil_B3_V1]     Optimal iCoil = 28.1
[coil_B3_V1]     Plot iJunc(iCoil)
[coil_B3_V1]     Output = /home/oper/B3/r1_B3_V1.coil.png

[coil_B3_V1] Step 4: Plot iJunc(vJunc)
[coil_B3_V1]     Acquire I(V) on [0 mV, 4 mV]
[coil_B3_V1]     Plot iJunc(vJunc)
[coil_B3_V1]     Output = /home/oper/B3/r1_B3_V1.junc.png

[coil_B3_V1] Thread exits

==end of tune
```

7.1.3 Graph examples





7.2 recng-dump

This program dumps the receiver internal values on the standard output.

7.2.1 Syntax

```
Recng Print Detailed status
Print detailed status of the receiver on the standard output
Usage: recng-dump [options]
Options:
  -v        Display version information
  -h, -?    Display help
```

7.2.2 Example

```
$ recng-dump
# hostname = ant22b
# Tue Feb 5 16:35:41 2013
H_ifLevel               adc_H_ifLevel.actualValue                 1.07
V_ifLevel               adc_V_ifLevel.actualValue                 1.87
band1.GunnBias          dac_B1_GunnBias.requestedValue            7.80
band1.HarmMixerBias     dac_B1_HarmMixerBias.requestedValue       2.10
[…]
```

7.3 recng-reset-lo1Ref

This program reset the Lo1Ref

7.3.1 Syntax

```
$ recng-reset-lo1Ref -h
Recng ResetLo1Ref - Reset Lo1Ref
Usage: recng-reset-lo1Ref [options]
Options:
  -c=channels     Specify channels to reset. channels is a coma-separated list.
                       Valid channels are {A,B}
  -v              Display version information
  -h, -?          Display help

Example:  recng-reset-lo1Ref -c=A,B

Return code:
        - 0: OK, resetting is done
        - 1: An error has happened
```

7.3.2 Example

```
$ recng-reset-lo1Ref -c=A,B
ant12b:resetLo1Ref:    Arguments:
ant12b:resetLo1Ref:            channels= A,B
ant12b:resetLo1Ref:    reset Lo1Ref A
ant12b:resetLo1Ref:    reset Lo1Ref B
ant12b:resetLo1Ref:    OK tuning done
```

7.3.3 Wrapper resetLo1Ref

For convenience, there is a wrapper called `resetLo1Ref` to avoid typing the –c argument

```
Trivial wrapper for recng-reset-lo1Ref
    'resetLo1Ref ch1' <=> 'recng-reset-lo1Ref -c=ch1'
```

```
'resetLo1Ref ch1 ch2' <=> 'recng-reset-lo1Ref -c=ch1,ch2'

example: resetLo1Ref 1 2
```

7.4 recng-lo

This program is a wrapper that runs recng-lo-gunn or recng-lo-yig according to the band number.

7.4.1 Syntax

Without arguments, `recng-lo` prints the help for `recng-lo-gunn` and `recng-lo-yig`.
`recng-lo` passes all its arguments to the real tuning subprogram.

7.5 recng-lo-gunn

This program tunes the gunn local oscillator.

7.5.1 Syntax

```
$ recng-lo-gunn -h
Recng LoGunn - Tune Local Oscillator (gunn)
Usage: recng-lo-gunn [options]
Options:
  -b=bandNum      Specify band to tune (1,2,3)
  -f=frequency    Specify the LO frequency in GHz (with multiplier)
  -d=[-|+]        Specify the deltaF, '+' or '-'. Default is '-'.

  -v              Display version information
  -h, -?          Display help

Example:    recng-lo-gunn -b=1 -f=90.0
```

7.5.2 Example

```
$ recng-lo -b=1 -f=90
ant12b:loGunn:    ReceiverID = 100
ant12b:loGunn:    Setting for this tuning:
ant12b:loGunn:            bandNum = 1
ant12b:loGunn:            flo     = 90.000 GHz
ant12b:loGunn:            deltaF  = MINUS
ant12b:loGunn:    FGUNN= 90 GHz
Load settings from h183
ant12b:loGunn:    Step 1: Configure the LO in safe mode
ant12b:loGunn:
ant12b:loGunn:    Step 2: Set Motors and DACs
ant12b:loGunn:    LO settings:
ant12b:loGunn:            FGunn = 90
ant12b:loGunn:            GunnBias = 7.8
ant12b:loGunn:            HarmMixerBias = 2.7
ant12b:loGunn:            HarmMixerPower = 3.3
ant12b:loGunn:            LoFreq = 4.854
ant12b:loGunn:            LoPower1 = 3.5
ant12b:loGunn:            LoPower2 = 2
ant12b:loGunn:            LoopGain = 5.01
ant12b:loGunn:            PowerGunn = 6.7
```

```
ant12b:loGunn:
ant12b:loGunn:    Step 3: Close Loop and optimize LoFreq
ant12b:loGunn:    Optimal loFreq = 4.833
ant12b:loGunn:
ant12b:loGunn:    Step 4: Set Offset Voltage and Gunn Bias
ant12b:loGunn:    Increase loFreq until OffsetVoltage < 5.00
ant12b:loGunn:    loFreq = 4.8370 ; offsetVoltage = 4.7887
ant12b:loGunn:    Increase gunnBias until OffsetVoltage > 5.00
ant12b:loGunn:    Optimal gunnBias = 7.83968
ant12b:loGunn:
ant12b:loGunn:    Step 5: Find max PllIfLevel(harmMixerBias)
ant12b:loGunn:    Max found: harmMixerBias = 0, pllIfLevel = 5.74997
ant12b:loGunn:    Tuning OK
ant12b:loGunn:
ant12b:loGunn:    End of tuning
```

7.6 recng-lo-yig

This program tunes the YIG local oscillator.

7.6.1 Syntax

```
$ Recng LoYig - Tune Local Oscillator (YIG)
Usage: recng-lo-yig [options]
Options:
  -b=bandNum      Specify band to tune (4)
  -f=frequency    Specify the LO frequency in GHz (with tripler)
  -q              Optimize only the PLL. The PLL must be already locked

  -v              Display version information
  -h, -?          Display help
```

7.6.2 Example

```
$ recng-lo-yig -b=4 -f=300
ant12b:loYig:    Setting for this tuning:
ant12b:loYig:            bandNum = 4
ant12b:loYig:            flo     = 100.000 GHz
ant12b:loYig:
ant12b:loYig:    ReceiverID = 100
Load settings from 'vbias_b4YigLo0'
ant12b:loYig:    Step 0: Load nominal values
ant12b:loYig:            vdb = 3
ant12b:loYig:            md = 0
ant12b:loYig:            vde = 3
ant12b:loYig:            vge = -0.3
ant12b:loYig:            vd1 = 0
ant12b:loYig:            vg1 = -0.3
ant12b:loYig:            vd2 = 0
ant12b:loYig:            vg2 = -0.3
ant12b:loYig:    Step 1: Open PLL
ant12b:loYig:    Step 2: Apply Vbias = -0.4
ant12b:loYig:    Step 3: Tune Yig (try to lock PLL)
ant12b:loYig:       OK: PLL is locked for yig frequency = 16.7241 GHz
ant12b:loYig:    Step 4: Optimize Yig
ant12b:loYig:       OK: Yig PLL is optimized ( 16.7241 GHz )
ant12b:loYig:    End of tuning
```

7.7 recng-park-lo

This programs parks the local oscillator in a safe position (parking) that does not interfere with others LOs.

LO Type	Parking position

Gunn	loop=0, sweep=0,
	loPower2=0

Yig	pll_open=1, yig_freq=15GHz,
	vd1=0, vd2=0

7.7.1 Syntax

```
Recng ParkLo – Send LO to a parking position that does not interfere with
others LOs.
Usage: recng-park-lo [options]
Options:
  -b=bandNum     Specify band to park [1,4]

  -v              Display version information
  -h, -?          Display help

Example: recng-park-lo -b=1
```

7.7.2 Example

```
$ recng-park-lo -b=4
ant22b:parkLo:    Park LO bandNum = 4
ant22b:parkLo:    End of parking LO
```

7.8 recng-mixer

This program tunes the harmonic mixer in the cryostat.
The graphic files are generated in the subdirectory B1, B2, B3, B4 according to the band number.

7.8.1 Syntax

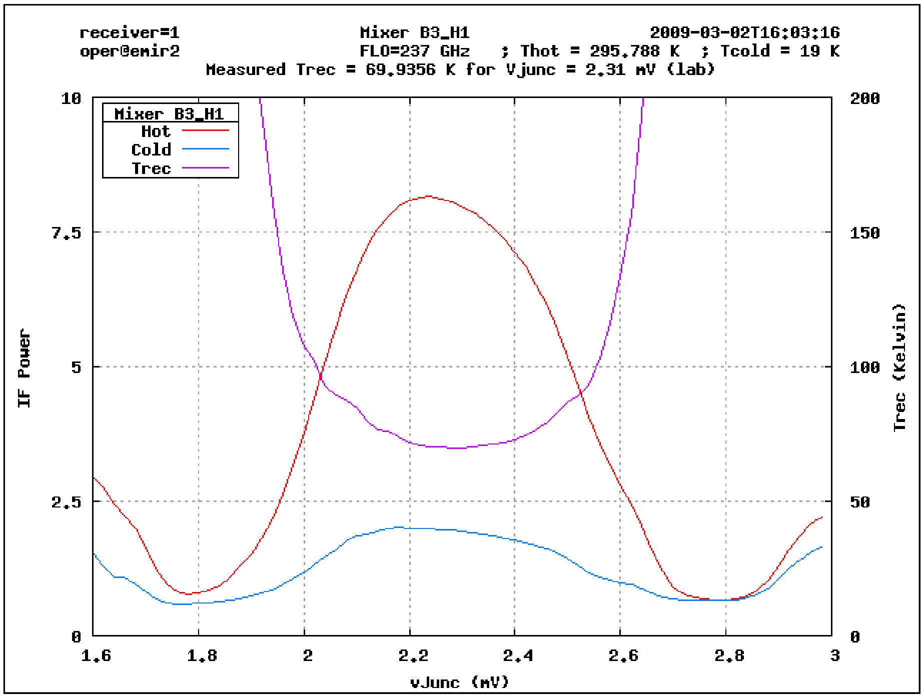
```
$ recng-mixer -h
Recng mixer - Tune Mixer
Usage: recng-mixer [options]
Options:
  -b=bandNum     Specify band to tune [1,4]
  -f=frequency   Specify the frequency LO1 in GHz
  -s=[L|U]       Specify the sideband (L)ower or (U)pper
  -r             Skip tuning, redraw only Trec graphs

  -v             Display version information
  -h, -?         Display help
```

7.8.2 Example

```
$ recng-mixer -b=3 -f=237 -s=L
ant12b:mixer:    ReceiverID = 100
ant12b:mixer:    Setting for this tuning
ant12b:mixer:       BandNum = 3
```

```
ant12b:mixer:       Frequency Lo = 237.000 GHz
ant12b:mixer:
ant12b:mixer:    ReceiverID = 100
ant12b:mixer:    Backshort Mixer
ant12b:mixer:    Load LO lab settings
Load settings from h244
ant12b:mixer:    LO Settings:
ant12b:mixer:            FGunn = 79
ant12b:mixer:            GunnBias = 7.8
ant12b:mixer:            HarmMixerBias = 0
ant12b:mixer:            HarmMixerPower = 0.4
ant12b:mixer:            LoFreq = 2.581
ant12b:mixer:            LoPower1 = 6.5
ant12b:mixer:            LoPower2 = 6.5
ant12b:mixer:            LoopGain = 4.835
ant12b:mixer:            PowerGunn = 5.73
ant12b:mixer:
ant12b:mixer:    Load settings from m560-554.b3.lsb
ant12b:mixer:    Mixer settings:
ant12b:mixer:            Attenuation_B3_H1 = 7
ant12b:mixer:            Attenuation_B3_V1 = 7
ant12b:mixer:            Backshort_H = 8.27
ant12b:mixer:            Backshort_V = 8.43
ant12b:mixer:            Flo1 = 237
ant12b:mixer:            Ij_H = 25.8
ant12b:mixer:            Ij_V = 23.2
ant12b:mixer:            TCold_H = 19
ant12b:mixer:            TCold_V = 23
ant12b:mixer:            Vj_H = 2.31
ant12b:mixer:            Vj_V = 2.31
ant12b:mixer:
ant12b:mixer:    Step 1: Starting
ant12b:mixer:    Polarize junctions
ant12b:mixer:    Wait for calibration moving to 03_Amb ... OK
ant12b:mixer:
ant12b:mixer:    Step 2: Set Motors and Attenuators
ant12b:mixer:    Step 3: Optimize mixer current
ant12b:mixer:    Target current = 23.2 uA on junction B3_V1
ant12b:mixer:    Step 4: Compute Trec and draw graphs
ant12b:mixer:    Wait for calibration moving to 03_Amb ... OK
[mixer_B3_V1] Start acquiring IF level with Amb load
[mixer_B3_V1] Acquisition range: [1.6, 3]
[mixer_B3_H1] Start acquiring IF level with Amb load
[mixer_B3_H1] Acquisition range: [1.6, 3]
ant12b:mixer:    Wait for calibration moving to 03_Cold ... OK
[mixer_B3_V1] Start acquiring IF level with Cold load
[mixer_B3_V1] Acquisition range: [1.6, 3]
[mixer_B3_H1] Start acquiring IF level with Cold load
[mixer_B3_H1] Acquisition range: [1.6, 3]
[mixer_B3_H1] Output = /home/oper/B3/r1_B3_H1.mixer.png
[mixer_B3_V1] Output = /home/oper/B3/r1_B3_V1.mixer.png
Step 5: Restore observing mode
Polarize junctions
Wait for calibration moving to 03_Sky ... OK
End of tuning
```

7.8.3 Graph



7.8.4 Utilities

Front-end excel files must be converted to mixer data files with the following scripts:

- recng-MixerBshort_import-excel.py
- recng-Mixer2sb_import-excel.py

```
$ recng-Mixer2sb_import-excel.py
Convert an Front-End Excel File (text Tab format) to a mixer 2SB data file
Syntax:
        Mixer2sb_import-excel.py filename.txt

How to do:

1- Open your file in excel.
2- Copy and paste your array in a new excel document
3- "File | Save As", Save as type: Text (Tab delimited)
(give the name you want, for example "toto.txt"

Now execute the following command line:
Mixer2sb_import-excel.py toto.txt > my_mixer_filename.b1

The converted data are in my_mixer_filename.b1
```

```
$ recng-MixerBshort_import-excel.py
Convert an Front-End Excel File (text Tab format) to a mixer 1SB data file
Syntax:
        MixerBshort_import-excel.py filename.txt

How to do:

1- Open your file in excel.
2- Copy and paste your array in a new excel document
```

```
3- "File | Save As", Save as type: Text (Tab delimited)
(give the name you want, for example "toto.txt"

Now execute the following command line:
MixerBshort_import-excel.py toto.txt > my_mixer_filename.b2.lsb

The converted data are in my_mixer_filename.b2.lsb
```

7.9 PlotJunction

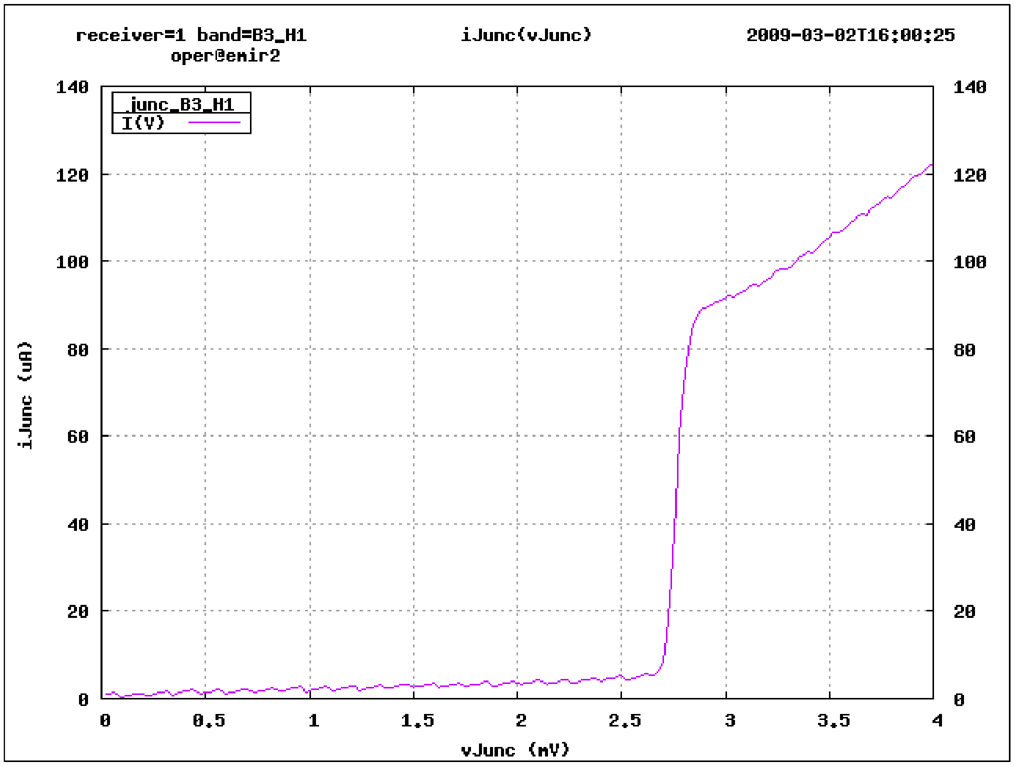
This program is used to plot the junction graph I(V)

7.9.1 Syntax

```
$ recng-plot-junction -h
Recng Plot Junction - Plot junction graph I(V)
Usage: recng-plot-junction [options]
Options:
        -b=bandNum    Specify band to plot
        -c=a,b,c      Specify channel names to plot. If empty, plot all junctions

        -v            Display version information
        -h, -?        Display help

Channel names:
        Band 1: B1_V1, B1_H1
        Band 2: B2_V1, B2_H1
        Band 3: B3_V1, B3_H1
        Band 4: B4_V1, B4_V2, B4_H1, B4_H2
```

7.9.2 Example

```
$ recng-plot-junction -b=3 -c=B3_H1
Setting for this tuning
        BandNum = 3
        ChannelList = B3_H1

Step 0: Decrease loPower2
Starts individuals threads
[junc_B3_H1] Thread starts
[junc_B3_H1] Unprotect junction
[junc_B3_H1] Plot iJunc(vJunc)
[junc_B3_H1]     Acquiring I(V) on [0 mV, 4 mV]
[junc_B3_H1]     Output = /home/oper/B3/r1_B3_H1.junc.png
[junc_B3_H1] Thread exits
==end of plot
```



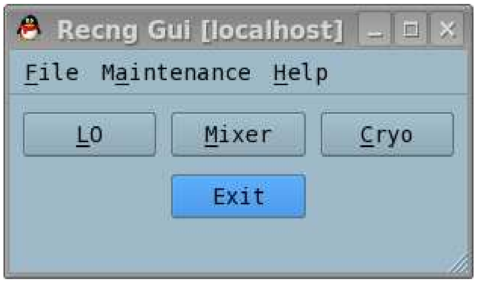
7.10 recng-gui

This program is specially designed for the front-end laboratory. It provides a graphical user interface (gui) for each receiver component.

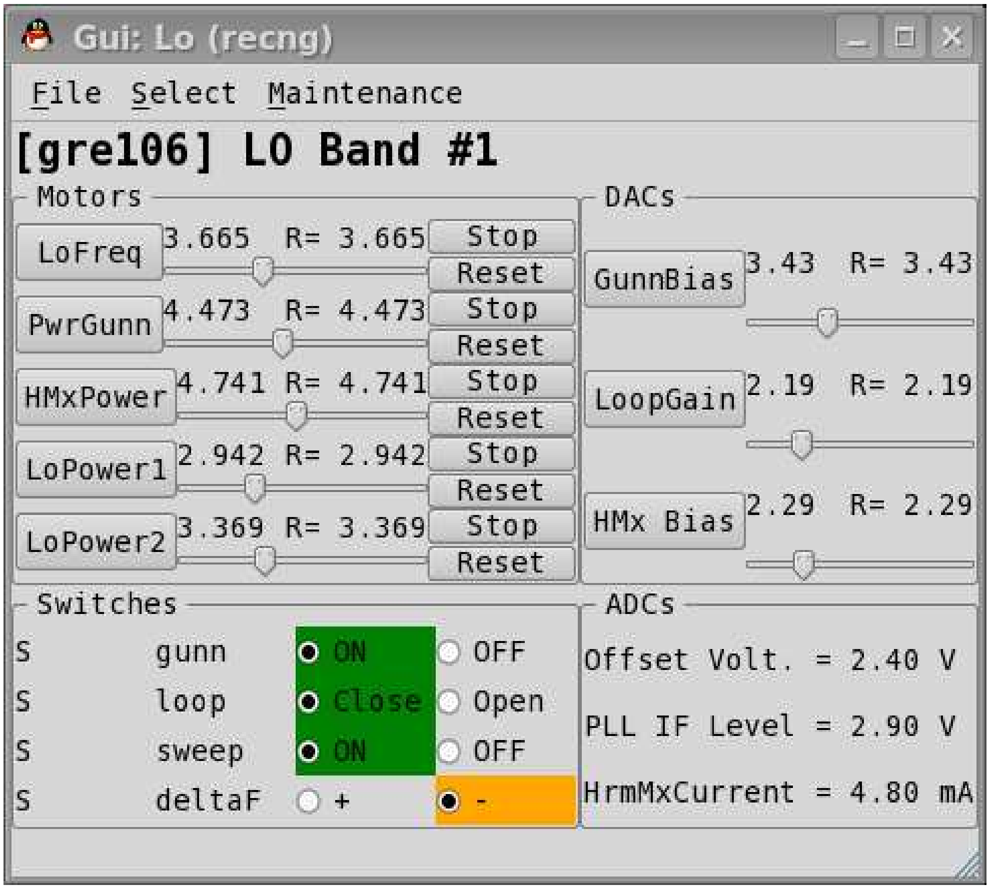
7.10.1 Syntax

```
$ recng-gui
```

7.10.2 Main window



7.10.3 Lo window (Gunn)



[gre106] LO Band #1

Motors

LoFreq 3.665 R= 3.665

PwrGunn 4.473 R= 4.473

HMxPower 4.741 R= 4.741

LoPower1 2.942 R= 2.942

LoPower2 3.369 R= 3.369

DACs

GunnBias 3.43 R= 3.43

LoopGain 2.19 R= 2.19

HMx Bias 2.29 R= 2.29

Switches

S gunn ● ON ○ OFF

S loop ● Close ○ Open

S sweep ● ON ○ OFF

S deltaF ○ + ● -

ADCs

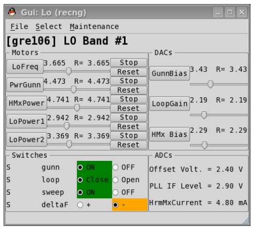
Offset Volt. = 2.40 V

PLL IF Level = 2.90 V

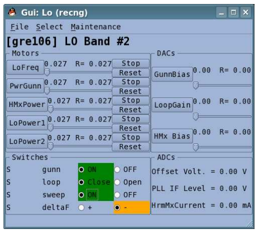
HrmMxCurrent = 4.80 mA

To avoid mistakes, each band has its own color:

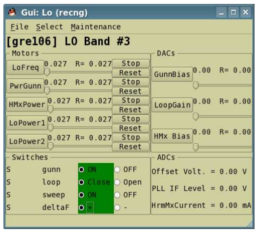
Band1



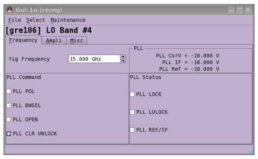
Band2



Band3



Band4



[gre106] LO Band #4

Yig Frequency 15.000 GHz

PLL

PLL CorV = -10.000 V

PLL IF = -10.000 V

PLL Ref = -10.000 V

PLL Command

□ PLL POL

□ PLL BWSEL

□ PLL OPEN

□ PLL CLR UNLOCK

PLL Status

□ PLL LOCK

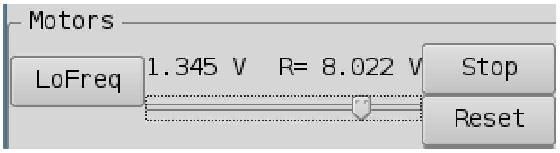
□ PLL LULOCK

□ PLL REF/IF

Motor widgets (LoFreq, PwGunn, HmxPwr, LoPower1, LoPower2)

This widget is used to drive the motor, and to display the current position.

Description



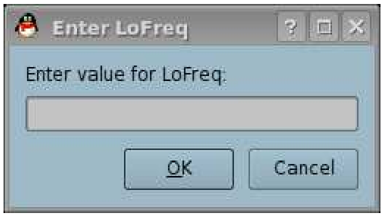
LoFreq 1.345 V R= 8.022 V

The requested value is displayed with a R prefix (here: R= 8.022).
The current position with no prefix (here: 1.345).
There is also a stop button and a reset button.

To modify the motor position, you can:

- Click on the button name to open a dialog box to enter the new motor value

- Move the slider

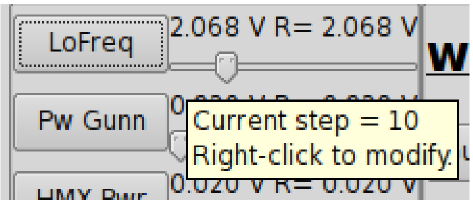


Enter LoFreq

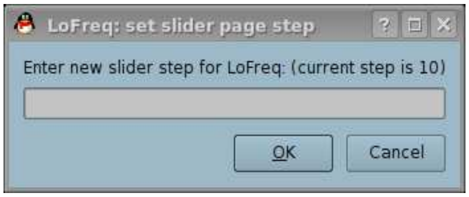
Enter value for LoFreq:

Motor dialog box

You can change the slider step by right-clicking on it



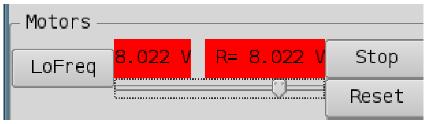
Current step = 10
Right-click to modify.



LoFreq: set slider page step

Enter new slider step for LoFreq: (current step is 10)

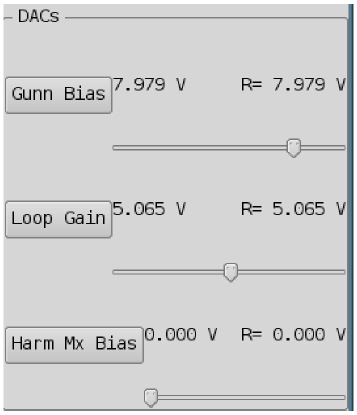
The unit for the slider is in motor raw unit (totally different from the motor unit which is displayed in Volt)



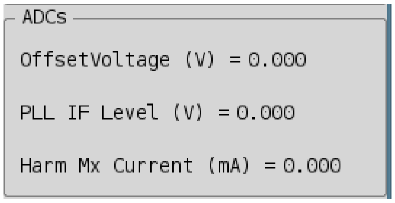
If the motor is disconnected from the CAN bus, the labels become red to indicate a problem.

DAC widget (GunnBias, LoopGain, HarmMxBias)
This widget is used to drive the DAC, and to display the current position.

Description
From the user point of view, a DAC is similar to a motor with an infinite speed.
So, the motor widget description applies also to the DAC widget



ADC widget (OffsetVoltage, PLL IF Level, HarmMxCurrent)



ADCs

OffsetVoltage (V) = 0.000

PLL IF Level (V) = 0.000

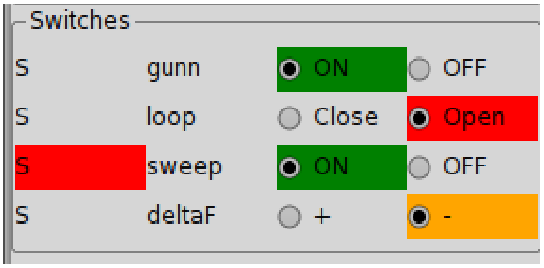
Harm Mx Current (mA) = 0.000

This widget is used to display the current ADC value.

Lo Switches widgets
This widget is used to command the LO switches and to display the status.

Description

Each radio-button pair represents a LO switch. It is a realistic representation of the LO physical front panel: color and switch order are taken from the LO hardware.



Switches

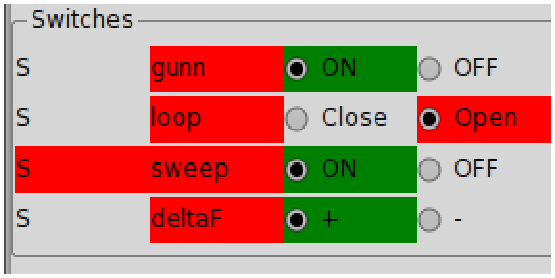
S gunn ● ON ○ OFF

S loop ○ Close ● Open

S sweep ● ON ○ OFF

S deltaF ○ + ● -

The radio button displays the current command applied on this switch.
For each switch, if the command matches the status, the status label (the S letter on the left side) is displayed with a normal background; otherwise this label is displayed with a red background.
Here the status for Sweep is red. It means that the status does not match the command.



Switches

S gunn ● ON ○ OFF

S loop ○ Close ● Open

S sweep ● ON ○ OFF

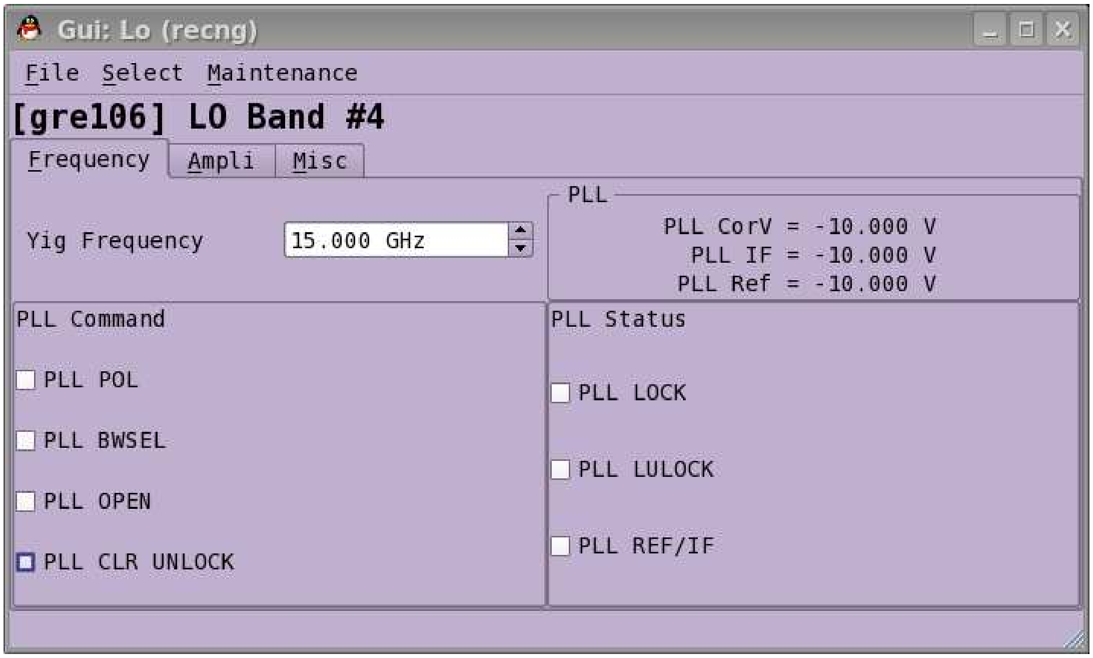
S deltaF ● + ○ -

If the LoSwitches device does not answer at all, the switch names (gunn, loop, sweep, deltaF) become red to indicate a problem.

7.10.4 Lo window (Yig)

For band #4, the window is different because the local oscillator is a Yig. There are 3 tabs: frequency, ampli and misc.

7.10.4.1 Frequency tab



[gre106] LO Band #4

Yig Frequency 15.000 GHz

PLL
PLL CorV = -10.000 V
PLL IF = -10.000 V
PLL Ref = -10.000 V

PLL Command
□ PLL POL
□ PLL BWSEL
□ PLL OPEN
□ PLL CLR UNLOCK

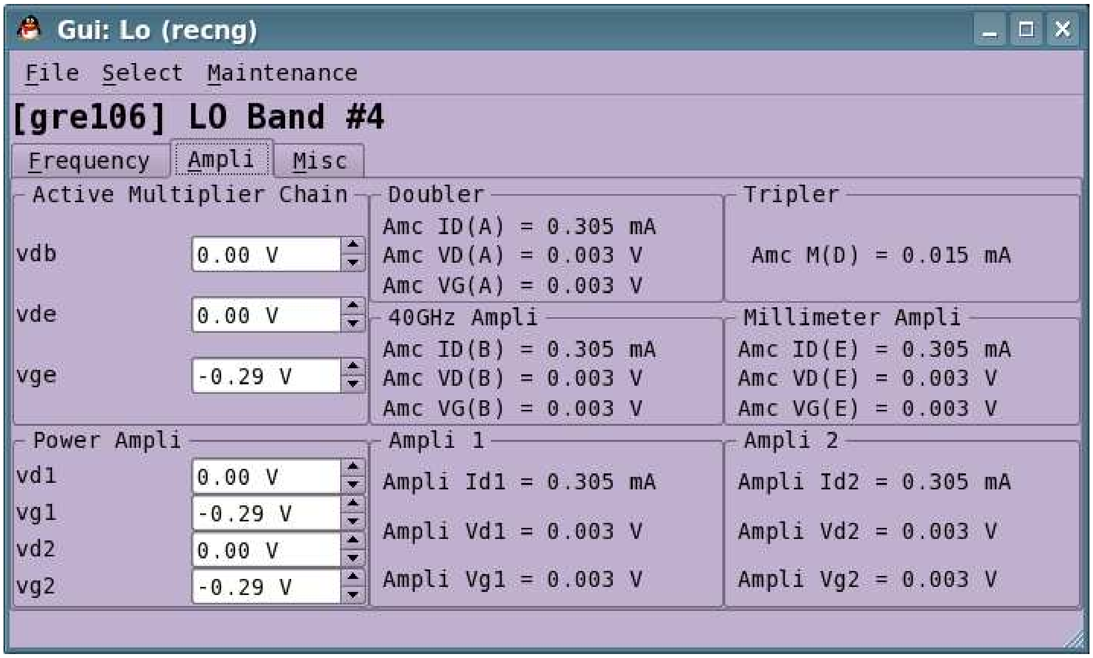
PLL Status
□ PLL LOCK
□ PLL LULOCK
□ PLL REF/IF

This window shows:

- The YIG frequency
- The PLL command and status

- The analog values of the PLL

7.10.4.2 Ampli tab



[gre106] LO Band #4

Active Multiplier Chain
vdb 0.00 V
vde 0.00 V
vge -0.29 V

Doubler
Amc ID(A) = 0.305 mA
Amc VD(A) = 0.003 V
Amc VG(A) = 0.003 V

Tripler
Amc M(D) = 0.015 mA

40GHz Ampli
Amc ID(B) = 0.305 mA
Amc VD(B) = 0.003 V
Amc VG(B) = 0.003 V

Millimeter Ampli
Amc ID(E) = 0.305 mA
Amc VD(E) = 0.003 V
Amc VG(E) = 0.003 V

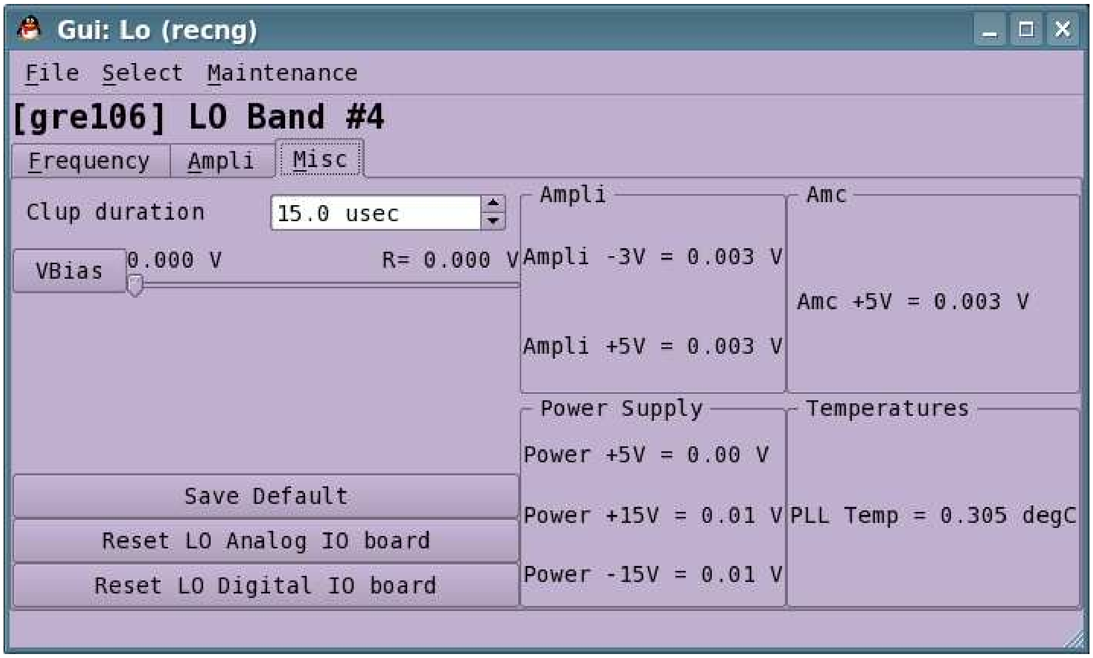
Power Ampli
vd1 0.00 V
vg1 -0.29 V
vd2 0.00 V
vg2 -0.29 V

Ampli 1
Ampli Id1 = 0.305 mA
Ampli Vd1 = 0.003 V
Ampli Vg1 = 0.003 V

Ampli 2
Ampli Id2 = 0.305 mA
Ampli Vd2 = 0.003 V
Ampli Vg2 = 0.003 V

There are software limit on the spinboxes to avoid damaging the millimeter amplifier.

7.10.4.3 Misc tab



[gre106] LO Band #4

Clup duration 15.0 usec
VBias 0.000 V R= 0.000 V

Ampli
Ampli -3V = 0.003 V
Ampli +5V = 0.003 V

Amc
Amc +5V = 0.003 V

Power Supply
Power +5V = 0.00 V
Power +15V = 0.01 V
Power -15V = 0.01 V

Temperatures
PLL Temp = 0.305 degC

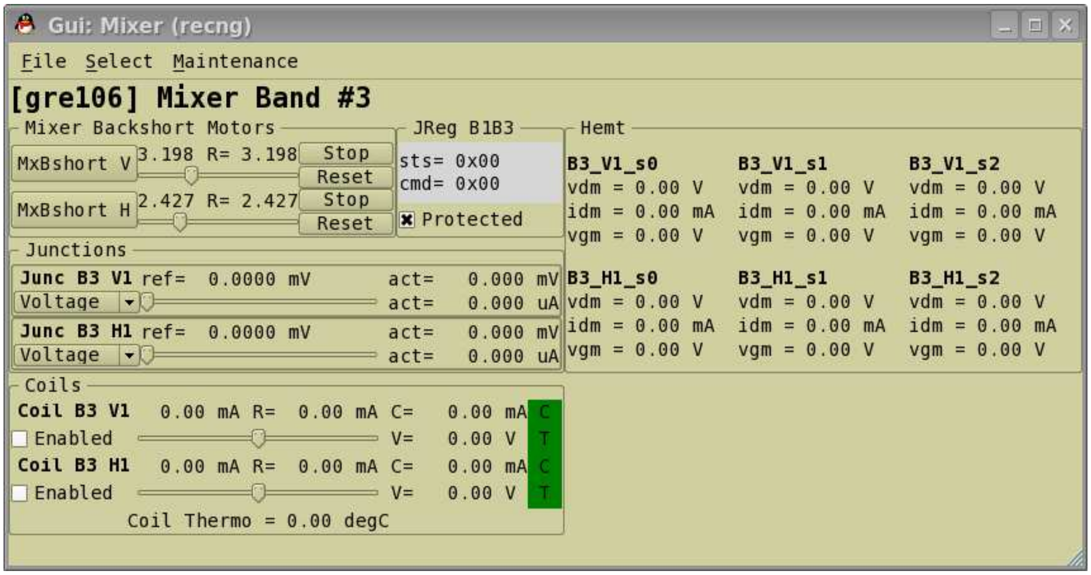
On this window, you can

- Read the power supplies
- Save default values in EEPROM. The EEPROM content is applied when the LO rack is powered. The saved value are:

1. YIG frequency
2. Ampli(vd1,vg1,vd2,vg2)
3. Amc (vdb,md, vde, vge), PLL commands)

- Reset the analog input/output board with button *Reset Lo Analog IO board*
- Reset the digital input/output board with button *Reset Lo Digital IO board*

7.10.5 Mixer window

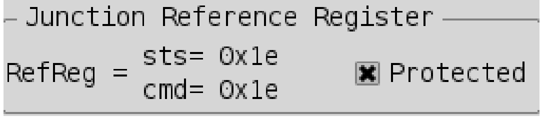


```
[gre106] Mixer Band #3
Mixer Backshort Motors                        JReg B1B3       Hemt
MxBshort V  3.198  R= 3.198                   sts= 0x00       B3_V1_s0          B3_V1_s1          B3_V1_s2
                                              cmd= 0x00       vdm = 0.00 V      vdm = 0.00 V      vdm = 0.00 V
MxBshort H  2.427  R= 2.427                                   idm = 0.00 mA     idm = 0.00 mA     idm = 0.00 mA
                                              ☒ Protected     vgm = 0.00 V      vgm = 0.00 V      vgm = 0.00 V
Junctions
Junc B3 V1  ref=  0.0000 mV        act=  0.000 mV             B3_H1_s0          B3_H1_s1          B3_H1_s2
Voltage                            act=  0.000 uA             vdm = 0.00 V      vdm = 0.00 V      vdm = 0.00 V
Junc B3 H1  ref=  0.0000 mV        act=  0.000 mV             idm = 0.00 mA     idm = 0.00 mA     idm = 0.00 mA
Voltage                            act=  0.000 uA             vgm = 0.00 V      vgm = 0.00 V      vgm = 0.00 V
Coils
Coil B3 V1    0.00 mA R=   0.00 mA C=    0.00 mA  C
□ Enabled                          V=    0.00 V   T
Coil B3 H1    0.00 mA R=   0.00 mA C=    0.00 mA  C
□ Enabled                          V=    0.00 V   T
           Coil Thermo = 0.00 degC
```

Mixer Backshort Motors

See description of LO motor.

Junction Reference Register

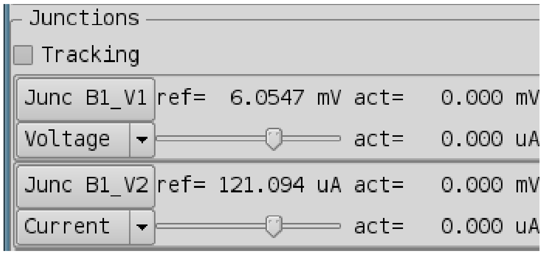


```
Junction Reference Register
RefReg =  sts= 0x1e        ☒ Protected
          cmd= 0x1e
```

This widget is used to display the junction reference register, and to protect/unprotect the junctions

Note: the is also a physical switch on the junction box

Junction

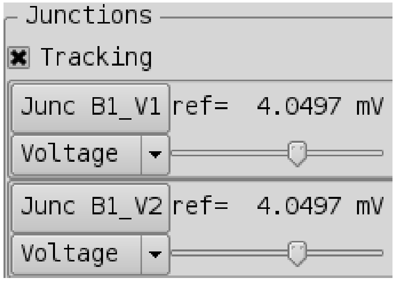


```
Junctions
□ Tracking
Junc B1_V1 ref=   6.0547 mV act=   0.000 mV
Voltage                      act=   0.000 uA
Junc B1_V2 ref= 121.094 uA act=   0.000 mV
Current                      act=   0.000 uA
```

This widget is used to control the junctions.
You can set the junction reference by clicking on the junction button name, or by moving the slider.

Use the combo box to switch from current/voltage reference.
On the screenshot, Junction B1_V1 has a voltage reference, and Junction B1_V2 has a current reference.

Tracking



Junctions

☒ Tracking

Junc B1_V1 ref= 4.0497 mV

Voltage

Junc B1_V2 ref= 4.0497 mV

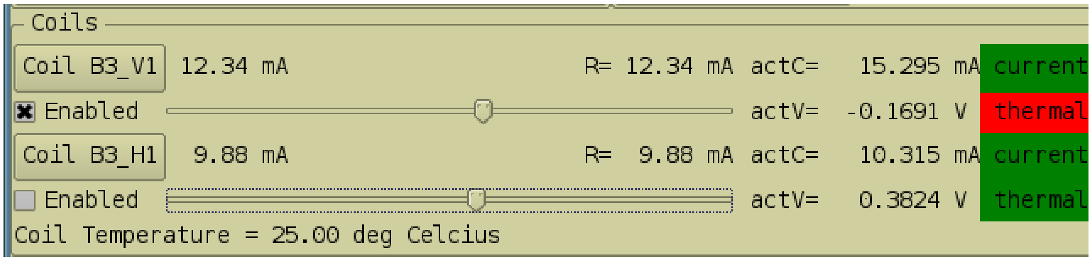
Voltage

For double side band mixers, the same-polarity junctions must have the same reference values, otherwise the tuning is wrong. Therefore, there is a special tracking mode to change simultaneously the two junction references.

If needed, you can disable the tracking mode by clearing the *Tracking* checkbox.

Coil

This widget is used to control the coils, to apply a magnetic field to cancel the Josephson current in junction.



Coils

Coil B3_V1 12.34 mA R= 12.34 mA actC= 15.295 mA 🟩 current

☒ Enabled actV= -0.1691 V 🟥 thermal

Coil B3_H1 9.88 mA R= 9.88 mA actC= 10.315 mA 🟩 current

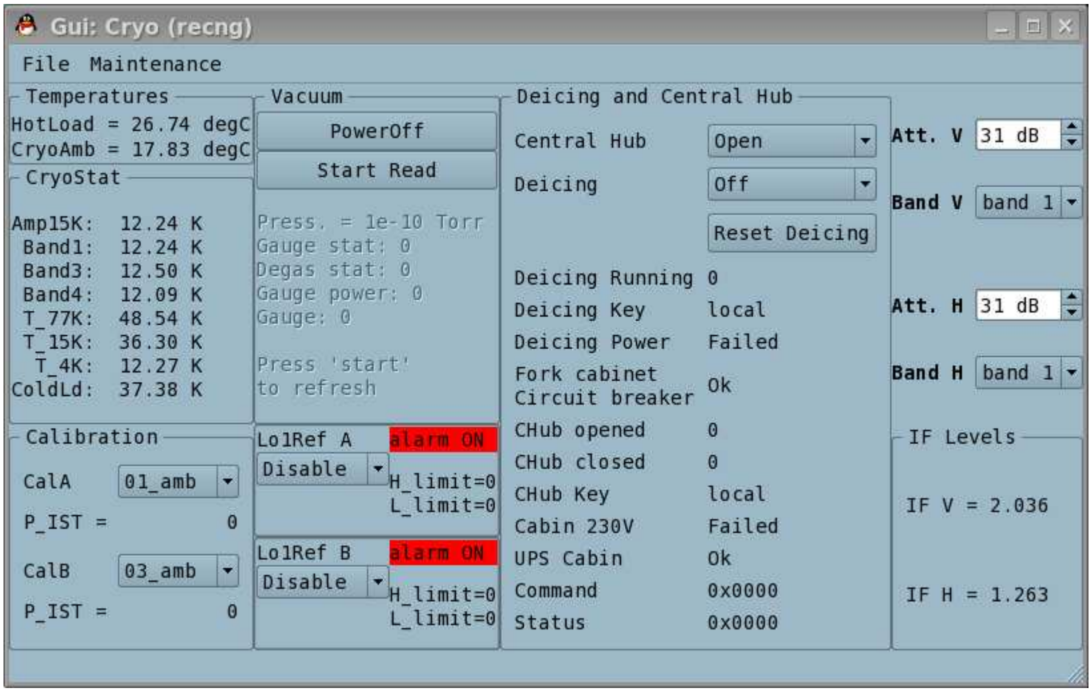
☐ Enabled actV= 0.3824 V 🟩 thermal

Coil Temperature = 25.00 deg Celcius

There are coils only in band 3 and 4.

Click on the *Enabled* check box to activate the coil, and then change the coil current with the slider or by clicking on the coil button. On the right side, you have *current* and *thermal* indicators. They become red, if a problem occurs.

7.10.6 Cryo window



Gui: Cryo (recng)

File Maintenance

Temperatures

HotLoad = 26.74 degC
CryoAmb = 17.83 degC

CryoStat

Amp15K: 12.24 K
Band1: 12.24 K
Band3: 12.50 K
Band4: 12.09 K
T_77K: 48.54 K
T_15K: 36.30 K
T_4K: 12.27 K
ColdLd: 37.38 K

Calibration

CalA 01_amb
P_IST = 0
CalB 03_amb
P_IST = 0

Vacuum

PowerOff
Start Read

Press. = 1e-10 Torr
Gauge stat: 0
Degas stat: 0
Gauge power: 0
Gauge: 0

Press 'start'
to refresh

Lo1Ref A alarm ON
Disable H_limit=0 L_limit=0

Lo1Ref B alarm ON
Disable H_limit=0 L_limit=0

Deicing and Central Hub

Central Hub Open
Deicing Off
Reset Deicing

Deicing Running 0
Deicing Key local
Deicing Power Failed
Fork cabinet Circuit breaker Ok
CHub opened 0
CHub closed 0
CHub Key local
Cabin 230V Failed
UPS Cabin Ok
Command 0x0000
Status 0x0000

Att. V 31 dB
Band V band 1
Att. H 31 dB
Band H band 1

IF Levels

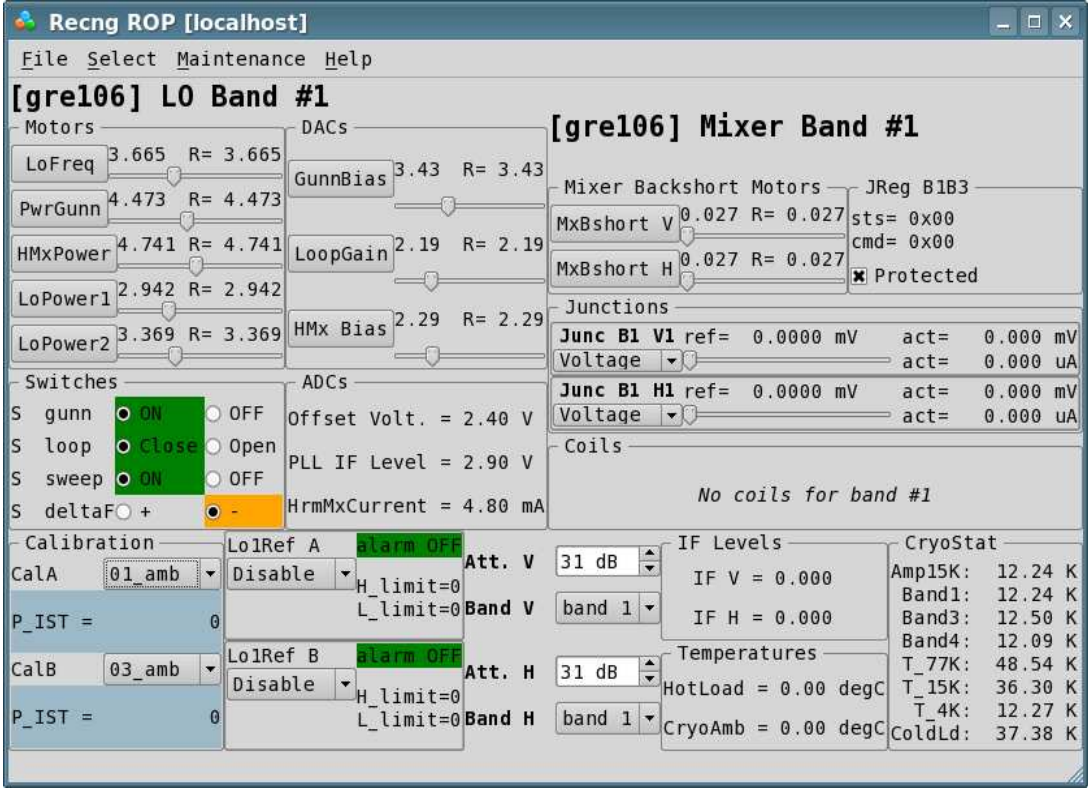
IF V = 2.036
IF H = 1.263

Vacuum

The vacuum widget has a auto power off timeout (120 secs), to avoid damaging the vacuum sensor.

7.11 Rop

Rop (Receiver OPerator) is a special version of recng-gui.
All components are grouped on only one window; rarely used devices (such Vacuum, Hemt) are not displayed. The target audience for this program is the IRAM telescope operators.



[gre106] LO Band #1

Motors
LoFreq 3.665 R= 3.665
PwrGunn 4.473 R= 4.473
HMxPower 4.741 R= 4.741
LoPower1 2.942 R= 2.942
LoPower2 3.369 R= 3.369

DACs
GunnBias 3.43 R= 3.43
LoopGain 2.19 R= 2.19
HMx Bias 2.29 R= 2.29

Switches
S gunn ● ON ○ OFF
S loop ● Close ○ Open
S sweep ● ON ○ OFF
S deltaF ○ + ● -

ADCs
Offset Volt. = 2.40 V
PLL IF Level = 2.90 V
HrmMxCurrent = 4.80 mA

[gre106] Mixer Band #1

Mixer Backshort Motors
MxBshort V 0.027 R= 0.027
MxBshort H 0.027 R= 0.027

JReg B1B3
sts= 0x00
cmd= 0x00
☒ Protected

Junctions
Junc B1 V1 ref= 0.0000 mV act= 0.000 mV
Voltage act= 0.000 uA
Junc B1 H1 ref= 0.0000 mV act= 0.000 mV
Voltage act= 0.000 uA

Coils
No coils for band #1

Calibration
CalA 01_amb
P_IST = 0
CalB 03_amb
P_IST = 0

Lo1Ref A alarm OFF
Disable
H_limit=0
L_limit=0

Att. V 31 dB
Band V band 1

Lo1Ref B alarm OFF
Disable
H_limit=0
L_limit=0

Att. H 31 dB
Band H band 1

IF Levels
IF V = 0.000
IF H = 0.000

Temperatures
HotLoad = 0.00 degC
CryoAmb = 0.00 degC

CryoStat
Amp15K: 12.24 K
Band1: 12.24 K
Band3: 12.50 K
Band4: 12.09 K
T_77K: 48.54 K
T_15K: 36.30 K
T_4K: 12.27 K
ColdLd: 37.38 K

7.11.1 Syntax

```
$ recng-rop
```

7.12 recng-check-software

recng-check-software.py checks that all the needed processes to use the receiver are running.

7.12.1 Syntax

```
recng-check-software.py
```

7.12.2 Example

```
$ recng-check-software.py
==========================
Check Recng software
==========================
Check Can Controllers:
tpmc816drv          14216   4
Can Controllers: Ok
Check CanManager:
```

```
CanManager: Ok
Check database:
Database: Ok
Check recng-server
recng-server: Ok
Check 22GHz receiver
22GHz: Ok
Check Subreflector
subreflector: Ok
RECNG Receiver: Ok
```

7.13 Rmc

rmc is an alias for program `receiverMultiCommand.py`. It is useful to run the same command on all the antenna.

7.13.1 Syntax

```
Run a command on several antennas
Version $Revision: 1.1.2.10 $ $Date: 2010/11/15 11:12:51 $

Usage:
        receiverMultiCommand.py [-nofilter] [-antenna=XYZ..] [-timeout=X] -
command="cmd args"
Arguments:
        -nofilter : do not filter output for antenna
                Default: filtered output on the antenna name

        -antenna=XYZ : list of antenna
                Default: use the $TELESCOPE environment variable
                Example: -antenna=1,2,3,4,5,6

        -timeout=T : timeout in seconds
                Default: no timeout
                Example: -timeout=2

        -command="cmd args" : command to run.
                Example: -command="lo.py -b=1"

Arguments can be abbreviated.
Example:
        receiverMultiCommand.py -n -a=1,2,3 -t=2  -c="lo.py -b=1"
```

7.13.2 Examples

To reboot all the antenna

```
rmc -a=1,2,3,4,5,6  -c="reboot"
```

To tune all the lo on band #1

```
rmc –a=1,2,3,4,5,6 -c="recng-lo –b=1"
```

8 Daily Operation

8.1 Modify database configuration

The configuration settings are stored in several SQL files in directory /home/introot/recng/data/ Normals users are interesting only by the filenames starting with prefix *“conf-”*. The others filenames are for developers only.

To modify a setting, edit the appropriate file and then run recng-update-db.sh to update the database.

```
$ recng-update-db.sh
    cd /home/introot/recng/data
    rm -f recng.db
# Update recng.db ...
    sqlite3 recng.db < ./00-canid.sql
    sqlite3 recng.db < ./conf-calibration.sql
[...]
    sqlite3 recng.db < ./devices/YigGenerator.sql
    sqlite3 recng.db < ./receiverId.sql
# OK
```

Then you have to restart the applications.

Note: You should notify me by email each time you modify SQL file, so that I can archive it in the SVN repository. Otherwise your modifications may be lost after a reinstallation.

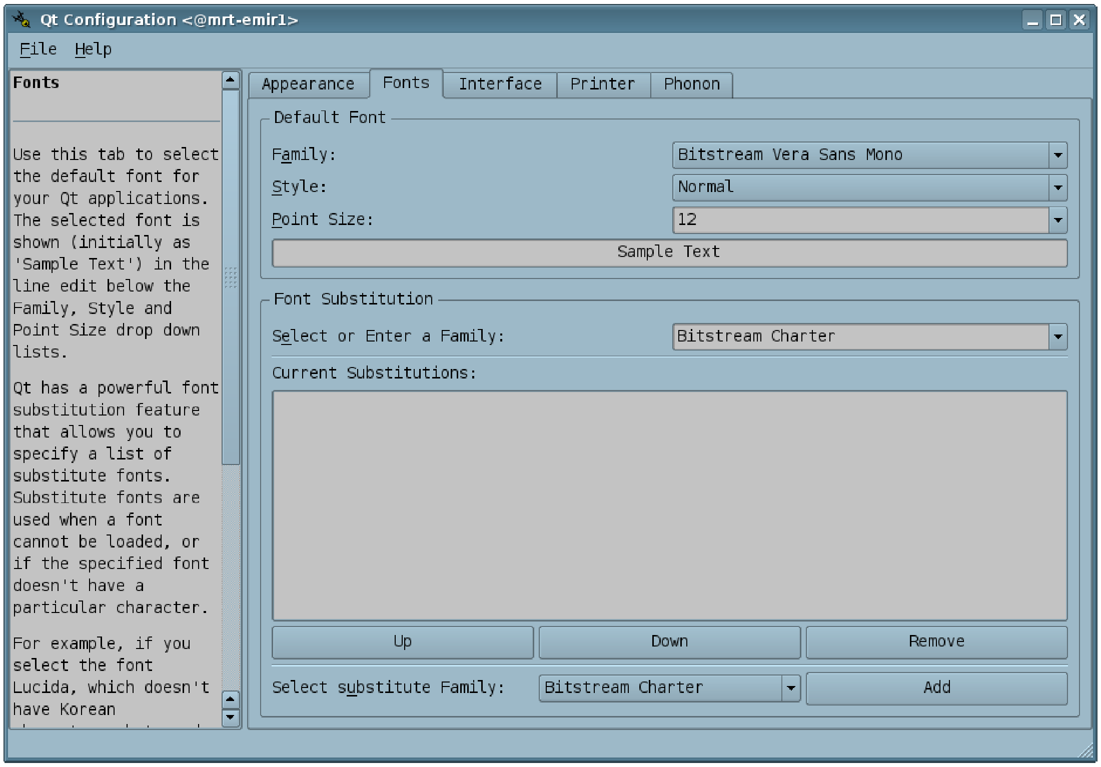
8.2 LO and Mixer data files

The data files for LO and Mixer are in /home/introot/recng/data/tuning
If you add a new file in this directory, you have to specify its names in the appropriate SQL configuration file (/home/introot/recng/data/conf-*.sql).

9 Tips

9.1 How to change the fonts size?

The graphical program use the default setting from your window manager, but you can change the default font with the program qtconfig



Fonts

Use this tab to select the default font for your Qt applications. The selected font is shown (initially as 'Sample Text') in the line edit below the Family, Style and Point Size drop down lists.

Qt has a powerful font substitution feature that allows you to specify a list of substitute fonts. Substitute fonts are used when a font cannot be loaded, or if the specified font doesn't have a particular character.

For example, if you select the font Lucida, which doesn't have Korean

Family: Bitstream Vera Sans Mono

Style: Normal

Point Size: 12

Sample Text

Select or Enter a Family: Bitstream Charter

Current Substitutions:

Select substitute Family: Bitstream Charter

To have a nicer display, you should select a fixed-width font. I recommends to use *Bitstream Vera Sans Mono*, with size=10 or size=12.

10 Bure1

All the previous chapters concern the software that run on the antenna computer.
As opposite, this section explain the software that run on bure1 (the main computer of the interferometer)

10.1 Build software on bure1

```
$ mkdir ~/develSVN
$ cd ~/develSVN
$ svn co svn://svn.iram.fr/PdB/ReceiverNG/trunk ReceiverNG
```

Then use the Makefile, to extract automatically the dependencies

```
$ cd ReceiverNG
$ qmake
$ make get_deps
```

Then build all the code

```
$ ./build.sh
```

To install the software, library and includes files in `/control/recng`

```
$ su -c "./install.sh bure1"
```

What is installed?

Description	Filename
Root directory	`/control/recng`
Function prototypes	`include/Fortran_Rpc_Client.hpp`

Libraries (binaries)	`libs/`

The libraries are available in two versions:

- Release version: to be used in the final product
- Debug version: they print debug messages when running

11 Bure1 programs

11.1 recng-StatusReceivers

This program gets status from all receivers in each antenna and stores the results in the shared memory.
The program uses one thread per connection, so you can add/remove antennas without restarting the program.

11.1.1 Syntax

```
$ recng-StatusReceivers -h
Status Receivers - Get Status from Receivers and store result in the shared
memory.
Usage: recng-StatusReceivers [options]
Options:
  -s=serverName  Specify a server name. Can be used several times.
                 If none, use default server names.

  -v             Display version information
  -h, -?         Display help
```

The –s option is useful for test.

11.1.2 Example

```
$ recng-StatusReceivers
```

The program connects to the default server names (ant12b, ant22b, … ant62b) and stores the results into the shared memory "RECE"

11.2 recng-print-status

This program prints the content of the shared memory

11.2.1 Syntax

```
$ recng-print-status -h
recng Print Status - Print Status Information
Print Status information from the RECE shared memory
Usage: recng-print-status [options]
Options:
  -r=N            Print only receiver N (N in [1,6])

  -v              Display version information
  -h, -?          Display help
```

11.2.2 Example

```
$ recng-print-status -r=1
rec1.band1.calibrationMode  0 (invalid)
rec1.band1.flo              109.35365300
rec1.band1.isLoLocked       1
rec1.band1.pllIfLevel       0
rec1.band2.calibrationMode  0 (invalid)
rec1.band2.flo              163.69999700
rec1.band2.isLoLocked       0
rec1.band2.pllIfLevel       0
rec1.band3.calibrationMode  3 (cold)
rec1.band3.flo              100.00000000
rec1.band3.isLoLocked       0
rec1.band3.pllIfLevel       0
rec1.band4.calibrationMode  0 (invalid)
rec1.band4.flo              321.12345700
rec1.band4.isLoLocked       0
rec1.band4.pllIfLevel       -9.99969
rec1.centralHub.isRemote    0
rec1.centralHub.position    0 (interm)
rec1.deicing.isActive       0
rec1.deicing.isPowerOk      0
rec1.deicing.isRemote       0
rec1.deicing.mode           0 (Off)
[...]
rec1.lastRefreshing 1288357198 (2010-10-29T14:59:58)
```