IRAM-COMP-054

Revision: 5
2015-09-29

Contact Author

Institut de RadioAstronomie Millimétrique

Emir software

Owner Sebastien BLANCHET

Keywords: receiver, software

Change Record

REVISION	DATE	AUTHOR	SECTION/ PAGE AFFECTED	REMARKS
5	2015-09-29	Blanchet	Emir-nika	New command for NIKA mirror
4	2013-09-09	Blanchet	Emir-plot-junction	Add new command line options
3	2012-03-12	Blanchet		Add new users programs (WarmJunction, emir-print-detailed-status, etc)
2	2011-11-08	Blanchet		Update for emir upgrade
1	2009-10-22	blanchet		Update build instructions

Content

1 Introduction

In 2009, IRAM has installed a new receiver called EMIR (Eight MIxer Receiver) at the IRAM 30M telescope. http://www.iram.es/IRAMES/mainWiki/EmirforAstronomers
In 2011, Emir has been upgraded with new dual side bands mixers.
This document is the documentation reference for the EMIR software: installation, technical documentation, daily usage and troubleshooting.

2 Main features

1. C++/Qt source code.
2. Two specific graphical interfaces for laboratory and observing.
3. Automatic lo, mixer and coil tuning.
4. Built-in Simulator.
5. XML RPC server for easy integration with observing software.
6. Embedded SQL database to store configuration.

3 Installation

3.1 Requirements

Hardware

The minimal requirements for hardware are:

- CPU: x86 CPU at 500 MHz
- RAM: 512 MB
- CAN controller: TPMC816 PMC card from Tews Technologies GmbH

For software development, the CAN controller is optional, because the software provides a CAN simulator.

Operating system

The software targets Linux Debian 5.0 (Lenny) i386 as main platform, but it should run on any computer with Linux 2.6.x or newer.

Mandatory libraries

The following libraries are required to build the software

Name	Description	Version	Download
g++	GNU C++ compiler	>= 4.1	http://gcc.gnu.org
subversion	Subversion is an open source version control system.	>= 1.5	http://subversion.tigris.org
Qt	C++ Cross-platform application framework	>= 4.4.3	http://www.qtsoftware.com
Sqlite	Embedded SQL database	>= 3.3.6	http://sqlite.org
CxxTest	Unary testing framework for C++	>= 3.10.1	http://cxxtest.tigris.org
Xmlrpc-c	XML-RPC for C and C++.	>= 1.06	http://xmlrpc-c.sourceforge.net
etherlabmaster	Ethercat Master	>= 1.5	http://www.etherlab.org

All these libraries are very common (except CxxTest and etherlabmaster), and you should find easily ready-to-install packages for your favorite Linux distribution.

Recommended software

The following programs are strongly recommended to modify easily the code.

Name	Description	Version	Download
Eclipse/CDT	C and C++ Integrated Development Environment (IDE) for the Eclipse platform.	>= 3.5	http://eclipse.org/cdt
doxygen	Automatic documentation system	>= 1.5.6	http://doxygen.org

All these software are very common, and you should find easily ready-to-install packages for your favorite Linux distribution.

3.2 Installation instructions for requirements

This procedure explains how to install the EMIR receiver software requirements on a fresh Debian 5.0 (Lenny) installation.

Install Linux Debian 5.0 for i386 on a computer.

Install development tools and libraries

```
# apt-get install rsync gcc g++ doxygen make gnuplot
# apt-get install manpages-dev graphviz sqlite3
# apt-get install libqt4-dev qt4-doc-html libqt4-sql-sqlite \
 libxmlrpc-c3-dev qt4-qtconfig libcurl4-openssl-dev subversion
```

Note: the Linux headers are required to build the CAN driver:

```
# apt-get install linux-headers-`uname -r`
```

Install useful packages:

```
# apt-get install openssh-server nmap xxdiff sqlite3-doc
```

Install cxxtest

```
$ tar xfz cxxtest-3.10.1.tar.gz
$ cd cxxtest
$ su
# mv cxxtestgen.py /usr/bin
# mv cxxtest /usr/include/
```

Create the oper account

```
$ su -c "adduser oper"
```

3.3 Build Emir software

Extract code from the repository

```
$ mkdir ~/develSVN
$ cd ~/develSVN
$ svn co svn://svn.iram.fr/30M/emir/trunk emir
```

Then use the Makefile to extract automatically the dependencies

```
$ cd emir
$ qmake-qt4
$ make get_deps
```

Build the TPMC816 driver (not required if you use only the simulation)

```
$ cd ~/develSVN/tpmc816
```

```
$ 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, but by default there will not be recreated at startup.

Therefore, create a startup script to recreate devices on boot:

```
# echo "rsync -a --devices /etc/udev/devices/ /dev/" > /create_devices.sh
# chmod +x /create_devices.sh
```

Modify /etc/rc.local to load driver and to call /create_devices.sh

```
# echo "/sbin/modprobe tpmc816drv" >> /etc/rc.local
# echo "/create_devices.sh" >> /etc/rc.local
```

Build the emir code

Note: If you do not use Debian 5.0, you have to modify firstly the file *Utils/conf/conf.pri* to specify the libraries locations

```
$ cd ~/develSVN/emir
$ ./build.sh
```

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

```
$ make doc
```

To install the software **and the default data** in /home/introot/emir

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

Warning:

If you wish to update only the software **without reinstalling the default data**, use:

```
$ make -f Makefile.install programs
```

Nevertheless it is safer to backup */home/introot/emir* first, so you can restore the original installation in case of errors.

```
$ tar cvfz ~/emir-backup-`date --iso`.tar.gz /home/introot/emir/
```

3.4 Configure environment

To run the EMIR software, several environment variables must be set.
The most convenient way to set them is to edit */etc/profile* to add the following lines:

```
# Define environment variable for EMIR software
export CAN_DB_FILE=/home/introot/emir/data/emir.db

INTROOT=/home/introot
PATH=${INTROOT}/bin:${INTROOT}/emir/bin:$PATH
export PATH
```

3.5 Initial test

For this initial test, we run

1. The CanManagers to enable the Can Over IP protocol
2. The emir 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_*"
$ emir-init-can.sh
$ xterm -e emir-simul &
```

(click on LO button to activate the LO simulation)

Now run the LO autotuning program:

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

3.6 Start application automatically

Add */home/introot/emir/bin/emir-init-receiver.sh* to */etc/rc.local* to initialize the emir software at the startup.

```
#!/bin/sh
# ############################################################################
# Copyright (C) 2008 Institut de RadioAstronomie Millimetrique
#
# $Id: init-receiver.sh 3717 2011-09-29 06:41:36Z blanchet $
#
# ABSTRACT: Initialize emir programs
#
# AUTHOR: Sebastien BLANCHET
#
# CREATION DATE: May 16, 2008
#
# $URL: svn://svn.iram.fr/30M/emir/trunk/scripts/init-receiver.sh $
# ############################################################################

cd `dirname $0`
source receiver_name.pri
PATH=/sbin:/usr/sbin:/bin:/usr/bin:/home/introot/${RECEIVER_NAME}/bin/
export CAN_DB_FILE=/home/introot/${RECEIVER_NAME}/data/${RECEIVER_NAME}.db

echo "Initialize Can BUS"
${RECEIVER_NAME}-init-can.sh
sleep 5

echo "Start EMIR calibration server"
emir-calibration-server &
sleep 30

emir-init-ampli

echo "Start ${RECEIVER_NAME} XML-RPC Server"
${RECEIVER_NAME}-server &
```

4 Internal programs

This section describes internal programs that drive the Emir receiver. These programs are executed automatically, therefore normal users are not expected run them directly.

4.1 CanManager

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

```
$ CanManager -h
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 -t1=100000
```

For the EMIR software, CanManager must listen on port udp/2500 and udp/2501

```
CanManager –d=/dev/tpmc816_0 –p=2500 –t1=20000
CanManager –d=/dev/tpmc816_1 –p=2501 –t1=20000
CanManager –p=2502 –t1=20000
```

- Port 2500 (device /dev/tpmc816_0) is used by the calibration system
- Port 2501 (device /dev/tpmc816_1) is used by the receiver
- Port 2502 (virtual CAN bus) is used by the calibration server (see appropriate section)

The CAN bus is initialized with the script */home/introot/emir/bin/emir-init-can.sh*

4.2 Calibration Server

To avoid collisions, emir-calibration-server is the **only** program that drives directly the calibration motors.

When starting, it initializes the calibration system (that has 3 motors: car1, car2 and filt).
Then it listens to calibration requests from other programs, and executes them.

Warning: For safety reasons, the program exits when an external program sends a power off command to a Middex motor. In this case you have to restart *emir-calibration-server*.

4.2.1 Syntax

```
$ emir-calibration-server -h
EMIR CalibrationServer – Server for the calibration system
Usage: emir-calibration-server [options]
Options:
  -v        Display version information
  -h, -?    Display help
```

4.2.2 Example

In this example, you can see the motor initializations and a calibration request: *setPosition(01_amb)*

```
$ emir-calibration-server
PID is 28831
CAN_MANAGER_TARGET='localhost'
Connect to CanManager localhost:2500
Find 23 records in CalibrationPosition
Find 11 records in MiddexPosition
Execute "/home/introot/emir/bin/release/emir-middex-init -m=filt"
[…]
Initializing motor filt ... OK
filt: reference is now initialized
```

```
Execute "/home/introot/emir/bin/release/emir-middex-init -m=car1"
[…]
Power Off car2 before initializing car1
Initializing motor car1 ... OK
car1: reference is now initialized
Execute "/home/introot/emir/bin/release/emir-middex-init -m=car2"
[…]
Initializing motor car2 ... OK
car2: reference is now initialized
Thu Feb 19 11:23:29 2009 CalibrationSystem::setPosition(01_amb)
Move 'middex_filt' to 'parking'      (do not wait)
Move 'middex_car2' to 'parking'      (wait)
Move 'middex_car1' to 'parking'      (wait)
Move 'middex_car1' to 'al1'          (wait)
Move 'middex_car2' to 'parking'      (wait)
Ok: new position is 01_amb           (it takes 1.5 sec)
```

4.2.3 Motor movements

The FILT motor can be driven independently, but the CAR1 and CAR2 motors must be perfectly synchronized. Therefore the following 4-steps procedure has been chosen to move the CAR1/CAR2 pair.

When a new request arrives:

1. move CAR2 to parking
2. move CAR1 to parking
3. move CAR1 to position
4. move CAR2 to position

4.2.4 Automatic tests

Automatic tests are provided to prove that motors never hit each other and to measure the maximum transition time.
Note: Emir-server is required to run this test. See emir-server section for more details.

```
$ TestCalibration.py -h
Syntax:
    TestCalibration.py [ -p | -tib1 | -tib2 | -ta1 | -ta2 ]

    -p          Test all positions
    -tib1       Test all intra-band transitions (1-way mode)
    -tib2       Test all intra-band transitions (2-way mode)
    -ta1        Test all transitions (1-way mode)
    -ta2        Test all transitions (2-way mode)

    -h          Display help
```

For example, test all positions

```
$ TestCalibration.py -p
========================================
Test all positions
========================================
(1/21) Test position '01_amb'
(2/21) Test position '01_cold'
(3/21) Test position '01_sky'
 […]
(21/21) Test position '24_sky'
```

To measure the maximum transition time:

Note: The following output comes from the simulator and the real receiver performances may be different.

```
$ TestCalibration.py -tib2
========================================
Test intra-band transitions in 2-way mode
========================================
1/42 test transition ['01_amb', '01_cold']      4.916760 sec
2/42 test transition ['01_amb', '01_sky']       2.007568 sec
[…]
39/42 test transition ['24_cold', '24_amb']      6.111430 sec
40/42 test transition ['24_cold', '24_sky']      3.008608 sec
41/42 test transition ['24_sky', '24_amb']       3.011832 sec
42/42 test transition ['24_sky', '24_cold']      3.015123 sec
The slowest transition is ['12_amb', '12_cold'], it takes 8.913323 seconds
```

4.3 Emir-dump

This application dumps the content of the EMIR shared memory.
This shared memory is used by several processes. For example to communicate between emir-server and emir-ethercat-plc.

4.3.1 Syntax

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

4.3.2 Example

```
$ emir-dump
band1.polarV.warmExpTime = 0 (1970-01-01T01:00:00)
band1.polarH.warmExpTime = 0 (1970-01-01T01:00:00)
band2.polarV.warmExpTime = 0 (1970-01-01T01:00:00)
band2.polarH.warmExpTime = 0 (1970-01-01T01:00:00)
band3.polarV.warmExpTime = 1329400801 (2012-02-16T15:00:01)
band3.polarH.warmExpTime = 1329400799 (2012-02-16T14:59:59)
band4.polarV.warmExpTime = 0 (1970-01-01T01:00:00)
band4.polarH.warmExpTime = 0 (1970-01-01T01:00:00)
nikaMirror 0
```

4.4 Emir-ethercat-plc

This program controls the EtherCAT PLC.

4.4.1 Syntax

```
$ emir-ethercat-plc -h
EthercatPLC Controller
Run EthercatPLC controller
Usage: emir-ethercat-plc [options]
Options:
  -v        Display version information
```

```
  -h, -?    Display help
```

4.4.2 Example

```
$ emir-ethercat-plc
emir-ethercat-plc PID is 10016
Configuring PDOs...
Activating master...
```

4.5 Emir Server

Emir server is an XML-RPC server to remotely control the EMIR 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).
Emir-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

4.5.1 Syntax

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

4.5.2 API Description

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

```
$ ListMethods.py
receiver.getCalibration
receiver.getPositionList
receiver.getStatus
receiver.setAttenuator
receiver.setCalibration
receiver.setLoSwitch
system.listMethods
system.methodHelp
system.methodSignature
system.multicall
system.shutdown
```

```
$ Introspection.py
```

```
Name       : receiver.getCalibration(  )
Return Type: struct
Description: Get Calibration Information
Return struct =
{
    string lastPosition;
    boolean isArrived;
    struct { int absPosition; int velocity } car1;
    struct { int absPosition; int velocity } car2;
    struct { int absPosition; int velocity } filt;
}
Syntax:  GetAttenuator( )

------------------------------------------------------------------------
Name       : receiver.getPositionList(  )
Return Type: array
Description: Get Calibration Position List
Return Array of string.
Syntax:  GetPositionList( )

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

------------------------------------------------------------------------
Name       : receiver.setAttenuator( int, string, int )
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' by bandNum
        1 => B1_V1 , B1_V2 , B1_H1 , B1_H2
        2 => B2_V1 , B2_H1
        3 => B3_V1 , B3_H1
        4 => B4_V1 , B4_V2 , B4_H1 , B4_H2

- 'attenuationDecibel' must be in range [0,15]

------------------------------------------------------------------------
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_sky,  02_amb,  02_cold,  02_sky,  03_amb,  03_cold,  03_sky,
04_amb,  04_cold,  04_sky,  12_amb,  12_cold,  12_sky,  13_amb,  13_cold,  13_sky,
24_amb, 24_cold, 24_sky, init, parking

------------------------------------------------------------------------
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, 4]
 - switchName must be in: { gunn, deltaF, loop, sweep }
 - switchValue must be 0 or 1
```

```
--------------------------------------------------------------------
Name       : receiver.warmJunction( int, string, int )
Return Type: int
Description: Warm a junction during a given duration.
Return code is always zero.
Syntax: warmJunction(int bandNum, string polarity, int duration)
 - bandNum is always 3
 - polarity is 'V' or 'H'
 - duration is the warming duration in seconds

--------------------------------------------------------------------
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.

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

4.6 XML RPC client examples

4.6.1 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" )
```

4.6.2 Python examples

See directory *~/develSVN/emir/python/xmlrpc* for other XML-RPC python examples.

4.6.3 C++ client example

I have written a small C++ client example: *emir-client-GetStatus*
The source code is available in emir/apps/GetStatus

4.7 Emir-middex-init

This program initializes a Middex motor. Middex motors are used for the calibration system.
To initialize itself, the motor moves slowly toward its zero reference.

Features:

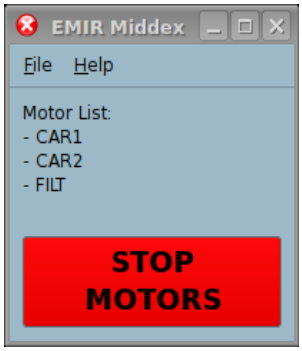
- Timeout protection to avoid never ending initialization.
- Safe behavior: the program powers off motor car2 when initializing motor car1 to avoid any collisions

This program is used by *emir-calibration-server*, so normal users should not run it manually.

4.7.1 Syntax

```
$ emir-middex-init -h
EMIR Middex Init - Initialize Middex motor reference
Usage: emir-middex-init [options]
Options:
  -m=MotorName  Specify motor to initialize {car1,car2,filt}

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


Example:
        emir-middex-init -m=car1
```

4.8 Emir-middex-stop



When clicking on the STOP button, it power off all the motor in the list.

For safety reasons, *emir-calibration-server* exits when the motor are powered off. So after clicking on the stop button, you have to restart *emir-calibration-server.*

Note: The window is always on the top, and cannot be hidden by others windows.

4.8.1 Syntax

```
$ emir-middex-stop -h
Middex Stop - Emergency Stop button for Middex motors
```

```
Usage: emir-middex-stop [options]
Options:
  -v                  Display version information
  -h, -?              Display help

Note: the environment variable CAN_DB_FILE must be set
to the Sqlite database that holds Middex parameters.
```

4.9 Emir-middex-util

Emir-middex-util is a graphical utility to manually drive a Middex motor.

The target audience for this program is the laboratory staff, to find all the adequate motor parameters (velocities, positions, etc.)

Warning: Even if the program has software limits from the emir database for positions and velocities, it must be use very carefully because the program does not check the other motor positions (unlike emir-calibration-server).

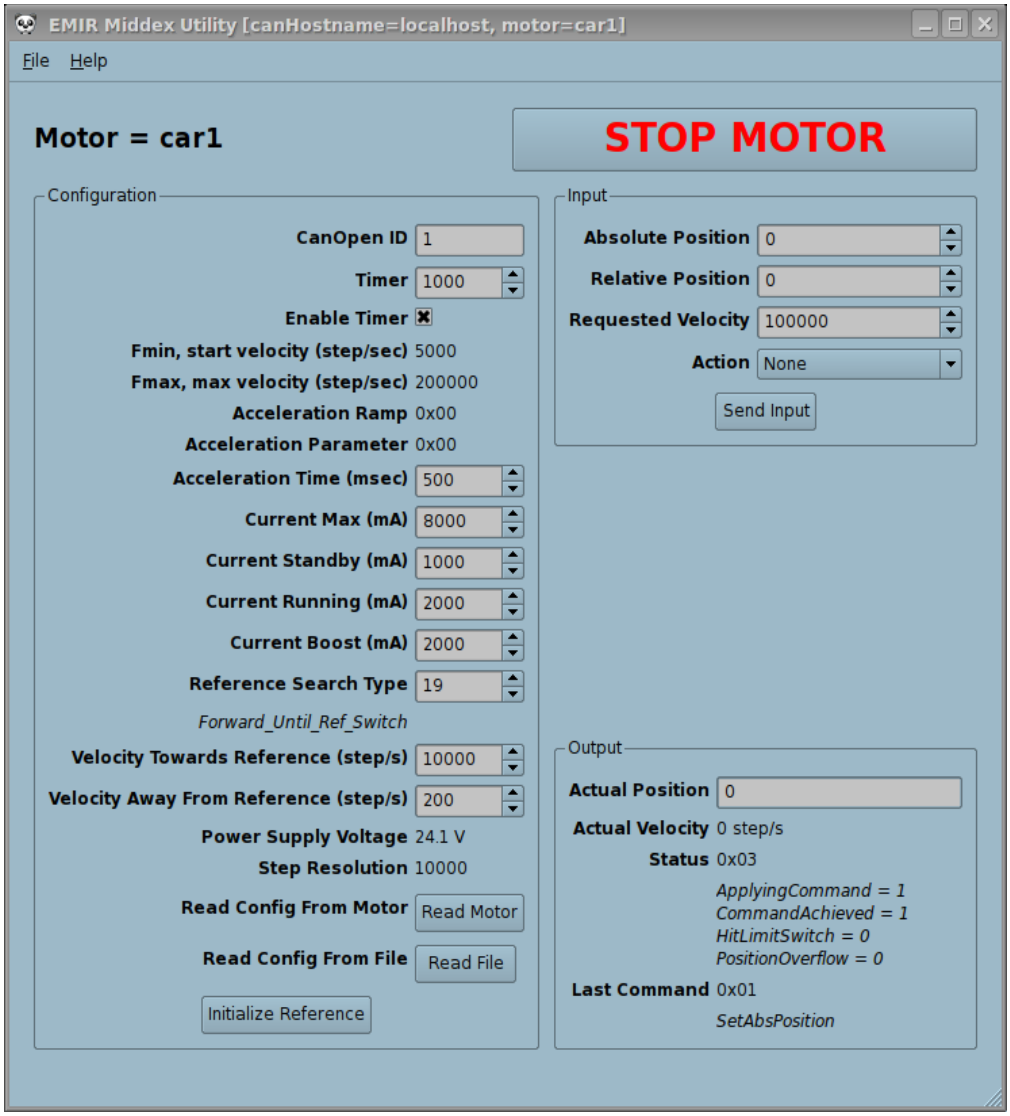
4.9.1 Syntax

```
$ emir-middex-util -h
Middex Utility - Control manually Middex motor
Usage: emir-middex-util [options]
Options:
  -m=name             Motor name to drive (Mandatory)

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

Example:
        emir-middex-util -m=car1

Note: the environment variable CAN_DB_FILE must be set
to the Sqlite database that holds Middex parameters.
```

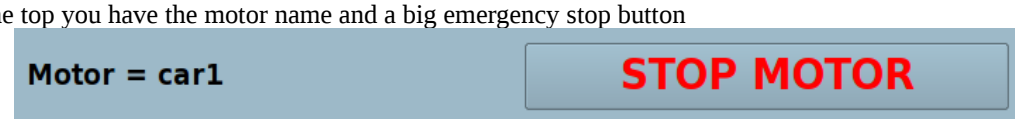
4.9.2 Usage

When the program starts, it:

1. Reads the motor parameters
2. Applies the settings from database
3. Displays the following window

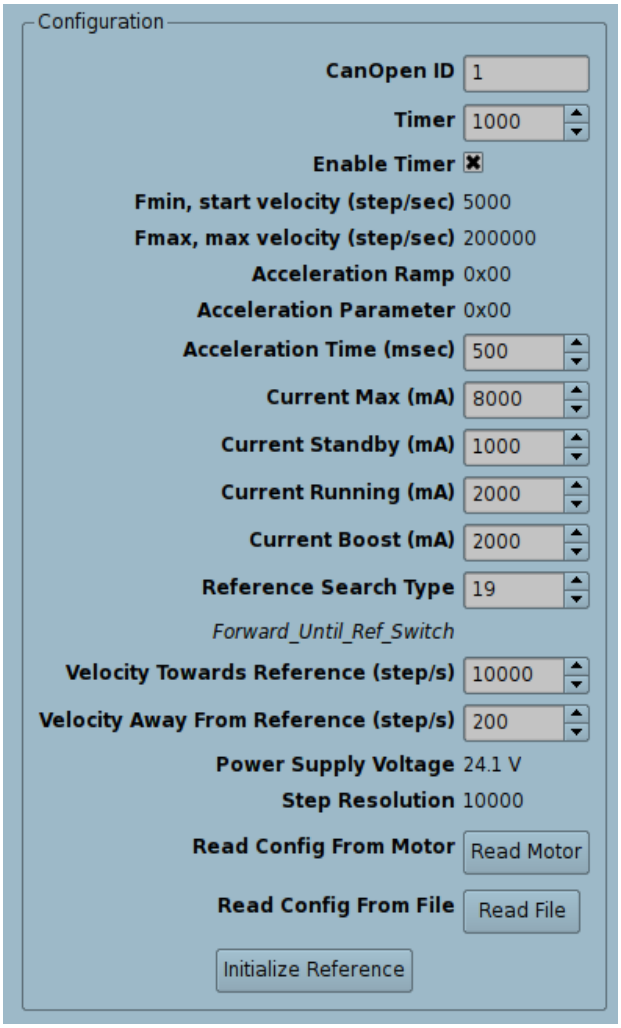


On the top you have the motor name and a big emergency stop button



It you stop the motor (in fact it powers off the motor), you have to reinitialize the motor reference.

Configuration panel



This panel displays the motor parameters.
See the motor manufacturer documentation for a full description of each parameter.

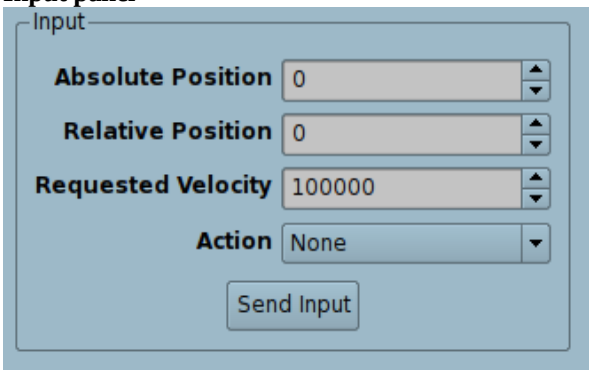
When you change a parameter, it is immediately send to the motor.
The displayed parameters are always up-to-date, because Middex-utility monitors the CAN traffic to know the modifications sent by other programs.

There are 3 buttons:
Read Motor: Read the motor parameters and update the display.
It is useful when motor parameters have changed without sending any CAN command. For example if you power OFF and then power ON the motor, its parameters goes back to their default (without any, but you does not want to restart middex-utility

Read File: Load database parameters into the motor.

Initialize Reference: Start the command sequence for reference searching.

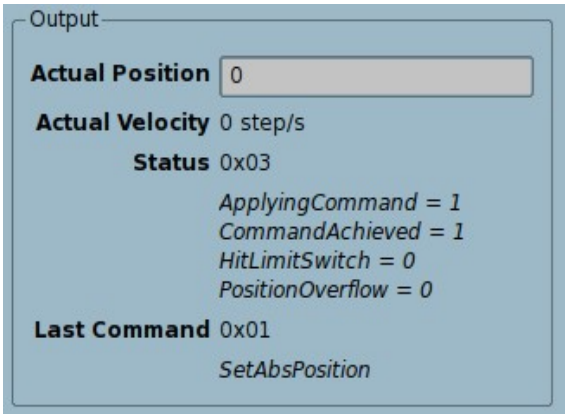
Input panel



This panel is used to send command to the motor.
Enter the command parameters, in the spinboxes, select an action in the list, and then click on *Send Input*

Available actions are:
None, *SetAbsPosition*, *SetRelPosition*, *SetVelocity*, *Stop*, *SearchReference*, *PowerOn*, *PowerOff*, *SetActualAsRequest*

Output panel



This panel displays:
The actual position
The actual velocity
The status register
The last sent command

4.10 Simulator

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

Syntax:
emir-simul

4.10.1 Main window

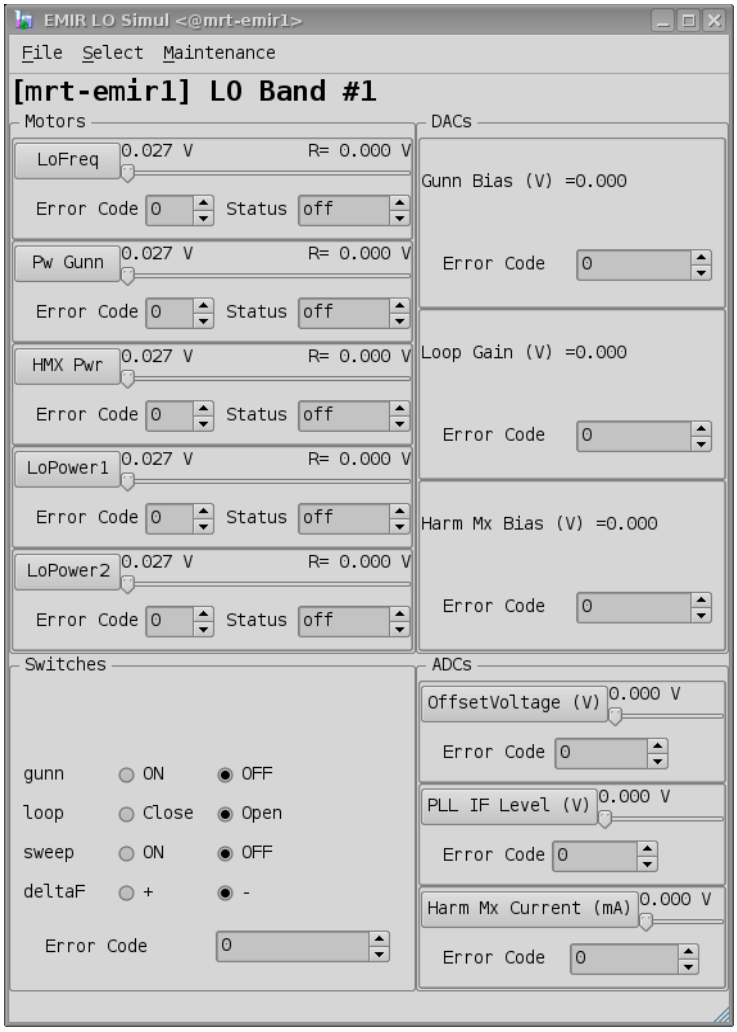
The main window has three buttons, click on them to display/hide simulation window for LO, Mixer or Cryo.

The simulation occurs only when the associated subwindow is opened.

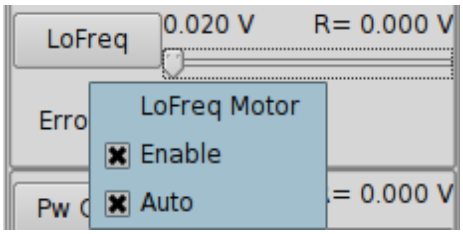
For example, if you want to simulate only the LO, open only the LO window.

4.10.2 LO Simulation Window

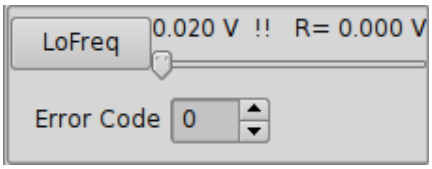
The LO window display the simulator for the LO.
The window can display only one band, but the 4 LO bands are simulated together. You can display the other bands with the *Select* menu.



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"*

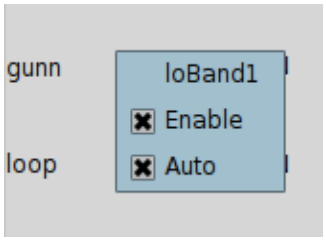


When "Auto" is unselected, a !! symbol appears.

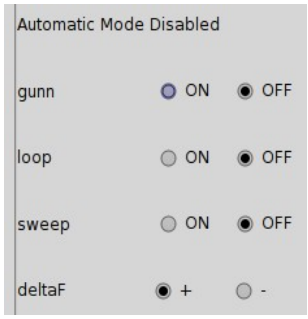
Adc simulator
It simulates a CAN ADC.
You can simulate a device missing error: right-click and unselect *"Enable"*

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.

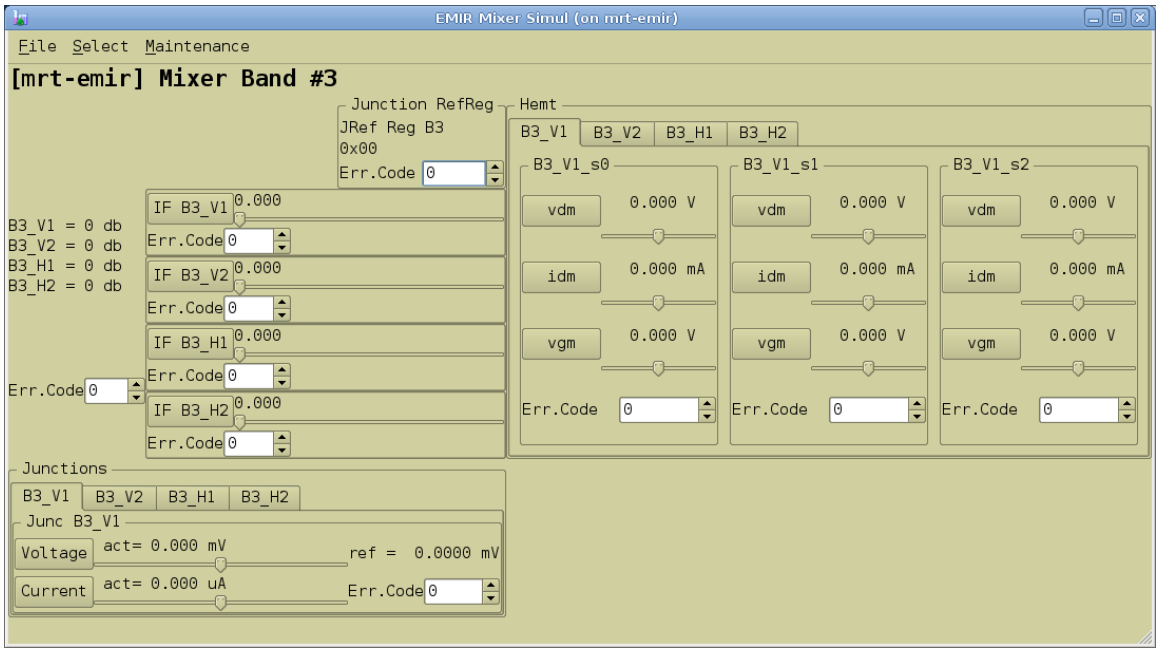


You can simulate the following errors:
A device is missing error: right-click and unselect *"Enable"*
The device answers, but values do not switch: right-click and unselect *"Auto"*



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

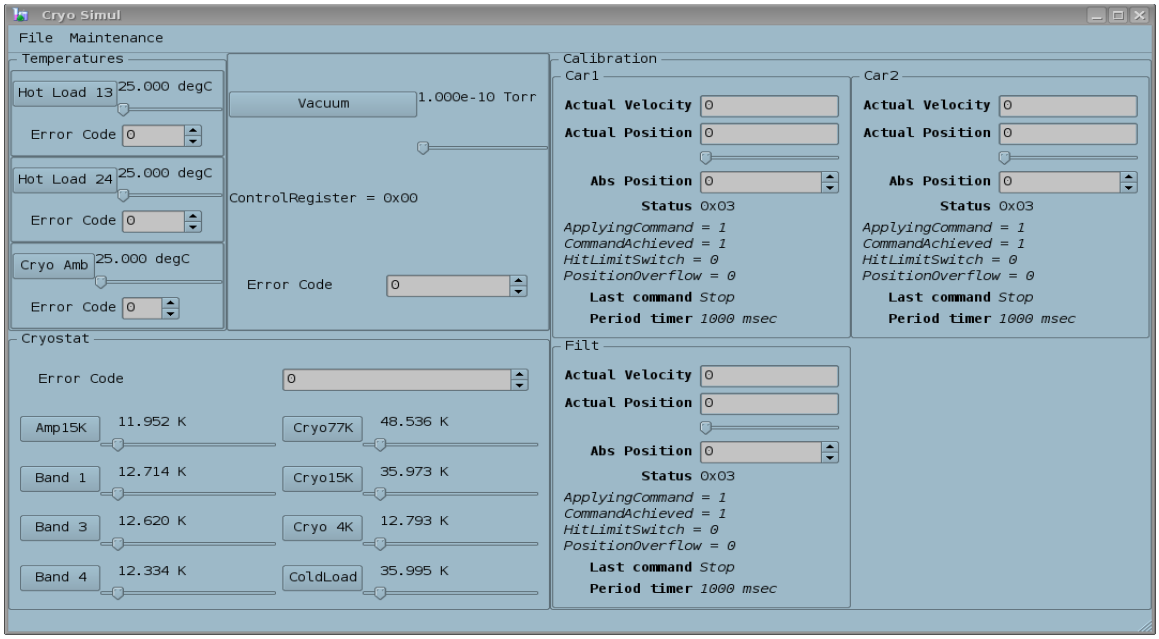
4.10.3 Mixer Simulation Window



This window simulates:

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

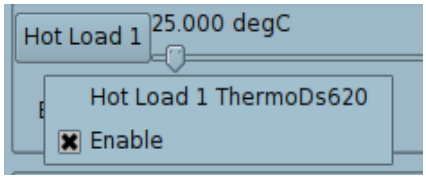
4.10.4 Cryo Simulation Window



This window simulates:

1. Hot load temperature
2. Cryostat temperature
3. Vacuum sensor
4. Calibration motor (CAR1, CAR2, FILT)

Hot Load /Cryo Amb
You can simulate the following failure:
The sensor returns an error code: enter the error code in the Error Code spinbox.
The sensor is missing: right-click, and unselect *"Enable"*



Cryostat
This widget simulates the different cryostat temperature.
You can simulate the following failure:
the sensor is missing: right-click, and unselect *"Enable"*

Vacuum
This widget simulates the cryostat vacuum sensor.
You can simulate the following failure:
the sensor is missing: right-click, and unselect *"Enable"*

5 User Programs

This section describes the user programs. These programs are installed in */home/introot/emir/bin*
All the programs exist in two versions:

- normal version (without suffix), for daily use
- debug version (with a .Debug suffix) for debugging

The data files are installed in */home/introot/emir/data*

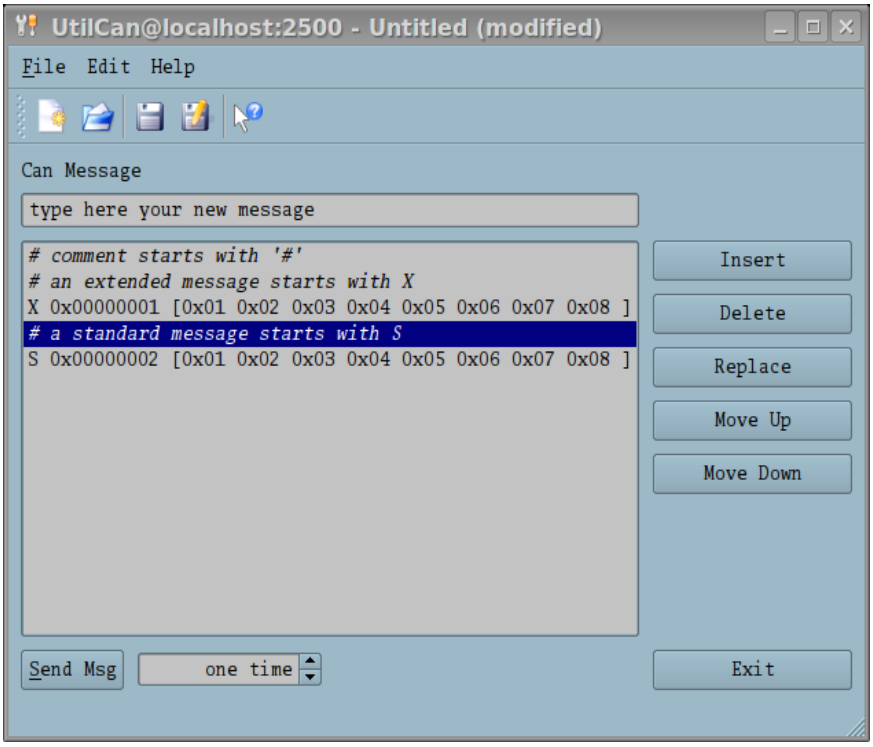
5.1 UtilCan

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

5.1.1 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
```

5.1.2 Screenshot



5.2 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

5.2.1 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.
```

5.2.2 Example

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

Load data from /home/introot/emir/data/emir.db
CAN_MANAGER_SERVER='localhost'
Connect to CanManager localhost:2501
2009-03-09T16:27:11.471: msg 1 : io_B1_LoSwitches_getStatus : X 0x01000100 []
2009-03-09T16:27:11.471: msg 2 : adc_B1_V1_ifLevel : X 0x13040100 []
2009-03-09T16:27:11.471: msg 3 : adc_B1_V2_ifLevel : X 0x13040101 []
2009-03-09T16:27:11.472: msg 4 : adc_B1_H1_ifLevel : X 0x13040102 []
```

5.3 Coil

This program tunes coils to remove the Josephson effect in junctions. It also draws graphs to check that the automatic setting is correct. *emir-coil* can be applied only on band #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.

5.3.1 Syntax

```
$ emir-coil -h
ReceiverID = 1
EMIR Coil - Tune coils to remove the Josephson effect in junctions
Usage:
   emir-coil [options]

Options:
   -b=bandNum    Specify band to tune
   -c=a,b,c      Specify channel names to tune. If empty, tune all coils

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

Channel names:
   Band 4: B4_V1, B4_V2, B4_H1, B4_H2
```

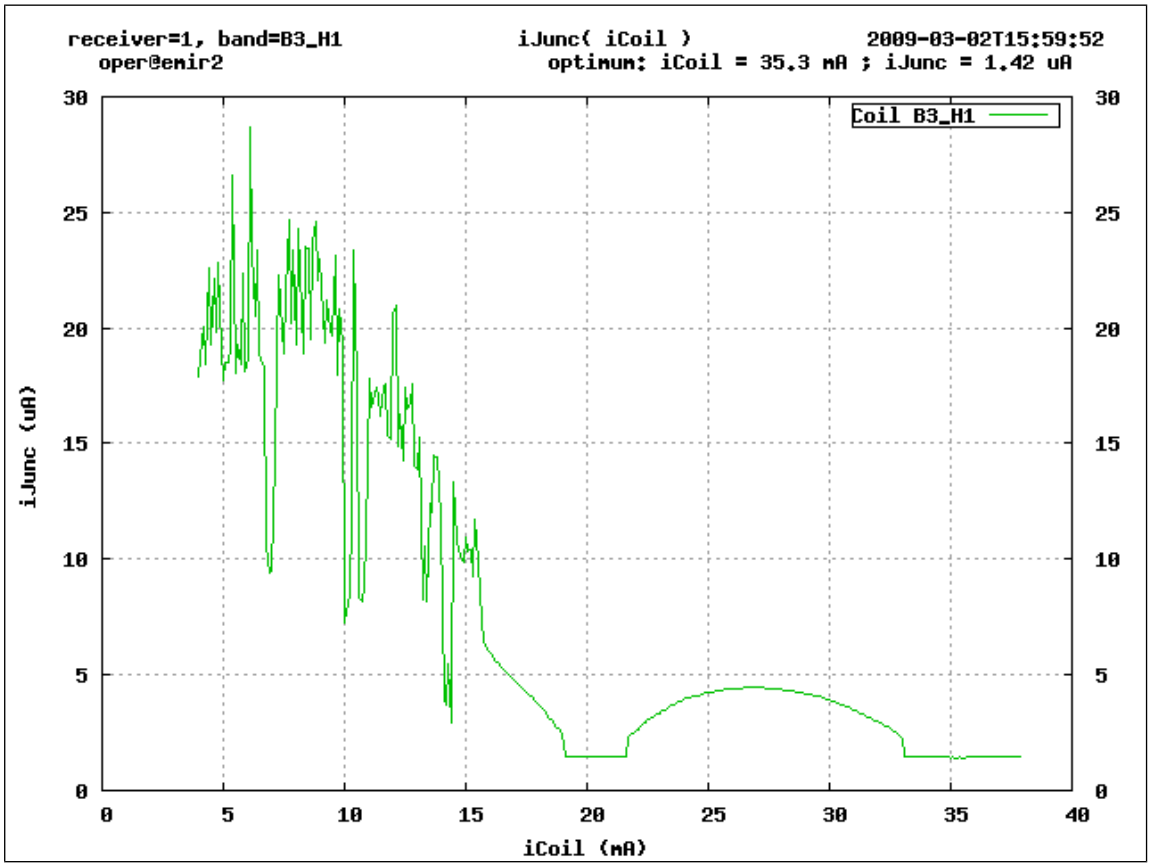
```
Example:
   emir-coil -b=4 -c=B4_V1,B4_H1
```

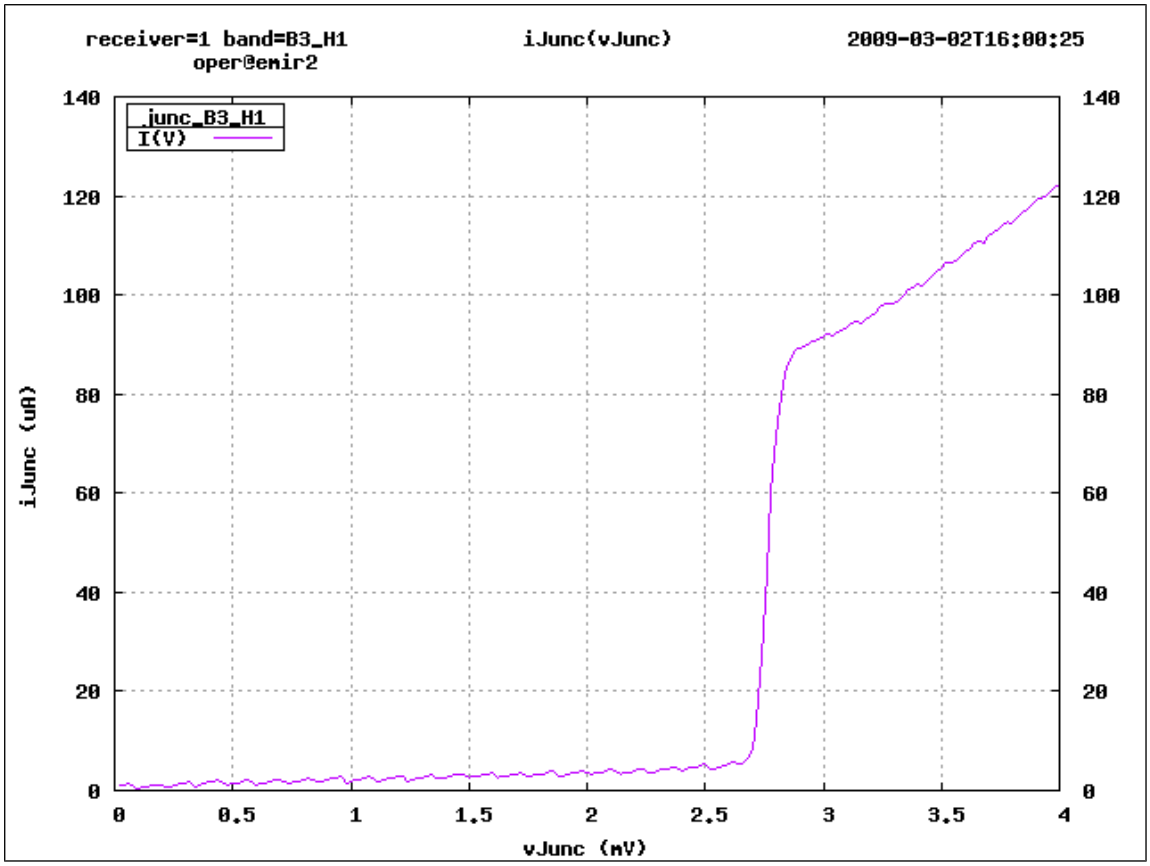
5.3.2 Example

```
$ emir-coil -b=4 -c=B4_V1
Setting for this tuning
        BandNum = 4
        ChannelList = B4_V1

CAN_MANAGER_SERVER='localhost'
Connect to CanManager localhost:2501
Step 0: Decrease loPower2
Starts individuals threads
[coil_B4_V1] Load parameters from database:
[coil_B4_V1]         iJuncMargin          =  0.10
[coil_B4_V1]         juncAcqRef           =  0.20
[coil_B4_V1]         juncStdRef           =  2.30
[coil_B4_V1]         juncZeroRef          =  0.01
[coil_B4_V1]         minCoilRef           =  4.00
[coil_B4_V1]         minFlatWidth         =  2.00
[coil_B4_V1]         maxCoilRef           = 40.00
[coil_B4_V1]         maxDerived           =  0.15
[coil_B4_V1]         maxJosephsonCurrent  =  1.50
[coil_B4_V1]         iCoilNegExploration  = -1.00
[coil_B4_V1]         iCoilPosExploration  =  2.00

[coil_B4_V1] Thread starts

[coil_B4_V1] Step 1: Polarize junction with ref= 0.2 mV

[coil_B4_V1] Step 2: Start Coils and acquire iJunc(iCoil)
[coil_B4_V1]     Clear memory effect
[coil_B4_V1]     Acquiring iJunc(iCoil) on [4 mA, 40 mA]

[coil_B4_V1] Step 3: Apply magnetic field
[coil_B4_V1]     Searching for a flat level range inside [  4.0 , 21.9 ]
[coil_B4_V1]         Searching parameters:
[coil_B4_V1]                  minCoilRef =  4.00
[coil_B4_V1]                  maxCoilRef = 21.90
[coil_B4_V1]                  maxDerived =  0.15
[coil_B4_V1]                 iJuncMargin =  0.10
[coil_B4_V1]                minFlatWidth =  2.00
[coil_B4_V1]                       minMin =  -inf
[coil_B4_V1]         Found minimum ( iCoil = 16.4 , iJunc = -20.2214 )
[coil_B4_V1]         Flat level range candidate: [ 14.7 , 18.4 ]
[coil_B4_V1]         OK: the flat level range is accepted (width = 3.7)

[coil_B4_V1]     Searching for a flat level range inside [ 22.1 , 40.0 ]
[coil_B4_V1]         Searching parameters:
[coil_B4_V1]                  minCoilRef = 22.10
[coil_B4_V1]                  maxCoilRef = 40.00
[coil_B4_V1]                  maxDerived =  0.15
[coil_B4_V1]                 iJuncMargin =  0.10
[coil_B4_V1]                minFlatWidth =  2.00
[coil_B4_V1]                       minMin =  -inf
[coil_B4_V1]         Found minimum ( iCoil = 30.2 , iJunc = -20.2336 )
[coil_B4_V1]         Flat level range candidate: [ 27.2 , 30.2 ]
[coil_B4_V1]         OK: the flat level range is accepted (width = 3)
```

```
[coil_B4_V1]     Build list of candidates
[coil_B4_V1]     Applying optimal iCoil
[coil_B4_V1]     Optimizing Josephson current
[coil_B4_V1]     Optimal iCoil = 28.1
[coil_B4_V1]     Plot iJunc(iCoil)
[coil_B4_V1]     Output = /home/oper/B4/r1_B4_V1.coil.png

[coil_B4_V1] Step 4: Plot iJunc(vJunc)
[coil_B4_V1]     Acquire I(V) on [0 mV, 4 mV]
[coil_B4_V1]     Plot iJunc(vJunc)
[coil_B4_V1]     Output = /home/oper/B4/r1_B4_V1.junc.png

[coil_B4_V1] Thread exits


==end of tune
```

5.3.3 Graph examples





5.3.4 Debugging

The emir-coil program creates temporary files in the subdirectory B**N** where **N** is the band number.
With these files you can understand how the program has chosen the optimum tuning.

5.4 Init-ampli

This program initializes the CalTech amplifiers for the Mixer band #4.
The amplifiers must be initialized otherwise mixer band #4 does not work.

5.4.1 Syntax

```
$ emir-ampli -h
EMIR Init Ampli - Init Ampli
Usage: emir-init-ampli [options]
Options:
  -v              Display version information
  -h, -?          Display help
```

5.4.2 Example

```
$ emir-init-ampli
Step 1: Initialize the ampli
End of initialisation
```

5.5 Lo

This program tunes the local oscillator.

5.5.1 Syntax

```
$ emir-lo -h
EMIR Lo - Tune Local Oscillator
Usage: emir-lo [options]
Options:
  -b=bandNum      Specify band to tune [1,4]
  -f=frequency    Specify the LO frequency in GHz
  -d=[-|+]        Specify the deltaF, '+' or '-'. Default is '-'.

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

Example:
        emir-lo -b=1 -f=90.0
```

5.5.2 Example

```
$ emir-lo -b=1 -f=90.0
Setting for this tuning
        BandNum = 1
        Frequency LO = 90.000 GHz
        DeltaF = MINUS

CAN_MANAGER_SERVER='localhost'
Connect to CanManager localhost:2501
FGUNN= 90 GHz
Load settings from h196
Step 1: Configure the LO in safe mode

Step 2: Set Motors and DACs
LO settings:
        FGunn = 90
        GunnBias = 7.8
        HarmMixerBias = 0
        HarmMixerPower = 1.7
        LoFreq = 4.817
        LoPower1 = 4
        LoPower2 = 4.5
        LoopGain = 5.065
        PowerGunn = 4.22

Step 3: Close Loop and optimize LoFreq
Optimal loFreq = 4.795

Step 4: Set Offset Voltage and Gunn Bias
Increase loFreq until OffsetVoltage < 5.00
loFreq = 4.7950 ; offsetVoltage = 0.0000
Increase gunnBias until OffsetVoltage > 5.00
Optimal gunnBias = 7.97968

Step 5: Find max PllIfLevel(harmMixerBias)
Max found: harmMixerBias = 0.4, pllIfLevel = 5.27466
Tuning Ok
```

```
End of tuning
```

5.6 Mixer

5.6.1 Syntax

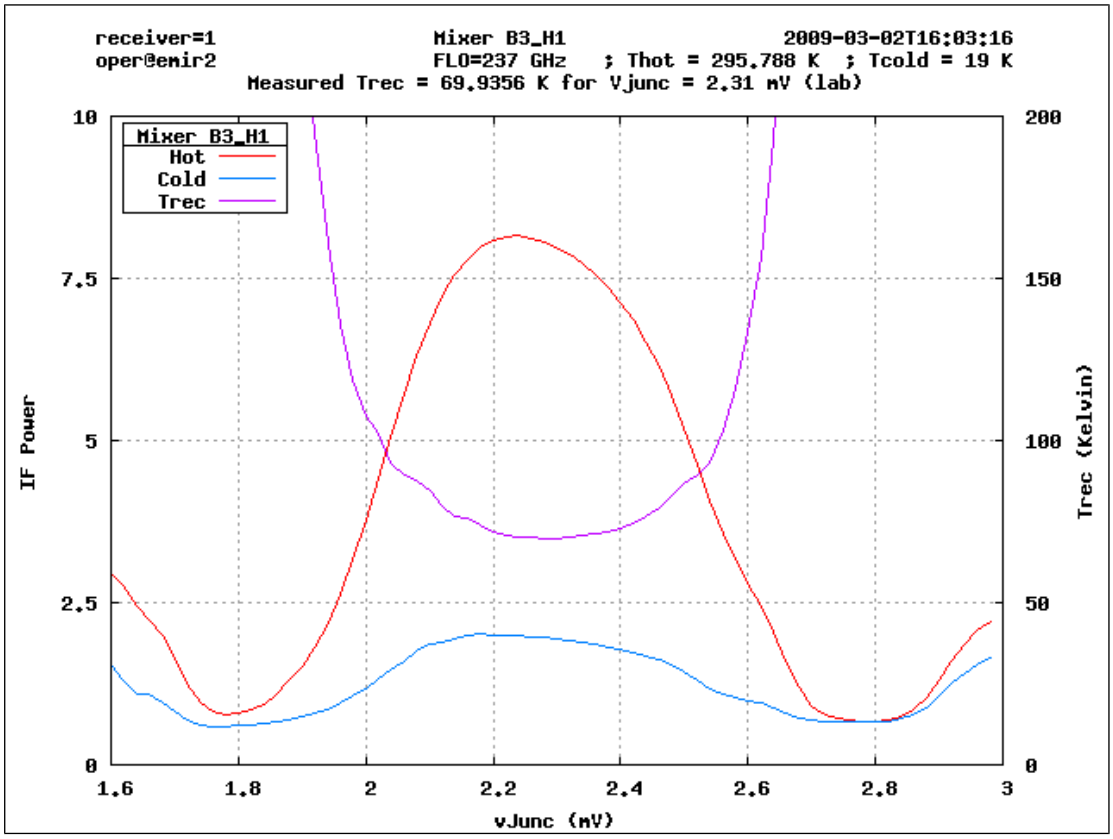
```
$ emir-mixer -h
EMIR mixer- Tune Mixer
Usage: emir-mixer [options]
Options:
  -b=bandNum     Specify band to tune [1,4]
  -f=frequency   Specify the frequency LO1 in GHz
  -s=[L|U]       Specify the sideband (L)ower or (U)'
  -r             Skip tuning, redraw only Trec graphs

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

Example:
        emir-mixer -b=1 -f=90.0
```

5.6.2 Example

```
$ emir-mixer -b=3 -f=237 -s=L
Setting for this tuning
   BandNum = 3
   Frequency Lo = 237.000 GHz

Backshort Mixer
CAN_MANAGER_TARGET='localhost'
Connect to CanManager localhost:2501
Connect to CanManager localhost:2502
Connect to CanManager localhost:2500
Load LO lab settings
Load settings from h244
LO Settings:
        FGunn = 79
        GunnBias = 7.8
        HarmMixerBias = 0
        HarmMixerPower = 0.4
        LoFreq = 2.581
        LoPower1 = 6.5
        LoPower2 = 6.5
        LoopGain = 4.835
        PowerGunn = 5.73

Load settings from m560-554.b3.lsb
Mixer settings:
        Attenuation_B3_H1 = 7
        Attenuation_B3_V1 = 7
        Backshort_H = 8.27
        Backshort_V = 8.43
        Flo1 = 237
        Ij_H = 25.8
        Ij_V = 23.2
        TCold_H = 19
        TCold_V = 23
```

```
        Vj_H = 2.31
        Vj_V = 2.31

Step 1: Starting
Polarize junctions
Wait for calibration moving to 03_Amb ... OK

Step 2: Set Motors and Attenuators
Step 3: Optimize mixer current
Target current = 23.2 uA on junction B3_V1
Step 4: Compute Trec and draw graphs
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]
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
```

5.6.3 Graph



5.6.4 Utilities

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

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

```
$ 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
```

```
$ 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
```

5.7 PlotJunction

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

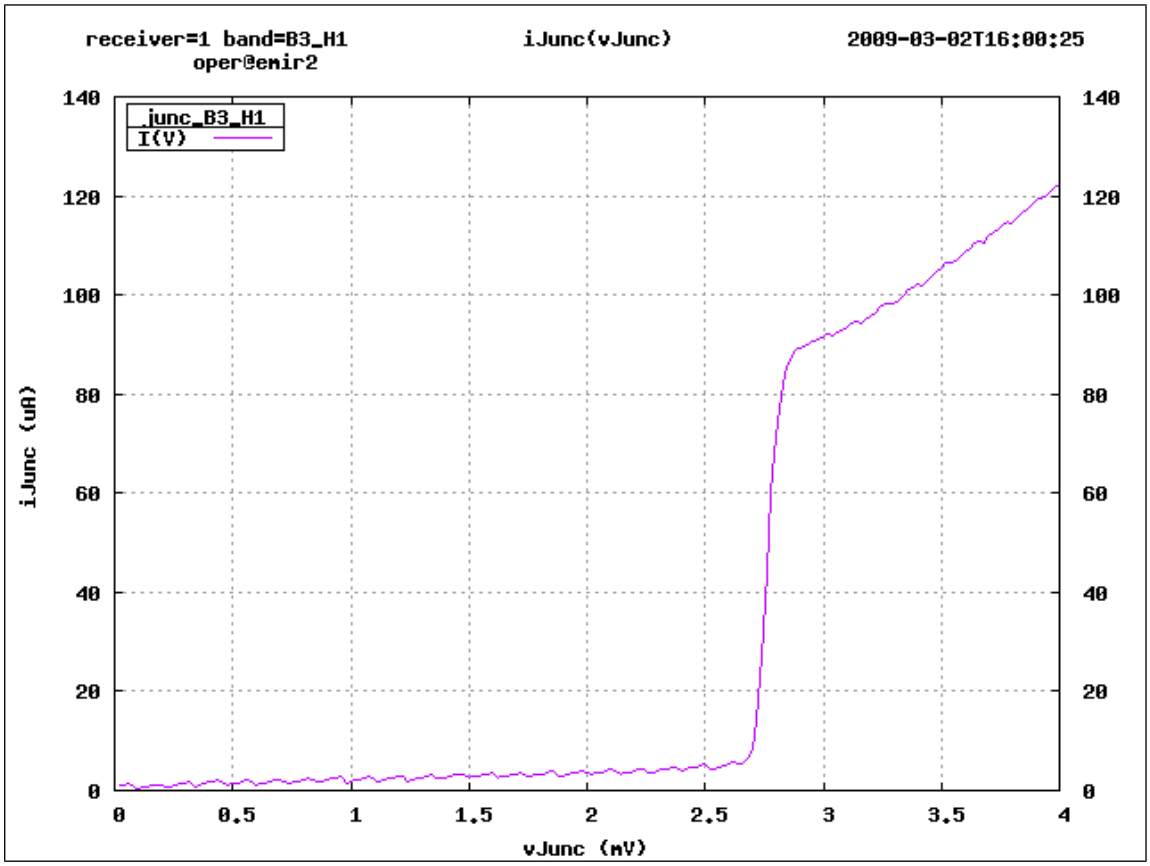
5.7.1 Syntax

```
$ emir-plot-junction -h
EMIR Plot Junction - Plot junction graph I(V)
Usage: emir-plot-junction [options]
Options:
  -b=bandNum          Specify band to plot
  -c=ch1,ch2,ch3      Specify channel names to plot. If empty, plot all junctions
  -k                  Keep LO Power at its actual level
  -iv                 Plot I(v) (default)
  -vi                 Plot V(i)
  -s=sampleNum        Number of sample per point. Default = 3
  -r=resolution       Resolution of acquisition. Default = 0.02
  -d=delay            Delay in seconds between 2 acquisitions. Default = 0.05
```

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

Channel names:
  Band 1: B1_V1, B1_V2, B1_H1, B1_H2
  Band 2: B2_V1, B2_H1
  Band 3: B3_V1, B3_V2, B3_H1, B3_H2
  Band 4: B4_V1, B4_V2, B4_H1, B4_H2

Example: emir-plot-junction -b=4 -c=B4_V1,B4_H1
```

5.7.2 Example

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

CAN_MANAGER_SERVER='localhost'
Connect to CanManager localhost:2501
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
```



5.8 emir-nika

Emir-nika control the NIKA mirror to select EMIR or NIKA instrument. Before moving the NIKA mirror, the EMIR calibration is sent to parking.

5.8.1 Syntax

```
$ emir-nika -h
emir-nika  --  Set NIKA mirror from EMIR computer
Usage: emir-nika {0|1}
  1 -> enable NIKA mirror
  0 -> disable NIKA mirror
Options:
  -v        Display version information
  -h        Display help
```

5.8.2 Example

```
$ emir-nika 1
Move EMIR calibrator to parking...
Send command 1 to NIKA
```

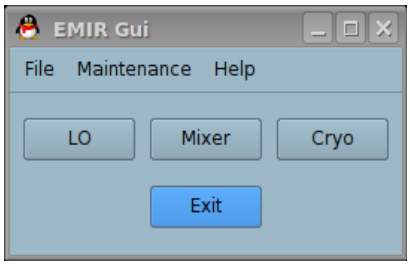
5.9 Gui

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

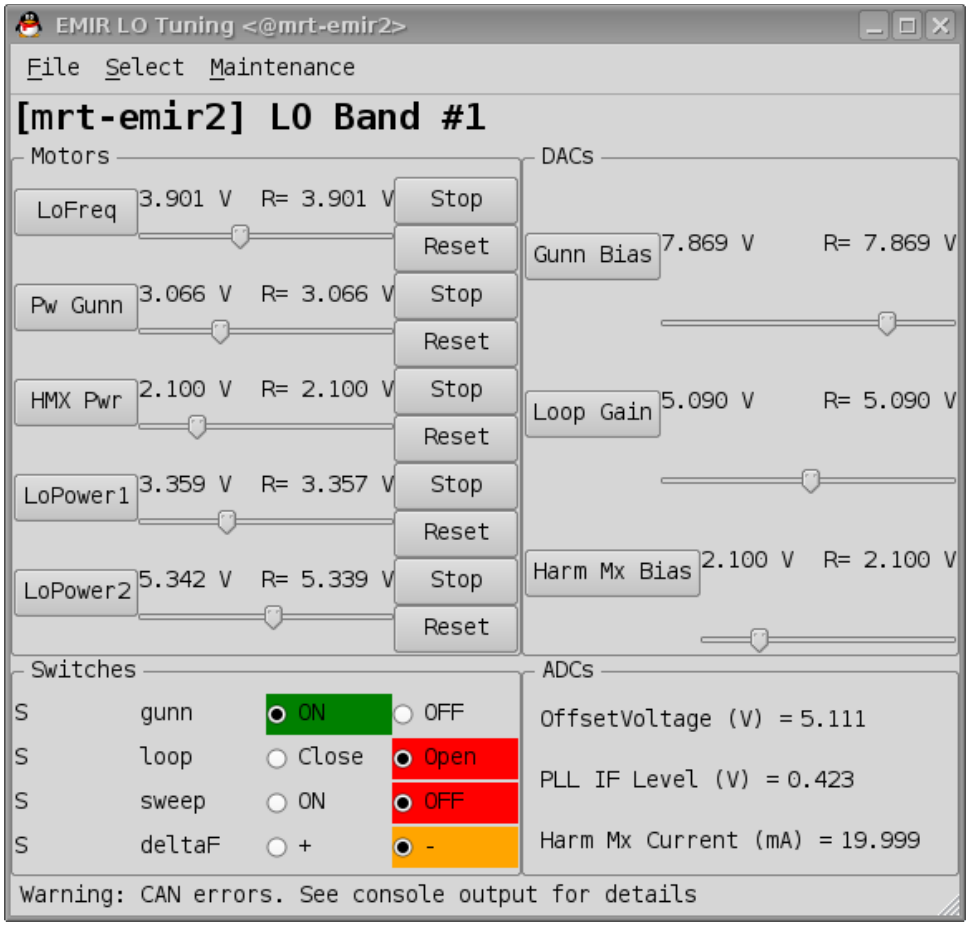
Syntax

emir-gui

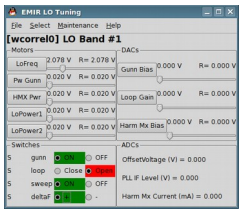
5.9.1 Main window

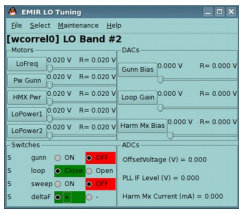


5.9.2 Lo window



To avoid mistakes, each band has its own color.


Band 1


Band 2

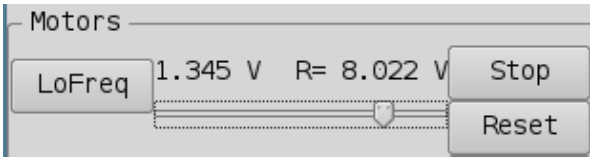

Band 3


Band 4

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

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

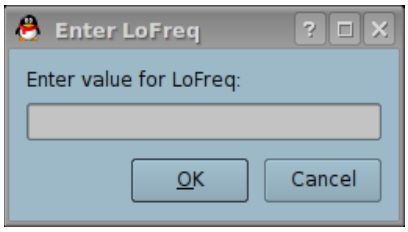
Description



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.

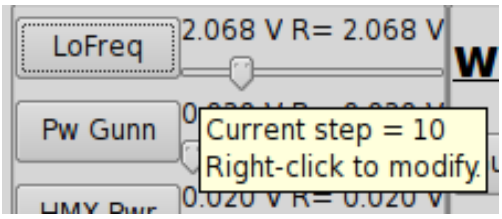
To modify the motor position , you can:

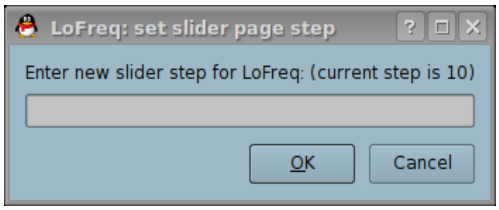
- Click on the button name to open a dialog box to enter the new motor value
- Move the slider



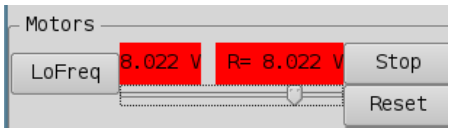
Motor dialog box

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



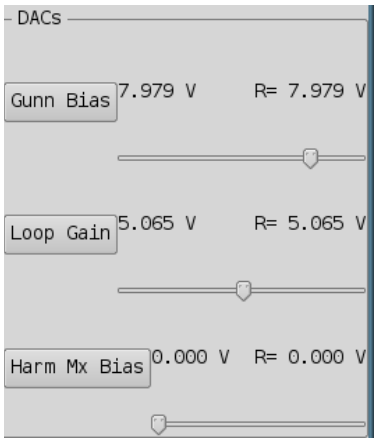


The unit for the slider is in motor raw unit (totally different from the motor unit which is displayed in Volt)



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

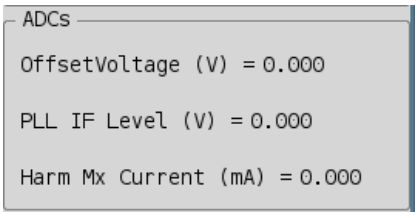
DAC widget (GunnBias, LoopGain, HarmMxBias)



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)



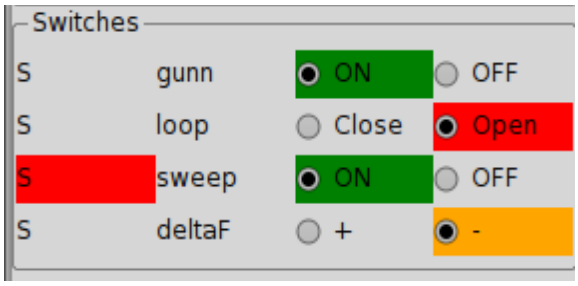
This widget is used to display the current ADC value.

Lo Switches widgets

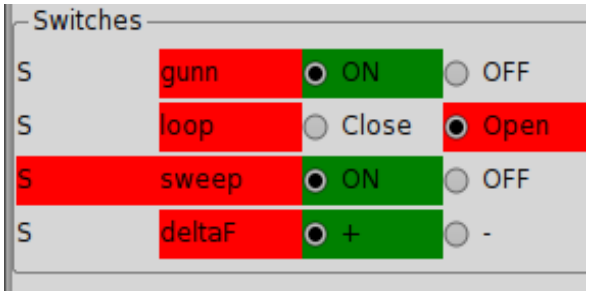
This widget is used to command the LO switches and to display the status.

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.

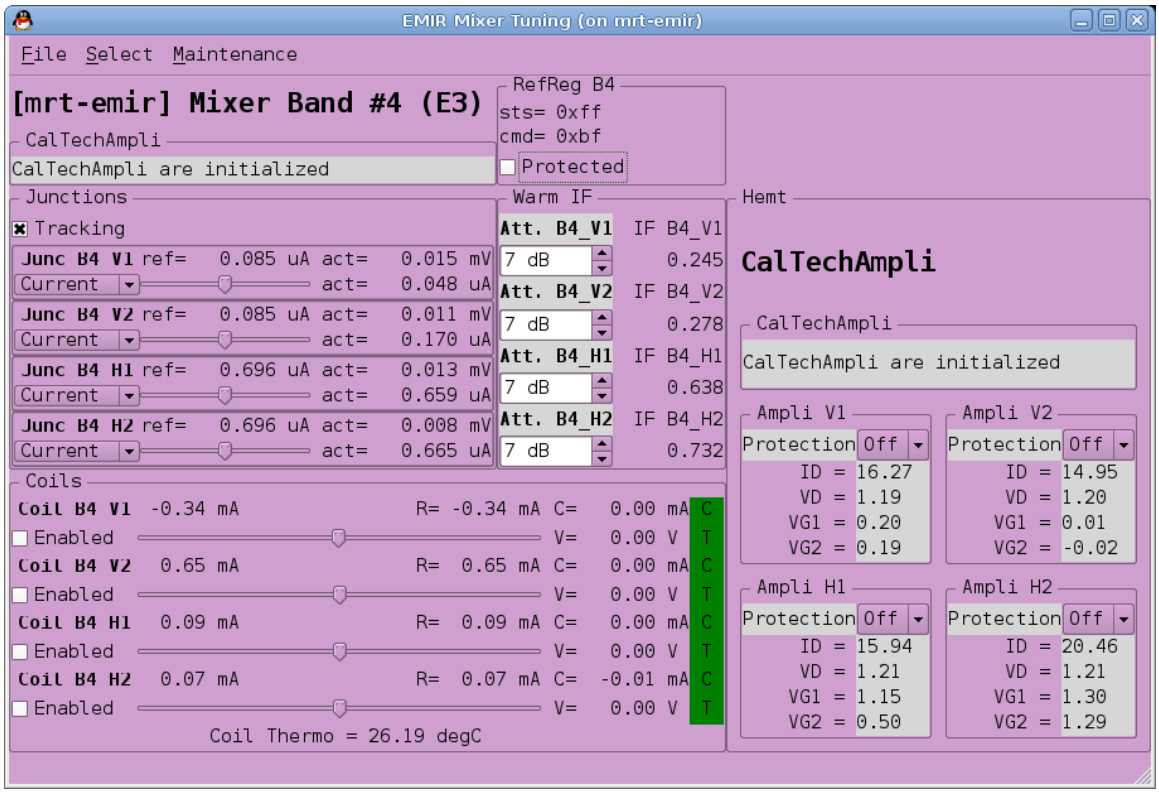


The radio button displays the current command applied on this switch.
For each switch, if the command matches the status, the status label (the S letter on the left side) is displayed with a normal background; otherwise this label is displayed with a red background.
Here the status for Sweep is red.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

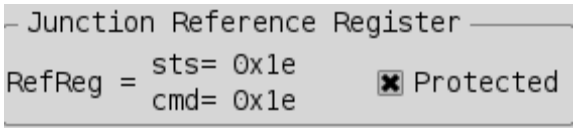


If the LoSwitches device does not answer at all, the switch names (gunn, loop, sweep, deltaF) become red to indicate a problem.

5.9.3 Mixer window



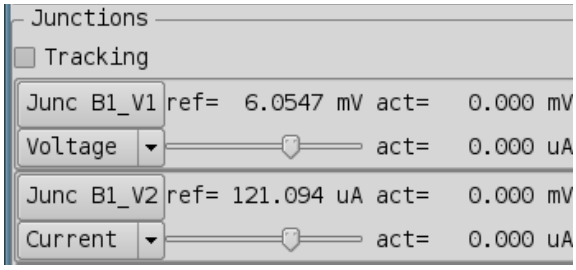
Junction Reference Register



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

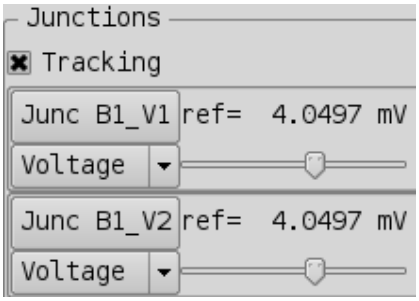
Junction



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

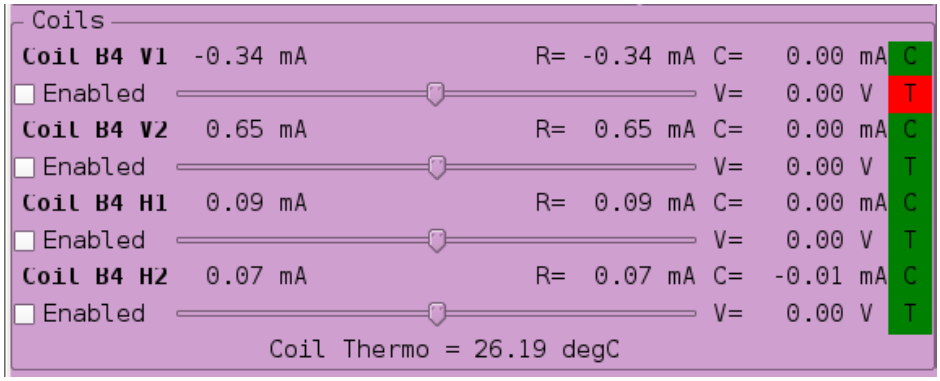


For double side band mixer, the same-polarity junctions must have the same reference values, otherwise the tuning is wrong. Therefore, there is a special tracking mode to change simultaneously the two junction references.

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

Coil

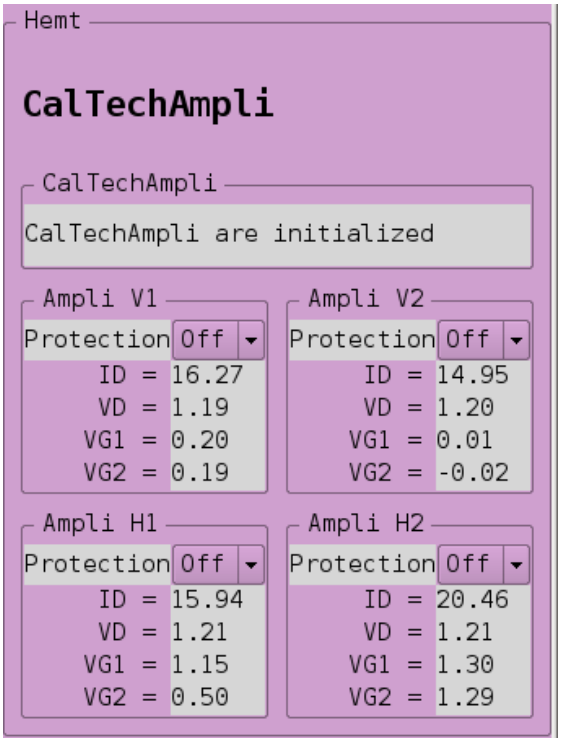
This widget is used to control the coils, to apply a magnetic field to cancel the Josephson current in junction.



There are coils only in band #4.

- Click on the *Enabled* check box to activate the coil
- To change the the coil current use the slider or click on the Coil name (for example *Coil_B4_V1*)to enter a value.
- On the right side, you have a current (C) and a thermal (T) indicators. They becomes red, if a problem occurs.

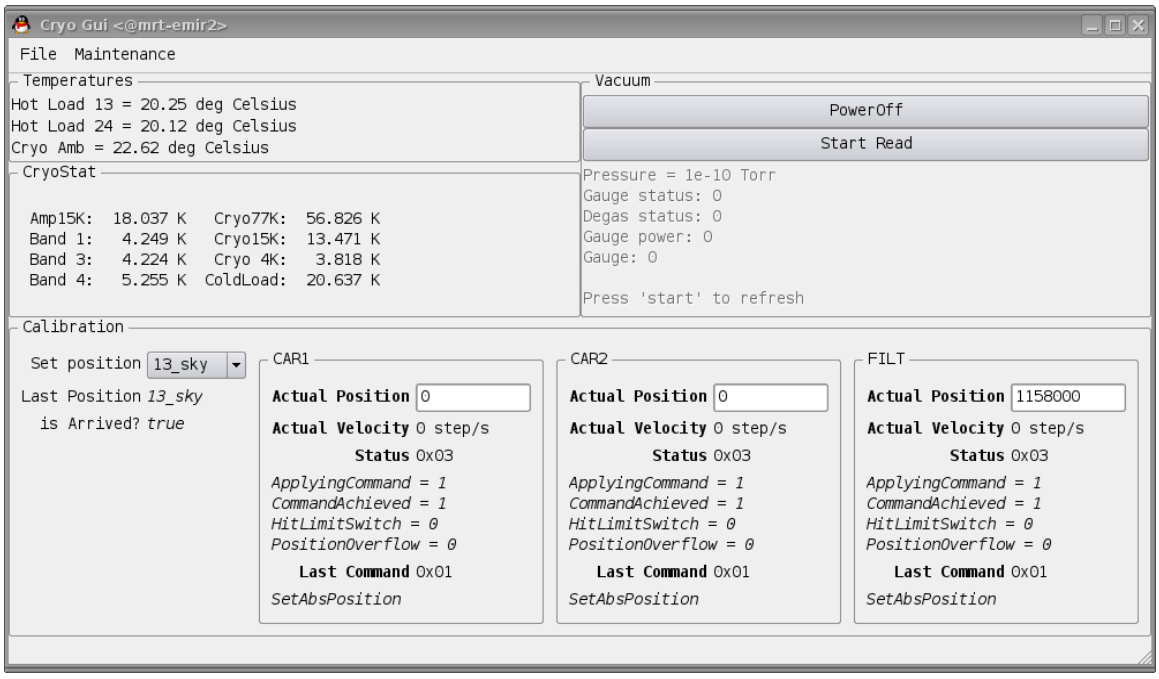
CalTech Amplifiers



If the amplifiers are initialized, the message "*CalTechAmpli are initialized*" appears. Otherwise a warning message appears with a button to initialize the amplifiers.

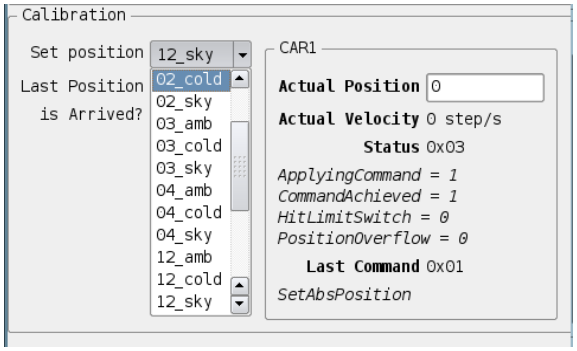
Each amplifier can be protected/unprotected with the combo box.

5.9.4 Cryo window



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

Calibration

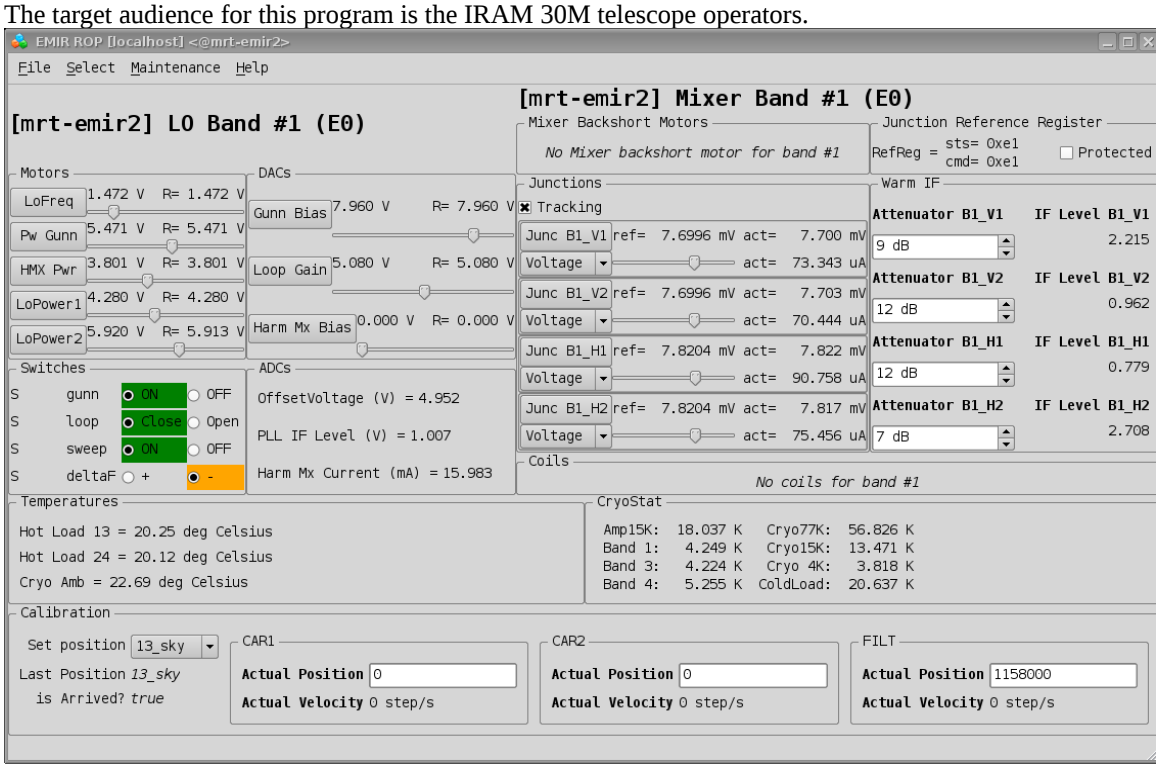


Use the menu to select a calibration position.
You can see the motor position and speed in the box on the right.

5.10 Rop

Rop (Receiver OPerator) is a special version of emir-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 30M telescope operators.



See emir-gui section for emir-rop usage instructions.

5.11 GetStatus

emir-get-status.py is a small program that send a “receiver.getStatus()” to the emir-server, and then print the result on the standard output.

5.11.1 Syntax

There is no options, so you just have type the program name.

```
$ emir-get-status.py
```

5.11.2 Example

```
$ emir-get-status.py
status.band1.attenuation_B1_H1     10.0
status.band1.attenuation_B1_H2     11.0
status.band1.attenuation_B1_V1     12.0
status.band1.attenuation_B1_V2     12.0
status.band1.deltaf                0
status.band1.gunn                  0
status.band1.ifLevel_B1_H1         0.188904
status.band1.ifLevel_B1_H2         0.258179
status.band1.ifLevel_B1_V1         0.172424
status.band1.ifLevel_B1_V2         0.170746
status.band1.loop                  0
status.band1.sweep                 0
status.band2.attenuation_B2_H1     11.0
[...]
```

5.12 Print Detailed Status

emir-print-detailed-status prints all the detail of the receiver into a text format. This program prints the value of more than 350 parameters.

5.12.1 Syntax

```
$ emir-print-detailed-status -h
EMIR Print Detailed status
Print detailed status of the receiver on the standard output
Usage: emir-print-detailed-status [options]
Options:
  -v        Display version information
  -h, -?    Display help
```

5.12.2 Example

```
$ emir-print-detailed-status
# hostname = gre106
# Mon Mar 12 16:52:56 2012
band1.GunnBias            dac_B1_GunnBias.requestedValue            0.00
band1.HarmMixerBias       dac_B1_HarmMixerBias.requestedValue       0.00
band1.HarmMixerCurrent    adc_B1_HarmMixerCurrent.actualValue       0.00
band1.LoFreq              motor_B1_LoFreq.requestedValue            0.027
band1.LoFreq              motor_B1_LoFreq.actualValue               0.027
band1.LoHarmMixerPower    motor_B1_LoHarmMixerPower.requestedValue  0.027
band1.LoHarmMixerPower    motor_B1_LoHarmMixerPower.actualValue     0.027
[...]
```

5.13 WarmJunction

Emir-WarmJunction.py is a small program that send a “receiver.warmJunction()” to the emir-server, to warm up a junction.

5.13.1 Syntax

```
$ emir-WarmJunction.py -h
```

```
Usage: emir-WarmJunction.py [options]

warm Junction

Options:
  -h, --help            show this help message and exit
  -s SERVER_NAME, --server=SERVER_NAME
                        XML-RPC server name. Default=localhost
  -b BAND, --band=BAND  band number
  -p POLARITY, --polarity=POLARITY
                        Polarity: 'V' or 'H'
  -d DURATION, --duration=DURATION
                        Duration in seconds
```

5.13.2 Example

```
$ emir-WarmJunction.py --band=3 --polarity=H --duration=10
# Connect to http://localhost:1080
Call server.receiver.warmJunction( 3, 'H', 10)
```

5.14 Check software

emir-check-software.py is a small program to check if all the EMIR programs run normally:

5.14.1 Syntax

```
$ emir-check-software.py -h
emir-check-software.py - Check emir software status
Usage: emir-check.py [options]
Options:
  -h,-?             Print this help
  -v                Print version

Run emir-check.py without options to start checkings.
```

5.14.2 Example

```
$ emir-check-software.py
==========================
Check EMIR software
==========================
Check Can Controllers:
tpmc816drv              8108  4
Can Controllers: Ok

Check CanManager:
CanManager: Ok

Check database:
CAN_DB_FILE='/home/introot/emir/data/emir.db'
Database: Ok

Check emir-calibration-server
emir-calibration-server: Ok

Check emir-server
```

```
emir-server: Ok

EMIR Receiver: Ok
```

6 Telescope Database

The IRAM 30 telescope has a network database, that holds all the telescope parameters. To access this database you have to do the following things:

1. Mount mrt-lx1.iram.es:/ncsServer into /ncsServer
2. Execute the telescopeStatus program (/ncsServer/mrt/ncs/lib/python/telescopeStatus.py)

6.1 TelescopeStatus

TelescopeStatus is a python program written by Walter Brunswig to get the telescope status from the on-line database.
To avoid users enter manually the LO frequency for emir-lo and emir-mixer, a script can get the parameters from TelescopeStatus to compute the LO frequency, and then call automatically emir-lo and emir-mixer with the right parameters.

6.1.1 Syntax

```
telescopeStatus.py key
```

6.1.2 Example

```
mrt-emir1:$ export PATH=/ncsServer/mrt/ncs/lib/python/:$PATH
mrt-emir1:$ telescopeStatus.py EMIR.frequency
rxCS.EMIR.frequency.ambientLoadTemp    296.115
rxCS.EMIR.frequency.bandwidth  4.000000
rxCS.EMIR.frequency.coldLoadTemp      0.000
rxCS.EMIR.frequency.doppler    0.999946852135
rxCS.EMIR.frequency.frequency  111.123456
rxCS.EMIR.frequency.harmonic   []
rxCS.EMIR.frequency.offset     0.0
rxCS.EMIR.frequency.skyFrequency 111.117550026
rxCS.EMIR.frequency.syntheziserFrequency []
rxCS.EMIR.frequency.timeStamp  2009-01-26T14:23:24.800
rxCS.EMIR.frequency.value
rxCS.EMIR.frequency.width
```

7 Daily Operation

7.1 Modify database configuration

The configuration settings are stored in several SQL files in directory */home/introot/emir/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 *emir-update-db.sh* to update the database.

```
$ emir-update-db.sh
```

```
    cd /home/introot/emir/data
    rm -f emir.db
# Update emir.db ...
    sqlite3 emir.db < ./00-canid.sql
    sqlite3 emir.db < ./conf-calibration.sql
    sqlite3 emir.db < ./conf-coil.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.

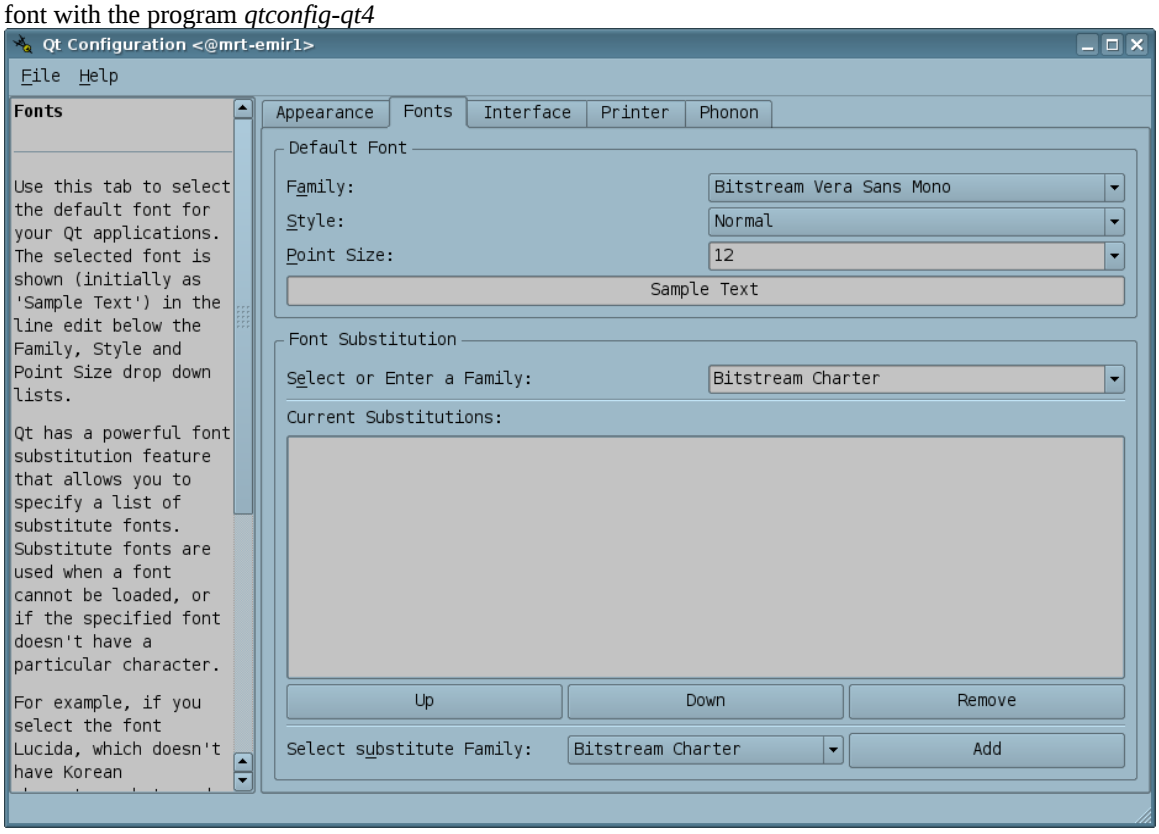
7.2 LO and Mixer data files

The data files for LO and Mixer are in */home/introot/emir/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/emir/data/conf-*.sql*).

8 Tips

8.1 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-qt4*



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.
(on Debian this font is in the *ttf-bitstream-vera* package)