IRAM-COMP-073

Revision: 0
2010-06-10

Contact Author

Institut de RadioAstronomie Millimétrique

Polar Switch Software

Owner Sebastien Blanchet

Keywords: polar switch, software

Approved by: A.Perrigouard	Date: June 2010	Signature: [illegible]

Change Record

REVISION	DATE	AUTHOR	SECTION/PAGE AFFECTED	REMARKS

Content

1 Introduction

In 2010, IRAM has installed a module called POLAR SWITCH, to select for each antenna which polarization (horizontal or vertical) has to be sent to the correlator.
This document is the reference for the polar switch software: installation, technical documentation, daily usage and troubleshooting.

You need also the hardware documentation of the polar switch from Philippe Chavatte: *POLAR_SWITCH.pdf*

2 Installation

2.1 Requirements

2.1.1 Hardware

The minimal requirements for hardware are:

1. CPU: x86 CPU at 500 MHz
2. RAM: 512 MB
3. CAN controller: TPMC816 PMC card from Tews Technologies GmbH

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

2.1.2 Software

Operating system
The software should run on any recent Linux (i386 or x86_64) with a 2.6.x kernel.

Mandatory software

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

All these libraries are very common (except CxxTest), 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	>= 3.5	http://eclipse.org/cdt

	Environment (IDE) for the Eclipse platform.		
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.

2.1.3 Host

For convenience, the software will be installed on `ifprocb.iram.fr` (this computer runs also the IF processor and the FO_RX software)

2.2 Building

2.2.1 Extract code from the repository

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

Then use the Makefile to extract automatically the dependencies

```
$ cd PolarSwitch
$ qmake
$ make get_deps
```

2.2.2 Build the TPMC816 driver

The TPMC816 driver is optional for the simulation. In this case, you can skip this section.

```
$ 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` .

2.2.3 Build the Polar Switch code

```
$ cd ~/develSVN/PolarSwitch
$ ./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 its default settings in `/home/introot/polswi`
Note: the shared libraries are installed in `/home/introot/lib`

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

Warning:

Normally, you need not to change the default settings, but if you wish to update only the software **without reinstalling the settings**, use

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

Nevertheless it is safer to backup `/home/introot/polswi` and `/home/introot/lib` first, so you can restore the original installation in case of errors.

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

2.3 Configure environment

You should add `/home/introot/polswi/bin` to your path.

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

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

2.4 Initial tests

For this initial test, we run

1. The CanManagers to enable the Can Over IP protocol
2. The polar switch simulator
3. The python xml-rpc client: `GetStatus.py`

First we remove the `/dev/tpmc816_*` devices, so the CanManager will run in simulation mode.

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

Now run the `GetStatus.py`

```
# Connect to http://localhost:1090
status.canErrors                  0
status.elapsedTimeInSeconds       9440
status.hvPolar                    0
status.moduleId                   305419896
status.powerSupply1.m5V           5.0
status.powerSupply1.p5V           5.09
status.powerSupply1.volt_3_3      3.29
status.powerSupply1.volt_5_0      5.2
status.powerSupply2.volt_6_5      6.4
status.serial                     136839169
status.temperature                25.0
status.temperatureDateTime        20100519T09:40:08
status.timer7                     3
# GetStatus() rpc call takes  0.00679993629456  seconds
```

2.5 Start application automatically

Add `/home/introot/fo/bin/polswi-init-device.sh` to `/etc/rc.local` to initialize the Polar Switch software at the startup.

This script starts the CanManager and the polswi-server.

3 Internal programs

This section describes internal programs that drive the Polar Switch device. These programs are executed automatically; therefore normal users are not expected to run them.

3.1 CanManager

For a complete description see the document IRAM-COMP-057 *CanIp*

```
$ CanManager -h
CanManager is a bridge between the CAN bus and the CAN/IP protocol
Usage: CanManager [options]
Options:
  -d=name        CAN controller device name to use. If missing,
                      the application runs in simulation mode
  -p=N           Listen to UDP port N
  -l=N           Limit write speed base CAN messages.
                      'N' is in message/sec. Default value = 0 (no limit)

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


Example:
        CanManager -d=/dev/tpmc816_0 -p=2500 -l=20
```

For the Polar Switch software, CanManager must listen on port udp/2501

```
CanManager –d=/dev/tpmc816_1 –p=2501 –l=50
```

3.2 Polar Switch Server

polswi-server is an XML-RPC server to remotely control the polar switch device.

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. This protocol is very simple to use, and can be used from any programming language (many opensource libraries are available)

For a detailed introduction to XML RPC see http://en.wikipedia.org/wiki/XML-RPC

polswi-server listens for XML-RPC calls on http://localhost:1090/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

3.2.1 Syntax

```
$ polswi-server -h
PolarSwitch Server - XML-RPC Server
Listen on port 1090
Usage: polswi-server [options]
Options:
  -v        Display version information
  -h, -?    Display help
```

3.2.2 API Description

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

```
$ cd ~/build/PolarSwitch/scripts/xmlrpc/
$ ./ListMethods.py
# Connect to http://localhost:1090
polarSwitch.getHvPolar
polarSwitch.getStatus
polarSwitch.getTemperature
polarSwitch.initIo
polarSwitch.reset
polarSwitch.setHvPolar
system.listMethods
system.methodHelp
system.methodSignature
system.multicall
system.shutdown
```

```
$ ./Introspection.py
# Connect to http://localhost:1090
----------------------------------------------------------------------
Name       : polarSwitch.getHvPolar(  )
Return Type: int
Description: Get HV Polar setting.
Syntax: polarSwitch.getHvPolar()
Return value : int.
    bit0:  antenna1, 0->H, 1->V
    bit1:  antenna2, 0->H, 1->V
    [...]
    bit11: antenna12, 0->H, 1->V


----------------------------------------------------------------------
Name       : polarSwitch.getStatus(  )
Return Type: struct
Description: Returns the polar switch status

----------------------------------------------------------------------
Name       : polarSwitch.getTemperature(  )
Return Type: struct
Description: Get PolarSwitch temperature
Syntax: getTemperature()
Return a struct
  {
    double   temperature // temperature in Celcius degree
    dateTime temperatureDateTime // Date time of the temperature reading
  }


----------------------------------------------------------------------
Name       : polarSwitch.initIo(  )
Return Type: int
Description: Init IO to their default values
Syntax: initIo()
Return code is always 0


----------------------------------------------------------------------
Name       : polarSwitch.reset(  )
```

```
Return Type: int
Description: reset polar switch
Syntax: reset()
Return code is always 0


----------------------------------------------------------------------
Name       : polarSwitch.setHvPolar(  )
Return Type: int
Description: Get HV Polar setting.
Syntax: polarSwitch.setHvPolar( int a_polarSetting )
    a_polarSetting:
        bit0:  antenna1, 0->H, 1->V
        bit1:  antenna2, 0->H, 1->V
        [...]
        bit11: antenna12, 0->H, 1->V
Return value : always 0.


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

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

3.3 XML RPC client examples

3.3.1 Python examples

3.3.1.1 Minimal example

XML RPC is very easy to use in Python.

For example to call method `polarScript.setHvPolar(12)` on the server, you need only the following lines.

```
import xmlrpclib
server = xmlrpclib.ServerProxy( "http://localhost:1088" )
print server.polarSwitch.setHvPolar( 12 )
```

3.3.1.2 Other Python examples

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

3.3.2 Fortran examples

Unfortunately, there are no native XML-RPC libraries for Fortran, therefore Fortran programs have to use the C library for XML-RPC. (http://xmlrpc-c.sourceforge.net)
For convenience, I have written a C library to hide the XML-RPC. Therefore Fortran programs can call remote procedures with simple C calls.

```
C Header:           libs/Rpc/PolarSwitch_Rpc_Client.hpp
Fortran types:      libs/Rpc/CF_PolarSwitch_Types.f
Compiled library:   bin/release/libpolswiRpc.so
Fortran example:    apps/Fortran/TestRpc
```

Notes:

- All the Fortran calls are described in the header file. There is one C function per XML procedure.
- The Fortran type file shares data types between C and Fortran. It is generated with `c2f`.
- The library has a C interface, but it is written in C++/QT. Therefore QT is required when linking with `polswiRpc`.
- If you wish to redevelop your own Fortran wrapper for `xmlrpc-c`, it is worth looking into the library source (`libs/Rpc`), to have an example.

For example to call method `polarSwitch.setHvPolar(12)` from a Fortran program.

```
PROGRAM test_polswi_set_hv_polar
CHARACTER(64) serverName
INTEGER output

serverName = 'localhost'
ouput = 1

CALL frpc_polswi_set_hv_polar ( TRIM(serverName), 12 )
STOP
END
```

3.3.3 Client/server performances

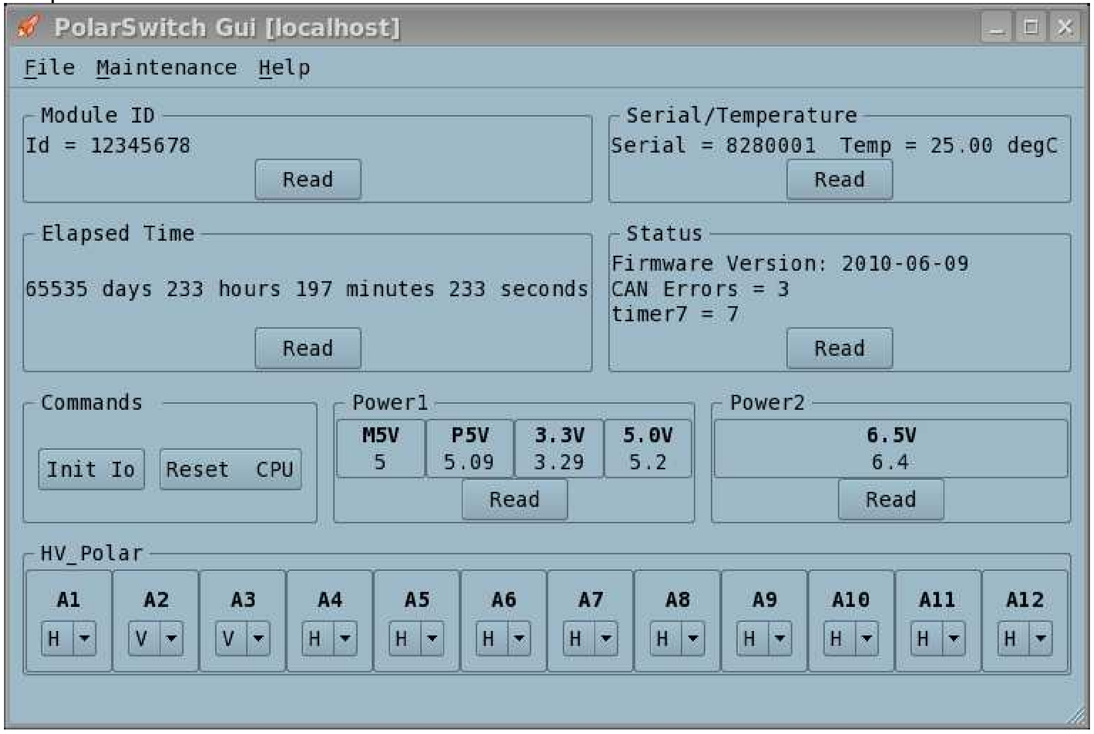
From the client point of view, the total call time (sending + request processing + status receiving) is slightly lower than 10 milliseconds. So we have very good performances.

3.4 Polar Switch Simulator

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

Syntax:

```
polswi-simul
```



Id = 12345678

Serial = 8280001 Temp = 25.00 degC

65535 days 233 hours 197 minutes 233 seconds

Firmware Version: 2010-06-09
CAN Errors = 3
timer7 = 7

M5V	P5V	3.3V	5.0V
5	5.09	3.29	5.2

6.5V
6.4

A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12
H	V	V	H	H	H	H	H	H	H	H	H

4 User programs

4.1 polswi-gui

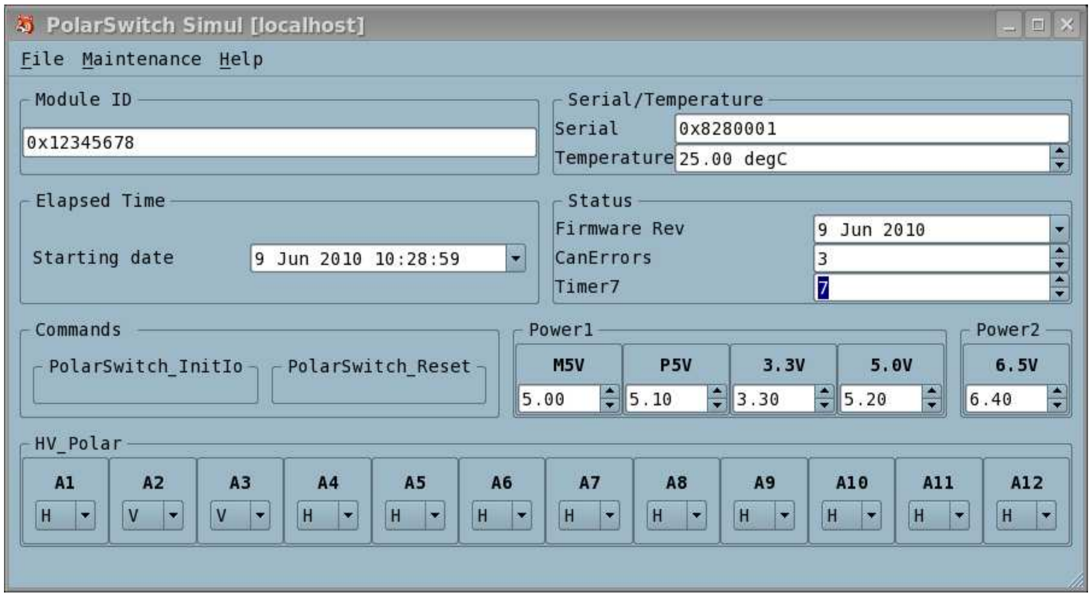
`polswi-gui` is a graphical software to send command to the polar switch device.

Syntax:

```
polswi-gui
```

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

```
$ ssh –Y backend@ifprocb.iram.fr
backend@ifprocb.iram.fr password:
$ polswi-gui
```



Module ID: 0x12345678
Serial: 0x8280001
Temperature: 25.00 degC
Elapsed Time – Starting date: 9 Jun 2010 10:28:59
Status – Firmware Rev: 9 Jun 2010; CanErrors: 3; Timer7: 7
Power1 – M5V: 5.00; P5V: 5.10; 3.3V: 3.30; 5.0V: 5.20
Power2 – 6.5V: 6.40
HV_Polar – A1: H; A2: V; A3: V; A4: H; A5: H; A6: H; A7: H; A8: H; A9: H; A10: H; A11: H; A12: H

With this software, you can

- read all the internal values. If the value is out of range, the widget background becomes orange.
- select the polarity (horizontal or vertical) for each antenna

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

5 Troubleshooting

5.1 CAN maintenance

See document IRAM-COMP-057 *CanIp*

5.2 Modules

You can check if the device drivers are loaded:

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

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

5.3 Processes

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

```
$ ps –f –u oper
```

Look for the following processes:

```
oper      1572  1571  0 Jun10 ?        00:01:01
/home/introot/polswi/bin/release/CanManager -d=/dev/tpmc816_1 -p=2501 -l=50
oper      1599     1  0 Jun10 ?        00:01:25 polswi-server
```

If one of these processes is missing, it means that it has crashed. If it happens, please contact the IRAM computer group. And then you can reboot the computer.