

**Nyalesund 13.2 m control system:
Installation and Basics**

F.J. Beltrán, L. Barbas, P. de Vicente
Informe Técnico IT-CDT 2016-17

Revision History

Version	Date	Author	Update
1.0	01-06-2016	F.J. Beltrán	First draft

Content

1	Summary	4
2	ACS and OS installation	4
3	Weather station installation and tests	9
	3.1. - Weather Station MET4A installation.....	10
	3.1.1. - Wiring.....	10
	3.1.2. - Configuration.....	12
	3.2. - Wind Sensor WMT701 installation.....	14
	3.2.1. - Wiring.....	14
	3.2.2. - Configuration.....	18
	3.3. - Data acquisition software.....	20
4	Counter installation and tests	22
	4.1. - Wiring.....	23
	4.2. - Configuration.....	24
	4.3. - Data acquisition software.....	26
5	Modifications at the control system	27
	5.1. - ACS customization for NyAlesund.....	27
	5.1.1. - ACS components.....	27
	5.2. - FS (Field System).....	30
6	The command line interface. Single dish observations	31
	6.1. - How to use ACS Command Center.....	31
	6.2. - How to use the command line interface.....	34
7	Observations with the Field System	35
8	Aperture efficiency and gain	36
A	Appendix	38

1 Summary

The Norwegian Mapping Authority (hereafter NMA) has started the erection of two geodetic antennas 13.2 m diameter at Ny Alesund. The antennas are basically identical to the 13.2m one in Yebes and will use the same control system as the Yebes one. This report summarizes the works performed to install and test the control system on three computers delivered temporarily by NMA to Yebes. Two of the computers will run the FS and one the telescopes control system. Other equipment was also sent to be tested and integrated in the control system: a MET4 weather station with Vaisala wind sensors and one Agilent counter to be used for the cable measurement and/or the GPS - maser comparison.

The installation of the control system for Ny Alesund was done at the same time that we upgraded the control system at Yebes 13.2 m. We believe that it will ease the installation at other RAEGE antennas and may provide a reliable path and system to similar antennas designed and built by MT-Mechatronics. The telescope control system runs under Debian/Linux and uses ALMA Common Software (ACS) infrastructure.

2 ACS and OS installation

The installation of Debian Jessie was performed on the three computers using a CD (**Debian 8.3.0 AMD-64** version for the control system and **Debian 8.3.0 netinst i386** for FS) and a network connection to upgrade the packages. The routes included in */etc/apt/sources.list* are listed below:

```
deb http://ftp.de.debian.org/debian/ jessie main contrib non-free
deb-src http://ftp.de.debian.org/debian/ jessie main contrib non-free

deb http://security.debian.org/ jessie/updates main contrib non-free
deb-src http://security.debian.org/ jessie/updates main contrib non-free

deb http://ftp.de.debian.org/debian/ jessie-updates main contrib non-free
deb-src http://ftp.de.debian.org/debian/ jessie-updates main contrib non-free

deb http://deb-multimedia.org/ jessie main non-free
deb-src http://deb-multimedia.org/ jessie main non-free
```

Several user accounts were created and their details are summarized in appendix A. Additional packages are required to allow the compilation of the ACS sources and of the components:

```
apt-get update
aptitude install deb-multimedia-keyring
apt-get update
apt-get upgrade
aptitude install g++ gfortran python-dev bison flex libxml2-dev libxslt1-dev
zlib1g-dev autoconf doxygen valgrind procmail openjdk-7-jdk ant python-mysqldb ksh
python-numpy python-scitools python-pip ipython libcfitsio3-dev libfftw3-dev git
libldap2-dev libsasl2-dev blt-dev time tclx8.4-dev emacs python-pexpect python-
libxml2 pychecker cppcheck python-pmw python-libxslt1 python-ldap python-gnuplot
python-pysnmp2 python-setuptools liblas-dev liblapack-dev libtool byacc automake
omniorb omniorb-idl omniorb-nameserver omniidl python-omniorb libomniorb4-dev
libreadline-dev make maven python-virtualenv graphviz tree icedtea-7-plugin
```

During the OS installation any time zone can be chosen, together with the keyboard mapping and the language. It is recommended to use the keyboard mapping that matches the physical available keyboard and English as language. Other languages might generate errors during running time due to a different locale. This is the case for Spanish where commas and dots with numbers have a different meaning. To solve this issue all C or C++ code requires using a locale setting:

```
setlocale(LC_NUMERIC, "C");
```

Choosing English helps avoiding this locale problem. Once the installation is completed the time zone should be changed to UTC reconfiguring package *tzdata*.

```
dpkg-reconfigure tzdata
```

Select first “None of the above” and then UTC.

The European Southern Observatory (ESO) provides support for ACS installation in RedHat and Scientific Linux. Debian is not officially supported, but Jorge Avarias has created a branch of the code that compiles in Debian. Information can be found in <https://github.com/javarias/ACS/wiki/Ubuntu-port>. The steps are described below:

- User account *nymgr* should be created and use to manage the ACS.
- Make the following link (as root):

```
ln -s /bin/tar /usr/bin/gtar
```

- Create an account in <https://github.com>
- Generate a public key:

```
ssh-keygen -t rsa -b 4096 -C your\_email@example.com
```

- Download ACS 2015.4 from github:

```
cd $HOME
git clone -b ubuntu-port --single-branch https://github.com/javarias/ACS.git ACS-2015.4
ln -s ACS-2015.4 ACS
```

- Create alma directory and change ownership (as root):

```
mkdir /alma
chown -R nymgr.nymgr /alma
```

- Copy the following environmental variables using ACS *.bash_profile*:

```
mkdir $HOME/.acs
cp $HOME/ACS/LGPL/acsBUILD/config/.acs/.bash_profile.acs $HOME/.acs/
source $HOME/.acs/.bash_profile.acs -r
```

- Clear variables:

```
unset JAVA
```

```
unset
```

- Add the next line at the end to file $\$HOME/.bashrc$:

```
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
#If a 32 bits OS is used, comment the line above and uncomment the following line
#export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-i386
export M2_HOME=/usr/share/maven
export M2=/usr/share/maven/bin
source $HOME/.acs/.bash_profile.acs -r
```

- Log out and in from *nymgr* account
- Install bash shell by reconfiguring package dash and selecting to use bash:

```
dpkg-reconfigure dash
```

- Download external products:

```
cd $HOME/ACS/ExtProd/PRODUCTS
./download-products.sh
```

- Compile external products. Products are an assorted bundle of packages required by the ACS.

```
cd $HOME/ACS/ExtProd/INSTALL
make all
```

- A correct compilation shows the following log:

```
WARNING: Do not close this terminal: some build might fail!
Create ACS-2015.4
buildTcltk                [ OK ]
buildTAO                   [ OK ]
buildDDS                   [ OK ]
buildOpenSpliceDDS        [ OK ]
buildJacORB                [ OK ]
buildPython                [ OK ]
buildOmniORB               [ OK ]
buildMico                  [ OK ]
buildEclipse               [ OK ]
WARNING: Now log out and login again to make sure that
        the environment is re-evaluated!

_oOo_
. . . 'all' done
```

- Although the output seems to be correct, OpenSplice DDS will not get compiled in a 64 bit architecture. The result of the compilation can be found in the different log files. Tcl/TK will also get some errors that we can neglect.
- Compilation of ACS:

Modify file $\$HOME/ACS/Makefile$ and leave the following line like:

```
#GMP = gmp
GMP =
```

Execute *.bashrc* to get the correct environment:

```
cd
. .bashrc
cd $HOME/ACS
export INTROOT=$ACSR00T
```

and compile:

```
make build
```

No errors should appear on the screen. A correct compilation shows the following log:

```
cat: /etc/redhat-release: No such file or directory
Evaluating current SCM tag
SCM tag is 2014-06-ubuntu-544-gc7c22b8
##### Clean Build Log File: build.log #####
##### Check directory tree for modules #####
##### Prepare installation areas #####
##### (Re-)build ACS Software #####
##### LGPL/Kit/doc SRC
##### LGPL/Kit/acs SRC
##### LGPL/Kit/acstempl SRC
##### LGPL/Tools/tat SRC
##### LGPL/Tools/expat WS
##### LGPL/Tools/loki WS
##### LGPL/Tools/extjars SRC
##### LGPL/Tools/antlr SRC
##### LGPL/Tools/hibernate SRC
##### LGPL/Tools/extpy SRC
##### LGPL/Tools/cppunit SRC
##### LGPL/Tools/getopt SRC
##### LGPL/Tools/FITS SRC
##### LGPL/Tools/astyle SRC
##### LGPL/Tools/swig SRC
##### LGPL/Tools/xercesc SRC
##### LGPL/Tools/xercesj SRC
##### LGPL/Tools/castor SRC
##### LGPL/Tools/gui MAIN
##### LGPL/Tools/xsdoc SRC
##### LGPL/Tools/extidl WS
##### LGPL/Tools/vtd-xml SRC
##### LGPL/Tools/oAW SRC
##### LGPL/Tools/shunit2 SRC
##### LGPL/Tools/log4cpp WS
##### LGPL/Tools/scxml_apache SRC
##### LGPL/CommonSoftware/jacsutil SRC
##### LGPL/CommonSoftware/xmljbind SRC
##### LGPL/CommonSoftware/xmlpybind SRC
##### LGPL/CommonSoftware/acseridl WS
##### LGPL/CommonSoftware/acsidlcommon WS
##### LGPL/CommonSoftware/acsutlpy SRC
##### LGPL/CommonSoftware/acsutl WS
##### LGPL/CommonSoftware/acstartup SRC
##### LGPL/CommonSoftware/loggingidl WS
##### LGPL/CommonSoftware/logging WS
##### LGPL/CommonSoftware/ac serr WS
##### LGPL/CommonSoftware/ac serrTypes WS
##### LGPL/CommonSoftware/ac sQoS WS
##### LGPL/CommonSoftware/acstthread WS
##### LGPL/CommonSoftware/ac scomponentidl WS
##### LGPL/CommonSoftware/cdbidl WS
##### LGPL/CommonSoftware/maciidl WS
##### LGPL/CommonSoftware/baciidl WS
##### LGPL/CommonSoftware/ac sncidl WS
##### LGPL/CommonSoftware/ac sjlog SRC
##### LGPL/CommonSoftware/repeatGuard WS
##### LGPL/CommonSoftware/loggingts WS
##### LGPL/CommonSoftware/loggingtsTypes WS
##### LGPL/CommonSoftware/jacsutil2 SRC
```

```

##### LGPL/CommonSoftware/cdb WS
##### LGPL/CommonSoftware/cdbChecker SRC
##### LGPL/CommonSoftware/codegen SRC
##### LGPL/CommonSoftware/cdb_rdb SRC
##### LGPL/CommonSoftware/acsalarmidl WS
##### LGPL/CommonSoftware/acsalarm SRC
##### LGPL/CommonSoftware/acsContainerServices WS
##### LGPL/CommonSoftware/acscomponent WS
##### LGPL/CommonSoftware/recovery WS
##### LGPL/CommonSoftware/basenc WS
##### LGPL/CommonSoftware/archiveevents WS
##### LGPL/CommonSoftware/parameter SRC
##### LGPL/CommonSoftware/baci WS
##### LGPL/CommonSoftware/enumprop WS
##### LGPL/CommonSoftware/acscallbacks SRC
##### LGPL/CommonSoftware/acsdemonidl WS
##### LGPL/CommonSoftware/jacsalarm SRC
##### LGPL/CommonSoftware/jmanager SRC
##### LGPL/CommonSoftware/maci WS
##### LGPL/CommonSoftware/task SRC
##### LGPL/CommonSoftware/acstime WS
##### LGPL/CommonSoftware/acsnrc WS
##### LGPL/CommonSoftware/acsnrcdds SRC
##### LGPL/CommonSoftware/acsdemon WS
##### LGPL/CommonSoftware/acsllog WS
##### LGPL/CommonSoftware/acstestcompcpp SRC
##### LGPL/CommonSoftware/acsexmpl WS
##### LGPL/CommonSoftware/jlogEngine SRC
##### LGPL/CommonSoftware/acspycommon SRC
##### LGPL/CommonSoftware/acsalarmpy SRC
##### LGPL/CommonSoftware/acspy SRC
##### LGPL/CommonSoftware/comphelpgen SRC
##### LGPL/CommonSoftware/XmlIdl SRC
##### LGPL/CommonSoftware/define WS
##### LGPL/CommonSoftware/acstestentities SRC
##### LGPL/CommonSoftware/jcont SRC
##### LGPL/CommonSoftware/jcontnc SRC
##### LGPL/CommonSoftware/nsStatisticsService SRC
##### LGPL/CommonSoftware/jacsalarmtest SRC
##### LGPL/CommonSoftware/jcontextmpl SRC
##### LGPL/CommonSoftware/jbaci SRC
##### LGPL/CommonSoftware/monitoring MAIN
##### LGPL/CommonSoftware/acssamp WS
##### LGPL/CommonSoftware/mastercomp SRC
##### LGPL/CommonSoftware/acspyexmpl SRC
##### LGPL/CommonSoftware/nctest WS
##### LGPL/CommonSoftware/acscmdcenter SRC
##### LGPL/CommonSoftware/acssim SRC
##### LGPL/CommonSoftware/bulkDataNT SRC
##### LGPL/CommonSoftware/bulkData SRC
##### LGPL/CommonSoftware/containerTests MAIN
##### LGPL/CommonSoftware/acscourse WS
##### LGPL/CommonSoftware/ACSLaser MAIN
##### LGPL/CommonSoftware/acsguis MAIN
##### Benchmark/util SRC
##### Benchmark/analyzer SRC
##### LGPL/acsBUILD SRC
##### DONE (Re-)build ACS Software #####
... done

```

A detailed log of the compilation can be found in *build.log*. If there an error related to jacorb is shown, there may be some problem with the corresponding external product. Have a look at *extprods.links.txt* in this case.

- Create *introot* directory structure:

```

cd
getTemplate

```

selecting the following options:

```
directoryStructure
createINTROOTarea
introot
"Press ENTER 3 times"
```

- Make the following link in order to use Python packages installed in Debian

```
ln -s /usr/lib/python2.7/dist-packages /alma/ACS-2015.4/Python/lib/python2.7/dist-packages
```

3 Weather station installation and tests

Ny Alesund weather equipment consists of a MET4A weather station and a WMT701 wind sensor from Vaisala. Fig.1 shows the basic connection scheme for this equipment. As explained later, the communication with the weather equipment is done via a Lantronix USD2100 device which converts two RS232 interfaces into an Ethernet interface. The RS232 configuration used is summarized in Fig. 2:

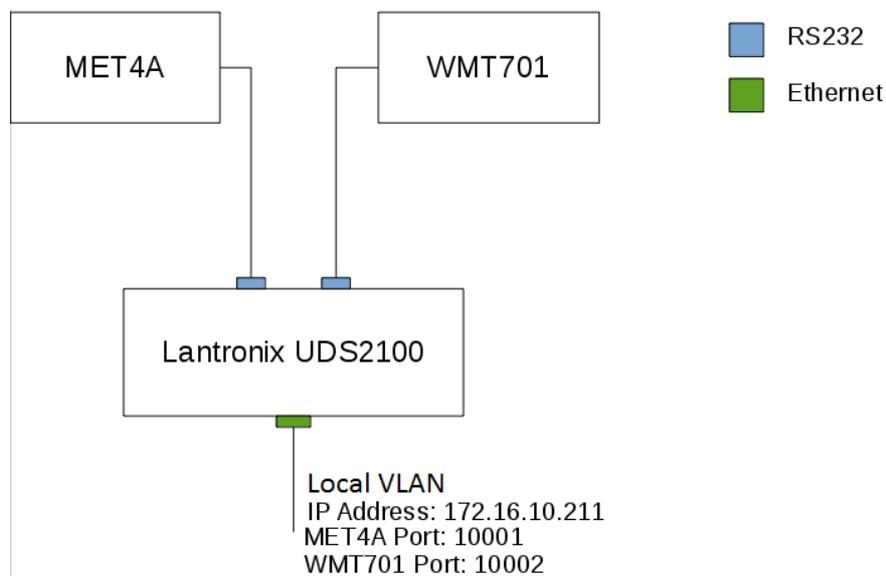


Figure 1: Weather equipment basic connection scheme.

RS232 configuration – MET4A and WMT701

Type	Value
Baud Rate	9600
Data Bits	8
Parity	None
Stop Bits	1
Flow Control	None

Figure 2: Weather equipment RS232 configuration.

3.1. - Weather Station MET4A installation

3.1.1. - Wiring

The remote communication is done via a RS232 or RS485 interfaces. We used an RS232 interface connected to a Lantronix UDS2100 device, which allows to monitor and control two RS232 ports via Ethernet: one device on TCP/IP port 10001 and the other one on 10002. Fig. 3 is a photograph of the device used during the tests in Yebes:



Figure 3: Lantronix UDS2100 device.

The RS232 communication parameters can be set at the Lantronix using its web interface:

- Baud Rate: 9600
- Data bits: 8
- Parity: None
- Stop bits: 1
- Flow control: None

The weather station is connected to serial port 1 and uses the TCP/IP port 10001.

Figs. 4, 5 and 6 show the wiring used in the Yebes Observatory while testing the station and the software:



Figure 4: The connector with the blue ring is below the main block. It is protected for outdoor operations. This cable supplies power and communications to the weather station.



Figure 5: RS232/RS485 weather station MET4A adapter. Power supply is on one side (voltage + ground). The two other ends are for RS232 connections: one from the station and the other to the receiving end. They can be easily identified from their tags.



Figure 6: Weather Station MET4A power supply

The adapter shown in Fig. 5 is connected at the end of the cable displayed in Fig. 4. The adapter allows RS485 or RS232. RS485 requires delivering power (5 volts) to the adapter, whereas this supply is not required for RS232. RS485 wires can be connected on the adapter bottom side of Fig. 5, where Rx and Tx sockets are tagged. The RS232 connector can be connected to the adapter connector tagged as RS – 232. Only one of these two modes should be connected since there is no switch that allows to choose

between both. RS485 mode is a good option for long distances since it uses two wires for the signal. However, since we are using a Lantronix converter and it will be located close by, the RS232 was selected as the operating mode for the weather station.

3.1.2. - Configuration

The weather station installed is a MET4A model by Paroscientific, Inc. This station yields pressure, temperature and relative humidity measurements with the following accuracy:

- Pressure accuracy is better than ± 0.08 hPa from 500 to 1100 hPa.
- Temperature accuracy is better than $\pm 0.2^\circ\text{C}$ from -50 to $+60$ deg C.
- Relative humidity accuracy is better than $\pm 2\%$ percent from 0 to 100%RH at 25 deg C.

The weather station can be polled, a request is followed by a measurement, or it can be monitored continuously. In the latter case a start command starts a periodic measurement until a stop command is received. Ny Alesund weather station was configured in poll mode. Strings commanded require a carriage return and a line feed as termination characters: '\r\n' respectively. ASCII commands used are summarized in the following table:

Command	Response	Description
*0100P9\r\n	\$WIXDR,P,<P>,B,<SN>,C,<T>,C,<SN>,<H>,P,<SN>	Polls the weather station. In the response: <P> is the pressure value in bars, <T> is the temperature value in deg C and <H> is the relative humidity value in percent. <SN> is not relevant.
*0100PP\r\n	\$WIXDR,P,<P>,B,<SN>,C,<T>,C,<SN>,<H>,P,<SN>	Monitors continuously the weather station ⁽¹⁾ . The response format is the same as the previous poll command.
*0100EW*0100UN=3\r\n	*0001UN=3	Sets the pressure measurement unit. This command sets the measurement unit to 3, which means bars . The response should have the same value as the command sent.
*0100UN\r\n	*0001UN=3	Gets the pressure measurement unit.
*0100EW*0100PI=1000\r\n	*0001PI=1000	Sets pressure and temperature integration times to 1 second. The integration time can be set with 1ms resolution approximately. The response should have the same value as the command sent.
*0100EW*0100TI=1000\r\n	*0001TI=1000	Sets only temperature integration time to 1 second. The integration time can be set with 1ms resolution approximately. The response should have the same value as the command sent.

(1): Pressure measurement unit should be **bars**.

Other commands to change the configuration, like baud rate, or to restrict the measurement to only one probe (temperature, humidity or pressure) are also available. Further information can be looked up at the weather station handbook.

3.2. - Wind Sensor WMT701 installation

3.2.1. - Wiring

Wind measurements are delivered by WMT701 wind sensor, one of the models from the WMT700 series manufactured by Vaisala. This model is built without moving parts. The maximum measurable speed is 40m/s. The characteristics of the sensor are coded in the model name: WMT701 B2A0A003B1A2:

- **Temperature range (B):** -40°C/+60°C.
- **Heating (2):** Heated transducers. The heating requires a 24/36V DC power supply delivering 40W power.
- **Digital communication interface (A):** RS485 isolated. It can be changed to RS232 using the wind sensor configuration.
- **Digital communication profile (0):** WMT70, Baud rate = 9600, Data bits = 8, Parity = None, Stop Bits = 1. This profile allows getting wind measures individually. Changing the wind sensor configuration, continuous measurements are possible.
- **Digital communication units (A):** Meters per second.
- **Analog output signals for wind speed channel (0):** Disabled.
- **Analog output signals for wind direction channel (0):** Disabled.
- **Cable (3):** Cable 10 m, cable connector, open leads on one end.
- **Mounting adapter (B):** Adapter 228869 with WMT70FIX70 (suitable also for inverted mounting). Standard adapter with general purpose fix.
- **Reserved for future purposes (1)**
- **Accessories (A):** None.
- **Manual (2):** English manual.

As specified above, the wind sensor has an RS485 interface as default digital communication interface. However, as explained in the previous section, since we are using short distances to connect a Lantronix device the configuration was changed via software to use RS232 as its primary interface.

This wind sensor comes with a 10 meter cable connected to the wind sensor end and exposed wires on the other end. Fig. 7 shows the open wires at the end of the 10

meter cable and their function:

Power Supply					Wire Colors	Pin
Operating Power Supply					White	1
Operating Power Supply Ground					Gray-Pink	11
Heater Power Supply					Gray	5
Heater Power Supply					Pink	6
Heater Power Supply Ground					Blue	7
Heater Power Supply Ground					Red	8
Enclosure Ground					Shield	Shield
Analog Outputs						
Analog Output AOUT2, Wind Direction					Brown	2
Analog Output AOUT1, Wind Speed					White-Green	13
Reference Input for AOUT2 (simulated potentiometer)					White-Gray	17
Analog Output Ground					Red-Blue	12
COM port	RS-232	RS-422	RS-485	SDI-12		
COM2	RS232Rx	RxB	RxB	-	Green	3
	RS232Tx	TxB	TxB	Data	Yellow	4
	-	TxA	TxA	-	Brown-Green	14
	-	RxA	RxA	-	White-Yellow	15
COM1 and COM2 Communication Ports Ground					Violet	10
COM1 (service port)	RS-485, B				Black	9
	RS-485, A				Brown-Yellow	16

Figure 7: Wires set at the end of the 10 meters' cable.

Communication with the wind sensor uses RS232 DB9 pins 2, 3 and 5 as shown in the table below:

Type	Cable color	RS232 DB9 Pin
Tx	Yellow	2
Rx	Green	3
Ground	Violet	5

The wires shown are cross connected, that is, the wind sensor Tx wire is connected to RS232 Rx pin (2) and the wind sensor Rx wire is connected to RS232 Tx pin (3). The RS232 DB9 pin model used to connect the wires is shown in Fig. 8 and the connection done in Fig. 9. In Fig. 9 the RS232 connector is reversed, so pins are reversed with respect to RS232 Pin model in Fig. 8.

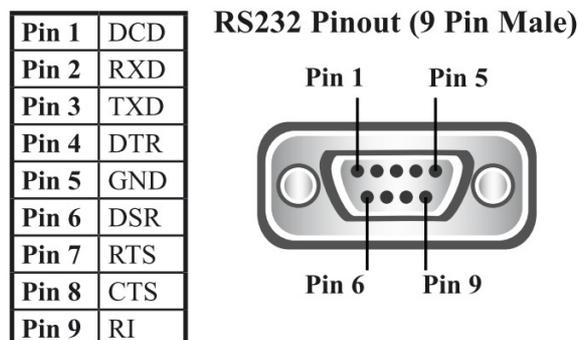


Figure 8: RS232 DB9 PIN model.

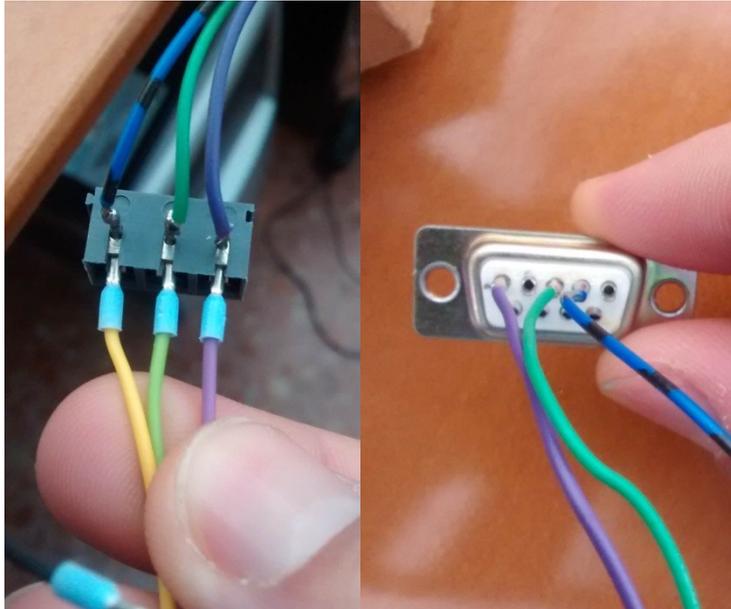


Figure 9: The wind sensor WMT701 communication connection

The wind sensor is connected to serial port 2 and uses the TCP/IP port 10002. This serial port is configured with the following parameters:

- Baud Rate: 9600
- Data bits: 8
- Parity: None
- Stop bits: 1
- Flow control: None

The wind sensor requires power supply for its operation and for the heater.

For the operating power supply the following connections are required:

- **White:** Operating power supply. Voltage must be between 9V and 36V. We used 20V as an intermediate value.
- **Grey-Pink:** Operating power supply ground.

Fig. 10 shows the wiring used to connect the wind sensor power supply to a lab power supply. Fig. 11 shows the voltage used in the lab power supply and the amperage limit set to avoid problems with the wind sensor. Fig. 12 shows the wind sensor current consumption while being connected. Fig. 13 shows the wind sensor current consumption once it has attained a stable value.

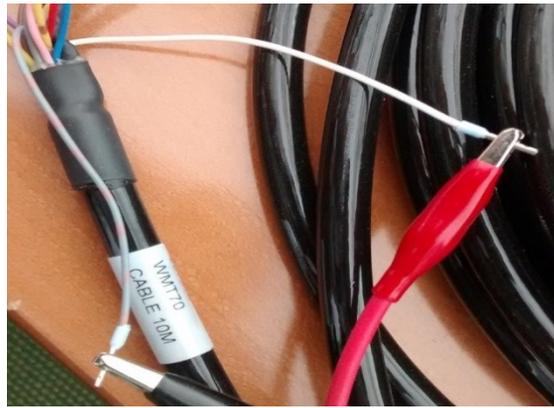


Figure 10: Operating power supply of the wind sensor.



Figure 11: Voltage and amperage used in lab power supply.



Figure 12: Wind sensor current consumption while being connected.



Figure 13: Wind sensor current consumption after stabilizing.

The heater power supply requires different connections:

- **Grey and Pink wires:** Heater power supply. Voltage required: 24/36V DC, power: 40W.
- **Blue and Red wires:** Heater power supply ground.

3.2.2. - Configuration

The wind sensor has two operation modes: the measurement mode and the configuration mode. The former is the standard mode used while measuring. The latter is used for setting the parameters.

When powering up the sensor the mode is set to measurement.

As explained before, string commands require a carriage return and a line feed as termination characters, '\r\n' respectively.

3.2.2.1. - Configuration mode

To enter into the configuration mode the following string should be issued:

```
$00PEN\r\n
```

The "0" character in the above string refers to the wind sensor address. The wind sensor address is "A" by default, but "0" allows addressing any device. So either "A" or "0" should be used.

While the configuration mode is on, there is a prompt (>) which is not present in the measurement mode. Commands available in configuration mode are:

Command	Description
?	Shows a list with all configuration commands.
CLOSE	Changes to measurement mode from configuration mode
G	Shows all configuration parameters
MEAS	Measures the wind with the current configuration. Data is not received until a POLL command is sent.
POLL <id_message>	Requests the delivery of the last wind measurement. The "id_message" is a number which indicates the message and data type to be received. The "id_messages" ranges from 1 to 32. Most types are fixed, but from 1 to 4 are configurable.
RESET	Resets the wind sensor.
S <parameter>,<value>	Modifies the value of one configuration parameter.
START	Starts to receive wind measurements continuously. To receive these measurements continuously the wind sensor should be set to respond automatically.
STOP	Stops the continuous measurement mode.

Operation of the wind sensor in poll mode and usage of RS232 interface required some configuration that we describe below. All changes performed are listed below and achieved using command "S <parameter>,<value>" within the configuration mode.

- **S autoPort,2:** This parameter indicates the serial port to be used to send and receive wind sensor data. By default, this parameter is set to port 1, which only

has a RS485 interface, while port 2 allows choosing between the RS485 interface and the RS232 interface.

- **S msg1,\$\ws,\wd,\wp,\wm\cr\lf**: This message type sends the following information: mean wind speed, mean wind direction, maximum wind speed and minimum wind speed. These measurements are done during a period of time that is specified in the parameter “wndAvg”, whose default value is 1 second. To get the maximum and minimum wind speed during the last 10 minutes, we save the values for these speeds during the last 10 minutes and we take the maximum and minimum value respectively.
- **S com2_interf,2**: Indicates the serial interface to use in port 2. By default, the interface is set to RS485 (0). Changing this value to 2 forces to use the RS232 interface.

3.2.2.1. - Poll mode

In order to use the poll mode to receive individual measurements the following two commands were used:

Command	Description
\$0meas	Requests the wind sensor to take a wind measurement.
\$0poll,1	Requests the wind sensor to return a wind measurement with the format specified with parameter “msg1”. Other formats are possible changing the parameter, 1 in in this case.

The time needed by the wind sensor to complete one measurement after having issued a “\$0meas” is 1 second. Configuration parameter “wndAvg” uses 1 second as default.

3.3. - Data acquisition software

The weather data acquisition software is installed in host “ny13ctl”, user account “meteo”. Details of this account are in the appendix.

The data acquisition is performed by a daemon (“*weatherd_socket*”) written in C++. The daemon commands the weather station and the wind sensor and writes the data received in shared memory. Continuous measurements have been avoided after testing it. In this mode some data can be lost if the periodicity of the weather station and the wind sensor are not exactly the same. Polled measurements avoid this problem and allow getting data without losses. The polling has a periodicity of 2 seconds and is done in several steps:

1. Requests to start measurement in both sensors. Two commands are sent, one per sensor.

2. Waits one second for the wind sensor to complete and requests the wind measurement.
3. Waits for one second and reads the responses from both sensors.

The wind sensor delivers the maximum and minimum speed of the wind for the integration period (2 seconds). Since we are interested in the maximum and minimum values for the last 10 minutes, the daemon stores these parameters in two arrays of 300 elements filling it like a FIFO. After a whole cycle of 300 measurements is completed, the maximum and minimum are computed and stored in shared memory together with the other data received.

There is a second daemon (“meteoServerNetMC”) written in C++ which delivers the weather data stored via UDP socket. A main UDP server opens, upon request, UDP ports where the weather data can be polled via secondary UDP servers. Once the main server receives a request a secondary server is opened in one of the available ports. Each secondary server is independent, so each client has a dedicated secondary server where it can poll data. To avoid a big amount of secondary servers opened, if these servers don’t receive a request in 2 minutes they close their port. The main server uses port 67010 and secondary servers’ ports range from 67011 to 67040.

An ACS component (“wStationComNet”) is used as client for the latter daemon. Every 2 seconds the client polls the weather data. This short period allows monitoring the wind speed and direction almost continuously.

Weather data are stored in a MySQL database. The host where database is stored and its characteristics are summarized in the following table:

Parameter	Value
HOST	
Name	ny13ctl
User	meteo
Password	met4g
MYSQL	
User	meteodbuser
Password	met4g
Database name	meteodb
Table name	weatherlog

The next table contains the description of the MySQL table used to store the weather data:

Parameter	Full name	Data type
Id	Row Identifier	Unsigned Integer
date	Date	Timestamp
MJD	Modified Julian Date	Double
Temp	Temperature	Double
Hum	Relative Humidity	Double

Pres	Pressure	Double
WndSp	Instantaneous Wind Speed	Double
WndDir	Instantaneous Wind Direction	Double
MaxWndSp	10 Minutes Maximum Wind Speed	Double
MinWndSp	10 Minutes Minimum Wind Speed	Double

The meteorological data is stored in the database from a script (“weatherlog”) which is executed every 5 minutes using Linux crontab. This database can be read using MySQL commands. If we want to do manual tests the first step consists in logging into the MYSQL:

```
mysql -u meteodbuser -p
```

After providing MySQL password one should select the database (“meteodb”) and later do a search in a specific MySQL table, (“weatherlog”):

```
USE meteodb;
SELECT * FROM weatherlog ORDER BY id DESC LIMIT 3;
```

The last three stored values will be printed. We provide an example Python script that queries the database and delivers the last 10 measurements:

```
#!/usr/bin/env python

import MySQLdb

header = ['MJD', 'Temp', 'Hum', 'Pres', 'WndSp', 'WndDir', 'MaxWndSp', 'MinWndSp']
row_format = "{:>19}" * (len(header) + 1)

DB_HOST = 'localhost'
DB_USER = 'meteodbuser'
DB_PASS = 'met4g'
DB_NAME = 'meteodb'

db_data = [DB_HOST, DB_USER, DB_PASS, DB_NAME]

conn = MySQLdb.connect(*db_data)
cursor = conn.cursor()

n_data = 10

cursor.execute("SELECT * FROM weatherlog ORDER BY id DESC LIMIT %d;" % n_data)

data = cursor.fetchall()

if len(data) > 0:
    print row_format.format("Date", *header)
    for row in data:
        aux = row[2:]
        print row_format.format(str(row[1]), *aux)
```

4 Counter installation and tests

Frequency counters and timers are used to measure frequency and time intervals. To measure the frequency drift of the station maser versus the GPS theoretical one, a

pulse per second (PPS) from the maser and the GPS are continuously compared. The time difference between both pulses varies with time and this difference as a function of time is used to determine the relative frequency error of the maser assuming that the GPS provides a stable 5 MHz signal. When a timer is used to do a dual input time interval measurement, the edge type of the square signal (positive or negative) for both input signals should be specified. According to the edges selected, the different time intervals that can be measured are shown in Fig. 14. Since we want to measure the time interval between the maser and GPS, the edges should have the same sign. For our purpose, we select the positive edges for both input signals.

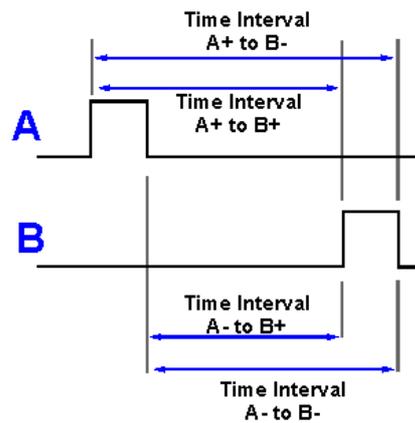


Figure 14: Time intervals according to the edges selected for both input signals.

4.1. - Wiring

Fig. 15 shows HP 5323A back panel. It is used to connect the power supply and the communication connectors. Remote communication is achieved with Ethernet.



Figure 15: 53230A counter back panel

5323A counter back panel has the following connections:

- Power supply.
- Ethernet interface.
- GPIB interface
- USB interface (type B).
- BNC connectors. These connectors can get or put different reference signals:
 - Ext Ref In
 - Gate In/Out
 - Int Ref Out
 - Trig In
- Optional connectors. In Fig. 2 these connectors are covered or empty.

Three connections are required for standard operations: the Ethernet interface, the power supply and the reference which should be a 5 MHz or a 10 MHz signal from the maser. If the reference is not connected the device will deliver non-accurate measurements.

The front panel is shown in Fig. 16. This panel has buttons that configure the counter manually, two BNC connectors (Ch. 1 and 2) where the signals to be measured are injected and a USB type A interface.



Figure 16: 5323A counter front panel

Fig. 16 shows an optional BNC connector (Ch. 3) used to measure signals whose frequency is up to 6 or 15 GHz, depending on the chosen option. This latter connector was covered in the counter used.

4.2. - Configuration

As mentioned previously time differences between a pulse per second (PPS) from a maser and a GPS are measured with a 53230A universal counter which works as a timer. HP 53230A universal frequency counter/timer has LXI Class C compliance. LXI is an instrumentation equipment standard that specifies Ethernet as the main communication interface. The characteristics of this counter are listed below:

- **Input channels:** Ch1 and Ch2 can measure signals until 350 MHz. Ch3 is an optional channel which can measure signals whose frequency is until 6 or 15 GHz, depending on the option chosen.
- **Frequency resolution:** 12 digits per second.
- **Single-shot time resolution:** 20 ps.

Further information about counter characteristics can be looked up at the counter handbook.

The counter accepts SCPI commands via Ethernet. String commands require a line feed ($\backslash n$) as termination character. The commands below set the counter configuration to measure time intervals between the maser and the GPS:

- Resets the device, clears status byte and disables status event and service request registers:

```
*RST\
*CLS\
*SRE 0\
*ESE 0\
STAT: PRES\
```

- Counter configuration:

```

CONF:TINT\n //Counter will measure time intervals
INP1:LEV:AUTO OFF\n //Triggers level won't set automatically
INP2:LEV:AUTO OFF\n
INP1:LEV1 <INP1_value> V\n //Sets triggers level to input arguments <INP1_value>
INP2:LEV1 <INP2_value> V\n //and <INP2_value> values
INP1:SLOP POS\n //Selects positive edges
INP2:SLOP POS\n
INP1:IMP 50\n //Sets 50 Ω as input impedance
INP2:IMP 50\n
INP1:COUP DC\n //Indicates direct current input
INP2:COUP DC\n
INP1:FILT OFF\n //Turns off input filters
INP2:FILT OFF\n
INP1:NREJ OFF\n //Disables noise rejection filters
INP2:NREJ OFF\n

```

- To poll the counter a request like the following string should be sent:

```
READ?\n
```

The counter answers automatically to the latter command providing the time interval measurement.

4.3. - Data acquisition software

Time intervals are also stored in a MySQL database. The host where the database is stored and its characteristics are summarized in the following table:

Parameter	Value
HOST	
Name	ny13ctl
User	maser
Password	C0unt3r
MYSQL	
User	maserdbuser
Password	C0unt3r
Database name	maserdb
Table name	maserdata

The next table shows the description of the MySQL table used to store the weather data:

Parameter	Full name	Data type
Id	Row Identifier	Unsigned Integer
date	Date	Timestamp
MJD	Modified Julian Date	Double
Diff_10m	Time difference between GPS and the maser in 10 minutes	Double
Rms_10m		Double

The data acquisition is performed by a C++ daemon ("*GpsMaserDif*"). The daemon sets the counter configuration upon start and gets and writes the data as follows: the counter is polled continuously and time intervals are stored in shared memory and added to an auxiliary variable, every second. The average of the PPS differences is computed every 10 minutes and this latter value stored in a MySQL database.

A second daemon ("*maserComNetMC*"), also coded in C++, delivers the counter data stored in shared memory via an UDP socket. Like the daemon used in the acquisition of meteorological data, there are two types of UDP servers. A main UDP server opens UDP ports upon request where stored time intervals can be polled via secondary UDP servers. Once the main server receives a request a secondary server is opened in one of the available ports. Each secondary server is attached and dedicated to an independent client. To avoid a large amount of secondary servers, these close their port after a 2 minute inactivity period. The main server uses port 67041 and secondary servers ports range from 67042 to 67071.

An ACS component ("*gpsMaserComNet*") is used as a client for the latter daemon. Every 2 seconds the client polls time differences between the GPS and the Maser. This short period allows monitoring the instantaneous time difference.

5 Modifications at the control system

5.1. - ACS customization for NyAlesund.

Environment variables are customized at specific directories where tools and databases are stored. These variables are set in "*/home/nymgr/.bashrc*" so that they are properly set after each login:

```
export INTROOT=$HOME/introot
export ACS_CDB=$HOME/nyalesund/NYALESUND_CDB
```

This file requires further tuning which will be explained later.

5.1.1. - ACS components

Components are tools (like network classes in the object oriented world) that allow each instance to communicate with ACS and to deliver services to humans or to other components. For example, "*wStationComNet*" component delivers meteorological data that can be used to monitor weather conditions by the telescope staff and by other components. Table 1 lists the components developed in this project:

Component Name	Source code	Lang.	Container Name	Description
MET4_COMNET	wStationComNet	C++	RCweather	Delivers meteorological data from weather station and wind sensor
GPS_MASER_COMNET	gpsMaserComNet	C++	RCmaser	Delivers time difference from 53230A counter
POW_1/2	powermeter	Java	RCjbackends	Controls gets power measurement from the powermeter
CURRENT_SCAN_1/2	scanObs	Python	RCscan	Selects parameters to use in observation scans
OBS_NYALE_1/2	raegeObserver	Python	RCobserver	Allows doing observations using the radiotelescope.

ANT_NYALE_1/2	raegeAntenna	Python	RCantenna	Used to move the antenna. It uses "ACU13M_1/2" in its methods.
OBS_DB_1/2	ObsDatabase	Python	RCvobs	Stores observation data in "nyinfo_1/2" tables of "observations" MySQL database.
REFR_13M	Atm	Java	RCatm	Delivers atmospheric data like opacity. Uses the program "\$HOME/bin/atm"
ACU13M_1/2	acu13m	C++	RCservos	Low level component that receives information and sends commands to ACU and HCU
mb_fitsW_1/2	fitsWriter	C++	RCdatawriter	Writes observation data in FITS format.
mbfitsExt_1/2	pipeline/mbfitsExtractor	Python	RCpywriter	Extracts and reads MBFITS data.
ampCal_1/2	pipeline/ampCalibrator	Python	RCpycal	Manages amplitude calibration.
gildasPipeline_1/2	pipeline/gildasProcessor	Python	RCpygildas	Rewrites input data to be compatible with Gildas FITS
FS13MCOMPY_1/2	fsNet	C++	RCfieldsystem	Allows communication between ACS and FS. These components work as client.
NONE (*)	efemAstroSource efemObservatorySite efemAstroTime efemSatellite	C++	RCefem	These components deliver information about situation and time of sources and the radiotelescope

All these components, together with their relevant information, are listed in file "*\$ACS_CDB/CDB/MACI/Components/Components.xml*". New components can be added modifying this file.

All component instances are included in a database composed of a directory and subdirectories, one per instance, which contains a file. Each file represents an instance and contains properties, defined as variables, like the IP address(es) or the connection port(s) required for its operation. The database allows using the same source code to initialize different component instances. A twin telescope can benefit from this policy. In our control system some components do the same function but in different radio-telescopes. For example, *ACU13M_1* and *ACU13M_2* share the same code but they differ in their internal variables. Also, components will only request services from other components that share the same number in their names. For example, *ANT_NYALE_1* will request services from *ACU13M_1*, but not from *ACU13M_2*. There are other components, like *MET4_COMNET*, that are used by components from both radio-telescopes.

Some components need common environment variables to work correctly. These variables are initialized in "*/home/nymgr/.bashrc*":

```

# Directory where data are stored
export NYALE1DATA=/home/nymgr/data1
export NYALE2DATA=/home/nymgr/data2

# Directory where modules are stored
export TEL13MROOT=$HOME/introot

# Directory where catalogs are stored
export NYALECATA=$HOME/nyalesund/Catalogs

# Default project ID of NYALE1 and NYALE2 (Should be change every year)
export NYALE1_PROJECT_ID="N-01.C-0001-2016"
export NYALE2_PROJECT_ID="N-02.C-0001-2016"

# Configuration file used by efemerides components
export JPLEFEM=$INTROOT/config/bintotal.405

# Gildas environment variables
export GAG_ROOT_DIR=/home/nymgr/gildas/gildas-exe-may13b
export GAG_EXEC_SYSTEM=x86_64-debian7-gfortran
source $GAG_ROOT_DIR/etc/bash_profile

```

ACS also provides the tools to implement and use clients. These clients are mostly developed in Java since most of them are graphical. The code is also stored in the same primary directory ("*/home/nymgr/nyalesund*") that contains all the control software. Two basic graphical clients and one line command one are used:

- **AcuClient13m_1/2**: Shows information received from the ACU when *ACU13M_1/2* are initialized respectively. To start both clients type: "*acsStartJava alma.servosystem.AcuClient13m_1*" and "*acsStartJava alma.servosystem.AcuClient13m_2*". The code for this client is located in "*\$HOME/nyalesund/acu13m/src*" directory.
- **obsmonitor**: Displays information of the current observation, including the state of the antenna, the observed source, weather data and information on the scan. The Java code is located in "*\$HOME/nyalesund/obsmonitor/src*" directory
- **nyale1/2**: Main radiotelescope command line interface that can be used to send commands to ACS. This client developed in python uses ipython to ease the operations. The code executes "*\$HOME/nyalesund/raegeObserver/src/tel13commands.py*".

5.2. - FS (Field System)

The link between the FS and the control system of the antennas is done via the station software.

Two host with the FS installed are available: *ny13fs1* and *ny13fs2*. Both hosts are prepared to control both radio-telescopes when being used in twin mode. If the telescopes are to be used independently each host will control its antenna.

The FS computers run Debian Jessie 8.3 with a 32 bit INTEL architecture.

"/usr2/control" directory FS contains configuration files. Three files contain specific

information for Ny Alesund:

- **stcmd.ctl**: List of implemented station commands.
- **fsnet.ctl**: Contains the IP address and port number where the ACS component is listening to commands from the FS when a radiotelescope observes individually.
- **fsnet_twin.ctl**: Same as before but when both radio-telescopes work together in twin mode. Two IP addresses and two ports one per radiotelescope, should be specified.

The station code is located, as usual, in *"/usr2/st"* directory. Only the user *"prog"* with password *"perl2016"* can modify or create tools. The customized files are:

- **antcn**: Antenna Control Program that allows FS to control the antennas.
- **fsNet**: Socket client that connects with the ACS component *"FS13MCOMPY_1/2"*. Used when the radio-telescopes work individually.
- **fsNetTwin**: Same as *"fsNet"*, but connects with both ACS components. Used when the radio-telescopes work together in twin mode.

Further information about tools developed can be looked up at *"/usr2/st"* directory.

6 The command line interface. Single dish observations

6.1. - Starting ACS services and containers: the ACS Command Center

Prior to observing, ACS should be started up. A graphical tool allows to start ACS, its services and the container required by the components. From a console type:

```
acscommandcenter
```

Fig. 17 shows a snapshot of the ACS Command Center window:

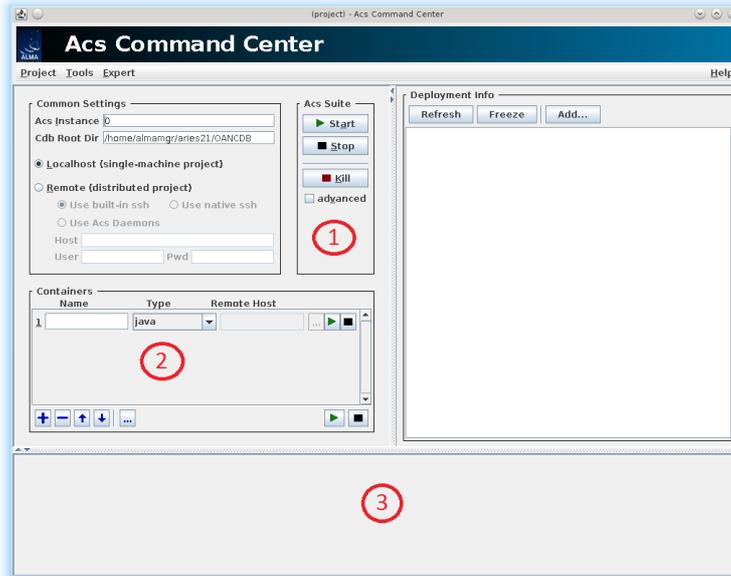


Figure 17: ACS Command Center GUI

Area 1 is used for managing the ACS. Three buttons allow to start, stop and kill the ACS and its services.

Area 2 is for managing the containers. There are two main frames: at the left the list of containers and the right a white window which will show the container and components managed by the former. In the left frame and at the right side of each container there are three buttons: edit container configuration, start container and stop container. At the bottom there are five more buttons that allow adding, removing, changing the container order and starting and stopping all containers.

Area 3 logs information from ACS and the containers and components running. This is extremely useful since it provides real time information of the observation and allows to debug the code in case of errors.

The ACS Command Center allows to load a project that contains information on where and how to run the ACS and on the containers to be started. We have set a project for Ny Alesund: go to Project -> Open... and select "*\$HOME/nyalesund/startupNyalesund*". Areas 1 and 2 (left side) will get filled with information. To start the ACS the start button should be clicked. While starting information on the containers will start being displayed on the right side of area 2, as shown Fig. 18. Area 3 will contain one tab displaying information on the ACS services being started.

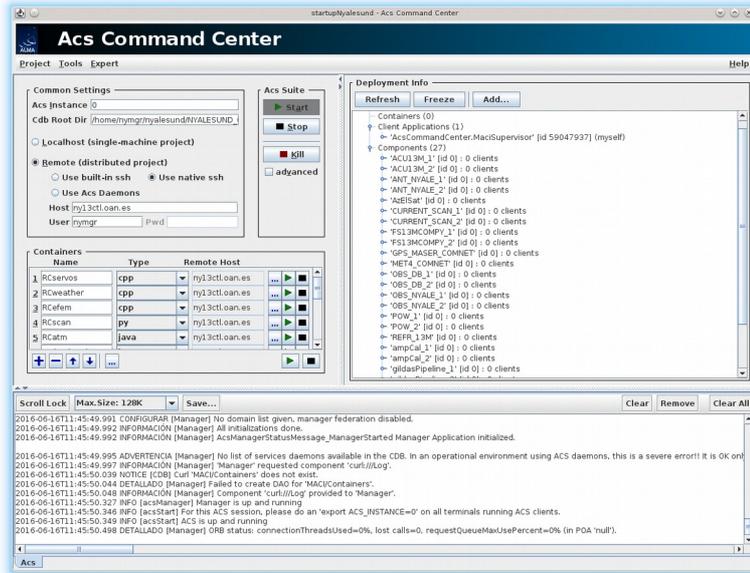


Figure 18: ACS Command Center after the start-up file was loaded and the start button clicked.

Containers are started from the start button at the bottom of the container list. Each container being started creates a tab in area 3 with logging information about that container. On the right side of area 2 information on live containers and clients will be shown. Once the process has completed the ACS Command Center should look like Fig. 19. At the right part there is the number of container running (indicated with red circle).

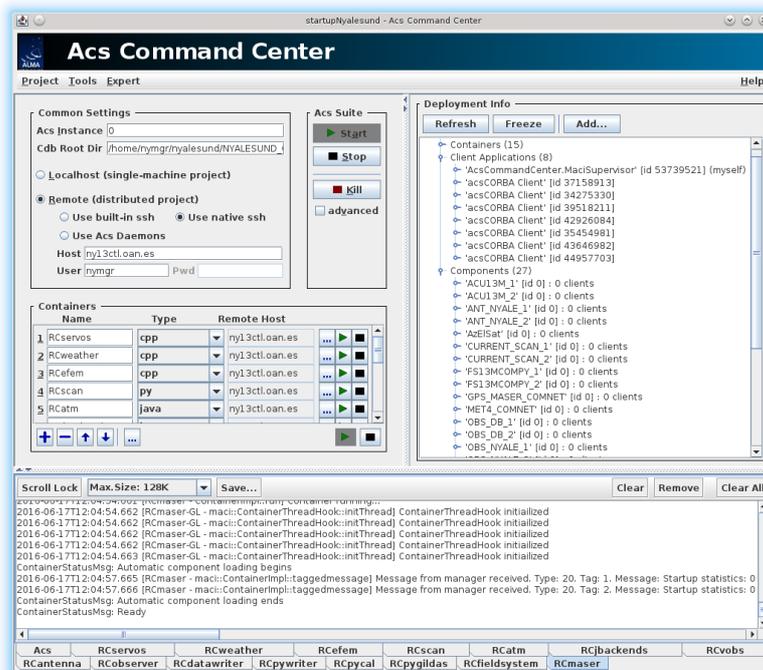


Figure 19: ACS Command Center with all containers running.

Containers manage components. Components can be started from the Object Explorer, a generic ACS client that can be launched from the ACS command center: select Tools -> Object Explorer. The Object Explorer window contains a list of all available components and instances. They are displayed in a tree view which can be collapsed or unfolded. An instance can be activated by unfolding its component and clicking on top of the name. Once clicked select “get a ‘sticky references’ for just this component” on the window that will pop up. The Object Explorer will boldface the instance name indicating it is active and it will show on the right side a list of methods which can be run just by clicking on top of them. Fig. 20 shows an example of such status. To close an instance right click on the instance and select “disconnect”

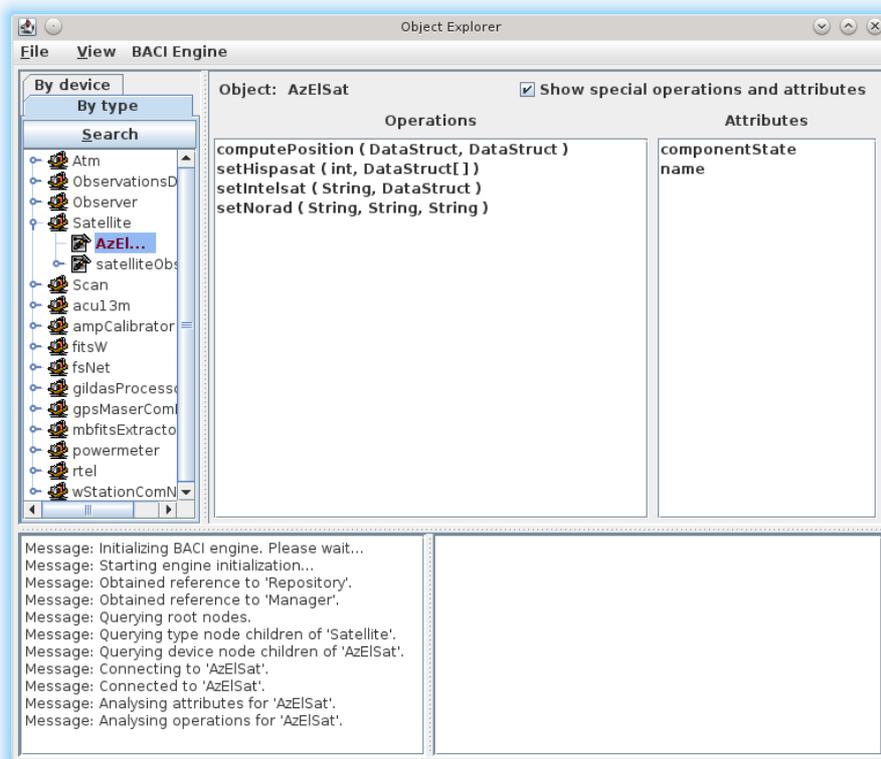


Figure 20: Testing component with Object Explorer

The Object explorer is useful for testing components, but it is not advisable to use it for standard operations. As mentioned in the previous section operations should be done from the line command clients “nyale1/2”. Clients *nyale1* and *nyale2* are launched from Linux console. “\$HOME/nyalesund/raegeObserver/src/nyale1.py” and “\$HOME/nyalesund/raegeObserver/src/nyale2.py” respectively.

6.2. - Command line interface basics

A whole report by de Vicente explains the usage of the line command interface. Here we just mention very basic commands used for single dish observations. :

- **sourcecats(catalogs="", mode = 'keep')**: Loads a catalog with source coordinates.
- **source(name="", x=(None, ""), y=(None, ""), pmx=0, pmy=0, system='EQ', epoch=2000.0, velocity=None, frame='LSR')**: Specifies the source to observed. If the source is already listed in the loaded catalog it can be selected only using its name.
- **on(intTime=30.0)**: Tracks a source for the specified time.
- **vlbi()**: Tracks the specified source until it sets.
- **cancel()**: Cancels the current scan.
- **bye()**: Releases all active components and exits the line command interface.

The ACS should never be stopped before closing the Command Line Interface of the antennas.

7 Observations with the Field System

The connection between the FS and the control system of the telescope requires the following steps:

- Start ACS and its containers as described in the previous section. The ACS should be started on host *ny13ctl*, user *nymgr*, password *Svalbard*.
- Activate components *FS13MCOMPY_1/2* from the Object Explorer.
- Execute client *fsNet* in a console window at *ny13fs1* and at *ny13fs2*, using the *oper* account to control the telescopes individually. If we want to activate the twin mode execute: *fsNetTwin* in any of the two computers.

Two basic FS commands related to the antenna may be used:

- **antenna=boot**: Boots the antenna. The antenna must be booted prior to being used. If client *nyale1/2* is running the boot can be done from the Line Command Interface and this command is not strictly necessary.
- **antenna=stop**: Stops the antenna and cancel the observation.

A Appendix

User accounts and passwords

Computer	User account	Password	Used to
ny13ctl	root	lat79long12	Super user
	nymgr	SvalbarD	Manage the ACS
	meteo	met4g	Manage the Weather Station
	maser	C0unt3r	Manage the Counter
ny13fsX	root	lat79long12	Super user
	oper	NyalesunD	Manage the FS
	prog	perl2016	Configure the FS

Databases

Computer: ny13ctl

Database	Table	User	Password	Info
mysql	mysql_tables	root	lat79long12	Super user mysql
observations	nyinfo_1	obsdbuser	0bsdbp4ss	Observations info for antenna 1
	nyinfo_2			Observations info for antenna 2
meteodb	weatherlog	meteodbuser	met4g	Weather Station info
maserdb	maserdata	maserdbuser	C0unt3r	Counter info