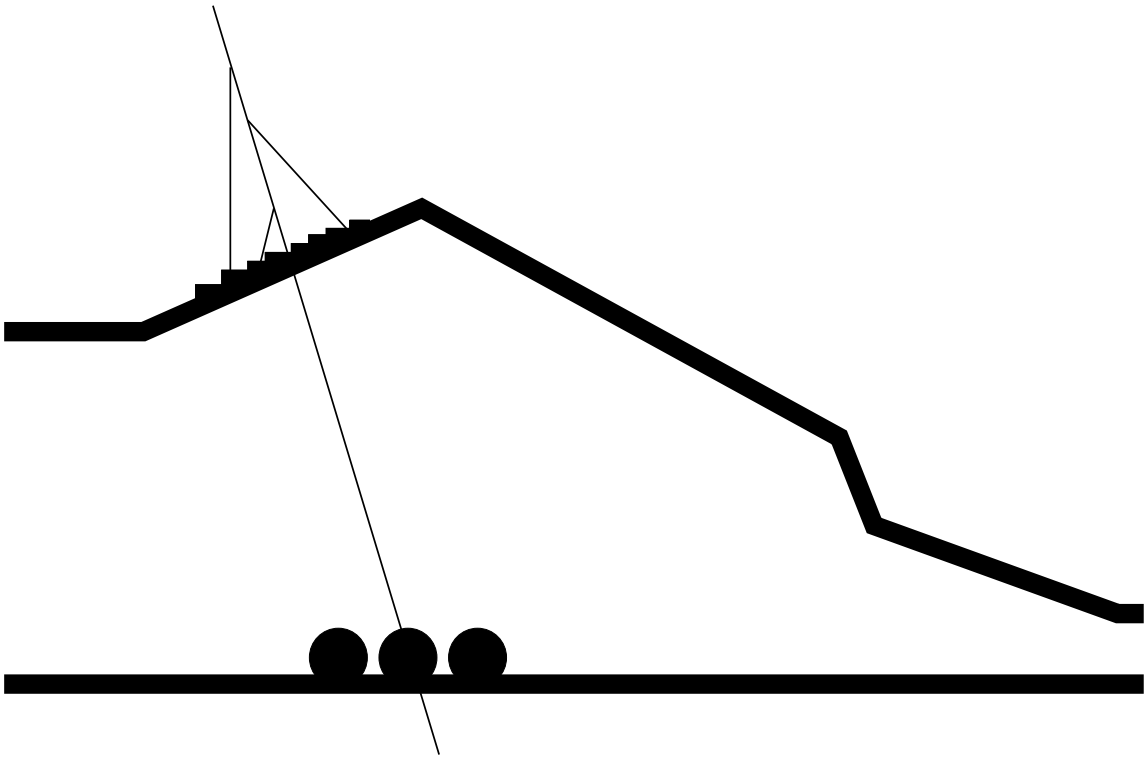
LNGS/TC-01/06
June 2006



V-DAS (A Versatile Data Acquisition Software)
The user interfaces

M. Di Paolo Emilio[a,b], S. Stalio[a]

INFN - Laboratori Nazionali del Gran Sasso

INFN - Istituto Nazionale di Fisica Nucleare
Laboratori Nazionali del Gran Sasso

LNGS/TC-01/06
June 2006

V-DAS (A Versatile Data Acquisition Software) The user interfaces

M. Di Paolo Emilio[a,b], S. Stalio[a]

[a] *INFN, Laboratori Nazionali Del Gran Sasso, Assergi (AQ) - Italy*
[b] *Dipartimento di Fisica, Universita' degli Studi dell'Aquila - Italy*

Abstract

A new data acquisition system (DAQ) named V-DAS and based on the VME bus is presented. The system has been developed for the data acquisition for the gravitational antennas belong of the ROG group. The system is supported by a text based user interface (TUI) and a graphic user interface (GUI), both allowing for an easy management of the DAQ. This document will describe in depth the two user interfaces.

1 Introduction

V-DAS is a software written using the C language for the management of VME based DAQ systems. The VME bus (Versa Module Europa) is a flexible open-ended bus system based on the Eurocard standard. It was introduced by Motorola, Phillips, Thompson, and Mostek in 1981. VME bus was intended to be a flexible environment supporting a variety of computing intensive tasks, and is now a very popular protocol in the computer industry. It is defined by the IEEE 1014-1987 standard. The system is modular and follows the Eurocard standard. VME card cages contain 21 slots, the first of which must be used as a crate manager.

The idea that led us to the realization of V-DAS (see figure 1) has been the necessity of creating, starting from the vme_universe drivers and libraries for the VME bus and the standard C libraries, a new set of functions and structures that assures the easy management of VME based DAQ systems (figure 1).
V-DAS has been originally developed for the acquisition of data generated by the gravitational antennas (Nautilus and Explorer) belonging to the ROG (WWW.lnf.infn.it/ROG) group. The system architecture relies on a VME crate managed by an Intel-based crate controller running the Linux operating system.

2 V-DAS architecture

V-DAS is composed of 5 subsystems, each having a specific function:

- VME bus interface: implements the communication with the boards mounted in the VME crate.
- Data writing: takes care of writing acquired data on structured data files.
- Configuration file interpreter: reads and parses the configuration file and sets up the DAQ.
- Error handler: manages errors that may show up during data taking (network problems, VME bus errors, disk access problems, ...)
- Network data transfer manager: takes care of transferring acquired data from the VME crate manager to an optional data storage host via an Ethernet connection.

3 The user interfaces

V-DAS has two user interfaces: a text based user interface (TUI) consisting in an ASCII configuration file and and a graphic user interface (GUI) available by means of any Web browser.

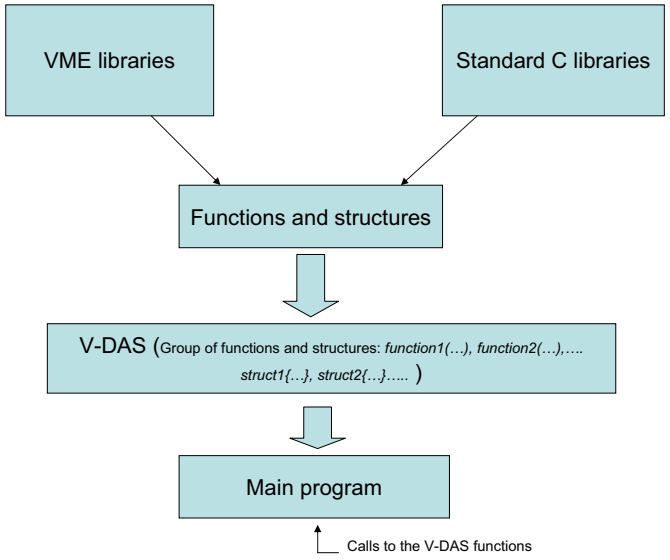


Figure 1: "Software structure"

Both interfaces permit DAQ management and customization without the need of recompiling the sources, thus granting full acquistion control also to inexperienced programmers. The configuration file is written in a high-level language (meta language) and is easily modified by the operator. V-DAS takes care of reading and parsing it and modifies the DAQ setup accordingly.
The GUI works at a higher level with respect to the ASCII configuration file and helps the operator in compiling the configuration file and in controlling the acquisition. The use of the Web interface does not require any knowledge of the configuration file sintax and avoids "grammatical" errors. It is up to the operator to choose the TUI or the GUI when modifying the DAQ setup.

3.1 The configuration file

V-DAS reads all the DAQ parameters from a configuration file (See figure 3 and 2). The configuration file is a text file organized in sections. Each section is delimited by a couple of strings; one is found at the beginning of the section, the other at its ending (See figure 4).
The configuration file sections and their delimiter strings are defined in the source code and can not be changed arbitrarily. Sections can contain three types of data: simple lists of objects, lists of objects characterized by a numerical parameter, lists of objects characterized by a numerical or textual parameter. Each section can contain a single type of data.

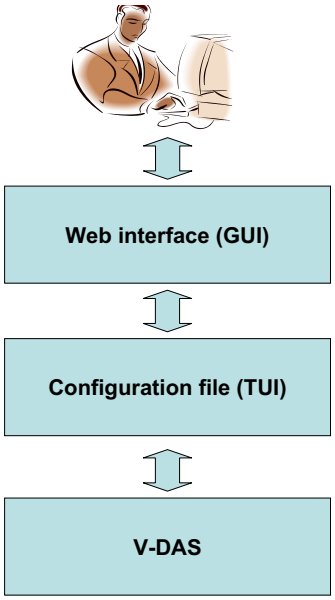


Figure 2: "V-DAS"

Comments in the configuration file begin with the "//" string and go to the end of the line (exactly as in the C++ programming language) and can be placed anywhere in the file.

In V-DAS the sequence of sections is organized in two parts: the first is declaratory, the second is executive.
In the declaratory part the elements of the acquisition are defined, ordered by level of complexity. At lower level we find the VME board registers (components), the definition of groups of registers (equipments) follow, at higher level groups of equipments (triggers) are created and filled.
In the executive part, the periodicity of each *trigger* is defined. When a *trigger* is encountered all the components belonging to the *trigger* equipments are executed.
The first section of the configuration deals with global parameters needed for the acquisition management (antenna name, type of run, data directory...).
This section is delimited by the following strings:

```
START_SECTION_PARAMETERS
...
END_SECTION_PARAMETERS
```

The contents of this section (a list of objects characterized by a numerical or textual parameter) will have to match the following syntax:

```
name1=value1
```

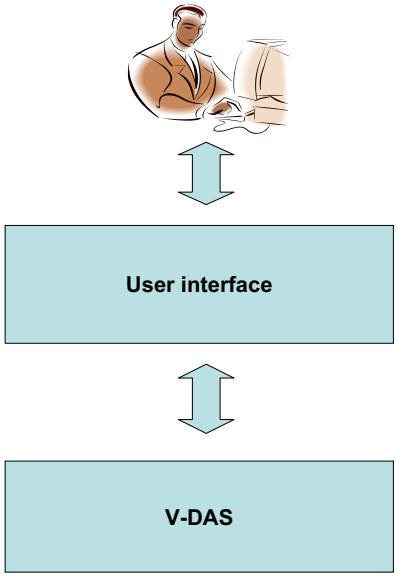


Figure 3: "The configuration file"

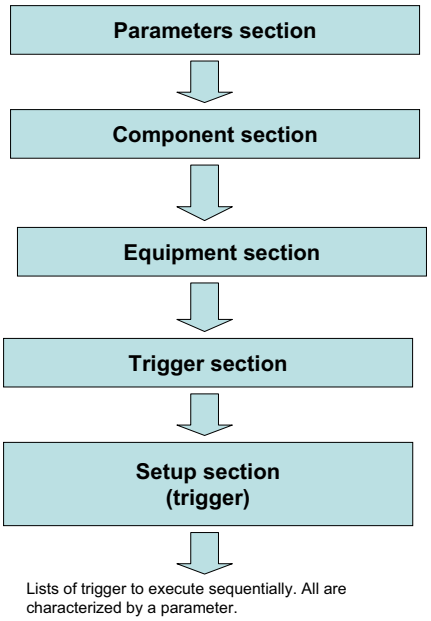


Figure 4: "Configuration file structure"

```
name2=value2
name3=value3
```

In the attached document (Appendix A) the list of parameters that need to be present in this section is found.
The VME section is delimited by the following strings:

```
START_SECTION_VME
...
END_SECTION_VME
```

This section is composed of four subsections:

3.1.1 The component section

The *components* subsection is delimited by the following strings:

```
START_COMPONENT_LIST
...
END_COMPONENT_LIST
```

In this section all the *components* used in the DAQ system will be described according to this template:

```
STARTCOMP name
module=val
register=val
dw=val
am=val
base=val
offset=val
size=val
bus=val
action=val
value=val
value_expected=val
END_COMP
```

The string *name* is chosen arbitrarily and represents the *component* name. Following is the meaning of the other variables:

- `module=val`

  the user will define the name of the VME board to which the register belongs.

- `register=val`

  This is an arbitrary name for the register.

- `dw=val`

  Width of the data word (8, 16, 32 or 64 bits). Accepted values are: VME_D8, VME_D16, VME_D32, VME_D64.

- `am=val`

  Access type to the VME bus. The address modifier for this register will have to be inserted here.

- `base=val`

  The VME board base address (hexadecimal).

- `offset=val`

  The offset of the register address with respect to the base address (hexadecimal).

- `size=val`

  Size in words of data stored in the register (usually 1 except for DMA registers).

- `bus=val`

  The incremental number of the VME crate to which the crate controller belongs.

- `value=val`

  The value to be written in the register (only used in write operations).

- `value_expected=val`

  The value that we expect to read from a register (only used in certain read operations, described later).

- `action=val`

  The type of operation to be performed on the register (read, write, ...).

Legal values for the **action** variable are:

- *read*: read data register (or memory area).
- *write*: write data (found on the **value** field) on the selected register.
- *read_verify*: read data register and compare the result with the value of the **value_expected** parameter. An error flag will be raised if the two numbers are different.
- *read_loop*: repeat readout of the data register until the result is equal to the value of the **value_expected** parameter.

3.1.2 The equipment section

The next subsection is the *equipment* subsection. Its delimiters are the following strings:

```
START_EQUIPMENT_LIST
...
END_EQUIPMENT_LIST
```

Here all the *equipments* that are used during DAQ must be defined. Each *equipment* must be declared this way:

```
STARTEQP equipment1
component1
component2
...
ENDEQP
```

Executing *equipment* **equipment1** means sequentially executing the operations defined by the **action** field of each component belonging to the *equipment*.

3.1.3 The trigger section

The *trigger* subsection is delimited by the following strings:

```
START_TRIGGER_LIST
...
END_TRIGGER_LIST
```

*Trigger*s are defined like this:

```
STARTTRIG trigger1
equipment1
equipment2
...
```

Executing *trigger* **trigger1** means sequentially executing the equipments listed inside the trigger itself.

3.1.4 The setup section

The last subsection belonging to the VME section deals with the DAQ setup. The delimiters strings are:

```
START_ACQ_SETUP_TRIG
...
END_ACQ_SETUP_TRIG
```

This section needs to be filled with the names of the *triggers* to be executed.

```
START_ACQ_SETUP_TRIG
trigger1 valpar
trigger2 valpar
...
END_ACQ_SETUP_TRIG
```

The **valpar** is an integer that controls the execution period of each *trigger*. If **valpar** is -1 the *trigger* is executed only once at the beginning of data taking. If **valpar** is 0 the *trigger* is executed whenever possible. If **valpar** has a positive value, this value represents the period (in seconds) of the trigger execution.

3.2 The Web interface

3.2.1 Interface structure

The Web interface has been realized by means of the *HTML* and *PHP* programming languages (figure 6, 5, 7, 8 and 9)). It requires running a Web server authorized to modify the V-DAS configuration file. Its main goals are the compilation of the configuration file and the management of the DAQ. The Web interface also makes possible browsing the configuration file sections.

3.2.2 Interface operation

We will give a step by step description of the operations to be performed in order to correctly fill a configuration file using the Web interface. All the operations described below are accessed via hypertext links found on the menu area on the left side of the main *HTML* page.

First of all an existing configuration file can be loaded. In this case its contents will be shown in a read-only text area. If no configuration file is loaded a default template will be used.

In order to fill the configuration file the menu section will have to be followed sequentially. First we can insert any comment describing the acquisition setup. This is done by clicking on the "Comments" link.

Then we can follow the "Parameters" link in order to assign values to general DAQ parameters and, if needed, comments related to this area of the configuration file.

We can now insert, delete or modify components. This is done by clicking on the "Components" link.

Components need to be grouped in equipments. The "Equipments" link allows us to operate on equipments.

Control on triggers is possible under the "Triggers" link.

The "Setup" link allows us to create a sequential list of the triggers to be executed during data taking. Each trigger is characterized by the **valpar** parameter described before.

The configuration file generated by this procedure can now be saved ("Save" link) and DAQ is ready to start.

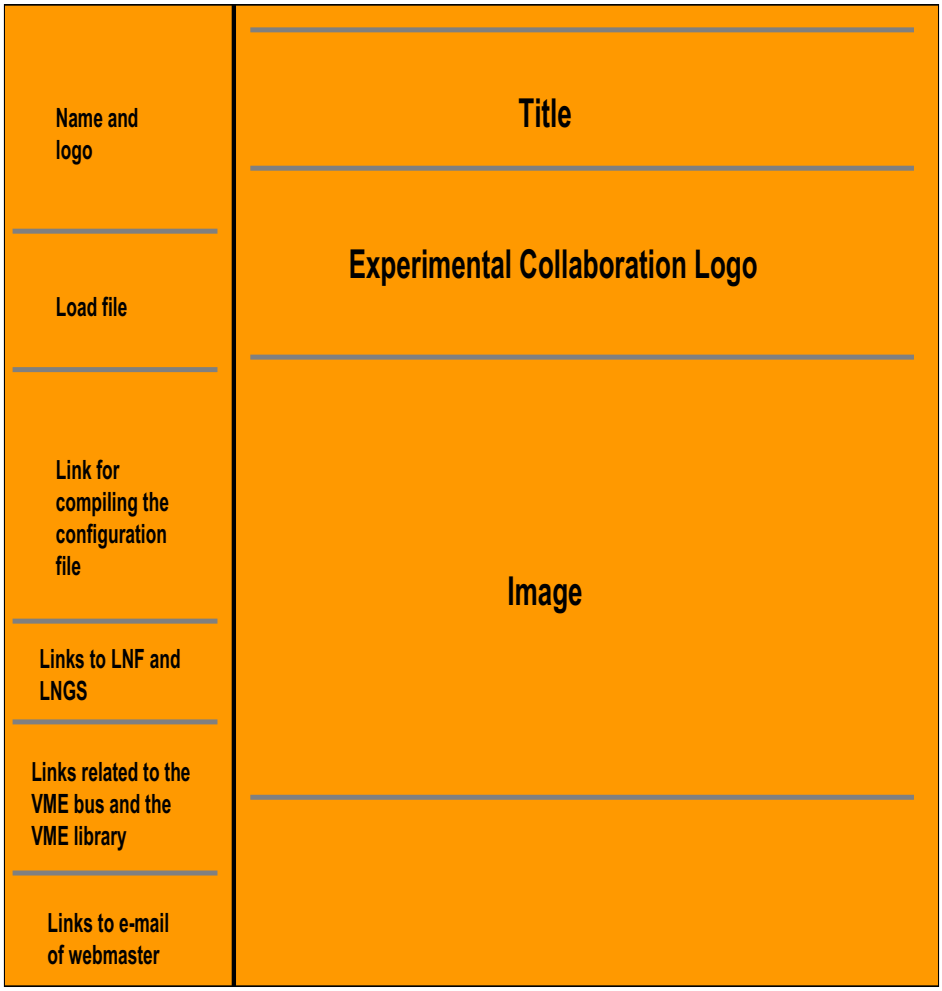


Figure 5: "Structure of the Web interface main page"

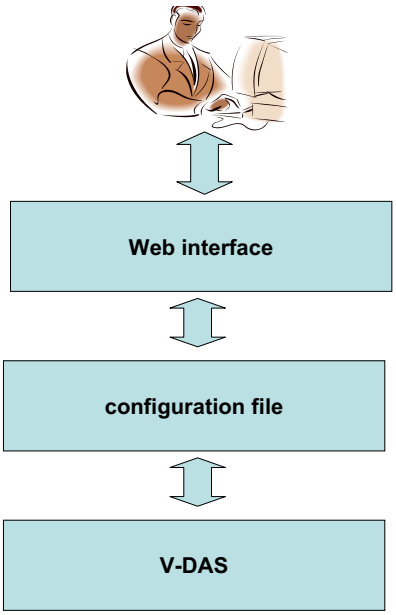


Figure 6: "The Web interface: V-DAS"

3.2.3 DAQ control via the Web interface

DAQ control (start, stop) can be performed via the web interface. The "Control" link gives access to a page where the "Start" and "Stop" buttons can be used for DAQ control and the running configuration may be browsed (see figure 10).

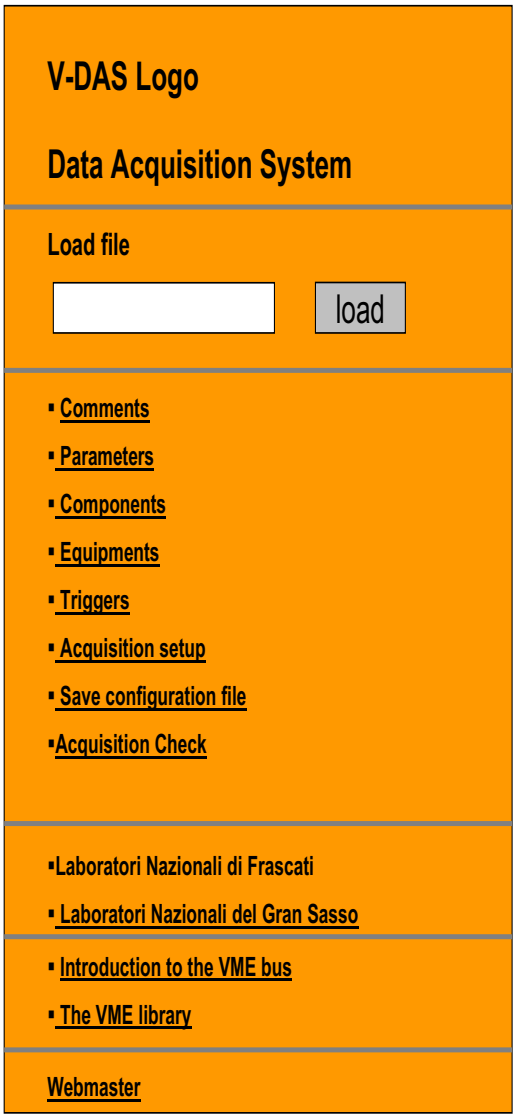


Figure 7: "Structure of the left frame of the Web Interface"

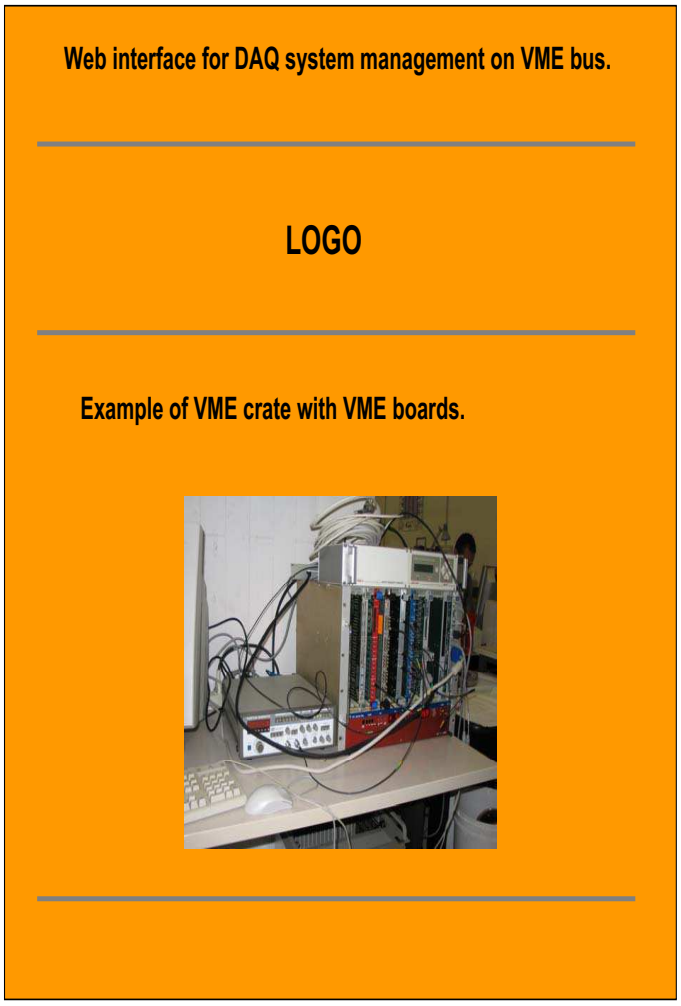


Figure 8: "Structure of the right frame of the Web Interface"

Figure 9: "First Page of the Web Interface"

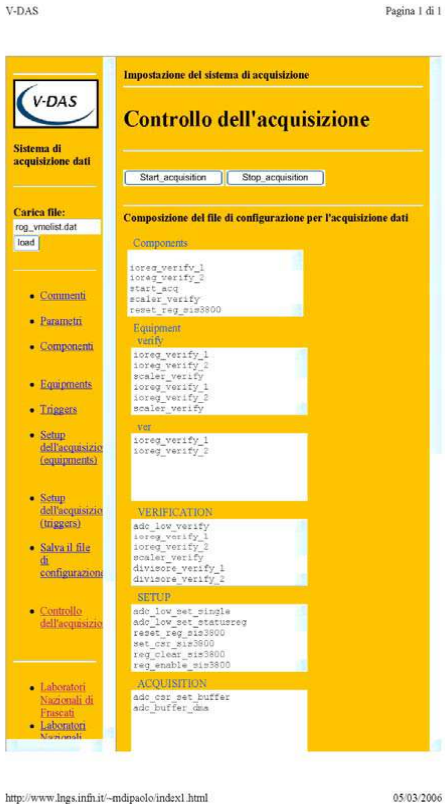


Figure 10: "Page of the Acquisition control"

Appendix A: Example of configuration file

The first part of the configuration file is composed of comments that describe the experimental setup and the DAQ system:

```
//This is a comment
//Lines cannot exceed 400 characters
//Start with module definition of the VME Components
//frequency in KHz
```

The first section we meet is the parameter section:

```
START_SECTION_PARAMETERS
//host name and communication ports
VME_HOST=rogdaq01
DATA_PORT=10000
DATA_HOST=rogdaq
CONTROL_PORT_1=10001
CONTROL_PORT_2=10002
PID_FILE=/tmp/rogdaq.pid
SELECT_TIMEOUT=2

//data file parameters
DATA_DIR=/tmp/
MAX_FILE_SIZE=100000000
//DATA_FILE=rogs
ADC_POLL_INTERVAL=10000
ADC_BUFFER_WORDS=65536
CHANNEL_FAST_ADC_1=8
CHANNEL_FAST_ADC_2=1
CHANNEL_SLOW_ADC_1=1
CHANNEL_SLOW_ADC_2=21
ACQ_MODE=calibration
ANTENNA=sfera
MAN_END_RUN=0
comment=first run

//daq parameters
DATA_BUFFERS=80

//running modes
DEBUG=0
DATA_CHECK=0
```

```
//definitions for interface.h

END_SECTION_PARAMETERS
```

The following section is the VME section. It is composed of 5 subsections. The first subsection is the component subsection:

```
START_SECTION_VME

START_COMPONENT_LIST

startcomp ioreg_verify_1
module=V977
register=register_dummy
dw=VME_D16
am=VME_A24UD
base=0xd00000
offset=0x2a
size=1
bus=1
action=write
value=0xdead
expected_value=0
endcomp

startcomp ioreg_verify_2
module=V977
register=register_dummy
dw=VME_D16
am=VME_A24UD

base=0xd00000
offset=0x2a
size=1
bus=1
action=read_verify
value=0
expected_value=0xdead
endcomp
startcomp start_acq
module=V977
register=singlehit
dw=VME_D16
am=VME_A24UD
```

```
base=0xd00000
offset=0x16
size=1
bus=1
action=read_loop
value=
expected_value=0x1
endcomp
startcomp scaler_verify
module=SIS3800
register=MIR
dw=VME_D32
am=VME_A24UD
base=0xe00000
offset=0x04
size=1
bus=1
action=read_verify
value=0
expected_value=0x38001000
endcomp
startcomp reset_reg_sis3800
module=sis3800
register=reset_reg
dw=VME_D32
am=VME_A24UD
base=0xe00000
offset=0x60
size=1
bus=1
action=write
value=0x1
expected_value=0x0
endcomp
startcomp set_csr_sis3800
module=sis3800
register=csr
dw=VME_D32
am=VME_A24UD
base=0xe00000
offset=0x0
size=1
bus=1
```

```
action=write
value=0xC
endcomp
startcomp read_counter_scaler
module=sis3800
register=dati
dw=VME_D32
am=VME_A24UD
base=0xe00000
offset=0x280
size=1
bus=1
action=read_loop
expected_value=0x99999
value=0x0
endcomp
startcomp reg_clear_sis3800
module=sis3800
register=clear
dw=VME_D32
am=VME_A24UD
base=0xe00000
offset=0x20
size=1
bus=1
action=write
value=0x1
endcomp

startcomp reg_enable_sis3800
module=sis3800
register=gc_enable
dw=VME_D32
am=VME_A24UD
base=0xe00000
offset=0x28
size=1
bus=1
action=write
value=0x1
endcomp
startcomp reg_control_sis3800
module=sis3800
```

```
register=csr
dw=VME_D32
am=VME_A24UD
base=0xe00000
offset=0x0
size=1
bus=1
action=read_verify
value=0x0
expected_value=0x800c
endcomp
startcomp divisore_verify_1
module=divisore
register=csr
dw=VME_D32
am=VME_A24UD
base=0x400000
offset=0x0
size=1
bus=1
action=write
value=0xba
expected_value=0
endcomp
startcomp divisore_verify_2
module=divisore
register=csr
dw=VME_D32
am=VME_A24UD
base=0x400000
offset=0x0
size=1
bus=1
action=read_verify
value=0
expected_value=0xba
endcomp

startcomp divisore_set_frequency_acq
module=divisore
register=registro_one
dw=VME_D32
am=VME_A24UD
```

```
base=0x400000
offset=0x0
size=1
bus=1

action=write
value=0x3e7
//value=0xd04
endcomp
startcomp divisore_set_frequency_acq_verify
module=divisore
register=registro_one
dw=VME_D32
am=VME_A24UD
base=0x400000
offset=0x0
size=1
bus=1
action=read_verify
value=0x0
expected_value=0x3e7
//expected_value=0xd04
endcomp
startcomp adc_reset
module=adc
register=csr0
dw=VME_D16
am=VME_A16U
base=0x8000
offset=0x4
size=1
bus=1
action=write
value=0x20
endcomp
startcomp adc_halt
module=adc
register=csr0
dw=VME_D16
am=VME_A16U
base=0x8000
offset=0x4
size=1
```

```
bus=1
action=write
value=0x40
endcomp
startcomp adc_waitready
module=adc
register=csr0
dw=VME_D16
am=VME_A16U
base=0x8000
offset=0x4
size=1
bus=1
action=read_loop
value=0x0
expected_value=0xff00
endcomp
startcomp adc_bid
module=adc
register=bid
dw=VME_D16
am=VME_A16U
base=0x8000
offset=0x0
size=1
bus=1
action=read_verify


value=0x0
expected_value=0x3d00
endcomp
startcomp adc_setramsize
module=adc
register=adc_dtobc
dw=VME_D16
am=VME_A16U
base=0x8000
offset=0x10
size=1
bus=1
action=write
value=0x11
```

```
//value=0xe
endcomp
startcomp adc_regmask
module=adc
register=adc_chen
dw=VME_D16
am=VME_A16U
base=0x8000
offset=0x18
size=1
bus=1
action=write
value=0xff
endcomp
startcomp adc_setmode
module=adc
register=adc_dbar
dw=VME_D16
am=VME_A16U
base=0x8000
offset=0x12
size=1
bus=1
action=write
value=0x06
endcomp

startcomp adc_triggersource
module=adc
register=adc_trig
dw=VME_D16
am=VME_A16U
base=0x8000
offset=0x0a
size=1
bus=1
action=write
value=0x4
endcomp
startcomp adc_error
module=adc
register=adc_error
dw=VME_D16
```

```
am=VME_A16U
base=0x8000
offset=0x22
size=1
bus=1
action=read_verify
value=0x0
expected_value=0x0
endcomp
startcomp adc_samplemode
module=adc
register=adc_csr0
dw=VME_D16
am=VME_A16U
base=0x8000
offset=0x4
size=1
bus=1
action=write
value=0x12
endcomp
startcomp adc_buffer_dma
module=adc
register=reg_dma
dw=VME_D16
am=VME_A24SB
base=0x0
offset=0x0
size=0x20000
//size=0x80000
bus=1
action=read
value=0x3e7
endcomp
startcomp adc_csr_set_buffer
module=adc_csr1
register=adc
dw=VME_D16
am=VME_A16U
base=0x8000
offset=0x06
size=1
bus=1
```

```
action=read
value=0x0
endcomp

startcomp test
module=adc_csr1
register=adc
dw=VME_D16
am=VME_A16U
base=0x8000
offset=0x06
size=1
bus=1
action=read_loop
value=0x0
expected_value=0xffff
endcomp
END_COMPONENT_LIST
```

Following is the equipment subsection:

```
START_EQUIPMENT_LIST
starteqp verify
ioreg_verify_1
ioreg_verify_2
scaler_verify
ioreg_verify_1
ioreg_verify_2
scaler_verify
scaler_verify
scaler_verify
scaler_verify
scaler_verify
scaler_verify
scaler_verify
endeqp

starteqp ver
ioreg_verify_1
ioreg_verify_2
endeqp

starteqp VERIFICATION
```

```
ioreg_verify_1
ioreg_verify_2
scaler_verify
divisore_verify_1
divisore_verify_2
adc_halt
adc_reset
adc_waitready
adc_bid
endeqp

starteqp SETUP
reset_reg_sis3800
set_csr_sis3800
reg_clear_sis3800
reg_enable_sis3800
reg_control_sis3800
divisore_set_frequency_acq
divisore_set_frequency_acq_verify
adc_setramsize
adc_regmask
adc_setmode
adc_triggersource
adc_error
adc_samplemode
start_acq

endeqp

starteqp ACQUISITION

adc_csr_set_buffer
adc_buffer_dma
endeqp
END_EQUIPMENT_LIST
```

This the trigger subsection:

```
START_TRIGGER_LIST
starttrig uno
verification
setup
endtrig
starttrig due
```

```
acquisition
endtrig
starttrig tre
verify
ver

endtrig

END_TRIGGER_LIST
```

The last subsection is the setup subsection:

```
START_ACQ_SETUP_TRIG

uno -1
due 0
tre 10
END_ACQ_SETUP_TRIG
END_SECTION_VME
```

References

[1] *Tesi di Laurea: ROG. Un esperimento per la Rivelzione delle Onde Gravitazionali. Il nuovo sistema di acquisizione*, M. Di Paolo Emilio A.A. 2004/2005.

[2] *Description and operation of the daga2 HF Acquisition System for gravitazional wave detectors*, S. D'Antonio, LNF-01/006 (IR) pubblicazioni LNF. Meth. **A345** (1994) 554.