EDOC133

EUROGAM PROJECT

Database for the VXI cards testing

October 1991
Last updated Thu 11 February 1992

M.M.ALEONARD, B.HUMBERT, C.DIARRA, P. KADIONIK, C.RING, G.ZEHNACKER.

CENBG Bordeaux
SATD
Centre de Recherche Nucleaire de Strasbourg
CNRS-IN2P3 France

Database for the VXI cards testing

Contact: Christophe DIARRA (diarra@borsu6.in2p3.fr)

October 1991; last updated Wed 12 Feb 1992

1 Introduction

To access a register in the VXI cards, a command line has to be generated. This may be done from a keyboard or sent by the Graphical User Interface (GUI). The first field of the command line is an executable file name (ex: read_database, write_vxi, load_vxi, ...). For each specific operation, an executable file is provided:

- read_vxi, to read a register
- write_vxi, to write a value into a register
- write_database, to save a value into the database
- ...

This programs always make an access to the database files to get some informations that will be used to make a call to the register server or to modify other files of the database.

```
+--------------------+      +--------------+      +-----------------+
|                    |      |              |      |                 |
| command line from a|      |              |      |                 |
| Keyboard or a GUI  |<====>|   Database   |<====>| Register Server |
|                    |      |              |      |                 |
+--------------------+      +--------------+      +-----------------+
```

Supposed you need the entry of the register 1 (this number must be converted into the effective register number) into to ROCI 1 of the germanium detector 1. You will have two possiblities:

- **with the GUI:** You click on the register one (ge card synoptique). The GUI will generates the following command line: **read_database germanium_roci_1 1 r1**
  The program read_database is executed and sends a reply to the GUI (see EDOC126: "Protocoles between electronic card and software tcard" for further informations).

- **from a keyboard:** you must type **read_database ge_roci_1 1 r1 <ENTER>**

2 Database files

There are two types of files: those common to the whole environment (Config file) and those specific to each card (Setup files).

2.1 The file "VXIConfig.target"

This file must be edited by hand and indicates the position of each crate.

Example: a configuration with 3 crates:

% cat VXIConfig.target

```
POSITION            NAME
%                   -
1                   peony
2                   daisy
3                   poppy
END                 END
```

In this configuration, the first crate is peony (POSITION=1). This crate is the one including the Master Trigger or the ROCO 1. The second crate is daisy and the last crate is poppy.

You don't need this file if you want to edit by hand the **VXIConfig** and **Array_Setup** files. But if you want to generate them with the command **update_vxi_database** (see the section **The different commands**) the first thing to do

2.2 The VXIConfig files

They can be built using the structure resulting of the VXI "Autoconfiguration" or edited by hand.
Format:

```
DET_NUM   CRATE         SLOT CHAN     SER_NUMB
%         -             -    -        -
<...................DATA...................>
END       END           END  END      END
```

Example of file VXIConfig.G

DET_NUM	CRATE	SLOT	CHAN	SER_NUMB
%	-	-	-	-
1	peony	4	1	0000
2	peony	4	2	0000
3	peony	4	3	0000
4	peony	4	4	0000
5	peony	4	5	0000
6	peony	4	6	0000
7	lily	7	1	0001
8	lily	7	2	0001
9	lily	7	3	0001
10	lily	7	4	0001
11	lily	7	5	0001
12	lily	7	6	0001
END	END	END	END	END

In this file, we have:

- one header: *"DETECTOR CRATE_NAME SLOT CHAN SER_NUMB"* (5 fields).
- one comment line begining by % (5 fields).
- the entry for each detector (5 fields)..
- the last line with all the fields set to **"END"** (5 fields).

This file can be updated every time you make the autoconfiguration by typing the command **update_vxi_database**.

But each time you generate the **VXIConfig** files, check them to be sure that never two cards have the same serial numbers in the same **VXIConfig** file. If you find the same serial number for all the cards, modify it in order to assign a diffe a card in the **VXIConfif** file but allway assign the same number to this card in case you have to move this cards to an other slot or an other crate.

2.3 The Array_Setup files

They can be built using the structure resulting of the VXI "Autoconfiguration" or edited by hand.

Format:

```
DET_NUM   DET_REF        PLUG
%         -              -
<............DATA.........>
END       END            END
```

This file is edited by hand before the setup and indicates the channels used on each card (PLUG/NONE). For example the Ge detector 7 must be on the first channel of the second

Ge card. If there is no detector on this channel or if there is a problem on this channel you will have a line like this: *"7 GUOCxxx NONE"* in the file Array_Setup.G. If the channel is unused, the detector reference may be any string or character. So, the entry for the detector 7 could be *7 _ NONE*. But you must always have at least one character in the field DET_REF.

```
DETECTOR    DET_REF     PLUG
%           -           -
1           GUOCxxx     PLUG
2           GUOCxxx     PLUG
3           GUOCxxx     PLUG
4           GUOCxxx     NONE
5           GUOCxxx     NONE
6           GUOCxxx     NONE
7           GUOCxxx     NONE
8           GUOCxxx     NONE
9           GUOCxxx     NONE
10          GUOCxxx     NONE
11          GUOCxxx     NONE
12          GUOCxxx     NONE
END         END         END
```

Note: in this file we could have replaced *"GUOCxxx"* per *"_"* from the detector 4 to 12 because these channels are unused.

This file is overwritten when you use the command **update_vxi_database**. To avoid this, use the command with the option [**NOTARRAY**] : **update_vxi_database NOTARRAY <ENTER>**.

Supposed you have edited by hand the file **Array_Setup.G**, and have updated in the field **DET_REF** the effective references for each connected detector. Executing the command **update_vxi_database** will overwrite the file **Array_Setup.G** and all the changes done by hand will be lost: the field **DET_REF** will be set to **"GERMANIUM"** for all the detectors and the field **PLUG** to **"PLUG"** for all the detectors.

2.4 Reg_Setup files

These files contain for each register the informations needed by tcard and the setup grograms.

Note that the name of the setup file integrates both the card ID (serial number) and a character for card selection (G, S, T, ...).

Examples: **Reg_Setup_0000.G**

1. Reg_Setup_0000.G

REGNAME	REGNUM	MIN	MAX	STEP	SETUP	UNIT	MODE
%	-	-	-	-	-	-	-
channel_0	-	-	-	-	-	-	-
%	-	-	-	-	-	-	-
CFDWidth	1	10	255	1	255	ns	W
FTSample	2	0	2000	8	340	ns	W
ValSample	3	0	10000	40	4750	ns	W
StartRdOut	4	0	10000	40	10000	ns	W
TestGen	5	0	65535	1	24941	keV	W
emptyFIFO	19	0	1	1	0	dec	RW
TstWFifo	21	0	65535	1	4750	dec	W
TstRFifo	22	0	65535	1	0	dec	R
%	-	-	-	-	-	-	-
channel_1	-	-	-	-	-	-	-
%	-	-	-	-	-	-	-
CFDThresh	23	0	5000	20	250	keV	W
CFDDelay	24	0	255	1	129	ns	W
PZAdj	25	0	255	1	80	dec	W
PDSGate	26	0	10000	40	4550	ns	W
RO20MTh	27	0	65535	1	0	dec	RW
RO4MTh	28	0	65535	1	0	dec	RW
ROTiTh	29	0	65535	1	0	dec	RW
ROCOTh	30	0	65535	1	0	dec	RW
ROAdd20MGr	31	1	70	1	1	dec	W_
ROAdd20MIt	32	0	7	1	1	dec	W
ROAdd4MGr	33	1	70	1	1	dec	W_
ROAdd4MIt	34	0	7	1	0	dec	W
ROAddTiGr	35	1	70	1	1	dec	W_
ROAddTiIt	36	0	7	1	3	dec	W
ROAddCOGr	37	1	70	1	1	dec	W_
ROAddCOIt	38	0	7	1	7	dec	W
ROTstR20M	39	0	65535	1	0	dec	R
ROTstR4M	40	0	65535	1	0	dec	R
ROTstRTi	41	0	65535	1	0	dec	R
ROTstRCO	42	0	65535	1	0	dec	R
ROSlSReg	51	0	65535	1	0	dec	R
ROTstReg	52	0	$ffffffff	1	6553	dec	W
%	-	-	-	-	-	-	-
channel_2	-	-	-	-	-	-	-
%	-	-	-	-	-	-	-
:							
:							
:							

```
%             -       -    -         -    -       -    -
channel_6     -       -    -         -    -       -    -
%             -       -    -         -    -       -    -
CFDThresh     23      0    5000      20   250     keV  W
CFDDelay      24      0    255       1    129     ns   W
PZAdj         25      0    255       1    80      dec  W
PDSGate       26      0    10000     40   4550    ns   W
RO20MTh       27      0    65535     1    0       dec  RW
RO4MTh        28      0    65535     1    0       dec  RW
ROTiTh        29      0    65535     1    0       dec  RW
ROCOTh        30      0    65535     1    0       dec  RW
ROAdd20MGr    31      1    70        1    1       dec  W_
ROAdd20MIt    32      0    7         1    1       dec  W
ROAdd4MGr     33      1    70        1    1       dec  W_
ROAdd4MIt     34      0    7         1    0       dec  W
ROAddTiGr     35      1    70        1    1       dec  W_
ROAddTiIt     36      0    7         1    3       dec  W
ROAddCOGr     37      1    70        1    1       dec  W_
ROAddCOIt     38      0    7         1    7       dec  W
ROTstR20M     39      0    65535     1    0       dec  R
ROTstR4M      40      0    65535     1    0       dec  R
ROTstRTi      41      0    65535     1    0       dec  R
ROTstRCO      42      0    65535     1    0       dec  R
ROS1SReg      51      0    65535     1    0       dec  R
ROTstReg      52      0 $ffffffff 1   6553    dec  W
END           END     END  END       END  END     END  END
```

Note: a register number is given by the mapping of the card.
The range (MIN, MAX) are fixed by the hardware and the step (also fixed by the hardware) is used as a conversion factor for writing and reading operations.
The step of the register 26 (PDSGate) is 40 (ns). This means that incrementing this register by 1 increments the PDSGate by 40 ns.
That is, all the values to be written (values given with a unit: ns, keV, ...) will be devided by the step before writing them. For a read operation, the value from the register is muliplied by the step to give a value with a unit (ns, keV, ...).

The value in the field **SETUP** is the value used to set the register (see the command **load_vxi**). This value is modified by the command **write_database** each time you click on the button **save** using the GUI. The value stored is the one selected by the slider. You can also restore the saved value by clicking on the button **restore**: the slider will move and indicates the value extracted from the database.

The values with **"$"** (ex: $ffffffff) are in hexadecimal.

The mode "W_" means that the register is write−able but the setup value depends to the location of the card in the crate: the value to be written is determined during the setup.
The separator "channel_", must be the same in all the setup files. But you can change it and

the change must be reported in the file "constantes.h". The variable CHANNEL_DELIMIT must be modified (default value: # define CHANNEL_DELIMIT "channel_").

Note: for the Ge and BGO cards separator is channel_< channel number on the card>. "channel_0" means that the data following are common to the card and channel_<i> means that the followind data are specific to the channel i of the card.

For the Master Trigger the separator will be **"groupe_"**.

2.5 The Button_Setup files

These files contain the entry for each switch. The field SETUP indicates the state (a string) of the switch at the setup, and the fields **STATE1** and **STATE2** the value (0 or 1) to be written.

Example: **Button_Setup_0000.G**

```
REGNAME         REGNUM STATE1      STATE2       SETUP     MODE
%                 -       -          -            -         -
channel_0         -       -          -            -         -
%                 -       -          -            -         -
CtlFEna         16     enable=1    disable=0    disable  RW
Endmodule       17     yes=1       no=0         no       RW
PileUpRej       18     mark=1      rejct=0      mark     RW
TstGenAtt       20     atten=1     _=0         atten    RW


%                 -      -           -            -        -
channel_1         -      -           -            -        -
%                 -      -           -            -        -
ROS1S20MEna     43      en=1        dis=0        dis      RW
ROS1S4MEna      44      en=1        dis=0        dis      RW
ROS1STiEna      45      en=1        dis=0        dis      RW
ROS1SCoEna      46      en=1        dis=0        dis      RW
RO20MEna        47      enable=1    disable=0    enable   RW
RO4MEna         48      enable=1    disable=0    enable   RW
ROTiEna         49      enable=1    disable=0    enable   RW
ROCOEna         50      enable=1    disable=0    enable   RW
CCRAmpext       53      int=0       ext=1        int      RW
CCRVetLem       54      lbus=0      lemo=1       lbus     RW
CCRTFAGai       55      high=0      low=1        high     RW
CCRBDCEna       56      on=1        off=0        off      RW
CCRTstEna       57      on=1        off=0        off      RW
CCRChDis        58      enable=1    disable=0    disable RW
%                 -      -           -            -        -
channel_2         -      -           -            -        -
:
:
```

%	-	-	-	-	-
channel_6	-	-	-	-	-
%	-	-	-	-	-
ROS1S20MEna	43	en=1	dis=0	dis	RW
ROS1S4MEna	44	en=1	dis=0	dis	RW
ROS1STiEna	45	en=1	dis=0	dis	RW
ROS1SCoEna	46	en=1	dis=0	dis	RW
RO20MEna	47	enable=1	disable=0	enable	RW
RO4MEna	48	enable=1	disable=0	enable	RW
ROTiEna	49	enable=1	disable=0	enable	RW
ROCOEna	50	enable=1	disable=0	enable	RW
CCRAmpext	53	int=0	ext=1	int	RW
CCRVetLem	54	lbus=0	lemo=1	lbus	RW
CCRTFAGai	55	high=0	low=1	high	RW
CCRBDCEna	56	on=1	off=0	off	RW
CCRTstEna	57	on=1	off=0	off	RW
CCRChDis	58	enable=1	disable=0	disable	RW
END	END	END	END	END	END

The entry for the register 16 (CtlFEna) indicates that the *Controle des Fenetres* must be disabled at the setup. And this is done by writing 0 (disable=0) to one bit in the "Controle des Fenetres" register.

2.6 The Button_Status files

This files are the copy of the **Button_Setup** files at the setup. They will contains at any moment the last state of each switch: each time you change the value of a switch in a VXI card, the string (disable, enable, low, ...) used to get the value (0 or 1) is saved in the field **SETUP** of the **Button_Status** file.

For your next setup, the files used are the **Button_Setup** files. Since these files are never modified, you will lost the last changes you made on the switches. So if you to retrieve the last states of each switch for a card, just copy the file **Button_Status** to the file **Button_Setup** for this card.

Example: cp Button_Status_0000.G Button_Setup_0000.G

2.7 The Reg_Setup_default_config and Button_Setup_default_config files

This files contain the name of the files you want to use for a new card.

For example the file **Reg_Setup_default_config.G** could contain the following line : **current/Reg_Setup_0000.G**.
This means that all new Ge cards will have a **Reg_Setup** file which is just a copy of the file **Reg_Setup_0000.G**.

Note that the name of the default file must begin by **current**

3 The different commands

The GUI executes programs using the protocole defined in EDOC126. After the execution completed, a reply is sent to the GUI in a format defined in EDOC126.

Note: All the registers and ”buttons” numbers from the GUI must be converted in real numbers corresponding to the numbers fixed by the mapping structure. The same lines are sent to the GUI.

If you type the commands by hand, You can give the effective register. In this case, the register number needs no conversion and is use directely in the call to Register Server. If you give directely the real register number, you must use an argument to indicate it: this argument is the last one.
For the Ge and BGO cards, this argument is any character or number.
For the Master Trigger card, this argument is the group of the register (see theMaster Trigger mapping structure).

Example: you want to read the CFD threshold DAC of the Ge detector number 5.

On the GUI this DAC has the number 2 and on the mapping structure the number 23 (real number).

When you click on this DAC on the GUI, the following command will be generated:
read_vxi ge 5 2

The register number **2** is converted into **23** which is the CFD threshold DAC.

But if you can use the number **23**, in this case, you have to type :

read_vxi ge 5 23 -

Here the character ”−” indicates that the register number **23** will not have to converted. ”− can be replaced by any other character. But for the Master Trigger card, ”− is replaced by only a number which specifies the group the register belong to. Remember that for the Master Trigger mapping structur
In all the following examples, we considered only the case where the command line is sent by the GUI.

Each field of the command line is used to found particular information.

- *<card_name> :* is used to found the files extension : *”G”, ”S”, ”T”, ...*
- *<detector_number> :* is used to get the VXI crate name, the slot number (with the ID of the card in this slot) and the channel number where the detector is based. These

informations are in the **VXIConfig** files.

- *<rbs><number> :* this number is the number of a **r**egister, a "**b**utton" or a **s**cope on the GUI and is converted (if necessary) to the effective number in the mapping.

- *<value> :* if specified, it is devided by the step of the register to give the value which will be written (conversion operation).

Each time a register is accessed (read_vxi, write_vxi, button_vxi, ...) a variable *request* of type data_in (see the document about the register server written by Patrice Kadionik) is given as the input parameter to a RPC which makes the call to the register server. The program (read_vxi, ...) extracts from the database all the values to set the fields of the structure request.
For all the following commands, the card type and the serial number of the card are used to find the rigth file.

3.1 The command dbformat

The files which must be handle with care are:
the **VXIConfig** files
the **Array_Setup** files
the **Reg_Setup** files
the **Button_Setup** files
You can edit them if needed but before you use them with the database commands (read_database, write_database, ...) you must use **dbformat**

Usage: **dbformat [-if] files ...**
Supposed you have edited the file **Reg_Setup_0000.G**. You will have to type one of the following commands:

% dbformat -i Reg_Setup_0000.G

The [-i] option is the default option of dbformat and this option need you to confirm your request.

% dbformat -f Reg_Setup_0000.G
In this case, no confirmation is needed.

If you have edited many files, you can type **dbformat -f *** and only the files that have the suitable format will be formated the other files wil not be modified by this command. **dbformat** test first the last line of a file to know if it can format it (the last line must have all the fields set to **"END"**.

3.2 read_database

Read from the database the entry of a register.

Syntaxe:

read_database <card_name> <detector_number> r<number> [option]
read_database <card_name> <detector_number> b<number> [option]
read_database <card_name> <detector_number> s<number> [option]

symboles

r used for registers
b used for buttons (switches)
s used for scope

[**option**] may be '-' for the Ge and BGO cards or the group of the register for the Master Trigger card.

The format of the reply line is :

commande: <the command you have generated>
status: <Status>
value: <reg_numb> <reg_name> <setup value> <unit> <min> <max>

Function:

sends to the GUI the entry of a register and is generated each time you click to adjust the value of a register. This cammand is also executed to restore the last saved value of a regsiter.

Examples:

- example 1: **read_database germanium 1 r1**
  This command may be typed by hand or generated by the GUI.

  Reply:

  commande: read_database germanium 1 r1
  status: ok
  value: 5 TestGen 24941 keV 0 65535

  signification:

  5: the register number in the mapping structure. Note that this register has the number 1 in the GUI.
  TestGen: the register name. Here it is the test generator DAC.

24941: the value loaded in the register at the *setup* (see load_vxi).
keV: the unit in which the values are given (24941 is in keV).
0 65535: the range of the test generator DAC in keV.

- example 2: **read_database germanium 1 r100**

  Reply:

  commande: read_database germanium 1 r100
  status: undefined register number

  There is no register number 100 recorded in the GUI.

- example 3: **read_database germanium 1 b1**

  Reply:

  commande: read_database germanium 1 b1
  status: ok
  value: 55 CCRTFAGai low _ high low

The signe "_" (underscore) is sent to make the reply compatible with the register's one. Here the *"button"* number 1 (b1) has an effective number 55 in the mapping structure.

3.3 write_database

Will save the value (in the field SETUP) of a register in a file **Reg_Setup** using the serial number (for the file name) of the card where the detector is.

Syntaxe:

write_database <card_name> <detector_number> r<number> <value> <unit> [option]

Function:

save in the database the value you want to write into a register at the setup.
The command **write_database** is executed when you click on the button **save**. The value indicated by the slider is saved for the right register in the rigth file.

Examples:

- example 1: **write_database germanium 1 r1 1000 keV**

This command may be typed by hand or generated by the GUI.

Reply:

commande: write_database germanium 1 r1 1000 keV
status: ok

Here we have only the status: "ok" means that the value "1000" (in keV) has been written in the database.

- example 2: **write_database germanium 1-3,10,17 r1 1000 keV**
  This command may be typed by hand or generated by the GUI.

  Reply:

  commande: write_database germanium 1-3,10,17 r1 1000 keV
  status: ok

In this example, the database is updated for many channels.

3.4 write_vxi

Write a value in a register in a VXI card.

Syntaxe:

write_vxi <card_name> <detector_number> <register> <value> <unit> [option]
write_vxi <card_name> <detector_numberi> [[−,]<detector_numberj>] <register_number> <value> <unit> [option]

Function:

write into a register (into the VXI card) the converted value.

example: **write_vxi germanium 1-5,22 1 665 keV**

reply:

commande: write germanium 1-5,22 1 665 keV
status: ok

The status *"ok"* indicates that the register 1 (5 in reality) has now the value 83 (665 keV) : 83 = (665 keV)(8 keV per bit)).
The step of the Test Generator DAC is 8 (8 keV per bit).

The DAC is modified for all the germanium cards where the detectors 1-5 and 22 are connected.

3.5 read_vxi

Syntaxe:
read_vxi <card_name> <detector_number> <register> [option]
read_vxi <card_name> <detector_numberi> <detector_numberj> <register_number> [option]
read_vxi <card_name> <detector_numberi> [[−,]<detector_numberj>] <register_number> [option]

Function:

read a register (from the VXI card) and sends the content in the reply structure.
The processing of the call depend on the register type. If the register is readable *(mode = "R" or "RW")* the register is read and the value converted (multiplied by the step) is sent back to the GUI.
If the register is a DAC, the output of the DAC is measured (in volts). The value measured is converted into a value with unit (keV, ns, ...).

Example: **read_vxi germanium 22 1**

reply:

commande: read_vxi germanium 22 1
status: ok
value: 665 keV

3.6 button_vxi

Syntaxe:
button_vxi <card_name> <detector_number <button_number <status> [option]

Function:

Make a call to the register server to set a bit to 0 or 1. The reply is the same as with write_vxi.

Example: **button_vxi germanium 1 1 low**

On the GUI the *"button" 1* is used to select the TFA gain (low or high). The real register number of the TFA gain select bit is 55.
The problem is to know if "low" means setting this register to 0 or 1. The entry of this register in the file Button_Setup_xxx.G is used to determined at which value the bit must be set.

The entry of the register 55 ("button 1") is : *"CCRTFAGai 55 high=0 low=1 high RW"* and indicates that to set the TFA Gain low, t 1.

Each time you execute the **button_vxi** program, the new status of the button is stored in the **Button_Status** file.

3.7 scope_vxi

Syntaxe:

scope_vxi <card_type> <detector_number> <test_point> <scope_channel>

Function: initializes the structure request and make a call to the register server to connect the test point to a channel of the scope.
The reply remains the same as in the write_vxi operation.

Example: scope_vxi bgo 1 1 L2

reply:

commande: scope_vxi bgo 1 1 L2
status: ok
value:

This command will has as effect to connect the parameter output of the test point 1 to the logical input 2 of the scope.

3.8 scope_reset_vxi

Syntaxe:

scope_reset_vxi <detector name> <detector number>

Function: disconnects all the parameters to the scope. This is done only for the crate where **<detector number>** is found.

3.9 load_vxi

This command is use to set a card with the values from the **Reg_Setup** and the **Button_Setup** files.
If a new card is found (a card having no setup files) **load_vxi** will create setup files for this card by copying the default files (whose names are in the files Reg_Setup_default_config, ...) to files whose names include the serial number and the type of the card.
load_vxi also creates the **Button_Status** files (*see the section* **Button_Status** *files*).

load_vxi has 3 usages:

Usage 1: loading of only one card

Usage: **load_vxi <card name> <detector number>**

<detector number> is used to get a slot number (using the file VXIConfig.<?>). The card in this slot will be loaded using the setup file. All the channels of the card are loaded if possible (the file Array_Setup tells if a channel could be use or not).

Example: **load_vxi ge 1** will load the germanium card where the detector 1 is found.

Usage 2: loading of all the cards of the same type in all the crates

Usage: **load_vxi <card name>**

Example: **load_vxi ge** will load all the germanium cards in all the crates.

Usage 3: loading of all the cards in all the crates

Usage: **load_vxi "all"**

Example: **load_vxi all** will load all the cards (Ge, BGO, Trigger) in all the crates.

3.10 init_database and save_current_database

The command **save_current_database** copy all the files from the directory $DATABASE/current into the directory $DATABASE/default.

The command **init_database** restore all the files from the directory $DATABASE/default into the directory $DATABASE/current.

These command are available from a menu of the GUI.

Be very carefull when using the command **init_database**: if the directory $DATABASE/default contains corrupted files that have the same name as in $DATABASE/current, you will overwrite the files in the directory $DATABASE/current and then loose some of the files you need to work.

If possible, do these two operation by hand.

4 Directories

If your home directory is $HOME, you must set the environment variables $DATABASE, $DATABASE_VXI, $COMMON, $DBMS:

$DATABASE = $HOME/DATABASE
$COMMON = $HOME/DATABASE/SOURCE_C/common
$DBMS = $HOME/DATABASE/SOURCE_C/dbms

In the directory $DATABASE be sure that the subdirectory **current** exist and contain all the files necessary for the database.

The environment variable $DATABASE_VXI indicates where to find the files created after the VXI autoconfiguration. This directory will contain a file for each crate (example: the file VXIConfig.peony refers to the crate peony and tells which slot is occupied by which card (card type, serial number)). Use **setenv DATABASE_VXI $HOME/DATABASE_VXI** to set the environment variable $DATABASE_VXI.

In the directory $DATABASE/current you will find all the files you need to configure your experiment. Some of this files have a specific format and you must be very carefull when editing them by hand. Each time a file is edited by hand, the format of this file is altered and you have to use the command **dbformat** to update the format.
To set $DATABASE, use **setenv DATABASE $HOME/DATABASE**

$COMMON is the directory where are all the source files common to all the Makefiles. Use **setenv COMMON $DATABASE/SOURCE_C/common** to set this variable.

$DBMS is the directory where you will find the functions (the sources) to extract (or modify) data from (into) the database.
Use **setenv COMMON $DATABASE/SOURCE_C/dbms** to set this variable.

The following organisation of the files must be respected, but you can't modify it if you set correctly your environment variables.

```
$HOME --------------------> DATABASE ->---> default
                           :          :
                           :          :----> current
                        SOURCE_C
                           :
                       :-------:
            common <- :       :---> plus1
                      :       :
      read_database <- :       :----> test
      write_database <-:       :
            read_vxi <-:       :----> update_VXIdb
           write_vxi <-:       :----> save_current
           scope_vxi <-:       :---> restore_default
                 :      :       :
        scope <-:       :       :
        reset <-:       :       :
                 mt <-:       :
              setup <-:       :--> dbms --:-> h
          button_vxi <-:                   :-> dbformat
                                           :->fdbase
```

5 Warning

The commands to access to the database file (extract values or modify values) are linked to the files structures. For example when you want to extract the entry of a register for the channel 5, the **read_database** command knows how many lines to skip. All this lines before the "separtor" **channel_5** are spikped and the search of the entry of the specified register can begin.

That is, you can't remove or add any line in the files **Reg_Setup**, **Button_Setup** and **Button_Status** if the changes are not reported in the file **dbconst.h** in the directory $DBMS/h.

If you make modifications in the file $COMMON/constantes.h or the file $DBMS/h, you will have to build new executables commands: move in the directories, look for the file makefile or Makefile and type **make**.
After this is done, copy the executable file in the directory $HOME/bin.

In all the files, all the entries(lines) must have the same number of fields (or columns). If the line is a comment line, the first field must be set to **"%"** and the other fields set to **"-"**.

In the files **Reg_Setup** and **Button_Setup**, the separator **"channel_"** don't begin by **"%"** because the separator is considered as usefull information.