AB-CO-FC Note-2005-xxx TECH

Hardware configuration management for the DSC:

A functional description

Alain Gagnaire

Abstract

This is the review of the model to describe the DSC hardware configuration management and the presentation of the services provided to record it and manage it.

The purpose of this document is to provided a quite complete description of the configuration model, of the relation between elements and the services expected from the database. This should be a base to plan the integration of this model in the general hardware database and also to think of an upgrade to support Hardware configuration new features.

Version update: 8/11/2005

1. foreword:

This is the review of the model to describe the DSC hardware configuration management and the presentation of the services provided to record it and manage it.
The purpose of this document is to provide a quite complete description of the configuration model, of the relation between elements and the services expected from the database. This should be a base to plan the integration of this model in the general hardware database and also to think of an upgrade to support Hardware configuration new features.

2. Introduction to the DSC hardware configuration management:

2.1. the why of the hardware configuration management system:

The control system requires tens of front end computers each of them with a different hardware configuration which determines the start up of the system. The start up of the system based on LynxOS is made of a script file so called rc.local, this start up command sequence depends on the configuration to start the expected i/o module drivers with the good parameters (base address, interrupt vector, interrupt level, initial parameters), to start the application program with the good parameters (priority, initial parameters, control parameters).
The management of so many different file by hand is not affordable, tedious, and not reliable, therefore it was decided to use instead a relational database which provide powerful, reliable and comfortable tools to automatically store the information and extract, and let it process by specifics programs to generate the cryptic script files.
Moreover the automatic generation of the rc.local with the database introduced a standardisation in such a way it was possible to add more feature to support the logical I/O addressing schema and the run time physical I/O address resolution.

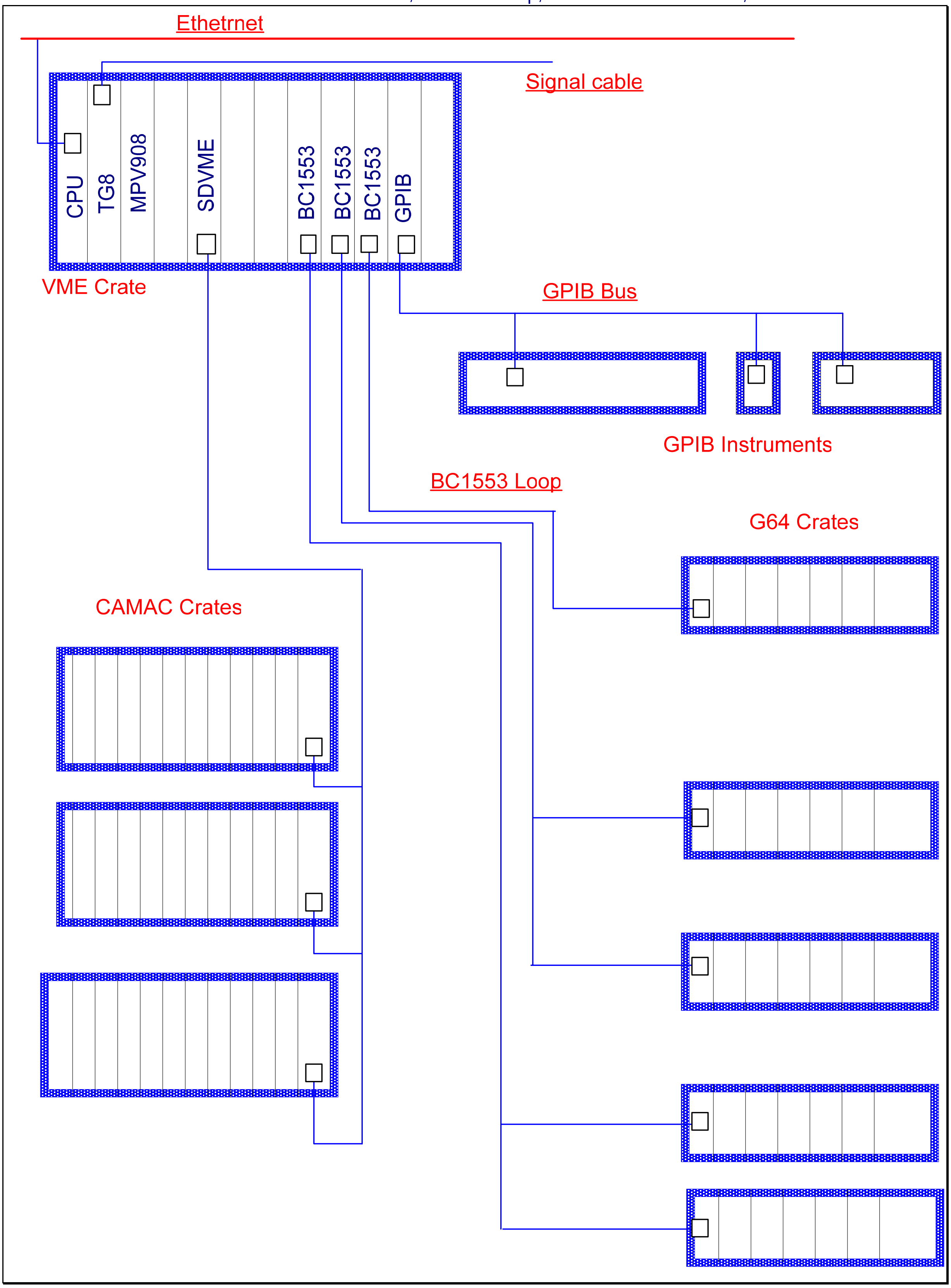
2.2. the purpose of the hardware configuration

The main purpose of the hardware configuration management is:

- to ease the exploitation by separating the responsibilities
  - Responsibilities of the hardware configuration installation: people in charge of this have to set up the module board strap fixing the hardware address of the module, to plug the configuration in the crate, to connect the good cables on the good devices and update consequently the data base without to fear at application run time any addressing trouble.
  - Responsibilities of the software development: the programs running in the DSC don't have to bother with device physical address: they just have to define the logical address of the targeted instrument, the final address resolution will be performed at the run time.
- to improve the system reliability: an a priori configuration checking is performed at the DSC starting up: this makes clear configuration errors and device fault because it comes up before any application run
- to make real time program portable according to physical device address: no need to compile or link and regenerate application program binary code when physical addresses changed
- to easily and reliably make configuration update: hardware update just request to perform the database configuration update, to deliver the start-up sequence script and to reboot the DSC, the application program will never see any change because the application just see logical addresses and the hardware resolution take place at the run time

3. A typical DSC hardware configuration:

This is a typical DSC configuration, where controls are performed via the DSC instrumentation interface placed in the VME crate: I/O and ADC modules, CAMAC Loop, MIL1553 bus controller, GPIB



4. The hardware configuration current model description:

The description is based on set of information collected which are stored in an Oracle database.

4.1. The Types and instances:

The DSC hardware configuration description is base on two kinds of elements (set of information):

- the Types: It gives the description of kind of object: the type name, and fix attributes associated to this type of object: computer Type, crate Type, module Type.
- the Instances: It gives the description of an actual object as part of the actual DSC specific configuration. The instance of an object is made of the object type reference, plus the specific attribute of the object, eg: crate VME X, Crate Camac Y. All instances for a given DSC will represent the actual hardware configuration for this DSC.

From the hardware configuration point of view, a DSC is at least the collection of the following objects associated to a ***computer type*** DSC: a ***VME crate*** and the VME ***modules*** in the crate. This minimum configuration may be extended with any element out the family of CAMAC *crates*, MIL1553 *crates*, GPIB instruments ***(***like crate***)***

4.2. the computer type:

COMTYPES = { type name, decscription}

The hardware configuration is dedicated for the computer type so called "DSC"

4.3. the crate type:

CRATETYPES = { *type name, category =* **CRATE***,*
Input bus type, Output bus type, crate slot count, crate first slot, crate strap pos, height}

This defines all name of supported crate with the characteristic of the crate where:

- Type name: name of the type of Crate
- category: = CRATE
- type serial number: serial number of the type
- intput bus type: ***non relevant for crate.***
- Output bus type: which bus type the crate slots are connected to
- Crate slot count: number of slot
- Crate first slot: slot numbering style (from 0 or 1)
- Crate Strap Pos:
- Height: size of the crate in module size unit

4.4. the module type:

This definition is made of several parts:

4.4.1. the module common part:

MODULETYPES *= { type name, category =* ***MODULE,***
type serial number,input bus type, output bus type, number of slot, subslot increment, number of channels, …

Where:

- type name: name of the module type
- category: = MODULE
- type serial number: serial number of the type

- **input bustype:** Module are providing facilities to perform I/O control against the associated instrument. This control is perform trough the electronic interface of the module with the host computer so called bus.
  A table contents all supported bus type name BUSTYPES = { type name, decscription} This provide all name of supported bus: VME, MIL1553, GPIB, CAMAC, …
- **output bus type :** which bus the module gives the access to (e.g. BC1553 give access to MIL1553 field bus)
- **Crate subslot count:** number of subslot (for mother board module only)
- **Crate first slot:** subslot numbering style (from 0 or 1)
- **Height:** size of the module in module size unit

4.4.2. the part dedicated to VME type modules and specific to VME bus modules:

…, mother board flag, SubSlot Increment, Channel Count, Width,
Driver type, Biscoto flag,
level, vector, vector increment, …

where

- **mother board flag:** when the module type supports plugged module (e.g.: VIPC610 and VIP mezzanines)
- **subslot increment :** the offset of each slot from the previous (to compute address of each slot after the mother board base address)
- **channel number:** number of channel which can be controlled
- **Width:** size of the module in slot number
- **Driver type:** is the name of the set of definition to tell how to build up the command to install the associated driver. This set is defined by the associated Driver Type.
- **Biscoto flag:** to tell if the module description is provided (this description enble the automatic generation of a direct access interface library based on a minimum driver
- ***Interrupt** = {level, vector, increment}*
  - **Level** : provide interrupt level where the ISR must run (used by the driver installation command)
  - **Vector:** provide the vector interrupt number generated by the module to raise an interrupt on the VME bus
  - **Increment:** when several module of the same type are declared, it provides the increment to get the vector number for the next module .
- ***Addr1, Address2** = {AM, BaseAddress, Range, Increment }:*
  this provide the VME base address of the module (provided the module strap are set according to this). VME module may have 2 address space therefore definition need 2 set.
  To ease the hardware setting of the VME module a default address principle was established for each module type see Technical note
  - **AM:** is the symbol of the address modifier : SH for short addressing, ST for standard addressing, EX for extended addressing, to be used for the corresponding module addressing space .
  - **BaseAddress:** the VME base address this module addressing space.
  - **Range:** the addres space size seen from the address
  - **Increment:** the increment to compute the base address of the next module

4.4.3. The part dedicated to PCI module:

…, Vendor ID, Device ID, …

Where:

- **Vendor ID:** is the PCI information found in the BARE 0 and giving the value for the Vendor ID of the Module

- Device ID: is the PCI information found in the BAR 0 and giving the ID of the Module

4.4.4. The part dedicated to CAMAC type modules and specific to the CAMAC bus module:

This information provide CAMAC addresses to perform the clear LAM on the module…,

CAMA1, CAMF1, Data1, CAMA2, CAMF2, Data2, …

4.4.5. The module addressability description:

When the biscoto flag is set in the description of a VME module, the module type record has an extension to give the description addressability of the module. This description is dedicated to the drivergen service to automatically produce library source code for module direct access interface (a set of header file , minimum direct access driver, functional access function.
Addressability is like union in c, the same are os the module space may be declred in different way, e.g.: consecutive register in the module space are seen as integer to enable named access or as a integer array to enable collective operation as reding of all of them. Two kind of definition are possible:

- ***The block definition:***

ModuleBloc= { block , Address part, offset, description}

Where:

- o Block: is the block number (arbitrary, like a name)
- o AddressPart: part number the block belong to
- o Offset: is the offset of the block from the module part base address

- ***The block element definition, or Block register definition:***

ModuleRegisterc= { block , Address part, Offset, Depth, TimeLoop, Wordsize, RW Access, Name }

- o Block: is the block number (arbitrary) the register belong to
- o Offset: is the offset of the block from the module part base address
- o Depth : number of Word in the element
- o TimeLoop: the delay loop number to temporise acces to the element
- o WordSize: size type (char, short, Long)
- o RW Access: access right (r for read, w for write, c for check, e for external)
- o Name: name of the element

4.5. The driver type : a software Type definition:

Starting device driver is not easy, even it becomes tricky specially when the parameters are multiple and provide critical hexadecimal values of the device.
In order not to have to define by hand the writing of the device driver installation command an automaitic mechanisme is included in the configuration description. It is based on the module declaration and a new kind of type definition: the driver type:

- ➢ the driver type:
this give the information to build the installation start up sequence for the driver. This type is referenced from a module type.

DRIVERTYPE *= {Driver Name, priority, master flag, subdirectory, file name, Module Type, Max modules , restart flag, Tags address, Tags next address, Tag vector, Tag level, Tag separator, Interrupt vector repeat, parameters, remarks}*

Where:

Driver name : the name to reference it from the module type declaration

Master flag: not used?

Subdirectory: path of the directory where driver object file and install program are fetched
Subdirectory: file name: the name of the installation program to invoke
Module Type : the module type of the module to drive
Max module number: max number of module the driver can control in the same major device
Restart: not used?
TAGS: the letters and characters of the installation program parameters options syntax . see the driver installation

5. The configuration declaration:

The hardware configuration of a DSC is made of the set of information for each elements present in the configuration: see the typical DSC schema, i.e.:

- ***DSC configuration*** *= { VME crate configuration [, MIL1553 crate configuration] [, GPIB crate configuration] [, CAMAC crate configuration]}*
- ***Crate configuration*** *= { module instance of a given type, …}*

The instance of a configuration is based on a set of information specific to the actual element of the actual configuration plus the information coming from the associated type definition.

5.1. Crate declaration:

This is list of the crate declaration associated to a given crate type name as define by the type definition attach to a given computer name.

Crate instance *= { computer name , crate type, Bus/Loop number, crate number, Building, room, rack, cable, label, function description}*

Where:

- **Computer name :** the name of the host computer
- **Crate type**: type of the reference crate
- **Bus/loop:** number of the crate set connected to the same bus
- **Crate number:** address of the crate in the loop
- **Geographic position information:** building, room, …

5.2. Modules declaration:

To declare a module in a crate, select with the wrting cursor the crate in the crate configuration table, and select the NextBlock of the menu and fill in a new record in the list of the crate module:

Module instance *= { slot, subslot , Module type, Lun, Tag modifier, Inhibit flag, Master type, master lun, Addr1 direct access flag, Addr2 direct access flag, remarks, more driver parameters, module instance ID}*

Where:

- **slot :** actual slot position of the module in the crate
- **subslot:** actual subslot position of the module on the mother board of the module (case of a mezzanine)
- **module type:** actual type of the plugged module
- **Lun :** logical unit number of this module (this is the lun part of the logical address of the module for the program level therefore, this number must be unique in the dsc for this module type)
- **Tag modifier:** to support old syntax driver install command for some module (e.g.: PLS_REC_FPI)
- **Inhibit:** instance generation control, according to the flag value: = I generation inhibitited, =D no driver started, N module not installed
- **Master:** type of the associated master
- **Lun of the master:** lun of the associated master module
- **Addr1 , Addr2:** flag to enable/disable direct access to the module Address space 1 / 2
- **Remarks:**
- **more driver parameters:** to insert more options in the driver install program command

5.3. Module Exceptions declaration:

This set of information is dedicated to manage the exception rules for the setting of the module address when required (see technical note). This information is linked to a given module instance as it was required at the configuration declaration..

MOD_EXCEPTIONS *= {Module ID, Driver Name, Interrupt Level, Base address1, Base address2 , Priority, Instance}*

Where:

- Drivername : substitutes for default driver name
- InterruptLevel: substitutes for default driver name
- Address1, Address2: substitutes for default values
- Instance: not used (this but could be used to have several installation of the driver in order to group modules working with the same triggers, e.g.: the mpv908 in dpsbinst)
- Priority: ?

5.4. Module Interrupts declaration : logical Events definition

This set of information is made for defining of the event logical number associated to an event source module. This information is linked to a given module instance as it was required at the configuration declaration..

MOD_INTERRUPTS *= {Num, SubAddress}*

- Number : this defines the event logical unit number associated to the module as event source.
- SubAddress: this is the name of the timing

5.5. Signals declaration:

This set of information is a reminder to connect given cable on the specified module name plug .
SIGNALS *= {connector, Signal}*

6. The VME module address management:

This the semantic is behind the all description, it takes place when hardware configuration data base is ask to produce output (file generation)

6.1. the default address principle:

A VME module is given his base address by straps on the Board, one module may have 2 address space: e.g.: the MPV908 module has a base address for the registers part of the module which is Short addressing mode area and another address space which is Standard addressing mode, the base address is selectable by straps, the memory area address is fixed by setting a specific register .
In order to ease the setting of the module address a default address principle was established : each module type is given a default base address. This setting is documented from the “control module” web page.

When several module of the same type are present in a configuration the default addressing schema give the way to set up the other addresses using the Increment field of the module type:.
eg for a configuration with n modules we will have:

Module 1 base address = default address from the module type

Module 2 base address = (module base address 1) + module type increment

…

Module n base address = (module base address 1) + (n-1)(module type increment)*

6.2. the exception: to solve conflicts at the configuration setting :

When configuration is plenty of different modules occurrence of default address setting conflict or overlap may appears. To solve this conflict it is possible at the crate configuration setting to bypass the default setting principle for some module

6.3. the dark part:

the purpose of the database is to provide on the one side services to record the DSC different Hardware configuration by means of the form application and on the other checking tools:

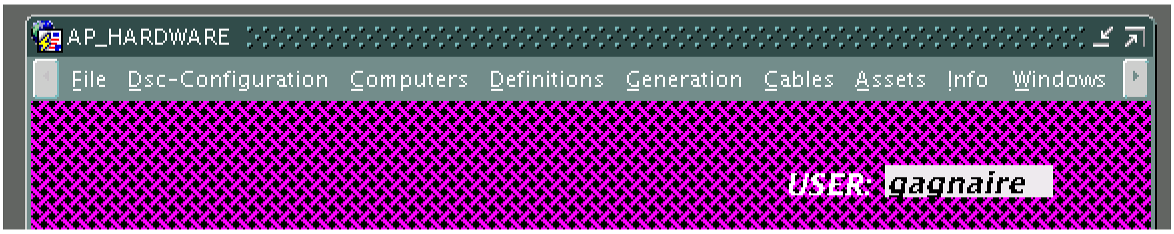
- generation of the DSC start up sequence; this include s command line to start software driver of module or family of modules.
- a priori checking to control discrepancy in the theoretical configuration description eg: a VME address cant be assigned to several module.
- to perform run time checking: does the expected module are present, how many module are available for building a pool,
- to ease I/O access from the program level running in the host computer (logical addressing principle as explain before)
- to start the driver required for some module.

7. Data entry

All data are entered into the database trough Forms.
The Data base user interface is reach by clicking in the Web page http://wwwpsco.cern.ch/private/db the hyper link Call Web Form which prompt you to login as data base user. Once logged you have to select the HARDWARE application which provide the Form interface.

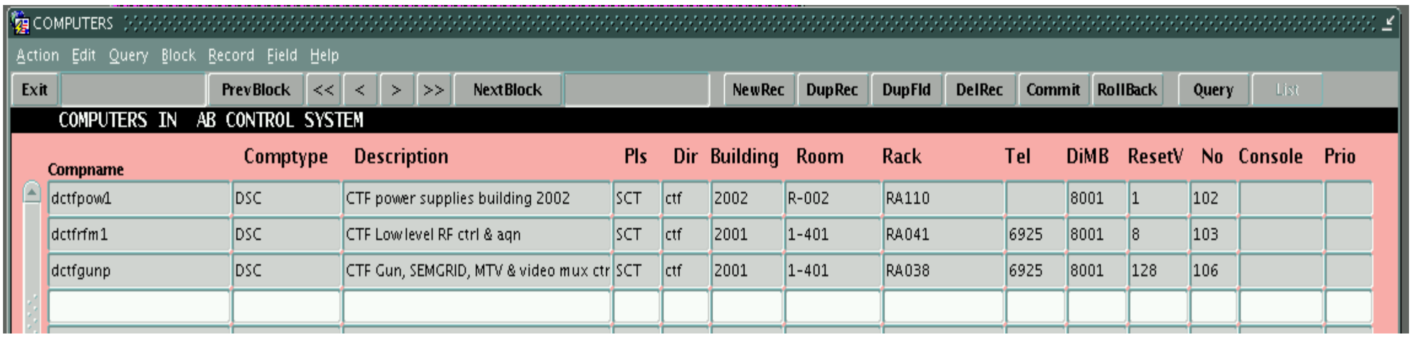
7.1. the data entry services:

It provides a menu bar



7.2. entering a new DSC in the database:

To add a new DSC in the configuration, the name of the associated computer must be first be added in the computer table. To record a new computer name, select the entry "Computer list " in the "Computer" pop up menu of the main menu bar. The query form view of the computer table appears, select cancel query to enter the form to fill up: computer name, computer type DSC, etc …



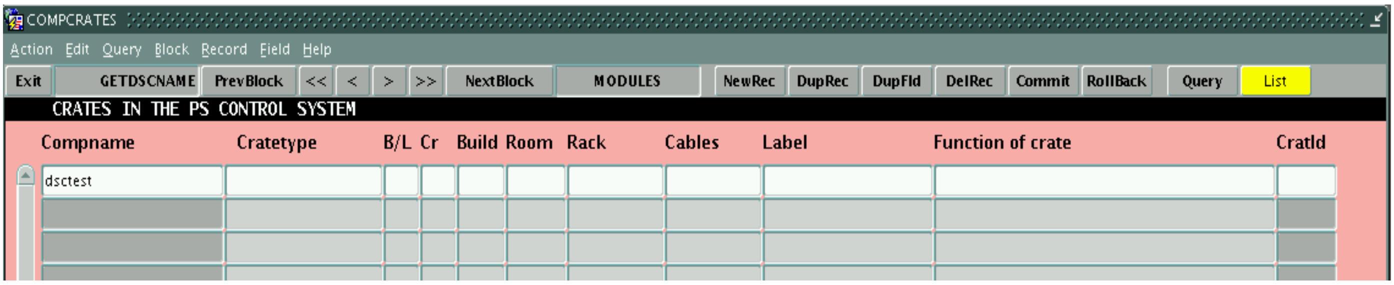
Once the dsc name, the configuration can be define

-

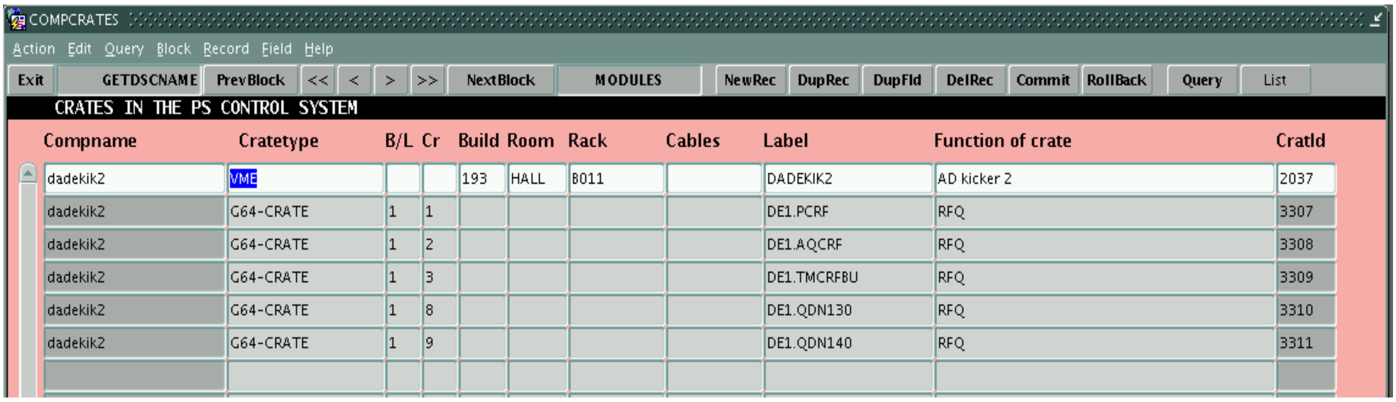
7.3. entering the DSC crate configuration in the database :

Select “Dsc Crate” entry in the “DSC-configuration” pop up menu of the main menu bar. A form view appears asking for the target dsc name, type the DSC name and click the DSC Crate button in the form. The DSC crate table form appears, select NewRec button in the menu bar to add the new crate record you and fill it up:

7.3.1. adding a new crate

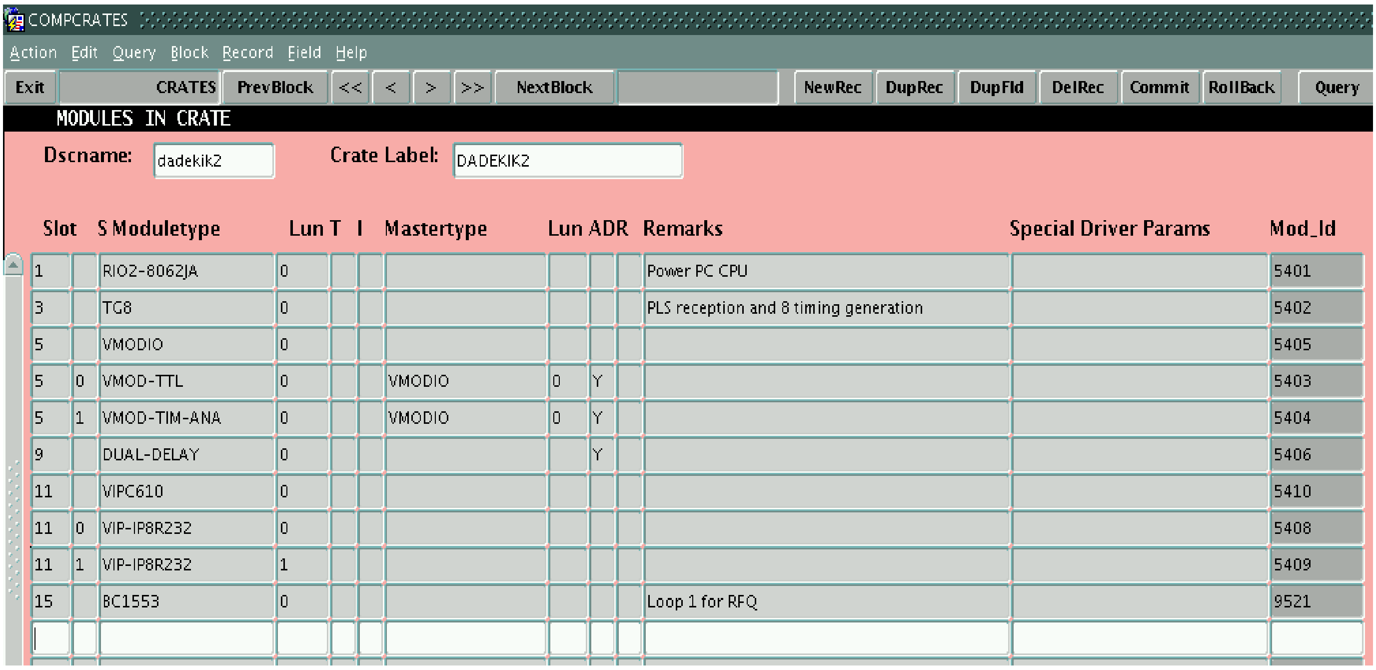


According to the actual target configuration one new record is filled up for each crate of the actual dsc configuration, e.g.: VME crate, G64 crate



7.3.2. configuring a crate

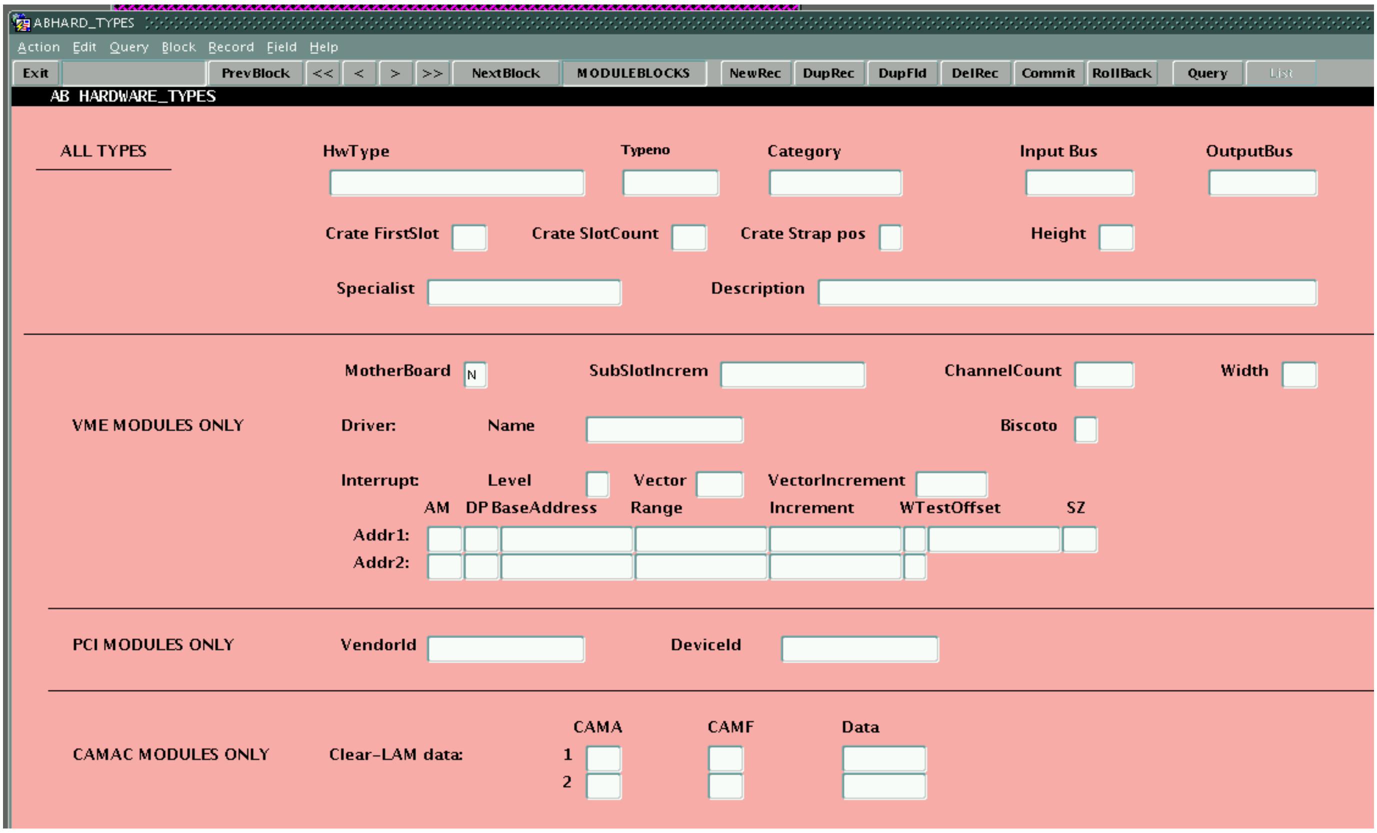
Once the new crate is defined, it must be given a module configuration. To record a crate configuration select the crate line you want to update, selecting in the main menu bar the NextBlock for the selected crate record . This open the module crate table, selecting the new record button will add a new line for the new module to add and fill it up:



Slot	S	Moduletype	Lun	T	I	Mastertype	Lun	ADR		Remarks	Special Driver Params	Mod_Id
1		RIO2-8062JA	0							Power PC CPU		5401
3		TG8	0							PLS reception and 8 timing generation		5402
5		VMODIO	0									5405
5	0	VMOD-TTL	0			VMODIO	0	Y				5403
5	1	VMOD-TIM-ANA	0			VMODIO	0	Y				5404
9		DUAL-DELAY	0					Y				5406
11		VIPC610	0									5410
11	0	VIP-IP8R232	0									5408
11	1	VIP-IP8R232	1									5409
15		BC1553	0							Loop 1 for RFQ		9521

7.4. Entering a new module type definition in the database:

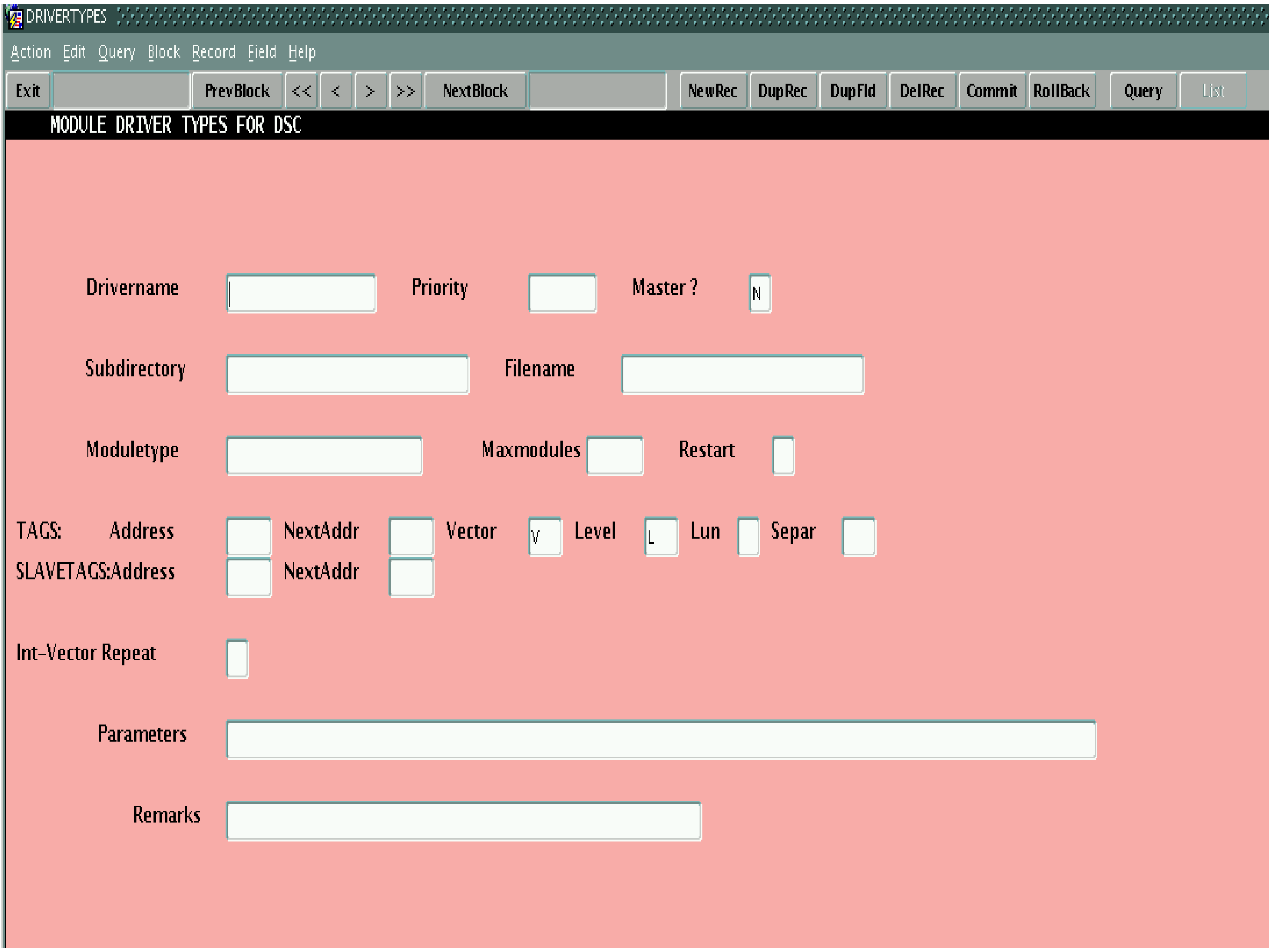
When a new module occurred in a DSC configuration the corresponding module type must declared.
To record a new module type, select "#AB_hardware_Types" entry in the "Definition menu" pop up menu of the main menu bar , cancel the proposed Query mode, an empty module type appear , e.g.:



-

7.5. Entering a new driver type definition

Select "Definition" entry in the "#List of Drivertypes" popup menu of the main menu bar, cancel the proposed query mode, select in the displayed menu bar NewRec button, an empty form is displayed, fill it

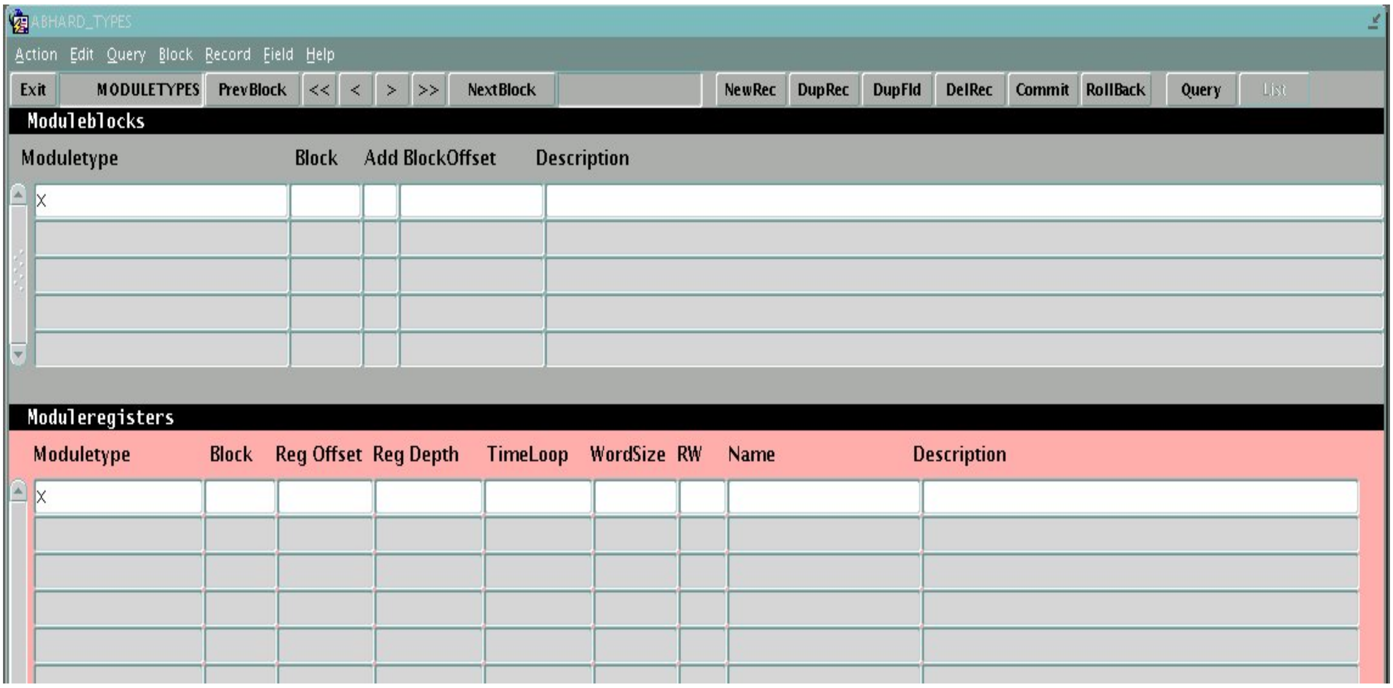


7.6. Entering a module addressability description :

When the biscoto flag is set to Y in the module type description, the user can define the description of the module as seen from addressing point of view.

This information is dedicated to the driverGen tool which extract the module type information and produce automatically after this the header file and minimum direct access library interface for the module.

Selecting NextBlock in the menu bar of the module type window displays the module space topology, e.g.:



in the upper block are entered the different addressable area of the module and lower part display the different address available in the selected block in the upper part

where:

- Module type is automatically generated from the previous block, the currently defined module
- Block: the block number definition
- Add: this tell which Address part (1 or 2, see in module type window) the description correspond to
- BlockOffset: the block offset from the base address

In the lower part is the description of the available address

Where :

- Module type is automatically generated from the previous block, the currently defined module
- Block: is automatically generated from the selected block in the upper part of the window
- RegOffset: the offset of the address from the block Offset
- Depth: 0 for scalar register , (-1) if fifo address, >0 addressable elements number from this address place(WordSize gives the element size)
- TimeLoop: delay loop number to face hardware delay answer
- WordSize: size of each element in the addressable place
- RW: access mode : r, w, rw and e for external variable
- Name: name of the addressable place

7.7. An example of module addressablity description: the CIBC

ABHARD_TYPES

Action Edit Query Block Record Field Help

Exit | MODULETYPES | PrevBlock | << | < | > | >> | NextBlock | NewRec | DupRec | DupFld | DelRec | Commit | RollBack | Query | List

Moduleblocks

Moduletype	Block	Add	BlockOffset	Description
SHARED_CIBC	0	1	0	Registers
SHARED_CIBC	1	1	0	Reset the module
SHARED_CIBC	2	1	0	All Paraisitc Registers counter
SHARED_CIBC	3	1	0	UTC Seconds and Microseconds
SHARED_CIBC	4	1	0	Data Trace (History Buffer)

Moduleregisters

Moduletype	Block	Reg Offset	Reg Depth	TimeLoop	WordSize	RW	Name	Description
SHARED_CIBC	0	0	0	0	long	r	STATUS	Current state of the module
SHARED_CIBC	0	0X4	0	0	long	r	DISBL_INPUT	Position of the on-board 'Disable swithches'
SHARED_CIBC	0	0X8	0	0	long	rw	MASK_INPUT	Mask Setting/Getting
SHARED_CIBC	0	0XC	0	0	long	r	INPUTS	Current state of the module inputs
SHARED_CIBC	0	0X10	0	0	long	rw	MATRIX_IN	Level inputs of the two Matrixes
SHARED_CIBC	0	0X14	0	0	long	r	RESET_time	Reset Occurrence time
SHARED_CIBC	0	0X18	0	0	long	r	PARASITIC_IN00	Parasitic counter for Input #00
SHARED_CIBC	0	0X1C	0	0	long	r	PARASITIC_IN01	Parasitic counter for Input #01
SHARED_CIBC	0	0X20	0	0	long	r	PARASITIC_IN02	Parasitic counter for Input #02
SHARED_CIBC	0	0X24	0	0	long	r	PARASITIC_IN03	Parasitic counter for Input #03
SHARED_CIBC	0	0X28	0	0	long	r	PARASITIC_IN04	Parasitic counter for Input #04
SHARED_CIBC	0	0X2C	0	0	long	r	PARASITIC_IN05	Parasitic counter for Input #05
SHARED_CIBC	0	0X30	0	0	long	r	PARASITIC_IN06	Parasitic counter for Input #06
SHARED_CIBC	0	0X34	0	0	long	r	PARASITIC_IN07	Parasitic counter for Input #07
SHARED_CIBC	0	0X38	0	0	long	r	PARASITIC_IN08	Parasitic counter for Input #08
SHARED_CIBC	0	0X3C	0	0	long	r	PARASITIC_IN09	Parasitic counter for Input #09
SHARED_CIBC	0	0X40	0	0	long	r	PARASITIC_IN10	Parasitic counter for Input #10
SHARED_CIBC	0	0X44	0	0	long	r	PARASITIC_IN11	Parasitic counter for Input #11

Block offset in Hex

Record: 1/? ... <OSC>

8. Exploitation of the DSC hardware configuration:

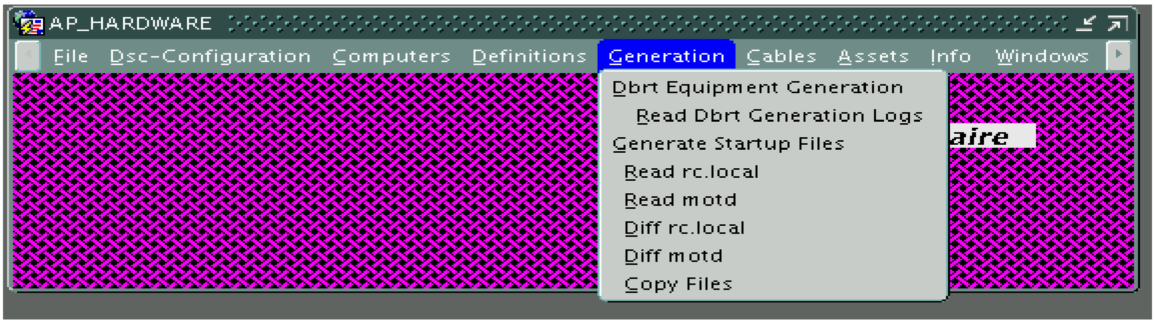
Once the dsc configuration is fully entered, exploitation tool may be used :
The generation command may be performed either from the Oracle form interface as displayed below or direct from shell script command

8.1. The shell command:

The generation command may be started from shell command which invoke script file starting the facility

8.2. the Oracle form tool :

- To generate the start up sequence file: the rc.local file for the associated dsc
- To distribute the rc.local file in the DSC serveur in order to start the DSC with the actual configuration as declared in the database
- Compare with the current operational version
- To save in the Dbrt (real time database) the actual dsc configuration table. These tables are dedicated to driverGen tools for automatic generation of a module direct access library interface (based on a direct access interface or a ioctl driver interface)



8.3. the generation of the DSC start up sequence:

The generation command may be performed either from the Oracle form interface as displayed below or direct from shell script command

8.3.1. From the shell command level:

To generate and deliver the rc.local of a given DSC, cd the DSC target directory and invoke the make program.:

>cd /ps/src/dsc/<machine>/< target dsc name>
> make rc.local

e.g.: to regenerate, check and deliver the dleibgen DSC rc.local file perform the following command:

>cd /ps/src/dsc/dleigen
>make rc.local

N.B. *:* this is done provided the AB-CO standard makefile and AB-CO Make include are available in the environment.

8.3.2. From the Oracle form interface level:

Selecting the entry "Generate Startup File" in the "Generation" pop up menu of the main menu bar starts a database application which extract the DSC configuration table, check the validity of this declaration, and generate the startup sequence after the configuration description, provided no error checking arise.

The generated file named `rc.local` is an extension of the `rc` file performed by system initialization after reboot. A text file is produced, its content is the script of the sequence to execute as local extension of the `rc` file after reboot .

8.3.3. analysis of the generated DSC rc.local file

This file is made of shell commands and special comments lines

- The TAG comment line: the TAG (#+#) is put in front of the line in order to enable the ioconfigInstall program to process these special comment line. The information put in the Tagged line summarise in a text form the information on the hardware element as declared in the DSC configuration:
    - VME module and associated info (addresses, vector, interrupt …)
    - CAMAC crate and and loop
    - GPIB loop and instrument
    - Logical event and associated device source
- The shell command: they perform all system command required to initialise the system, start the different driver required by the configuration and start up of the application programs .

Example:
Below extracted from the generated rc.local,

➢ the tagged comment line to bring the configuration information in the header of the rc.local file

```
#!/etc/bash
# dtstbd       startup file rc.local, generated 2005-JAN-20/15:51  .
export PATH=.:/etc:/dsc/local/bin:/usr/local/bin:/usr/local/rt:/bin:/usr/bin
#*****************************************************************************
#   WARNING :  File generated from database.
#              Can be overwritten at any time !
#
#*****************************************************************************

# ***** IOCONFIG Information *****

#ln mln mtno module-type    lu W AM DPsz basaddr1 range1  W AM DPsz basaddr2 range2 testoff  sz sl
ss
#+#  1   0 VME   23 TG8      0 N ST DP16   c00000     1000 N -- ----     0       0        0  2  4 -1
#+#  2   0 VME   59 IPP-1    0 Y EX DP32  2800000   400000 N -- ----     0       0   380007  0  8 -1
#+#  3   0 VME   22 MVME167  0 N -- DP16        0        0 N -- ----     0       0        0  0  2 -1
#+#  4   0 VME   55 VTSM     0 Y ST DP16   a00000    10000 N -- ----     0       0     8000  2  5 -1
#+#  5   0 VME    0 SDVME    0 Y SH DP16     f800      400 N -- ----     0       0        0  2  6 -1

#   ln sln      mtno module-type      lu evno   subaddr    A1 F1     D1     A2 F2     D2
#+#  6   1 EVT   23 TG8               0    1     20401
#+#  7   1 EVT   23 TG8               0    2     20402
#+#  8   1 EVT   23 TG8               0    3     20403
#+#  9   1 EVT   23 TG8               0    4     20404
#+# 10   1 EVT   23 TG8               0    5     20405
#+# 11   1 EVT   23 TG8               0    6     20406

#   ln mln      mtno module-type  lp cr
#+# 12   5 CAM  826 SCC-L2         1 11

# ***** Program Startup before drivers *****
```

➢ the automatically generated line for loading and installing drivers after the configuration

```
# ***** Driver Initialisation *****
cd /usr/local/drivers/sacvme; sacvmeinstall -R0 -M0 -V254 -L2

cd /usr/local/drivers/tg8; tg8install  -file /tmp/tg8infofile.out  -M0xc00000 -
V184 -L2

cd /usr/local/drivers/camacsdvme; camacsdvmeinstall  -Af800 -V160 -L2

cd /
# ***** Program Startup after drivers *****
# Install data used by ioconfig library
ioconfigInstall
```

8.3.4. Configuration checking at the file generation

At that time the actual service provide a priori checking only for the VME bus, CMAC bus and PCI bus. The actual checking are:
The checking depends on the bus type where the module are declared, as a matter of fact each bus is seen from the host computer trough an electronic interface depending on the bus type. For each type the a priori checking may lead to different algorithms.

- VME info, modules and base address checking for the VME configuration
- Camac crate and camac slot checking for the CAMAC configuration
- Interrupt checking for the host computer
- Driver generation checking for the start up sequence of the drivers.

These checking are performed at the generation of the Start up sequence of the DSC.
When discrepancy or errot is detected the file is not generated

8.4. generation of the Dbrt equipement :

when the "Dbrt Equipement Generation" in the "Generation" pop up menu is selected, it starts a database application which update the Dbrt with the equipement definition of the databse. This concern the module type definition with the Biscoto extension.
The Dbrt module type information is exploited by the `driverGen` program
This update must be done each time a module type is changed.

> WARNING: Currently for security reason the Web form can't operate this action, to perform it you have to do it on a Linux system and invoke the program dbrt_gen

N.B.: beware:
any change in the module type description for a module using the Biscoto flag, may lead to discrepancy with the associated code previously produced by `driverGen.`

8.5. The driverGen program

Once a new module type is fully entered in the database and when Biscoto flag is set to Y, the drivergen program may be used to automatically the all set of file of a direct acces library interface for he corresponding module.
The driverGen facility generate after Dbrt information all files required , i.e.:

- Depositery directory and subdrectory
- Header files, library and direct access driver source files
- Make file to compile and deliver the object file; drivers and library object file

9. The hardware configuration at the DSC runtime:

9.1. installation of the hardware configuration: the `iocongfigInstall` program

at the start up of the DSC running the `ioconfigInstall` program install locally the interface with the hardware configuration and perform the run time checking of the installation as describe in the Tagged comment lines of the rc.local file, i.e.:

- it checks the syntax of the line
- it stuffs the hardware configuration converted in table in a shared memory segment.
- It stuffs a table with the description of the logical event in the shared memory segment and sets up the mapping window to access VME modules.
- It performs the presence of modules checking when requested.

9.2. The hardware configuration library

The hardware configuration ioconfigInstall put in the shared memory segment is not directly visible from the user's program, an interface library (`ioconfiglib`) is available for that and it provides several services:

- Function to find element after the logical address i.e.: {module type, Lun}
- Function to get the direct access pointer to a module specified by its logical address (`IocModulPointer()`).
  N.B. This function returns an error if the module was not allowed to be directly access at the configuration declaration.
- Function to list the all configuration used in services as `ioconfigDisplay` program
- Function to find the link between a module and its master module
- Function `IocGetEventInfo()` to find the information associated to a logical event i.e.: module type and lun of the device source).

Example:

Simplest programming example
using the logical device address and the address resolution trough the `ioconfiglib` functions

```
int L_cc;
unsigned char* lptr;
int llu, plu, tlu, ldata;
int L_val;
int loffset;
char *L_errmsg;
     /*  */ ...
llu = 0;
tlu = IocModuleType;
loffset = <value of the offset in the module space>
     ...
L_cc = 0;
L_cc = IocModulPointer(tlu, llu, plu, &lptr); /* getting the direct pointer to the
module space*/
if (L_cc){
    IocGetErrorMessage(L_cc , &L_errmsg);
    printf("IocModulPointer error: %s\n", L_errmsg);
      return(L_cc);
}
      ...
printf("value to write= 0x%lx\n", ldata);
L_val = ldata;
L_ptr = luptr + offset;

(*((long *)L_ptr))= (long)L_val;  /* writing module through the pointer */

printf("wrote:  0x%x \n", L_val);
L_data = (int)(*((long *)L_ptr)) /* reading the  module through the pointer */

printf("read back:  0x%x \n", L_data);
```

10. References:

- Using driverGen (AB-CO-FC Note 2005 Y.Georgievskiy, A. Gagnaire)
- Argument for HardwareConfiguration managementSystem.doc 25/11/2004 A. Gagnaire
- PS-CO note and ICALPEPCS'95 document : Automatic Generation of Configuration Files for a Distributed Control System by J. Cuperus and A. Gagnaire.
- PS-CO/Note 93-080 : DSC's configuration management (Alain GAGNAIRE)