EPICS Oracle Database Tutorial

\\DOSERVER4\projects\Online_Computing\Tutorials\HdbTutorial

Stan Krzywdzinski
May 2, 2001

Introduction

The EPICS Database, implemented as a relational Oracle database, serves as a repository of the following objects:

- EPICS record types, as given by the .dbd files
- EPICS templates, as given by .dbt files
- EPICS generators, as given by .dbg files
- instances of EPICS records, which is the information contained in EPICS .db files

In addition to storing the EPICS records, related to all of the front-end nodes (IOC's) used at D0, the database provides a framework for structuring these records.
A number of records could be grouped into devices, e.g. an entire power supply. The devices in turn, and thus their records, could be further grouped according to the following categories:

- detector type, e.g. CALC, CFT, CPS, ICD, MUOC, SMT
- device type, e.g. RM, RMI, LVCA, LVCB, VBD, VRB
- templates, e.g. rm.dbt, rmib.dbt, lvca0l.dbt, lvcb2r.dbt vbdb.dbt, vrb.dbt
- front-end node, the records pertain to, e.g. d0olctl09 d0olmuo25
- location of a device, or node, in terms of its house, rack, crate e.g. MCH-3/300/B2, PN/08

Naming

A device is referred to by its unique name. The name should follow the adopted convention for naming devices:

<det>_<devtype>_<loc>

Likewise, a record is referred to by its unique name. Record names in .db, or .dbt, files, which belong to a device named:

<device>

should inherit the device name in the following way:

<device>/<attr>

i.e. the record name should be the device name followed by a "/" separator and an <attr> extension which is unique among the records of that device.

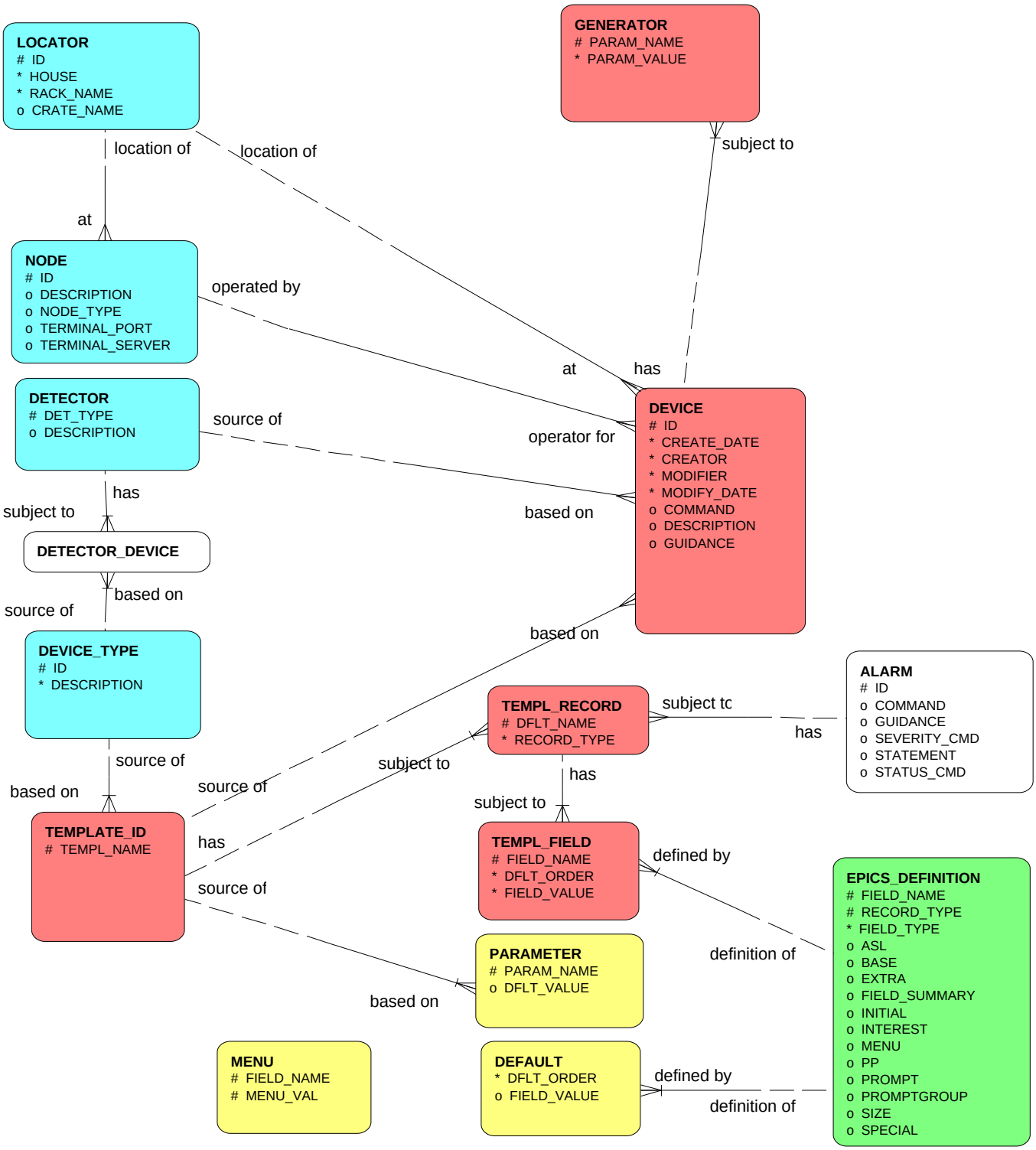
Content & Relations

The database tables, their content, and relations between them, can be depicted by the Entity-Relationship Diagram.

The database employs the standard features:

- constraints, e.g. to enforce:
  - unique device and record names,
  - record types and their field names as defined by EPICS .dbd file
- triggers to update, or delete all related tables upon either action on the parent table
- view supported by a stored custom function, to create record instances based on a stored template and a corresponding set of substitution parameters.

E-R Diagram


LOCATOR
ID
* HOUSE
* RACK_NAME
o CRATE_NAME
GENERATOR
PARAM_NAME
* PARAM_VALUE
location of
location of
subject to
at
NODE
ID
o DESCRIPTION
o NODE_TYPE
o TERMINAL_PORT
o TERMINAL_SERVER
operated by
at
has
DETECTOR
DET_TYPE
o DESCRIPTION
source of
operator for
DEVICE
ID
* CREATE_DATE
* CREATOR
* MODIFIER
* MODIFY_DATE
o COMMAND
o DESCRIPTION
o GUIDANCE
has
subject to
based on
DETECTOR_DEVICE
based on
source of
based on
DEVICE_TYPE
ID
* DESCRIPTION
TEMPL_RECORD
DFLT_NAME
* RECORD_TYPE
subject to
has
ALARM
ID
o COMMAND
o GUIDANCE
o SEVERITY_CMD
o STATEMENT
o STATUS_CMD
source of
based on
source of
subject to
has
subject to
TEMPLATE_ID
TEMPL_NAME
has
source of
TEMPL_FIELD
FIELD_NAME
* DFLT_ORDER
* FIELD_VALUE
defined by
EPICS_DEFINITION
FIELD_NAME
RECORD_TYPE
* FIELD_TYPE
o ASL
o BASE
o EXTRA
o FIELD_SUMMARY
o INITIAL
o INTEREST
o MENU
o PP
o PROMPT
o PROMPTGROUP
o SIZE
o SPECIAL
definition of
PARAMETER
PARAM_NAME
o DFLT_VALUE
based on
MENU
FIELD_NAME
MENU_VAL
DEFAULT
* DFLT_ORDER
o FIELD_VALUE
defined by
definition of

Tables & Views

Tables	Count as of 11/19/01
ALARMS	1
DEFAULTS	1648
DETECTORS	33
DETECTOR_DEVICES	0
DEVICES	4502
DEVICE_TYPES	75
EPICS_DEFINITIONS	1983
GENERATORS	34958
LOCATORS	284
MENUS	40
NODES	87
PARAMETERS	789
TEMPLATE_IDS	137
TEMPL_FIELDS	25797
TEMPL_RECORDS	1517
Views	
DEV_INSTANCES	897858

Utilities to Access

Custom utilities were created, to enter, maintain and extract the data from the EPICS Database:

- hdbWeb: interactive-type Web-based GUI
- hdbBatch: batch-type Python scripts

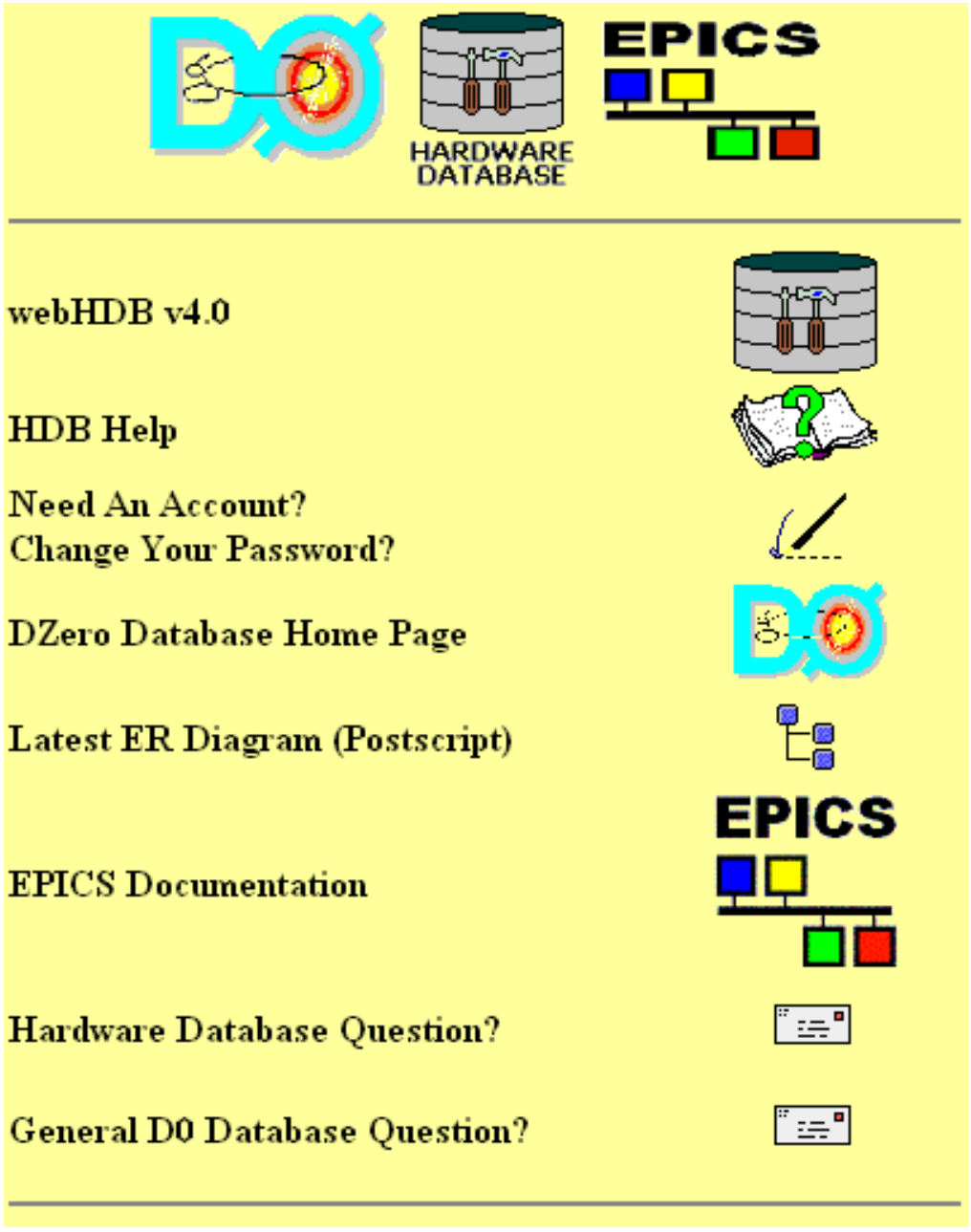
Oracle account (username/password) to the D0 Production Database, d0onprd, with either hdb_operator, or hdb_administrator role granted, is needed in order to use these utilities.

Oracle Enterprise Manager, a powerful GUI interface available on NT and Unix, allows a DBA to do almost anything to a database, including manipulation of database definitions and data in tables.

One can always resort to SQL*Plus …

Where is it ?

http://www-d0online/hdb/



HDB Help

http://www-d0ol/hdb/doc/

Batch Utilities (1) …

On all of the **D0 online** nodes and **d0mino**:

setup hdb

defines everything needed to run the scripts. The scripts require input arguments to do the work.
Typing script name with no arguments displays its usage, except sqlplus.py.
Upon typing script name with the required arguments, username and password are prompted for in general, prior to the execution. The exceptions to the prompting are:

hdb_extract.py, db_sort.py, db_sort_recs.py, dbt_params.py, db_compare.py .

The scripts are supported by two libraries:

- hdblib.py - library of functions for accessing the database
- parslib.py - library of functions for parsing generic EPICS .db / .dbt files, and enhanced .hdb / .hdbg files

… Batch Utilities (2) …

Delete, extract, insert, replace:

- hdb_delete.py - batch delete of fields, records and devices from the database
- hdb_delete_dbg.py - batch delete of devices, defined by Epics generator .dbg file, from the database
- hdb_extract.py - to make EPICS flat ascii files from the database and complementary listings to terminal screen
- hdb_insert.py - batch load of data from EPICS .db file, supplemented by supporting data, into the database
- hdb_insert_dbg.py - load data from Epics generator .dbg file, supplemented by supporting data, into the database
- hdb_insert_dbt.py - load data from Epics template .dbt file into the database, or replace template already stored

… Batch Utilities (3)

- hdb_insert_defaults.py - populate DEFAULTS table using data from EPICS_DEFINITIONS table
- hdb_insert_epics_defs.py - load data from EPICS .dbd file into the database
- hdb_list.py - batch listing of device(s) info, including supporting data, from the database

Miscellaneous:

- db_sort.py - sorts EPICS .db file
- db_sort_recs.py - from a list of record names generates sorted list of corresponding devices
- dbt_params.py - lists substitution parameters of a template file
- db_compare.py - database versus reference comparison of node.db and template .dbt files
- sqlplus.py - wrapper around Oracle sqlplus

hdb_extract.py (1) …

Extracts:

- EPICS database .db file for a given front-end node
- EPICS database .db file for a given device
- EPICS template .dbt file for a given template file name
- EPICS generator .dbg file for a given front-end node

Usage for schema HDB:

```
> hdb_extract.py -d device [output.db]
> hdb_extract.py -n  node [output.db]
> hdb_extract.py  -t  template [output.dbt]
> hdb_extract.py -g  node [output.dbg]
> hdb_extract.py  -lt dev_type
Adding l to -d/-n/-t/-g lists devices/templates to stdout
Adding s to -d/-n/-t sorts .db/.dbt
Adding o terminates prompting for more
```

If an optional, output file name, argument is omitted, the output file inherits its name from the preceding argument. In case of listings, when the 'l' flag is used, the wildcard character '%' can be embedded into 'device' and 'dev_type' arguments.

… hdb_extract.py (2)…

Example 1a – extract epics.db file for node d0olctl20, using the utility directly

```
d0olb:krzyw> hdb_extract.py -n d0olctl20 epics.db
EPICS records from HDB.TEMPL_RECORDS and HDB.TEMPL_FIELDS tables, for
    node "d0olctl20":
  225 Epics records selected, containing 2088 fields total
      8 record types encountered
File "epics.db" created, of 2763 lines
================================================================
    ===
Enter arguments, or CR to quit: hdb_extract.py
d0olb:krzyw>
```

Example 1b – extract epics.db file for node d0olctl20, using Makefile in the ioc directory on host:

```
d0ol02:krzyw> pwd
/online/ioc/ppc/mv2300/d0olctl20
d0ol02:krzyw> setup onl_ioc
d0ol02:krzyw> gmake reinstalldb
Generating epics.db from Oracle for d0olctl20
.
File "epics.db" created, of 2763 lines
d0ol02:krzyw>
```

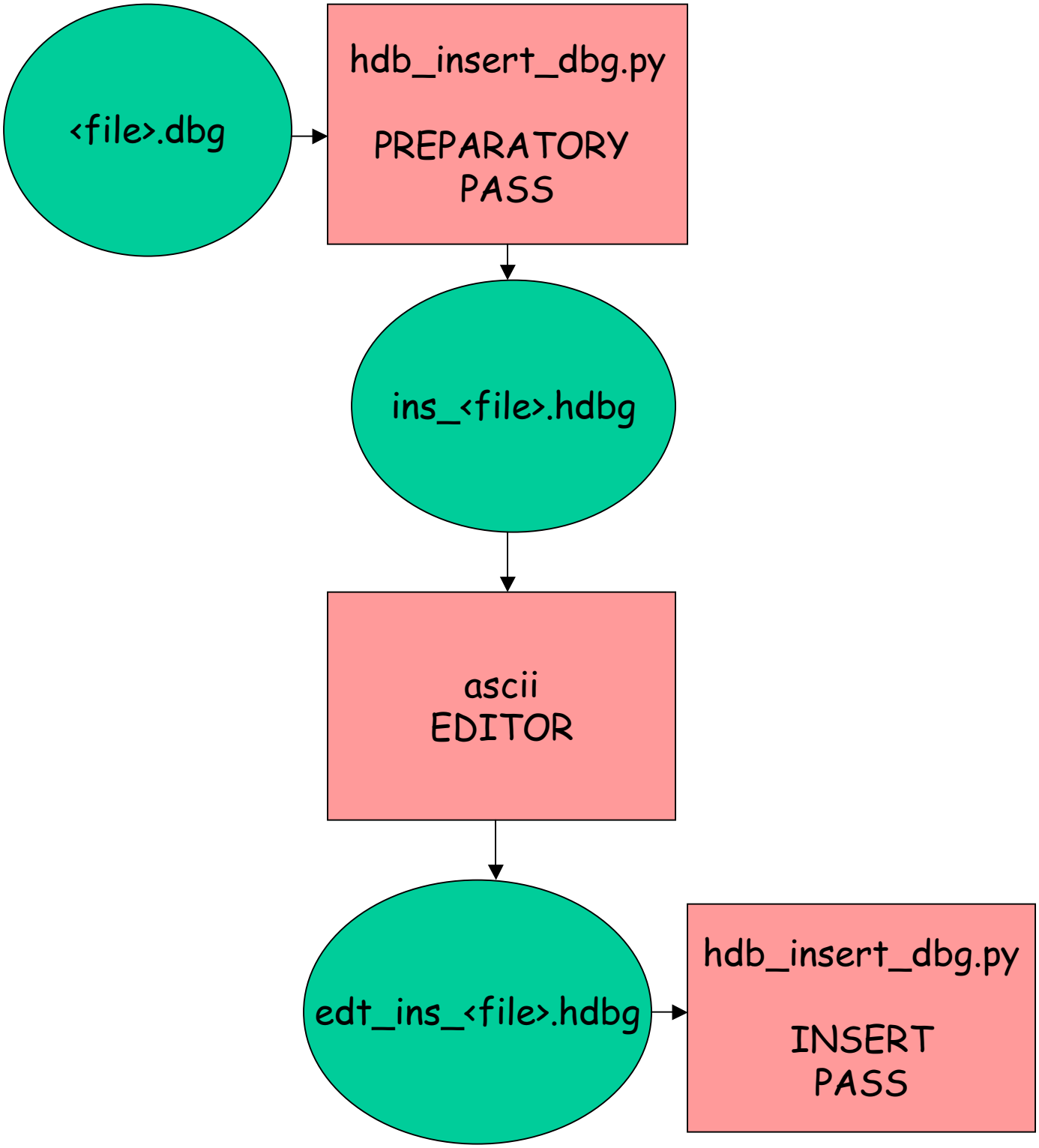
… hdb_extract.py (3)

Example - list all templates stored in the database:

```
d0olb:krzyw> hdb_extract.py -lt %

DEV_TYPE                        TEMPLATE          #EPICS RECS
=================================================================
          ADC                        adcrad.dbt                        23
          AFE                            afe.dbt
3
         CCCT                           ccct.dbt
9
         CCCT                       ccctctrl.dbt                        12
         CMCA                      cmca10.dbt                        14
            .                                .
.
         VRBC                           vrbc.dbt                         4
         VRBC                       vrbc_cft.dbt                        13
         VRBC                         vrbcn.dbt                         3
         VRBC                      vrbcnwu.dbt                         8
87 template files (1453 epics records total) for device type containing string '%'
=================================================================
Enter arguments, or CR to quit: hdb_extract.py
d0olb:krzyw>
```

hdb_insert_dbg.py (1) ...



… hdb_insert_dbg.py (2) …

Inserts data from an Epics generator .dbg file into the database.
Two passes are needed to accomplish the task. In the PREPARATORY PASS, an intermediary file is generated with the assembled devices, in a format required, and the missing supporting data indicated by '???*' - to be filled in by a user. Having the intermediary file edited and saved, it should then be run through the utility again, which recognizes the INSERT PASS if all the missing data has been provided.

Usage for schema HDB:

```
> hdb_insert_dbg.py <file>.dbg [node] [locator]
    PREPARATORY PASS, to make preliminary  ins_<file>.hdbg – to  be
    edited!
> hdb_insert_dbg.py edt_ins_<file>.hdbg
    INSERT PASS, to insert rows to HDB.DEVICES and
    HDB.GENERATORS tables.
```

Optional arguments, may be used to fill in the node and/or locator data already at the PREPARATORY PASS. The data is extracted from the database, so it needs to be there prior to running the utility; otherwise the missing tokens would be inserted, as in the case of no optional arguments provided.

… hdb_insert_dbg.py (3) …

Also note, that the same node and/or locator would be assigned to every device assembled! As for the node, this is usually the situation since .dbg files tend to be node oriented. Locator, however, specifying house/rack/crate could be different for devices in a batch. One can use predefined 'global', but less specific locators like AAANULL (null locator), MCH-1, MCH-2, MCH-3.

Lastly, if a device is already present in the database, the node and/or locator arguments will be ignored.

Example - insert data from an example.dbg file containing the following:

```
file ../../../templates/rm.dbt {
    { det=CALN, loc=PN11A, chan=0, rt=6, scan="1 second", phas=0,
      bmask=0x9, bwc0=1 }
    }
file ../../../templates/rmic.dbt {
    { det=CTL, loc=PN11, rm=CALN_RM_PN11A, schan=03, cchan=01,
      scan="1 second", phas=0 }
    }
```

Running the utility on this file produces the following to the terminal:

… hdb_insert_dbg.py (4) …

```
d0olb:krzyw> hdb_insert_dbg.py example.dbg d0olctl11
Username: krzyw
Password:
PREPARATORY PASS
================================================================
===Template = rm.dbt     1 set(s) of parameters found:
 ----------------------------------------------------------------
----------------------------------------
 {'chan': '0', 'det': 'CALN', 'phas': '0', 'scan': '1 second', 'bmask': '0x9',
  'bwc0': '1','loc': 'PN11A', 'rt': '6'}
 #1 device is CALN_RM_PN11A
================================================================
===Template = rmic.dbt     1 set(s) of parameters found:
----------------------------------------------------------------
----------------------------------------
{'det': 'CTL', 'phas': '0', 'rm': 'CALN_RM_PN11A', 'scan': '1 second',
 'schan': '03', 'cchan': '01', 'loc': 'PN11'}
 #2 device is CTL_RMI_PN11
================================================================
===
          Input file used: example.dbg , of 7 lines
      2 template files found
      2 set(s) of parameters found
          Output file created: ins_example.hdbg , of 61 lines
      2 HDB devices assembled
================================================================
===Another file, or CR to quit:
d0olb:krzyw>
```

Depending on what supporting data is already stored in the database, the created ins_example.hdbg file may look as follows:

… hdb_insert_dbg.py (5) …

```
devblock(1)   {
            device("CALN_RM_PN11A")  {
                        description("???dev_desc")
                        template("rm.dbt") }
            generator  {
                        chan=0,
                        det=CALN,
                        phas=0, scan="1 second",
                        bmask=0x9,
                        bwc0=1,
                        loc=PN11A,
                         rt=6 }
            detector("CALN")  {
                        description("Calorimeter North")  }
            devtype("RM")  {
                        description("Rack Monitor")  }
            node("d0olctl11")  {
                        locator("MCH-3/300/C2")  {
                                    house("MCH-3")
                                    rack("300.)
                                    crate("C2")  }
                        nodetype("MVME2301")
                        termserv("t-d0-mch3")
                        termport("5")
                        description("Calorimeter MVME2301 PPC in rack
                                    M300,crate C,partition 2") }
            locator("???loc")  {
                        house("???house")
                        rack("???rack")
                        crate("???crate")  }
}
```

... hdb_insert_dbg.py (6)

```
devblock(2)   {
          device("CTL_RMI_PN11")  { .
}
```

This file should then be edited by filling out all the missing data, indicated by '???*', and saved, say, as edt_ins_example.hdbg .

Running the edited file through the same utility again, finally loads the information into the database:

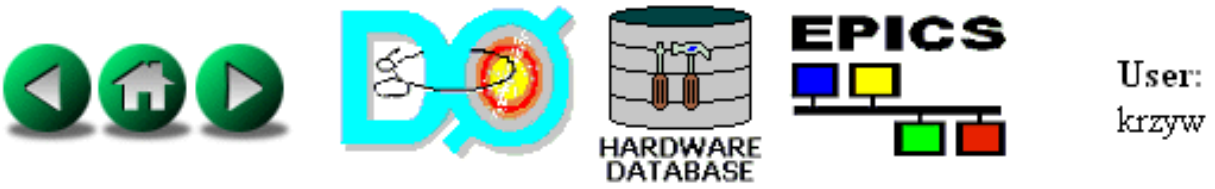
```
d0olb:krzyw> hdb_insert_dbg.py edt_ins_example.hdbg

Username: krzyw
Password:

INSERT PASS
------------------------------------------------------------
------------------------------------
Processing devblock( 1 ) ...
... device CALN_RM_PN11A
------------------------------------------------------------
------------------------------------
Processing devblock( 2 ) ...
... device CTL_RMI_PN11
============================================================
        Input file used: edt_ins_example.hdbg , of 69 lines
     2 HDB device blocks found
------------------------------------------------------------
------------------------------------
     2 rows inserted into HDB.DEVICES table
    15 rows inserted into HDB.GENERATORS table
============================================================
Another file, or CR to quit:
d0olb:krzyw>
```

Web GUI (1) …

Web-based CGI interface. Uses Python scripts on the server to generate HTML forms, with some JavaScript enhancements. These forms are used to input and view data from the Oracle Database.



MAIN MENU

Add Device (Templated)	Add Device (Custom)
Edit Existing Template	Template Name [ ] LIST
Edit Existing Device	Device Name [ ] LIST
Delete Existing Device	Device Name [ ] LIST
Clone Existing Device	Device Name [ ] LIST
View Existing Device	Device Name [ ] LIST
Find Existing Device	

Dzero Hardware DB Interface

…Web GUI (2)

- Main Interface
  - hdbEnter.py – Login and Main Menu
  - getDev.py – Popup pick lists from the Main Menu
  - hdbAdd.py – Add templated devices
  - hdbSngl.py – Add custom devices
  - hdbTemplEdit.py – Edit template
  - hdbEdit.py – Edit device
  - hdbClone.py – Clone device
  - hdbDel.py – Delete device
  - hdbView.py – View device in EPICS format
  - hdbFind.py – Query for a device

- Libraries
  - hdbLib.py – Common methods
  - hdbOraLib.py – Common db methods
  - webCGI.py – CGI form and parsing helper

- Configuration
  - conf.py – per instance global variable configuration file

Main Menu

Provides access to different functions, by clicking the appropriate button.
Some functions (Edit, Delete, Clone, View) require a device name to be entered in the corresponding blank.
Clicking the 

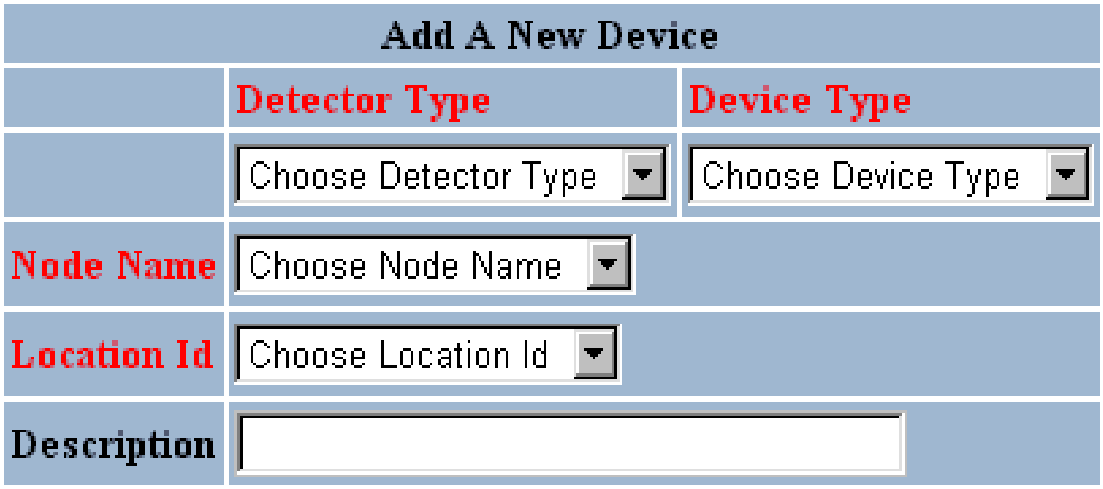
icon brings up a list of all devices currently in the database. Choose one and press the select button. This will fill the corresponding blank with the selected device name.
Pointing mouse over  will provide a more complete description of the function:

- Add Device (Templated) - Add new device and records using templates
- Add Device (Custom) - Add new device and/or records without templates
- Edit Existing Template - Edit a Template
- Edit Existing Device - Edit device and/or records
- Delete Existing Device - Delete a device and/or records
- Clone Existing Device - Clone a device
- View Existing Device - Displays a device in EPICS format
- Find Existing Device - Query for a device

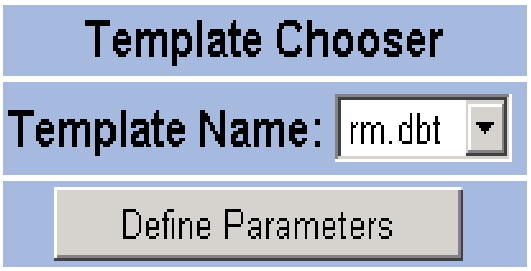
Clicking the  will bring page with help information on that topic.

Add Device (Templated) (1) …

By selecting this function one can add a device and it's records to the database using a predefined template. One is presented with a device building matrix:



It is required to make a selection or fill all the fields except 'Description'.
Click on 'Build Device >>' leads to choosing a template:



… Add Device (Templated) (2)

Select a template from the pull-down list and click on 'Define Parameters':

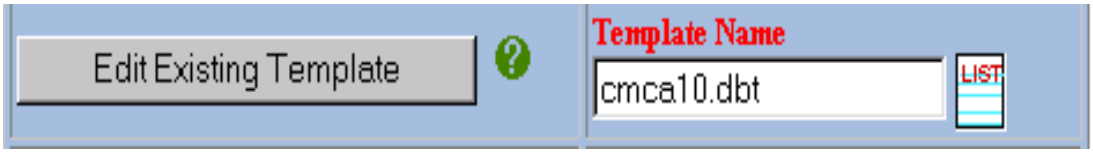
Define Parameters	
Parameter Name	User Value
$(phas)	0
$(rt)	0
$(bwc0)	1
$(det)	CALC
$(loc)	TEST
$(scan)	1 second
$(chan)	0
$(bmask)	0xf

Build Device

Add value for each parameter !
Once 'Build Device' is clicked, the device is build and stored in the database.

Edit Existing Template … (1)

By selecting this function one can edit the subsequent records of a template, or records of a custom (i.e. non-templated) device.
Enter the template name or click the list icon 

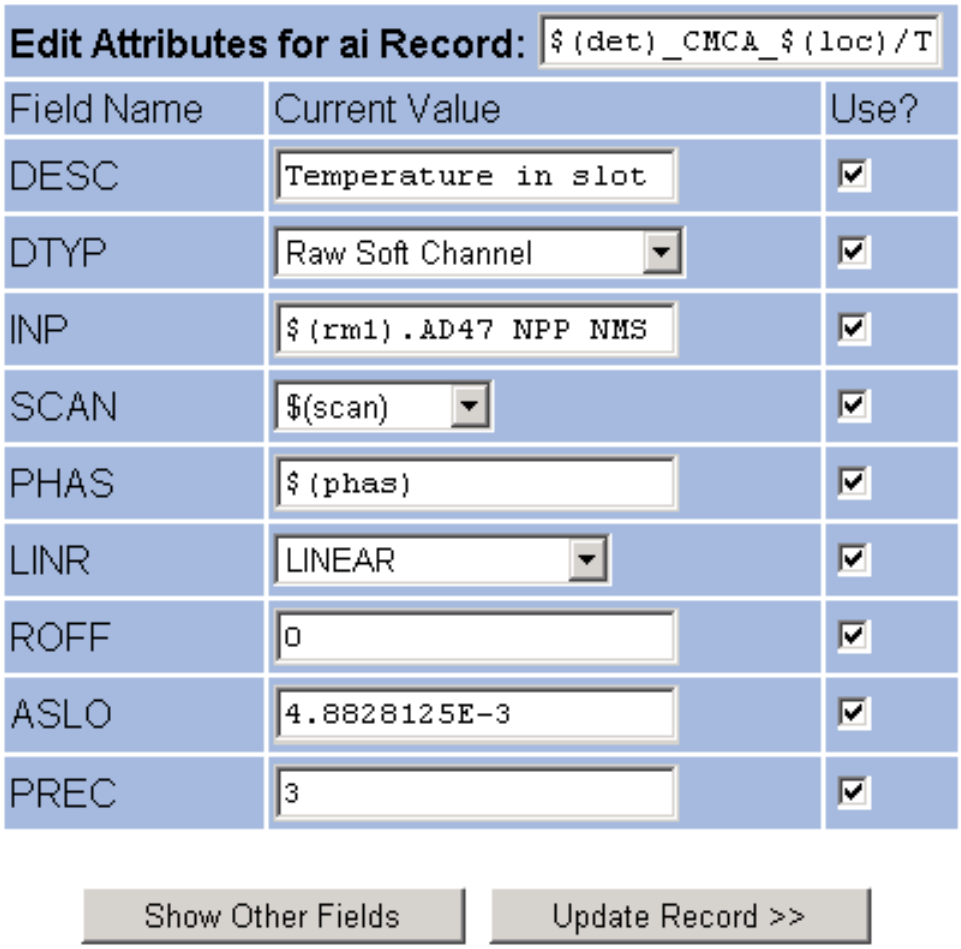


After clicking "Edit Existing Template", one is presented a list of possible records to edit:

Edit ai Record	$(det)_CMCA_$(loc)/TE00
Edit ai Record	$(det)_CMCA_$(loc)/TE01
Edit ai Record	$(det)_CMCA_$(loc)/TE02
Edit ai Record	$(det)_CMCA_$(loc)/TE03
Edit ai Record	$(det)_CMCA_$(loc)/TE04
Edit ai Record	$(det)_CMCA_$(loc)/TE05
Edit ai Record	$(det)_CMCA_$(loc)/TE06
Edit ai Record	$(det)_CMCA_$(loc)/TE07
Edit ai Record	$(det)_CMCA_$(loc)/TE08
Edit ai Record	$(det)_CMCA_$(loc)/TE09
Edit ai Record	$(det)_CMCA_$(loc)/TE10
Edit ai Record	$(det)_CMCA_$(loc)/TE11
Edit ai Record	$(det)_CMCA_$(loc)/TE12
Edit ai Record	$(det)_CMCA_$(loc)/TE13

… Edit Existing Template … (2)

Update field values of the record where needed:



The "Show Other Fields" button allows one to add more fields to the record if needed. Selecting this button provides a comprehensive list of all available fields not in the current record:

… Edit Existing Template (3)

Other Fields			
ACKS	Alarm Acknowledge Severity		☐
ACKT	Alarm Acknowledge Transient		☐
ADEL	Archive Deadband (DOUBLE)		☐
ALST	Last Archiver Monitor Trigger Value (DOUBLE)		☐
ASG	Access Security Group (STRING)		☐
DISA	Scan Disable Input Link Value (SHORT)		☐
DISP	Disable putFields (UCHAR)		☐

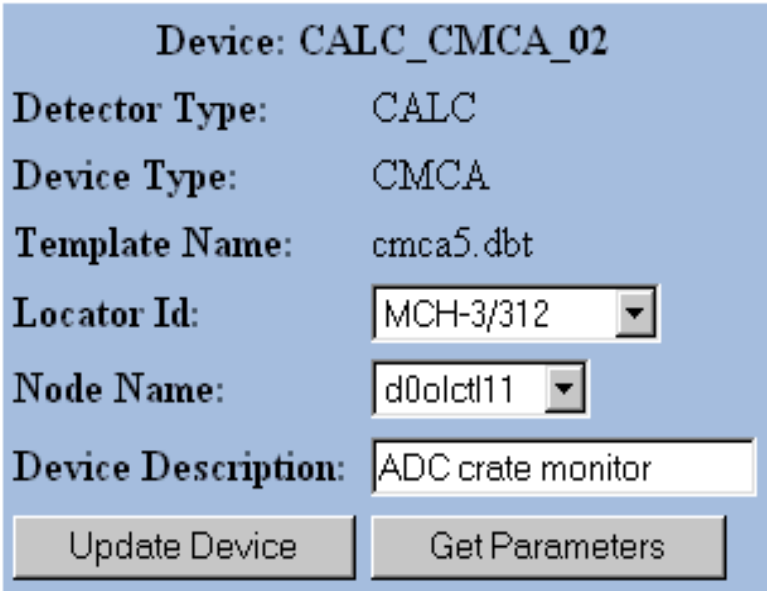
TPRO	Trace Processing if <> 0 (UCHAR)		☐
TSE	Time Stamp Event (SHORT)		☐
TSEL	Time Stamp Event Link (INLINK)		☐
UDF	Set if Record Value VAL Undefined (UCHAR)		☐
VAL	Engineering Value (DOUBLE)		☐

Update Record >>

Edit Existing Device … (1)

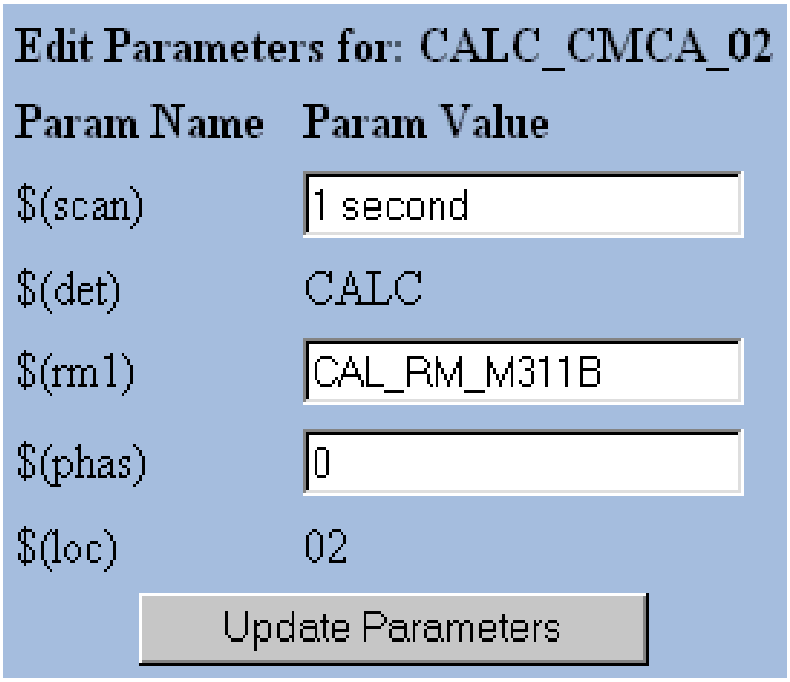
By selecting this function one can edit a device and/or substitution parameters in a template for that device. Enter the template name or click the list icon :

Editing a device:



Click "Get Parameters" to get a list of them. Update values of the parameters where needed:

… Edit Existing Device (2)



View Existing Device

By selecting this function one can view and/or print a device in it's fully expanded EPICS format.

Enter the template name or click the list icon 

:


```
Device Name: CALC_RM_PC17A
Detector Type: CALC
Device Type: RM
Locator Id: PC/17/A
Node Name: dOolctl11
Description: Rack Monitor

record(rm,"CALC_RM_PC17A")
  {
  field(DESC, "Rack monitor on CALC_PC17A")
  field(DTYP, "Rack Monitor")
  field(INP, "#@C2 R14")
  field(SCAN, "1 second")
  field(PHAS, "0")
  field(FLNK, "CTL_RMI_PC17/STAT")
  field(BWCO, "24")
  field(BMASK, "0x9")
  }
```