ThomX presentation : IHM_health_check, an interface to check state of ThomX

Alexandre Moutardier

Université Paris-Saclay, CNRS/IN2P3, IJCLab, 91405 Orsay, France.

March 26th, 2021

Goal

Create an interface to synthesise the state of all ThomX devices.

- Code in git : https://gitlab.in2p3.fr/ThomX/panneaux/-/tree/master/Tools
  - health_check.py : to check the system state
  - IHM_health_check.py : to launch the interface
  - All_Elements.yml (+ files call in it) : configuration files
- Documentation : https://gitlab.in2p3.fr/ThomX/panneaux/-/tree/master/Tools/doc_health_check/doc_health_check_config_file.txt

Structuration of ThomX as a Tree

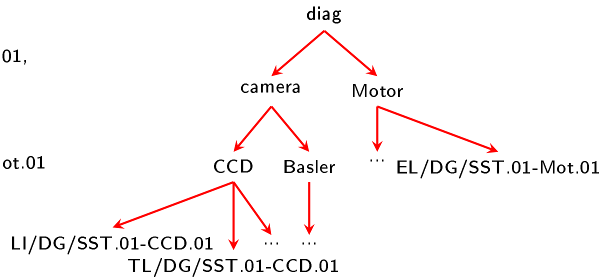
A structure of tree is use to represent ThomX. For instance, ThomX is decomposed into "diagnostics", "synchro", "vacuum", "laser" ..., them self decomposed as wanted.

Example with the diagnostics branch :

Structure of the diagnostics configuration file :
Elements_diag.yml

```
Camera:
  CCD:
   ds_name: LI/DG/SST.01-CCD.01,
TL/DG/SST.01-CCD.01,...
  Basler:
   ...
Motor:
  ds_name: ..., EL/DG/SST.01-Mot.01
...
```

The associate tree

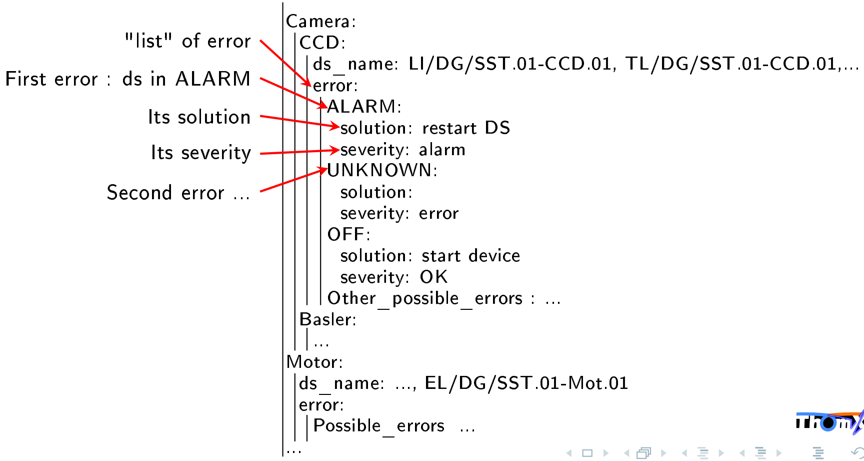


Real configuration file

The ThomX tree - or configuration file - contain all devices to be check and some other information according to the following structure :

- each node is only a name
- each leaf is :
  - A list of device name
  - A list of possible errors, with an associate severity and a possible solution

For instance, **Elements_diag.yml** is more like :



```
Camera:
  CCD:
    ds_name: LI/DG/SST.01-CCD.01, TL/DG/SST.01-CCD.01,...
    error:
      ALARM:
        solution: restart DS
        severity: alarm
      UNKNOWN:
        solution:
        severity: error
      OFF:
        solution: start device
        severity: OK
      Other_possible_errors : ...
  Basler:
    ...
Motor:
  ds_name: ..., EL/DG/SST.01-Mot.01
  error:
    Possible_errors ...
...
```

Generation of the structure of the configuration file(1/2)

Documentation : https://gitlab.in2p3.fr/ThomX/panneaux/-/tree/master/Tools/doc_health_check/doc_health_check_config_file.txt
Example of configuration files : https://gitlab.in2p3.fr/ThomX/panneaux/-/tree/master/Tools/doc_health_check
To create a configuration file, the easiest is to :

- List all device that are needed to be check
- Sort them in several "group" composed of similar devices
- Sort each group in one or several huge-group
- Repeat the last steep since you have only one root-group
  The name of the configuration file should be : "Elements_name_of_root-group.yml".
  "name_of_root-group" will become use as root name in the loading of the file.
- Write the structure defined above in the configuration file as follow :

```
root-group:
||first-sub-root-group:
||||...
||...||group1:
||...||||ds_name: ds_name_1,ds_name_2,...
||...||group2
...
||second-sub-root-group: ...
...
```

With "|" to visualized blank characters of the indentation.
Each new depth level has 2 more blank characters than the previous one.

Generation of the structure of the configuration file (2/2)

⚠ Be careful :

- You cannot have a "ds_name" and a sub-group in the same group, else the sub-group is ignored !!!
- A group name should be ONE word (example : "diag", "Li-TL-EL", "MyCamera1_10"...). Avoid any other character than -, _, **letters**, or **numbers**.
- There is **NO space BEFOR** ":" BUT there IS **ONE space AFTER** !

If the configuration file is **to large**, one can split it in several sub-configuration file.
To load a sub-configuration file in a file, add at the beginning of the file :
"**import: path/file_name, path2/file_name2, ...**" (see example below)

- path/ : the path from folder of configuration file to sub-configuration file (may be obliterate if both in the same file)
- file_name : Name of the file to load
  file_name separator is ","

  ⚠ Use only **1** import occurrence by file

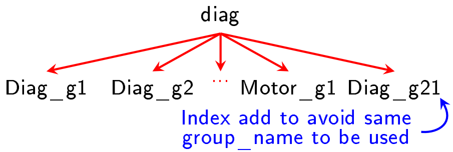
Configuration file : Element_diag.yml

```
import: Element_motor.yml
Diag_g1: ...
Diag_g2: ...
...
```

Sub-configuration file : Element_motor.yml

```
Motor_g1: ...
Diag_g2: ...
...
```

The tree structure :



Completion of the configuration file

The next steep is to add the error listing.
After each "ds_name", copy/past the error structure of one of the example.
The error structure is as follow, with "|" for showing the indentation and first indentation at the same level than the "ds_name" :

```
error:
||Error_name:
||||solution: you may write what you want (avoid accentuation and special symbol)
||||severity: one between : "OK" "alarm" "time_out" "error" (in order of severity)
```

- Common error name :
  - **DS_not_exist**: error raised if at least one of the server's name does not exist
  - **server_unknow_state**: error raised if a server's state is unknown
  - **server_error**: error raised if a server is in error state
  - **server_off**: error raised if a server is off
  - **DS_failed**: error raised if the server is on, and the DS does not respond
  - **ERROR**: error raised if a ds is in error state
  - **ALARM**: error raised if a ds is in alarm state
  - **UNKNOWN**: error raised if a ds is in unknown state
  - **TIME_OUT**: error raised if a ds does not respond after 10s
  - **INIT**: some ds have initialisation state that may need a "start" command to be use
  - **OFF**: error raised if a ds is OFF

If a device as an other usual state (name "MyDsError") it can be added in the same way as "OFF" for example. Be careful, it's **case sensitive** !
Let me know if some servers states must be added or if you thought of other common state. "MOVING" state (for motors) won't raise an error. (can be change if needed)

Test of the configuration file

To test your new file, execute in a terminal:

- cd /data/sharded/Interface/panneaux/Tools
- ipython health_check.py path/to/the/file/name_of_file.yml
    - should return a "general state" preceded by a representation of the tree (with states and severity)

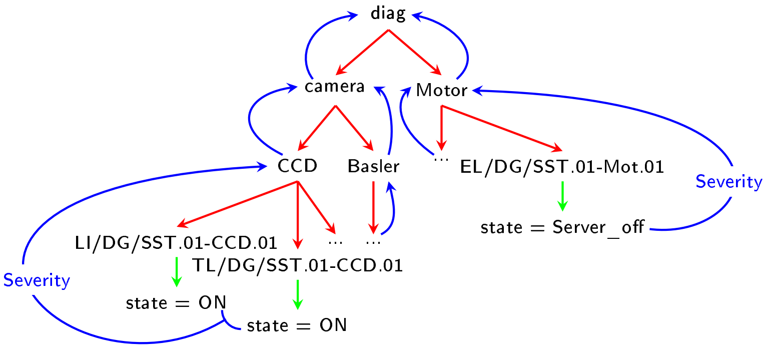
Example of output :

```
Diag
-Motor
–Li/DG/SST.01-MOT.01 : ALARM
— Solution : restart DS
–TL/DG/SST.01-MOT.01 : OK
–TL/DG/SST.02-MOT.01 : server_off
— Solution : start server
–Severity : alarm
-Severity : alarm
Severity : alarm

general state is :alarm
```

Principle of the computation

- Load a tree
- For each leaf :
  - Read the state of all the devices (or associate server if the device cannot answer)
  - Found the associate severity of the state (ever "OK" for "ON" state, or associate severity for an error state)
  - Concatenate all severity to obtain the severity of the leaf
- For each node : the associate severity is the concatenation of all children's severities



Reading of the devices' state

Process :

```
  try : state = read state of the device
    if the reading has failed, check of the state of the server
        state = "ds_failed"
        try : communicate with the server
        if the communication failed, there is a server error (it exist but loading is impossible)
            state= "server_error"
        else:
            if the ds name is uncorrecte
                state = "DS_not_exist"
            else:
                check of the server state
                if server state is "None":
                    state = "server_off"
               elif server state is "ON":
                 keep : state = "ds_failed"
                else:
                    state = "server_unknow_state"
```

Presentation of the interface

A window is associated to each node and leaf.
This window is composed of :

- The name of the previous node (ThomX for the root)
- The time
- A button to reopen the previous window (Does not exist on the root window)
- The date and time of the last update
- A button for update
  - Green button = automatic process launch
  - Red button = error in the automatic process (click on it to update once more)
- For leaf :
  - one line for each device in the list with :
    - A LED of the device state
    - The name of the device
    - The "state" of the device (add after screenshot...)
    - The solution for the error (if state is in error)
- For node :
  - A button for each child of the node with :
    - The name of the child node
    - The color depending of the severity associate to the node (see color description)

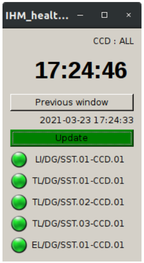


Figure 1: Example of leaf window

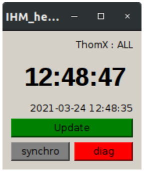


Figure 2: Example of the root window

Presentation of IHM_health_check

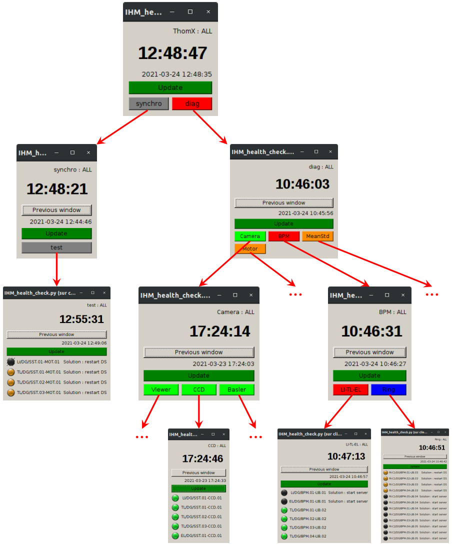


Figure 3: Representation of the tree structure of IHM_health_check

Color use base on Taurus color code : https://taurus.readthedocs.io/en/develop/devel/color_guide.html.
In order of growing gravity :

- OK : RGB(0,255,0)
- alarm : RGB(255,140,0)
- time_out : RGB(0,0,255)
- error : RGB(255,0,0)
- UNKNOWN : RGB(128,128,128)
- Uncoloured button (like "previous window") mean "no information"

Be careful, the color of a button may not correspond to the color of the device LED. For example, "test" button is grey because I have defined (only for the test) the error state to raise an UNKNOWN severity !

Severity concatenation

The concatenation of severity mean taken the worse severity possible in a list of gravity.
Possible gravity are (from worse to better) :

- UNKNOWN : raise if an unknown gravity is found
- error : worse known severity, some thing does is in a very bad state
- time_out : gravity usually associate to time out error
- alarm : "small" gravity, usually raise when a device is in a "alarm" state (for example a motor in switch limit)
- OK : no error (imposed for device state = "ON" or = "MOVING" for motors)

For example :

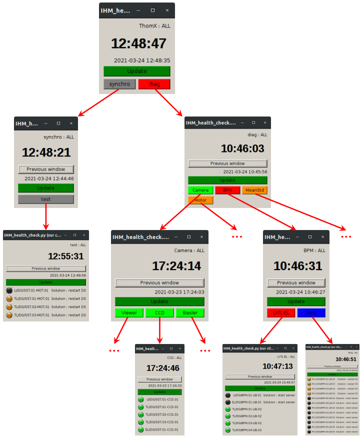
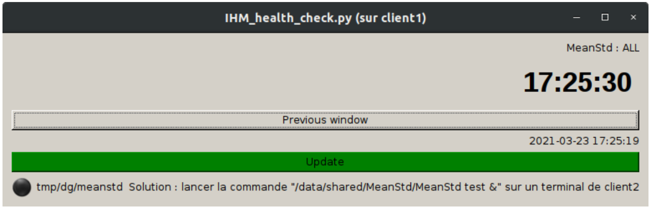


Figure 4: Representation of the tree structure of IHM_health_check

Examples of window

If one click on "meanStd" on the window "diag", the window below will open.
In this leaf of ThomX tree there is only one device "tmp/dg/meanStd". This device is "OFF" (LED black). A possible solution to resolve this is to follow the "Solution" presented on the window.



● tmp/dg/meanstd Solution : lancer la commande "/data/shared/MeanStd/MeanStd test &" sur un terminal de client2

Figure 5: Window of IHM_health_check for mean_std device.

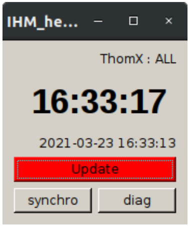


Figure 6: Window of IHM_health_check for ThomX : issue of loading.

On the window to the right, there is 2 "issues" :

- "update" button is red : this mean that the automatic update has crash at some point
- "synchro" and "diag" buttons are uncoloured : this mean that no severity has bean load for those node in the tree

A click on "update" should solved it as it will relaunch the automatic update of the window, and eventually read the share file containing severity.

Share file for the interface

Each interface share a common file : "/data/shared/log_health_check/tmp_health_check.yml".
This file is compose of :

- All the ThomX structure with reduction of errors information as much as possible
- last_update : the date of the last update
- is_running : a boolean to know if a process to update this file is running

Updating this file - hence checking all device - may take some time (very quick if all devices are "ON", may be long if some server are "time_out") and informatics resource.

For the actual diagnostics devices (40-50 devices), about 1 second if every thing is "ON" and less than 1 minute if several error occurs.

The goal is to update the shared file often enough to have follow modification, but not to often to avoid consuming to much informatics resources.
At the moment there is 10s between each new file update.

Idea of upgrade :
Save some share file (for example each hour) to keep information of ThomX health along the day.
A process to remove old save file (may be 2 days old) may be added to save informatics resources.

Update of the interface

To avoid several checking to be execute separately in each window, the interface work as follow :

- When a window is open or updated it will try to read the shared file
  - If this file exist, the interface load it
    - If is_running = True, the interface use it to defined itself (structure, button color...) (usual process)
    - Else : It launch a process that will update automatically the shared file (and impose is_running = True)
      This interface will have "(main)" added on the update button
  - Else : It launch a process that will create the shared file and launch the automatic update
    This interface will have "(main)" added on the update button
- The window launch a new update of itself some time later (3s later at the moment)

All automatic process launch by a window is kill when the window is closed.

This method must assure :

- to have only 1 process to execute the checking at once (as setting of is_running = True should be quick enough)
- to check that there is one and only one process of checking thanks to the "(main)" text on window with process running
- to open interface in several computer
- to have only one process that read the configuration file (the update of the shared file)
- to have a new update processing launch if the last one is kill

Conclusion

A method to give the healthiness of ThomX has been developed.

This method is used in an interface to give a visualization clear and simple.

Thing to do :

- Completed the configuration file with all equipment of ThomX. (Need help of those in charge of sub-systems)
- Some "segmentation fault" occur some time. Need to be investigated.
- Add the interface in the "PlateformeIHMThomX".

Thing to discuss :

- Add a save of some "tmp_health_check.yml" ?
- Other useful parameter for devices ? For example :
  - Add the name of the error ? (very useful if the error come from the server)
  - Add the Status of the device ? (more complex and may be useless)

Thanks