

KAMELEON – A BEHAVIOR-RICH, NON-MEMORYLESS AND TIME-AWARE GENERIC SIMULATOR

R. Fernandes[†], European Spallation Source, Lund, Sweden

N. Senaud, Commissariat à l'Energie Atomique et aux Energies Alternatives, Gif-sur-Yvette, France

Abstract

At the European Spallation Source (ESS), thousands of devices will be used to control both the machine and end-station instruments. To enable ongoing development when access to these devices is not possible (for whatever the reason), a simulator named Kameleon was implemented. The present paper describes this simulator, illustrates how it works through a practical example, and finally presents possible developments to further improve it.

INTRODUCTION

The ESS is currently in construction phase and it is expected to enter in operation in 2019. Consequently, the development of controls is ramping up sharply despite developers not having an easy access to devices most of the time (thus putting an additional burden on people). The reasons for this constrain are due to some of these devices being under construction, while others being purchased or not physically located at the ESS but at its in-kind partners (e.g., the Commissariat à l'Energie Atomique et aux Energies Alternatives – CEA). To overcome this situation, a simple yet flexible and powerful simulator was needed to secure a smooth development. With these goals in mind, Kameleon [1] is being developed at the ESS in the recent years. It is a behavior-rich, non-memoryless and time-aware generic simulator that handles clients through a TCP/IP connection. An instance of this client is an EPICS IOC or a Tango Device Server.

DESCRIPTION

Kameleon is a tool implemented in the Python programming language so that its development effort is kept at minimum but also to ease the development of simulators (that are used by Kameleon) by people not necessarily accustomed to software development (e.g., integrators). These simulators – also known as .kam files – model devices by describing the commands and statuses that they receive and send respectively. At the time of writing several .kam files are already available that may be used with Kameleon to simulate disparate devices such as power supplies, electronic boards, oscilloscopes and temperature controllers. Table 1 summarizes these.

The main features that originally characterize this tool are the following:

- Ubiquitous: multiple platforms such as Windows, Linux and Mac OS X are supported.
- Behavior-rich: predefined behaviors as well as user-defined are available.

- Non-memoryless: the state of the simulator can be preserved between events and/or elapsed time so that, e.g., a finite-state machine can be implemented.
- Time-aware: a status can be sent to the client either event-based – whenever a command is received – or time-based – after a certain elapsed time.
- Flexible: commands and statuses are described in a simple user-defined file – nothing is hard-coded in Kameleon.

Table 1: Devices Currently Simulated by Kameleon

Device
AK-NORD XT-Pico-SXL (electronic board)
FUG HCH 15K-100K (power supply)
FUG HCP 35-3500 (power supply)
Geiger Counter (electronic board)
Hameg HMO3034 (oscilloscope)
Julabo F25-HL (refrigerated/heating circulator)
Lake Shore 336 (temperature controller)
TDK Lambda Genesys 10-500 (power supply)

Workflow

Kameleon is based on a classic client-server model [2] where it essentially awaits for incoming clients (through a TCP/IP connection) and serves their requests by receiving commands and sending statuses. Fig. 1 depicts this model as well as the simulation workflow in Kameleon.

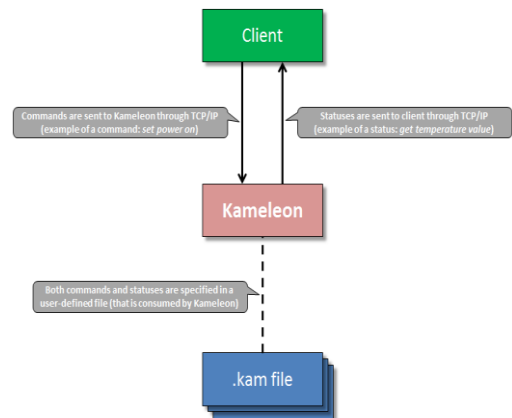


Figure 1: Simulation Workflow in Kameleon.

.kam file

A .kam file is technically a Python module (or .py file) that is defined by the person interested in having Kameleon simulating a certain device. Kameleon is able to read this type of file and evaluate its content. This approach, of evaluating the content, can be seen as an effective way to

[†] ricardo.fernandes@ess.se

extend Kameleon's capability to simulate the device described by a .kam file.

Typically, a .kam file contains two Python lists named `COMMANDS` and `STATUSES` (both recognized by Kameleon) that model commands and statuses respectively. Their generic forms – and detailed explanations – are:

```
COMMANDS = [[Description1, Command1, Status1,
Wait1], ..., [DescriptionX, CommandX, StatusX,
WaitX]]
```

- **Description:** string that describes the command (e.g., “set power on”).
- **Command:** string that represents the command (e.g., “AC1”). Only data (received from the client) that matches exactly the command is selected. Additional matching policies are available:
 - If command starts with `***`, any data that ends with command is selected.
 - If command ends with `***`, any data that starts with command is selected.
 - If command starts and ends with `***`, any data that contains the command is selected.
- **Status:** integer or list that specifies the index(es) of the status(es) (stored in the `STATUSES` list) to send to the client after the command is selected. If 0 or not specified, no status is sent.
- **Wait:** integer that specifies the time to wait (in milliseconds) before sending the status to the client. If 0 or not specified, the status is immediately sent (i.e., right after the command is received).

```
STATUSES = [[Description1, Behavior1, Value1,
Prefix1, Suffix1, Timeout1], ..., [DescriptionX,
BehaviorX, ValueX, PrefixX, SuffixX, TimeoutX]]
```

- **Description:** string that describes the status (e.g., “get temperature value”).
- **Behavior:** integer that specifies the behavior for generating the status. It can either be:
 - **FIXED:** sends a fixed value to the client.
 - **ENUM:** sends a value – belonging to an enumeration – to the client.
 - **INCR:** sends an incremented value to the client.
 - **RANDOM:** sends a random value to the client.
 - **CUSTOM:** sends a value from a user-custom function to the client.
- **Value:** value to send to the client. Depending on the behavior, it can either be an integer, float, string or list:
 - When **FIXED**, the value is expected to be an integer, float or string. Independently of how many times it is sent to the client, the value remains the same (i.e., does not change).
 - When **ENUM**, the value is expected to be a list. It represents a set of elements (enumeration). After sending an element of the list to the client, the next value to be sent is the next element in

the list. When the last element is sent, the next to be sent is the first element of the list.

- When **INCR**, the value is expected to be an integer, float or list. If an integer or float, the first value to be sent is a 0 and subsequent values to be sent are incremented by value. If a list, the lower bound, upper bound and increment values are defined by the first, second and third elements of the list, respectively.
- When **RANDOM**, the value is expected to be an integer or a list. If an integer, a random number between 0 and value is generated. If a list, the lower and upper bounds of the random number to generate are defined by the first and second elements of the list, respectively. The generated random number is sent to the client.
- When **CUSTOM**, the value is expected to be a string. It contains the name of a user-defined Python function to be called by Kameleon. The value returned by this function is sent to the client.
- **Prefix:** string that contains the prefix to insert at the beginning of the value to send to the client. If not specified, nothing is inserted.
- **Suffix:** string that contains the suffix to insert at the end of the value to send to the client. If not specified, nothing is inserted.
- **Timeout:** integer that specifies the time-out (in milliseconds) after which the status is sent to the client (i.e., time-based). If 0 or not specified, the status is only sent after receiving a command from the client (i.e., event-based).

Usage

Kameleon is primarily meant to be launched from a terminal where it listens for incoming clients through a TCP/IP connection on port 9999 – this can be configured through a parameter though – to serve their requests. When no .kam file is specified (upon launching it), Kameleon simply displays the commands received from the client and does not react to these (useful for connection testing purposes). By specifying a .kam file, Kameleon is able to simulate the device modelled by this file by recognizing commands and eventually reacting to these by sending statuses to the client.

To illustrate how Kameleon works, the following Python code (stored in a .kam file) defines some commands and statuses, as well as a user-defined function:

```
COMMANDS = [{"Set Power On", "AC1"},
{"Set Power Off", "AC0"},
{"Get Power", "AC?", 1},
{"Get FIXED", "FIXED?", 2},
{"Get ENUM", "ENUM?", 3},
{"Get INCR", "INCR?", 4},
{"Get RANDOM", "RANDOM?", 5},
{"Get CUSTOM", "CUSTOM?", 6}]

STATUSES = [{"Get Power", ENUM, [1, 0], "AC?"},
{"Get FIXED", FIXED, 18.3},
{"Get ENUM", ENUM, [10, 20, 30]},
```

```
["Get INCR", INCR, 2],
["Get RANDOM", RANDOM, 50],
["Get CUSTOM", CUSTOM, "func()"]]
```

```
i = 0
```

```
def func():
    global i
    i = i + 1
    return math.sin(0.25 * i) * 100
```

Assuming that an IOC (a typical client) is running and Kameleon is launched with the .kam file storing the previous Python code, Kameleon accepts a connection from the IOC, and starts receiving commands and sending statuses from/to it. Fig. 2 and Fig. 3 depict Kameleon serving the IOC and an OPI displaying PVs (belonging to the IOC) that are fed by Kameleon, respectively.

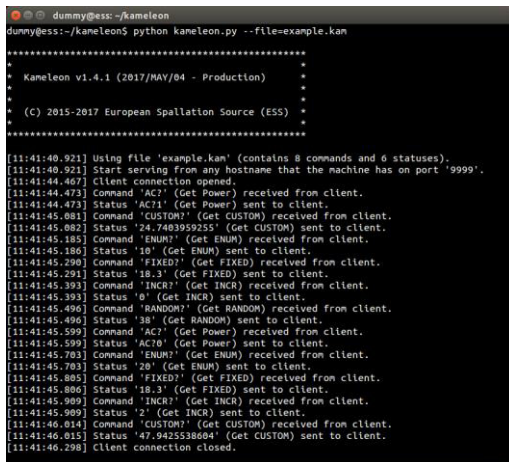


Figure 2: Kameleon serving an IOC.

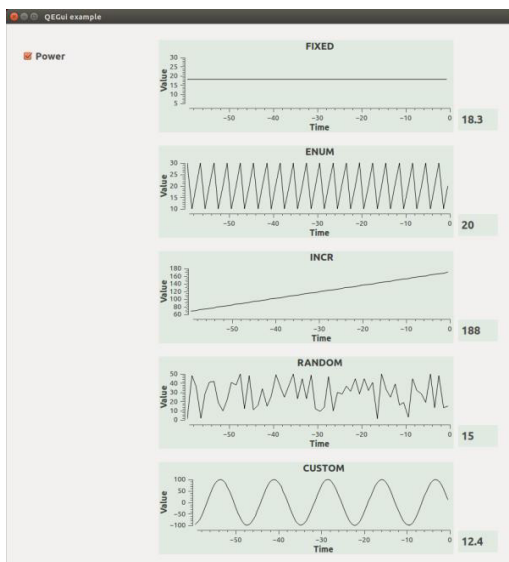


Figure 3: OPI displaying PVs fed by Kameleon.

FUTURE DEVELOPMENTS

As the development of controls intensifies at the ESS, new functionalities are expected to be requested and Kameleon will be developed accordingly. To anticipate

some of these requests, the following missing functionalities were identified and are candidates for development in the near future:

- Currently, Kameleon only supports serial-based devices. Although this type of devices covers a wide spectrum of the controls, register-based devices (e.g., MicroTCA AMC CPU) are equally important and will be supported as well.
- The predefined behavior RANDOM will be extended with additional random distributions, namely Normal (Gaussian), Uniform and Poisson.
- A new predefined behavior for the Channel Access [3] protocol will be added to ease the development of .kam files when these are meant to be used in EPICS.
- In addition to the devices enumerated in Table 1, many others – Leybold TD20, CAEN SY4527, MKS 946, Sorensen SGA 30X501D, to name a few – will have .kam files developed for them.
- While it is possible to run several instances of Kameleon at the same time using different ports, this tool will be updated to cope with more than one client (i.e., TCP/IP connection) at the time.

CONCLUSION

Many projects are using Kameleon where it has proven to be valuable. For instance, a team at the CEA uses it to develop IOCs and test these through continuous integration, while another one uses it to develop a gateway to have EPICS communicate with PLCs. SINE2020 is another project that uses Kameleon to develop a new protocol named Sample Environment Communication Protocol (SECoP) as it does not depend on an accurate model of a sample environment (i.e., real hardware).

At the ESS, Kameleon is instrumental for simulation of devices to successfully enable a myriad of scenarios such as development of EPICS devices support, IOCs, OPI screens, testing of IOCs and alarm workflows.

In addition to new functionalities (e.g., register-based devices support, predefined behavior for the Channel Access protocol), more .kam files are expected to be developed in the future. These will improve Kameleon and allow people to have access to an extensive collection of simulators that may be used with this tool out-of-the-box.

ACKNOWLEDGEMENT

The authors would like to thank all the people that contributed with ideas and participated in discussions to further improve Kameleon, in particular Han Lee at the ESS.

REFERENCES

- [1] Kameleon, <https://bitbucket.org/europeanspallationsource/kameleon>
- [2] Client-server model, https://en.wikipedia.org/wiki/Client%E2%80%93server_model
- [3] Channel Access, <http://www.aps.anl.gov/epics/docs/ca.php>