TASKOMAT & TASKOLIB: A VERSATILE, PROGRAMMABLE SEQUENCER FOR PROCESS AUTOMATION

L. Fröhlich*, O. Hensler, U. Jastrow, M. Walla, J. Wilgen,
Deutsches Elektronen-Synchrotron DESY, Germany

Abstract

This contribution introduces the TASKOLIB library, a powerful framework for automating processes. Users can easily assemble sequences out of process steps, execute these sequences, and follow their progress. Individual steps are fully programmable in the lightweight Lua language. If desired, sequences can be enhanced with flow control via well-known constructs such as IF, WHILE, or TRY. The library is written in platform-independent C++ 17 and carries no dependency on any specific control system or communication framework. Instead, such dependencies are injected by client code; as an example, the integration with a DOOCS server and a graphical user interface is demonstrated.

INTRODUCTION

Like other scientific and industrial facilities, particle accelerators consist of many subsystems that need to be coordinated. The operation of any such complex system is inevitably governed by *processes* of various kinds. These processes can be implicit ("operators typically do it like this"), explicit (checklists and step-by-step instructions), or automated (implemented in hard- or software). Automated processes have several advantages over their manual counterparts:

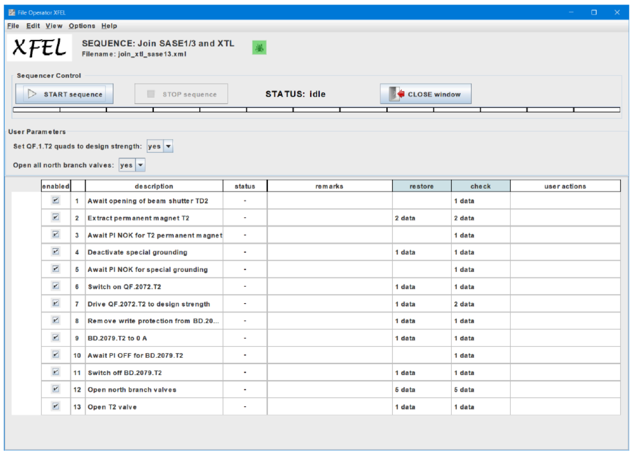
- Faster execution
- Better reproducibility
- Reduced work load for operators
- Improved documentation of what happened when

The particle accelerator community has traditionally been in a good position to profit from automation in software because of its early adoption of distributed control systems. Tools to execute process steps automatically, so-called *sequencers*, have a long history [1–7]. Even today, this field keeps spawning new developments due to changing requirements and user expectations [8–10].

At DESY, the main tool for the automation of processes for more than a decade has been the aptly named *Sequencer/File Operator* [11]. It can execute linear sequences of steps with limited support for branches and jumps between the steps. Each step can write values to the control system or read them until certain conditions are fulfilled. Apart from this, the tool also provides save&restore functionalities for control system properties.

The *Sequencer* consists of a Java client with a graphical user interface (Fig. 1) and a Java server component that mainly provides central data storage and version control. Although the tool has proven its usefulness across practi-

* lars.froehlich@desy.de



SEQUENCE: Join SASE1/3 and XTL
Filename: join_xtl_sase13.xml

STATUS: idle

Set QF.1.T2 quads to design strength: yes

Open all north branch valves: yes

enabled		description	status	remarks	restore	check	user actions
✓	1	Await opening of beam shutter TD2	-			1 data	
✓	2	Extract permanent magnet T2	-		2 data	2 data	
✓	3	Await PI NOK for T2 permanent magnet	-			1 data	
✓	4	Deactivate special grounding	-		1 data	1 data	
✓	5	Await PI NOK for special grounding	-			1 data	
✓	6	Switch on QF.2072.T2	-		1 data	1 data	
✓	7	Drive QF.2072.T2 to design strength	-		1 data	2 data	
✓	8	Remove write protection from BD.20...	-		1 data	1 data	
✓	9	BD.2079.T2 to 0 A	-		1 data	1 data	
✓	10	Await PI OFF for BD.2079.T2	-			1 data	
✓	11	Switch off BD.2079.T2	-		1 data	1 data	
✓	12	Open north branch valves	-		5 data	5 data	
✓	13	Open T2 valve	-		1 data	1 data	

Figure 1: A procedure in the legacy DESY sequencer.

cally all of DESY's accelerator facilities, several flaws have become apparent over the years:

- The sequences are written as XML files with a poorly documented set of tags and attributes. Users typically have to interact with a Subversion repository to edit these files. This presents a high entry bar for creating or maintaining sequences, so that this task is left to very few experts and a lot of potential for automation is wasted.
- The kinds of sequences that can be expressed are limited: Control flow can only be directed through conditional *goto* statements, and there is no support for variables or arithmetic or string operations.
- The client-server communication is based on the TINE protocol [12] which has reached end-of-life and is slowly phased out.
- Sequences are executed on the client and require the graphical user interface. This prohibits their use from other components in the control system.

In combination, these shortcomings would be hard to overcome with incremental improvements to the existing code. Therefore, we have decided to develop a new sequencer from scratch.

TASKOMAT: DESIGN

The new TASKOMAT sequencer models processes as sequences of individual steps like its predecessor. In contrast to it, it is designed around the following goals:

- **Integration** into the control system: Sequences can be started, controlled, and manipulated from the control system.

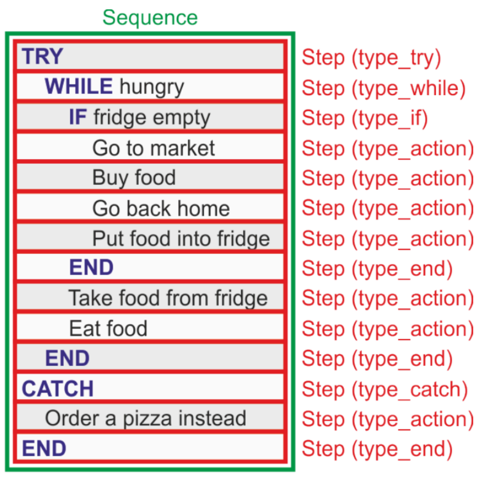


Figure 2: A process is modeled as a sequence of steps.

- **Control flow**: Sequences can contain well-known control flow constructs such as *if* or *while*, but no *goto*.
- **Programmability**: There is no inherent limit to the complexity of the procedures that can be expressed. Where necessary, users are able to program the desired functionality.
- **Ease of use**: The barriers for users to create, edit, and test sequences are as low as possible.
- **Separation** of control system dependent and independent code: As much of the code as possible has no dependency on the control system.

Figure 2 illustrates how a process is modeled as a sequence of steps: Typical steps just perform an *action*, but control flow steps like *if* or *while* can be used to express conditional execution or loops. Table 1 summarizes the available step types. In this high-level view, only the overarching control flow and user-defined descriptions of the individual steps are visible, providing a clear and unobstructed outline of the sequence.

What each individual step actually does is defined by a script written in the lightweight Lua extension language [13, 14]. Unlike most general-purpose programming languages, Lua has a clear and simple syntax that is easy to handle even for occasional users:

```
addr = "SOME/CONTROL/SYSTEM/ADDRESS"
for i = 1, 10 do
    print("Writing", i, "to", addr)
    dset(addr, i)
    sleep(0.5)
end
```

Here, TASKOMAT provides implementations of the functions *dset* (set a property in the control system) and *sleep*. Conditions for *if*, *elseif*, or *while* steps are also written in Lua. In these cases, the return value of the script determines if the corresponding branch is taken:

```
current = dget("ADDR/OF/A/CURRENT_MONITOR")
return current < 0.05
```

Table 1: Supported Step Types

Type	Description
action	Execute script
if	Conditional execution of block
elseif	Conditional execution of block
else	Conditional execution of block
while	Conditional loop execution
try	Execute block while intercepting errors
catch	Handle errors intercepted by try
end	Block-ending step

Individual steps are isolated from each other, but variables can be passed between them through explicit ex- or import from/to a *context* that is shared by the entire sequence.

IMPLEMENTATION

The TASKOMAT is implemented in standard C++ 17. The language was chosen for its platform independence and because bindings for all relevant control systems and for the Lua interpreter are available.

Taskolib Library

All of the control system independent code is gathered in a library that provides

- classes for modelling sequences and steps,
- a set of custom commands for Lua (e.g. *sleep* or *print* with output redirection),
- support for executing sequences asynchronously and aborting them at any time,
- support for receiving status updates and output from running sequences,
- support for serialization and deserialization of sequences.

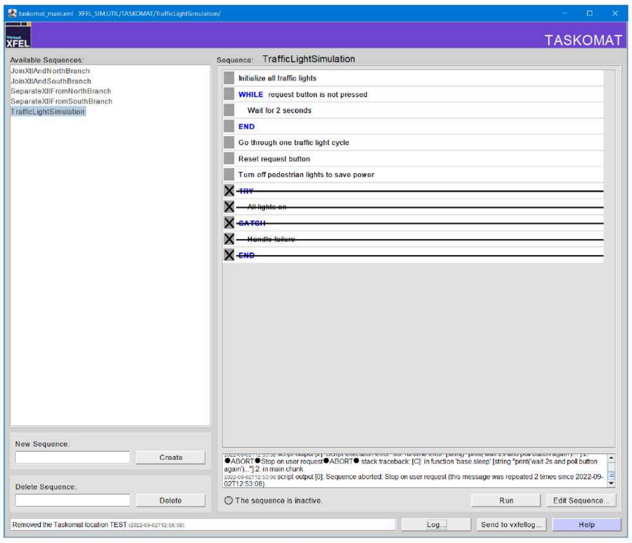
The library includes a copy of Lua 5.4.3 and of the associated Sol3 C++ wrapper [15, 16] in version 3.2.3. Its only external dependency is the GUL14 [17, 18] basis library.

Taskomat DOOCS Server

So far, we have implemented one server for the DOOCS control system [19–21] that makes use of the library. It provides a thin layer of DOOCS properties to allow the inspection, modification, and execution of sequences. It also injects the DOOCS-specific commands *dget* and *dset* into the Lua interpreter to facilitate control system access from within the scripts. Most of the basic data types can already be read and written, which should make the server useful for DOOCS environments such as the European XFEL or FLASH facilities.

Graphical User Interface

We have created a simple graphical user interface with the *jddd* [22, 23] user interface builder. Figures 3 and **??** show screenshots of the main panel and of the step editor. This user interface blends in well with the jddd-based operation



Available Sequences:
JoinXtlAndNorthBranch
JoinXtlAndSouthBranch
SeparateXtlFromNorthBranch
SeparateXtlFromSouthBranch
TrafficLightSimulation

Sequence: TrafficLightSimulation

```
Initialize all traffic lights
WHILE request button is not pressed
  Wait for 2 seconds
END
Go through one traffic light cycle
Reset request button
Turn off pedestrian lights to save power
~~TRY~~
  ~~All lights on~~
~~CATCH~~
  ~~Handle failure~~
~~END~~
```

New Sequence:

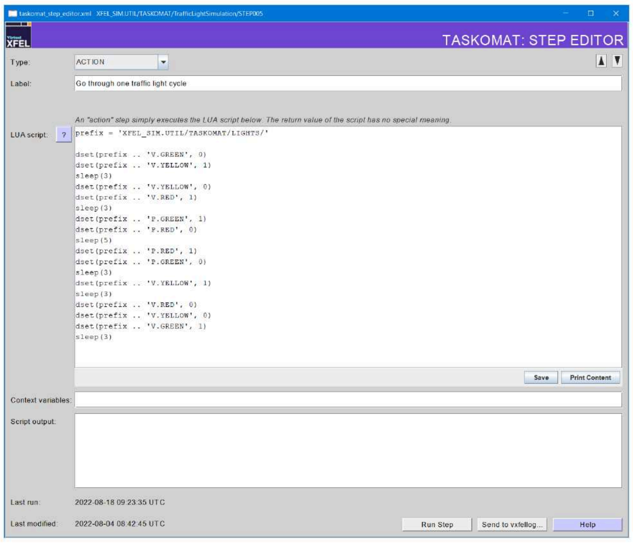
Delete Sequence:

```
●ABORT●Stop on user request●ABORT● stack traceback: [C]: in function 'base.sleep' [string "print('wait 2s and poll button again')..."]:2: in main chunk
2022-09-02T12:53:06 script output [0]: Sequence aborted: Stop on user request (this message was repeated 2 times since 2022-09-02T12:53:08)
```

The sequence is inactive

Removed the Taskomat location TEST (2022-09-02T12:56:56)

Figure 3: Main user interface for the TASKOMAT DOOCS server.


taskomat_step_editor.xml XFEL_SIM.UTIL/TASKOMAT/TrafficLightSimulation/STEP005
TASKOMAT: STEP EDITOR
?
Save
Print Content
Run Step
Send to vxfellog...
Help

Type: ACTION

Label: Go through one traffic light cycle

An "action" step simply executes the LUA script below. The return value of the script has no special meaning.

LUA script:

```
prefix = 'XFEL_SIM.UTIL/TASKOMAT/LIGHTS/'

dset(prefix .. 'V.GREEN', 0)
dset(prefix .. 'V.YELLOW', 1)
sleep(3)
dset(prefix .. 'V.YELLOW', 0)
dset(prefix .. 'V.RED', 1)
sleep(3)
dset(prefix .. 'P.GREEN', 1)
dset(prefix .. 'P.RED', 0)
sleep(5)
dset(prefix .. 'P.RED', 1)
dset(prefix .. 'P.GREEN', 0)
sleep(3)
dset(prefix .. 'V.YELLOW', 1)
sleep(3)
dset(prefix .. 'V.RED', 0)
dset(prefix .. 'V.YELLOW', 0)
dset(prefix .. 'V.GREEN', 1)
sleep(3)
```

Context variables:

Script output:

Last run: 2022-08-18 09:23:35 UTC

Last modified: 2022-08-04 08:42:45 UTC

Figure 4: Step editor for the TASKOMAT DOOCS server.

panels of our accelerator facilities and should make it easy for operators and subsystem experts to develop their own procedures. It is, however, tightly coupled to the DOOCS server, and can therefore not be reused for developments with other control systems.

OPEN SOURCE

The TASKOLIB library is open source. The source code is published on GitHub [24] under the LGPL-2.1 [25] license. We welcome collaboration and external contributions. The source code of the TASKOMAT DOOCS server can also be made available on request.

STATUS AND OUTLOOK

The TASKOLIB library, the TASKOMAT DOOCS server, and the jddd-based user interface have reached "minimum viable product" state. Their feature set should be sufficient to perform useful tasks in our accelerator environments. We are now going to focus on the writing of sequences together with domain experts to gather feedback and experience with the software. The further development will depend on the outcome of this field test. We expect that features like automatic version control or improved nesting of sequences will be the next logical steps.

As we have seen with our DOOCS server, the effort of writing a TASKOMAT application for a specific control system is relatively low because the library contains most of the core functionality. This is not necessarily the case for the graphical user interface, though: In our case, the jddd panels were easy to create, but they cannot be reused for other control systems. The creation of a full-fledged GUI application could help with this. It would also make it easier to add special features like syntax highlighting for Lua.

ACKNOWLEDGEMENTS

The authors would like to thank Pedro Castro (DESY) and Giulio Gaio (Elettra–Sincrotrone Trieste) for many helpful discussions and ideas on sequencers and on the automation of accelerator operations.

REFERENCES

[1] P. Clout, M. Geib, and R. Westervelt, "Automation tools for accelerator control – A network based sequencer", in *Proc. LINAC 1990,* Albuquerque, USA, Sep. 1990, pp. 764–766.

[2] L. R. Dalesio, A. J. Kozubal, and M. R. Kraimer, "EPICS architecture", in *Proc. ICALEPCS 1991,* Tsukuba, Japan, Nov. 1991.

[3] A. J. Kozubal, D. M. Kerstiens, and R. M. Wright, "Experience with the State Notation Language and run-time sequencer", *Nucl. Instr. and Meth. A,* vol. 352, no. 1, pp. 411–414, Dec. 1994.
doi:10.1016/0168-9002(94)91556-3

[4] J. A. Perlas, D. Beltrán, and J. Rosich, "Design and implementation of a finite state machine queuing tool for EPICS", in *Proc. ICALEPCS 1999,* Trieste, Italy, Oct. 1999, pp. 564–566.

[5] J. Patrick, "The Fermilab accelerator control system", in *Proc. ICAP 2006,* Chamonix, France, Oct. 2006, pp. 246–249.

[6] V. Baggiolini, R. Alemany-Fernandez, R. Gorbonosov, *et al.,* "A sequencer for the LHC era", in *Proc. ICALEPCS 2009,* Kobe, Japan, Oct. 2009, pp. 670–672.

[7] R. Alemany-Fernandez, V. Baggiolini, R. Gorbonosov, *et al.,* "The LHC sequencer", in *Proc. ICALEPCS 2011,* Grenoble, France, Oct. 2011, pp. 300–303.

[8] G. Gaio, P. Cinquegrana, S. Krecic, *et al.,* "A framework for high level machine automation based on behavior trees", in *Proc. ICALEPCS 2021,* Shanghai, China, Oct. 2021, pp. 534–539.
doi:10.18429/JACoW-ICALEPCS2021-WEAL02

[9] W. Van Herck, B. Bauvir, and G. Ferro, “Automated operation of ITER”, in *Proc. ICALEPCS 2021,* Shanghai, China, Oct. 2021, pp. 628-630. doi:10.18429/JACoW-ICALEPCS2021-WEPV006

[10] D. Marcato, G. Arena, M. Bellato, *et al.,* “Pysmlib: A Python finite state machine library for EPICS”, in *Proc. ICALEPCS 2021,* Shanghai, China, Oct. 2021, pp. 330–336. doi:10.18429/JACoW-ICALEPCS2021-TUBL05

[11] R. Bacher, “Commissioning of the new pre-accelerator control system at DESY”, in *Proc. PCaPAC 2008,* Ljubljana, Slovenia, Oct. 2008, pp. 171-173.

[12] P. K. Bartkiewicz and P. Duval, “TINE as an accelerator control system at DESY”, *Meas. Sci. Technol.,* vol. 18, pp. 2379–2386, July 2007. doi:10.1088/0957-0233/18/8/012

[13] R. Ierusalimschy, L. H. de Figueiredo, and W. Celes, “Lua – an extensible extension language”, *Software: Practice & Experience* vol. 26, no. 6, pp. 635–652, June 1996. doi:10.1002/(SICI)1097-024X(199606)26:6<635::AID-SPE26>3.0.CO;2-P

[14] Lua website, https://www.lua.org

[15] Sol3 documentation website, https://sol2.readthedocs.io/en/latest/

[16] Sol3 source code repository, https://github.com/ThePhD/sol2

[17] GUL14 (General Utility Library for C++14) documentation website, https://gul14.info

[18] GUL14 (General Utility Library for C++14) source code repository, https://github.com/gul-cpp/gul14/

[19] O. Hensler and K. Rehlich, “DOOCS: A distributed object oriented control system”, in *Proc. XV Workshop on Charged Particle Accelerators,* Protvino, Russia, 1996.

[20] L. Fröhlich, A. Aghababyan, S. Grunewald, *et al.,* “The evolution of the DOOCS C++ code base”, *Proc. ICALEPCS 2021,* Shanghai, China, Oct. 2021, pp. 188–192. doi:10.18429/JACoW-ICALEPCS2021-MOPV027

[21] DOOCS website, https://doocs.desy.de

[22] E. Sombrowski, A. Petrosyan, K. Rehlich, *et al.,* “jddd: A tool for operators and experts to design control system panels”, in *Proc. ICALEPCS 2013*, San Francisco, USA, Oct. 2013, pp. 544–546.

[23] jddd (Java DOOCS Data Display) documentation website, https://jddd.desy.de

[24] Taskolib source code repository, https://github.com/taskolib/taskolib

[25] GNU Lesser General Public License version 2.1, https://www.gnu.org/licenses/old-licenses/lgpl-2.1.html