Future Sequencer Library — Evolving Design

Purpose

The future sequencer library provides a framework for executing sequences of steps. Each step contains a small program written in a scripting language. Sequences can be started and are generally executed in the order of steps; control flow steps like IF and WHILE allow formulating more complex procedures. User code can inject custom function definitions that are made available to the scripts.

Stakeholders

Developers: .... Pedro Castro, Lars Fröhlich, Olaf Hensler, Marcus Walla

“Done” features

The following features are already implemented in the current release of the library:

- Step class (defines a step in a sequence)
  - has an embedded LUA script that can be set and retrieved as a string.
  - has one of the following types: *action*, *if*, *else*, *elseif*, *end*, *while*, *try*, *catch*. The type can be set and retrieved.
  - stores a timestamp for “last time this step was executed” and “last time this step was modified”. Both timestamps are initialized to invalid values (0) and have getters and setters.
  - Setting a new script automatically sets the “modified” timestamp to the current system time.
  - has a label that can be set and retrieved.
  - has an associated timeout for its execution. The timeout can be set and retrieved.
  - has a modifiable list of variable names to be imported from a context before execution.
  - has a modifiable list of variable names to be exported into a context after execution.
- Context class (defines a script context holding variables etc.):
  - holds an arbitrary number of variables.
  - Each variable has a name and a value.
    - Names are case sensitive, must start with a letter, and may contain only alphanumeric characters and underscores.
    - Each value can be of type double, long long, or std::string.
  - Variables can be set, retrieved, and removed.
- Free function execute_step(Step&, Context&)
  - runs the script contained inside a Step with the given Context, updating the “last run” timestamp.
  - first loads the script from the string and throws an exception if it is not syntactically correct. Then, the script is executed; any runtime error during execution is thrown as a C++ exception. If the script returns a value that

  evaluates to true, the function returns true. Otherwise, the function returns false.
  - interrupts the execution of the script if the step timeout is reached. In this case, an exception is thrown.

Immediate development goals

The following features should be implemented in the next release of the library:

- A context can store C/C++ functions just like other variables.
- A context variable can be flagged as “permanent”.
  - Permanent variables cannot be exported from steps.
  - Permanent variables are automatically imported into each step.

Short-term development goals/discussion items

These are goals for the next iterations of the server:

- Pass a username along with all modifying functions of the Step class

Long-term development goals/discussion items

These are goals for later iterations of the server or items needing further discussion.

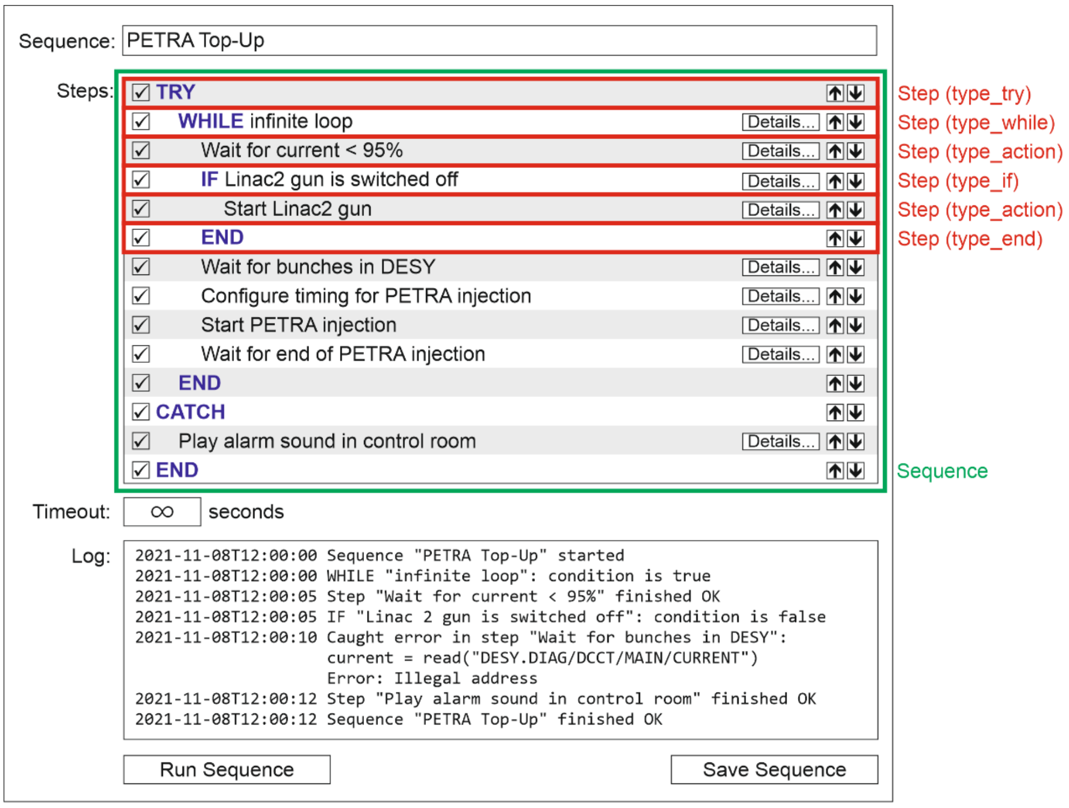
- Implement an “abort execution” functionality to interrupt running scripts
- Implement a Sequence class that contains a list of Steps and can execute them in order, following the control flow directions.

Not to be implemented

It has been decided that the following features are not to be implemented in this library (the list is obviously not complete):

- Direct control system dependencies (all control system specific functionality must be injected through an API)

Figures



Sequence: PETRA Top-Up

Steps:

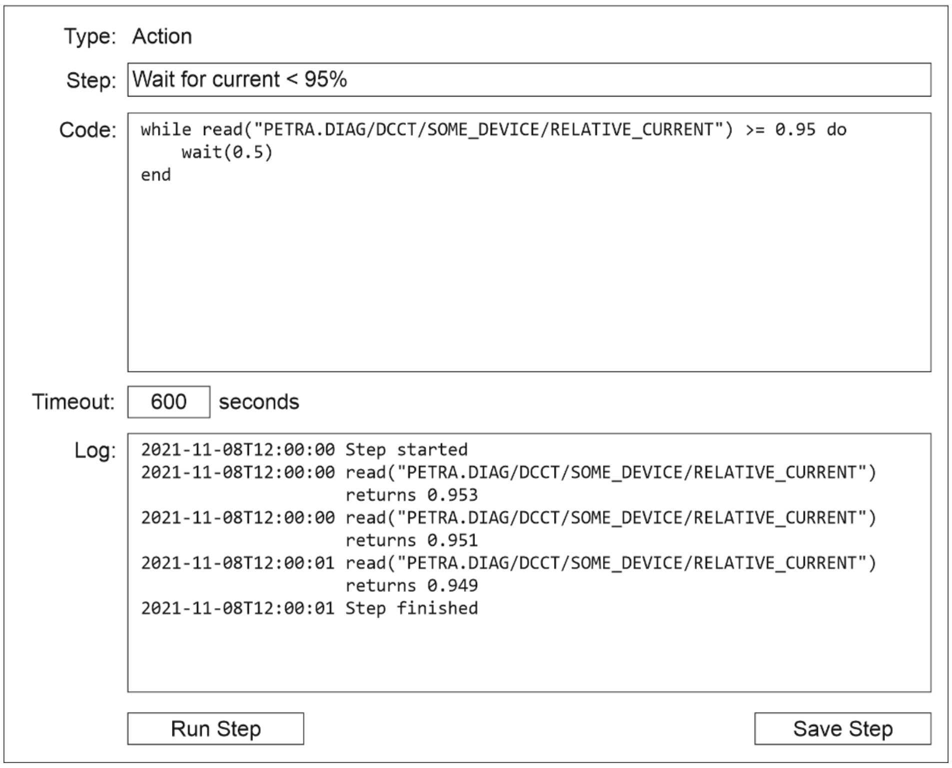
```
✓ TRY
✓   WHILE infinite loop
✓     Wait for current < 95%
✓     IF Linac2 gun is switched off
✓       Start Linac2 gun
✓     END
✓     Wait for bunches in DESY
✓     Configure timing for PETRA injection
✓     Start PETRA injection
✓     Wait for end of PETRA injection
✓   END
✓ CATCH
✓   Play alarm sound in control room
✓ END
```

Timeout: ∞ seconds

Log:

```
2021-11-08T12:00:00 Sequence "PETRA Top-Up" started
2021-11-08T12:00:00 WHILE "infinite loop": condition is true
2021-11-08T12:00:05 Step "Wait for current < 95%" finished OK
2021-11-08T12:00:05 IF "Linac 2 gun is switched off": condition is false
2021-11-08T12:00:10 Caught error in step "Wait for bunches in DESY":
                    current = read("DESY.DIAG/DCCT/MAIN/CURRENT")
                    Error: Illegal address
2021-11-08T12:00:12 Step "Play alarm sound in control room" finished OK
2021-11-08T12:00:12 Sequence "PETRA Top-Up" finished OK
```

Figure 1: Mockup of a sequence editor with associated classes



```
while read("PETRA.DIAG/DCCT/SOME_DEVICE/RELATIVE_CURRENT") >= 0.95 do
    wait(0.5)
end
```

```
2021-11-08T12:00:00 Step started
2021-11-08T12:00:00 read("PETRA.DIAG/DCCT/SOME_DEVICE/RELATIVE_CURRENT")
                    returns 0.953
2021-11-08T12:00:00 read("PETRA.DIAG/DCCT/SOME_DEVICE/RELATIVE_CURRENT")
                    returns 0.951
2021-11-08T12:00:01 read("PETRA.DIAG/DCCT/SOME_DEVICE/RELATIVE_CURRENT")
                    returns 0.949
2021-11-08T12:00:01 Step finished
```

Figure 2: Mockup of a step editor with associated attributes of the Step class