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 im- and exported from/to a context before/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.
  - A context can store a "LUA init function" that allows inserting functions, variables etc. into the LUA state before each script execution.
- 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.
- Free function execute_sequence(Sequence&, Context&)
  - validates the correct nesting of the steps within the sequence and throws an exception if syntax rules are violated, then
  - runs the sequence with the given Context.

Immediate development goals

The following features should be implemented in the next release of the library:
- Introduce a thread-safe message queue for communication between a thread executing a sequence and another thread.
- Add more Sequence methods to allow modifying/moving/inserting/deleting steps.

Short-term development goals/discussion items

These are goals for the next iterations of the server:
- Improve Doxygen documentation with examples.
- 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.
- Allow user-defined “libraries” of LUA scripts

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

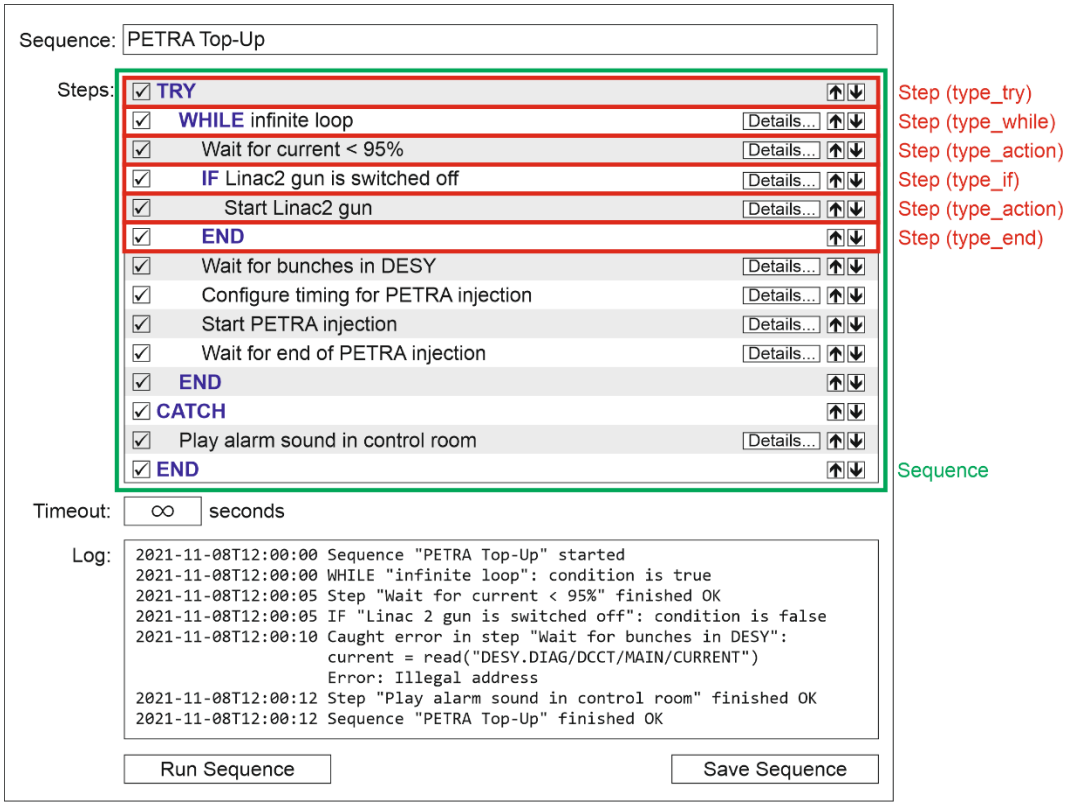

Figure 1: Mockup of a sequence editor with associated classes

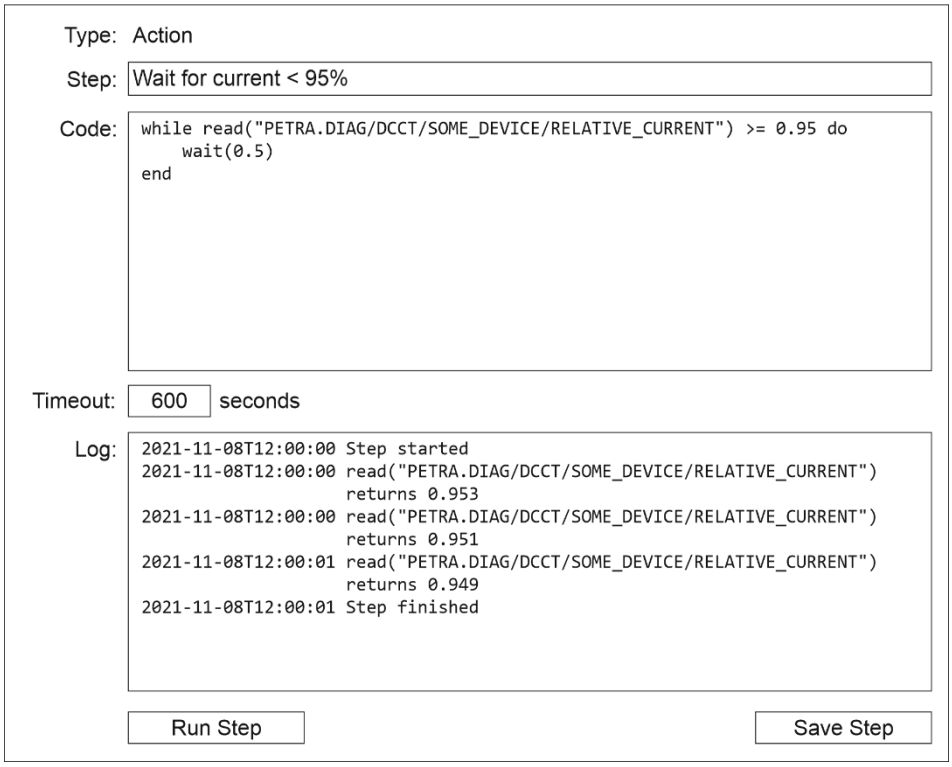

Figure 2: Mockup of a step editor with associated attributes of the Step class