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:

–

Immediate development goals

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

- Implementation of the Step class:
  - Each step has an embedded LUA script that can be set and retrieved as a string.
  - Each step has one of the following types: *task*, *if*, *else*, *elseif*, *end*, *while*, *try*, *catch*. The type can be set and retrieved.
  - Step 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.
  - Each step has a label that can be set and retrieved.
- Implementation of an Executor class:
  - A LUA state is embedded in each Executor object.
  - The Executor class has a member function to check if a script passed as a string is syntactically correct.
  - The class has a member function to run a script passed as a string. This function 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.
- Implementation of a free function execute_step(Step&, Executor&) to run the script contained inside a Step on the given Executor, updating the “last run” timestamp

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
– Implement a timeout in the Step class; the timeout is limited to a minimum and a maximum value

Long-term development goals/discussion items

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

– Implement a timeout for the Step class; when the timeout is reached, execution of the script is aborted and a timeout exception is thrown
– 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

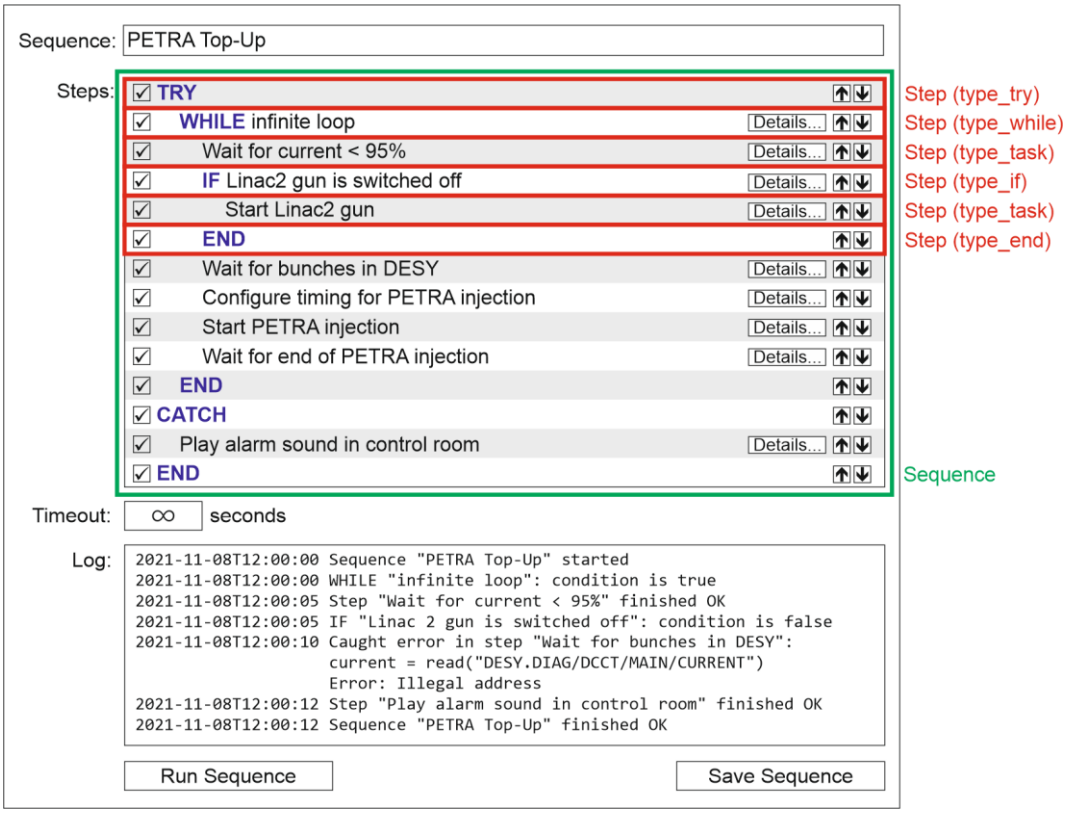

Figure 1: Mockup of a sequence editor with associated classes

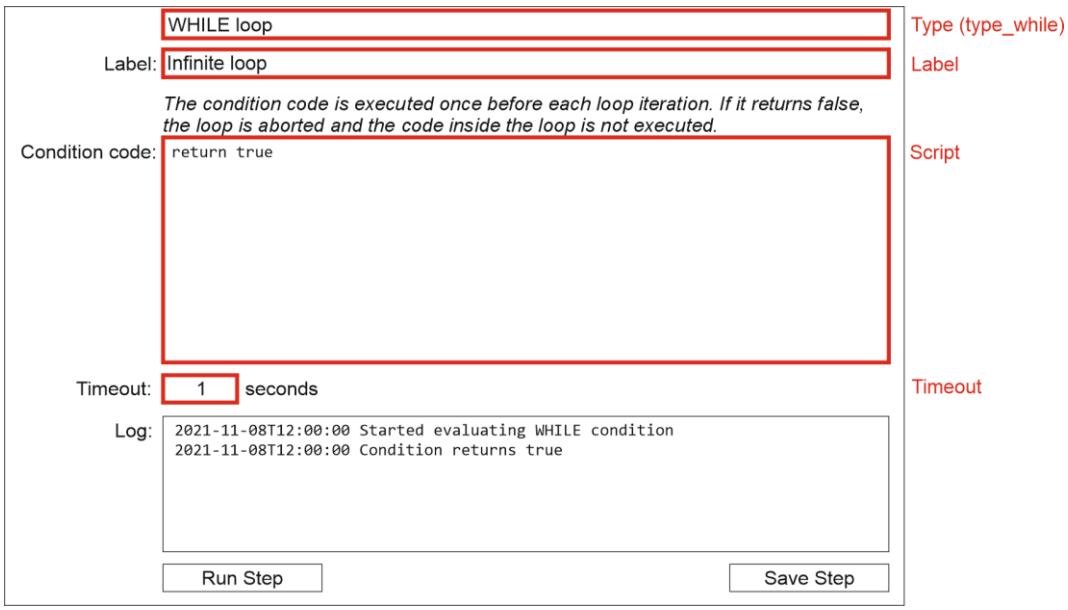

Figure 2: Mockup of a step editor with associated attributes of the Step class