Thylacine Code Roadmap

Evan Kirshenbaum

Tuesday, March 22, 2022

This document is a quick overview of the files that make up the Thylacine code base. In particular, it covers the part of Thylacine[1] called the "MPAM API" in the original proposal document. This is the part that is used to directly drive DMF boards and control their environment.

Obtaining the code

The code can be found on the internal HP GitHub at

https://github.azc.ext.hp.com/evan-kirshenbaum/thylacine.

The rest of the document will talk about files relative to the mpam subdirectory of this repository. Other potentially useful documents can be found in the doc subdirectory of the repo.

Prerequisites

The code is written in a combination of Python and ANTL4. I am currently using Python version 3.9.4. It is known to not work with versions of Python less than 3.9, and it does not work with version 3.10.

The following packages are required. They can be installed using **pip**:

- antlr4-python3-runtime
- clipboard
- matplotlib
- pyserial

Setting $PYPATH

The following directories must be included in your $PYTHONPATH environment variable:

- ${MPAM}/src
- ${MPAM}/tools
- ${MPAM}/target/generated-sources/antlr4

where ${MPAM} is the mpam directory.

Directory hierarchy

Within the mpam directory, you will find

- **grammar**: the grammar definitions for the macro language.

[1] The only part that exists at the moment.

- **inputs**: data files (e.g., macro definition files and JSON files) used by some of the tools.
- **opentrons**: Python code that runs locally on the Opentrons pipetting robot.
- **src**: the Python source for Thylacine.
  - **src/devices**: device definitions for various devices
  - **src/erk**: a Python package with more general support code.[2]
  - **src/langsup**: Python code that implements the macro language compiler and interpreter.
  - **src/mapm**: Python code for the MPAM proper
  - **src/quantities**: a Python package that supports general type-safe dimensioned quantity objects.
- **targets/generated-sources/antlr4**: files generated by ANTLR4 while compiling the grammar.
- **tests**: Python code for one-off tests.
- **tools**: Python code for useful tools, e.g., protocols, interactive exercisers, and the like.

Files

By directory, here is a brief description of the interesting files in each subdirectory. Files not mentioned are probably vestigial and, if so, should eventually be pruned.

grammar

- **DMF.g4**: the ANTLR4 definition of the syntax of the macro language.
- **pathLexer.g4**: the ANTLR4 definition of the lexical component of the macro language.

inputs

- **macros.dmf**: an example macro language input file, suitable for using as the --macro-file argument for the exercisers.
- **ot2.json**: a JSON file describing the configuration of an Opentrons OT-2 robot. Suitable for using as the --ot-config argument for the exercisers.
- **reagents.json**: a JSON file describing the placement of input reagents on the OT-2 for the combinatorial synthesis protocol in tools/pcr.py. Suitable for using as the --ot-reagents argument for the exercisers.

opentrons

These files are intended to be sent to the Opentrons OT-2 robot. Note that the robot runs an older version of Python (3.7.2, if I recall correctly).

- **looping_protocol.py**: an OT-2 “protocol” that loops, calling back to a local HTTP server to get instructions on what it should do next.
- **opentrons.support.py**: Python support code for the protocol.
- **schedule_xfers.py**: Python code to schedule robot trips. Includes keeping track of tips that have been used for reagents, filling from multiple source wells, and dispensing to multiple targets.

[2] “erk” is my initials.

src/devices

- **dummy_pipettor.py**: an instance of *Pipettor* that just spends time and prints messages. Used by default when there is no actual pipetting robot specified.
- **joey.py**: an instance of *Board* that describes the current Joey design. Note that this does not actually interact with a physical board.
- **opendrop.py**: an instance of *Board* that describes and controls an OpenDrop board.
- **opentrons.py**: an instance of *Pipettor* that runs the Opentrons OT-2 pipetting robot. Also includes an HTTP server for callbacks from the robot.
- **wombat.py**: a subclass of *Joey.Board* that talks to a Joey controlled using an OpenDrop controller.

src/erk

- **basic.py**: random useful Python routines and classes.
- **errors.py**: support for checking and reacting to possible conditions.
- **numutils.py**: number-related Python routines and classes.
- **stringutils.py**: string-related Python routines and classes

src/langsup

- **dmf_lang.py**: the back-end for the macro language compiler (implemented as an ANTLR4 visitor) and support classes for runtime execution.
- **type_supp.py**: support classes for the macro language, mostly having to do with supporting types.

src/mpam

- **device.py**: the main classes for the device model. This is the place to find things like *Board*, *Pad*, *Well*, and the like.
- **dilution.py**: optimal in-place dilution sequences for various dilution levels, as registered instances of *MixSequence* from processes.py
- **drop.py**: support for *Blob* and *Drop*, including most of the drop-motion inference as well as the traffic control that allows moving drops to yield to one another.
- **engine.py**: the underlying *Engine* that actually makes things work. It primarily includes three threads that run throughout:
  - *DevCommThread* performs interactions with actual system components.
  - *ClockThread* runs the ticking clock and contains hooks for performing actions before, on, and after each tick.
  - *TimerThread* schedules calls to happen after wall-clock delays.
- **exceptions.py**: various exception classes.
- **excerciser_tasks.py**: example instances of exerciser.*Task*.
- **exerciser.py**: a framework for standalone tools that can run multiple user-specified and highly parameterizable tasks. The main tools (e.g., wombat, joey, pcr) are built using this framework.
- **ga_regression.py**: support for running genetic algorithms to find optimal mixing sequences. Used by the dilution_ga_placed.py and mixed_ga_placed.py tools.

- **mixing.py**: like dilution.py, but for mixing of multiple reagents rather than dilution.
- **monitor.py**: the user interface, implemented using matplotlib.
- **paths.py**: classes to support complex drop motion paths. This is the primary mechanism by which I see protocols being specified.
- **pipettor.py**: the *Pipettor* class, which provides support for moving fluid onto and off of the board.
- **processes.py**: support for coordination of activities that involve multiple drops, such as mixing, dilution, and thermocycling. Implements synchronization of starting process automatically when all drops are in position and signaling when the process is complete.
- **thermocycle.py**: support for thermocycling processes.
- **types.py**: a large number of support classes and routines used all over the place. Includes things like *Reagent*, *Liquid*, *Mixture*, *Ticks*, *Location*, *OnOff*, and the like. In particular, it includes
  - *Delayed[T]*, a future-like mechanism that is the primary synchronization method used to sequence activity in the system.
  - *Operation* and *OpScheduler*,, used to schedule operations like turning on or off a pad or moving a drop.
  - *Barrier*, which allows activities to pause (or not) until a set number have arrived, at which point all futures are posted to, allowing activity to resume.

src/quantities

- **core.py**: the basic classes implementing the package.
- **dimensions.py**: definitions of the basic physical dimensions.
- **prefixes.py**: SI prefixes.
- **SI.py**: SI units
- **storage.py**: support for bits, bytes, bpm, as well as various flavors of binary prefixes.
- **temperature.py**: absolute temperatures in various systems.
- **timestamp.py**: absolute time points, as well as routines for getting the current time, the time after a delay, and the elapsed time since a timestamp.
- **US.py**: non-SI units, including relatively rare ones.

tests

Most of the contents of this directory are one-off tests that are unlikely to be useful to anybody else (or even me, these days), but which were checked in so that I had version control while I was using them. **scratch.py** is a (usually empty) file that I use for ad hoc testing. I try to remember to empty it out before merging it into the mainline.

tools

- **dilution_ga_placed.py**: a script that can be used to find optimal dilution sequences for full or partial dilutions of various orders. Useful when a protocol needs something that isn't in src/mpam/dilutions.py. The results can be put there.
- **joey.py**: an exerciser for a Joey board not hooked up to any actual hardware.

- **mix_ga_paced.py**: like dilution_ga_placed.py but for mixtures of different reagents rather than dilutions.
- **opendrop.py**: an exerciser for OpenDrop boards.
- **pcr.py**: an exerciser for Joey boards containing a few reasonably complicated protocols:
    - *CombSynth*: a complicated pipelined combinatorial synthesis protocol
    - *Prepare*: a parallel multi-drop protocol for making a precursor of one of the inputs for the combinatorial synthesis protocol.
    - *MixPrep*: a protocol to mix together the precursors made by *Prepare*.
- **wombat.py**: an exerciser for Wombat boards (Joey boards partially controlled by OpenDrop controllers).