Joey/Wombat Exerciser Tool: A Users’ Guide

Evan Kirshenbaum

*Tuesday, July 13, 2021*

Introduction

The Joey/Wombat exerciser tool is a pair of Python programs that allow you to run simple scripts on the Joey and Wombat boards as well as to interact with them directly. They provide a simple graphical interface and can be run with or without an actual board attached.[[1]](#footnote-1)

A note on nomenclature

To forestall confusion, here are definitions for several untransparent codenames:

**“Joey”** is the full board being developed at HP. It contains

* eight ~54-µL fully voidable wells (four on each side) that dispense 0.5-µL drops. Note that the interior electrodes in the wells on each side of the board are ganged together and move in lockstep. (The wells have independently-controlled gates),
* a 19-by-19 array of electrodes (aka “pads”). There is a five-by-three-pad region in the center of the board that contains non-functional pads.
* three independently-controlled heaters, each controlling two 3-by-seven-pad thermal zones,
* one magnet controlling two pads, and
* three “extraction ports” for the introduction and removal of fluid.

The pads can come pre-loaded with dried reagents. Schematically, the board is shown in ‎Fig. 1. with the magnetic pads indicated by “M”, the thermal pads indicated by their current temperature, and the extraction points indicated by small circles, and the “dead” pads in black. The coupled thermal zones are the ones directly above and below one another.

|  |
| --- |
| A picture containing text, crossword puzzle  Description automatically generated   1. Joey layout |

The full Joey system[[2]](#footnote-2) will also include an external environment including an Opentrons pipetting robot to refill the wells, a downward-facing camera, and a spectrometer.

**“Wombat”** is the Joey board hooked up to an OpenDrop controller. Due to limitations of the controller, only 128 electrodes can be controlled. The resulting layout is shown in ‎Fig. 2. Note that only four of the eight wells are functional. Note also that while the thermal zones, magnet, and one extraction point are represented in the figure (and in the on-screen display), they are non-functional in the actual Wombat board.

|  |
| --- |
| A picture containing text, crossword puzzle  Description automatically generated   1. Wombat layout |

**“Thylacine”** is the codename for an overarching project aimed to develop domain-specific languages for specifying protocols to be run on microfluidics systems at a variety of levels, from “essentially programming language” to “near-natural-language”. I wouldn’t mention it here except that the name shows up as the name of the Git repository.

**“MPAM”** is the codename for the lowest level of Thylacine, the “Microfluidics Platform Abstract Machine”. The current tools utilize a Python library that is being developed to provide an API to this abstract machine from programs written in Python.[[3]](#footnote-3) Note that the API is not (or at least not intended to be) Joey-specific.

Installation

In this section, we will cover how to get what you need to run the exercisers.

Python runtime and packages

To run the exerciser, you will need a sufficiently-recent version of Python. I am currently developing with version **3.9.4**, and I can guarantee that nothing before version 3.9 will work. The runtime can be acquired from <https://www.python.org/>.[[4]](#footnote-4)

In addition to a basic Python system, you will need to install a few packages. These can be installed using **pip**, which comes with the Python runtime. These packages include:

* numpy
* matplotlib
  + This may come with numpy.
* pyserial
  + Note that there is another, similarly-named package.

Thylacine code

You will also need to clone the Thylacine git repository, which can be found on the HP github at <https://github.azc.ext.hp.com/evan-kirshenbaum/thylacine>.

The git toolset can be found at <https://git-scm.com/>. For this, the version distributed by Cygwin will work just fine.

Assuming you have access to the HP github, you should be able to clone the repository by running

git clone git@github.azc.ext.hp.com:evan-kirshenbaum/thylacine.git

This will create several subdirectories, of which the relevant ones are

* **docs**: Contains documents, including this one
* **mpam**
  + **src**: the source code for the Python API
  + **tools**: source code for several tools, including the exercisers.

If you can’t clone the repository even though you have access to the HP github, it may be that you need to be granted access.

Running the exercisers

In this section, we will cover how to actually run the exercisers.

Setting up your environment

Before you can run the tools, you need to make sure that the **src** and **tools** directories are on your **PYTHONPATH** environment variable.[[5]](#footnote-5) You will want these to be absolute paths, in Windows syntax (e.g., “c:\foo\bar” rather than “/foo/bar”) and with elements separated by semicolons rather than colons.[[6]](#footnote-6)

Running the programs

To run the exerciser, simply run

python [*path to* ***mpam\tools***]\wombat.py *task* *options …*

substituting “joey” for “wombat”, if appropriate.

Task-first syntax

As indicated above, the basic syntax is to specify a *task* followed by *options*. If you don’t supply a task but do supply the option --help (or -h), you will get a list of the available tasks, which will be described below.

For all tasks, specifying --help (or -h) after the task name will give a list of the options available for that task.

Specifying options

All options (other than those that are positional rather than introduced by a keyword) can be specified using a keyword preceded by two dashes. Most can also be specified by one or two letters preceded by a single dash. All keywords and task names can be abbreviated to an unambiguous prefix. (For example, --update-interval could be abbreviated to –upd.)

Some options require values to be specified. These can be specified as a following argument (e.g., “--well 5”) or using an equals sign (e.g. “--well=5”). For single-letter arguments (only), the value can immediately follow the letter (e.g., “-w5”).

For values indicating physical quantities such as time or volume, both the value and units must be specified (e.g., “3uL”, “2drops”, or “1microliter”). The set of units and abbreviations is relatively large, and if you try one it doesn’t know, let me know and I’ll add it. Note that no space is allowed between the number and the units.

Options identifying wells take integers. The four wells down the left side of the Joey board are numbered 0 through 3, while the four down the right side are numbered 4 through 7. Note that this numbering is used for the Wombat board as well, so only wells 2, 3, 6, and 7 are available.

Common options

All tasks understand the following options:

* **--clock-speed** *TIME* (or **-cs**): The amount of time between clock ticks. The default is 100 ms. Several notes:
  + This is somewhat approximate, due to vagaries of Python, but all motion that should happen on a single tick will.
  + Python doesn’t appear to be able to go below about 60 ms.
  + The display runs asynchronously to the task, so at higher speeds, not all changes to the board state will be reflected in the display. They are, however, being sent to the board.
* **--initial-delay** *TIME*: The amount of time to wait after bringing the display up before running the selected task. The default is 5 seconds.
  + This parameter can be useful to make sure that you can see (on the display) the beginning of the task
* **--no-display** (or **-nd**): Don’t bring up the on-screen display.
* **--update-interval** *TIME*: The maximum amount of time between display updates. The default is 20 ms.
* **--min-time** TIME: The minimum amount of time to leave the display up, even if the task has finished. The default is 5 minutes.
  + This parameter is useful if you plan on using the display in interactive mode. You can specify a value of something like 7days to ensure that the display won’t accidentally disappear on you while you’re working.
* **--max-time** *TIME*: The maximum amount of time to leave the display up, even if the task hasn’t finished. The default is that there is no limit.

The wombat exerciser also has the following common option:

* **--port** *PORT* (or **-p**): The communication port (e.g., COM5) used to talk to the board. If this is not specified, only the display is run.

Identifying the COM port

For actually interacting with a Wombat board, you need to know the USB port that it is plugged into. This can be tricky, but a tool called USBDeview can help. It can be obtained at <https://www.nirsoft.net/utils/usb_devices_view.html>. The download link (near the bottom of the page) will download a zip file, and the program can be run directly from this file (without extracting it). Running it will bring up a display like the one shown in ‎Fig. 3.

|  |
| --- |
| 1. USBDeview |

The device name you want to look for for the OpenDrop controller is “Feather M0”, and the option value is found in the “Drive Letter” column, in this case COM3.

Interactively using the display

In addition to running tasks, the display can be used in an interactive manner to control the electrodes on the board. (If you only want to use the tool in this manner, use the display-only task.)

If you click on a pad or one of the sections in a well, that electrode will be turned on *and all other electrodes will be turned off*. This can be useful to walk drops around the board by clicking on an adjacent electrode. The electrode under the drop will be turned off and the one on the selected neighbor will be turned on simultaneously.[[7]](#footnote-7) Note that if you click on a well section (other than a well’s gate), this will also affect the corresponding well section in the other well on that side of the board.

If you click on a pad or a well section while holding down the control key, the state of the selected electrode will be toggled, but no other electrodes will be affected. Unless you have slowed the clock down significantly, you are unlikely to be able to make more than one simultaneous modification.[[8]](#footnote-8)

If you click on a pad (but not a well section) while holding down the shift key, the system runs a merge-and-split sequence assuming that that pad is between two (or more) nearly-adjacent drops. In particular:

* All of the eight neighbors of the selected pad are checked, and a note is made of any whose electrodes are currently on.
* Those electrodes are turned off and, simultaneously, the electrode of the selected pad is turned on. This will have the effect of drawing all of the drops under the neighbors to the selected pad and merging them.
* In the next clock tick, all of the previously-on neighbors are turned back on, and the selected pad is turned off. This will have the effect of splitting the merged drop back to the original locations. If all goes right, the volume of the resulting drops should be the same (as each other, not as they were initially).

Tasks

In this section, we will cover the most commonly used tasks and their options. All tasks also accept the common options described above, so they will not be repeated here.

display-only

The display-only task brings up the display but doesn’t do anything else. It is most useful when the user wants to interactively control the board by clicking on pads.

For example:

python wombat.py display-only --port=COM3 --min-time=1day --clock-speed=200ms

dispense

The dispense (or disp) task dispenses a drop from the specified well.[[9]](#footnote-9) The task options are

* **--well** *INT* (or **-w**): The well to dispense from. This is required, and there is no default.
* **--volume** *VOLUME* (or **-v**): The initial volume of the well, for display purposes only. The default is a full well.

For example:

python wombat.py dispense --well 2 --port=COM3

path

The path task walks a drop along a user-provided path, starting from a specified well or its current (specified) location. The task options are

* **--start-well** *INT* (or **-sw**): The well to dispense from. This is required, and there is no default.
* **--start-pad** *X Y* (or **-sp**): The (x, y) coordinates of the pad containing the drop.
  + Note that this option requires two following values.
  + The lower-left pad of the pad array is coordinate (0, 0).
* **--path** *PATH*: The path to follow. See below for details. The path must be specified, but it may be empty.
* **--volume** *VOLUME* (or **-v**): The initial volume of the well, for display purposes only. The default is a full well.

Note that exactly one of --start-well and --start-pad must be specified.

Paths are specified by a case-insensitive string of characters, each alphabetic character specifying a direction, optionally preceded by an integer indicating repetition. The characters “U”, “D”, “L”, and “R” indicate motion up, down, left, and right, respectively. The characters “N”, “S”, “E”, and “W” similarly indicate motion north (up), south (down), east (right), and west (left).

If the path ends in “A”, the drop should be on the pad adjacent to a well gate pad, and indicates that the drop should be absorbed into that well.

As an example, a path of “2RD10RUA” indicates

* walk two pads to the right,
* walk one pad down,
* walk ten pads to the right,
* walk one pad up, and
* be absorbed into the well.

For example:

python wombat.py path --well 2 --path 2R5D16RDA

should dispense a drop from well 2 (the upper usable well on the left side), walk it across to well 7 (the lower well on the right side) and absorb it in.

mix

The mix task dispenses n drops from well 2, treats them as each containing a distinct reagent (named R1 through Rn), mixes them together to a specified tolerance, and walks them to the wells on the opposite side. The lead drop is walked to well 6 and the other drops are walked to well 7.

The options are

* *NUM-DROPS*: a positional argument indicating the number of drops to mix.
* **--tolerance** FLOAT (or **-t**): The allowed tolerance, measured as the maximum deviation between the maximum and minimum proportion of any reagent concentration. The default is 10% (measured relative to the smaller concentration).
  + Note that at the moment, each number of drops only has a single mixing sequence for the single-drop case and for the fully-mixed case, and this is the shortest sequence to get to below a 10% tolerance, so specifying this option doesn’t really do anything.
  + If tighter tolerances are desired, let me know. It’s easy enough to find sequences for a specific number of drops and tolerance.
* **--full** (or **-f**): If this option is specified, all of the resulting drops will be well-mixed. Otherwise, only the lead drop (and the last drop it mixed with) are guaranteed to be well-mixed.
  + It is often the case that a shorter mixing sequence can be found if only one drop needs to be well-mixed.
* **--shuttles** *INT*: The number of times each mix-and-split operation is repeated. The default is zero (indicating a single mix and split). Specifying a larger number will likely result in better-mixed drops.
* **--pause-before** INT (or **-pb**): The amount of time (measured in the number of clock ticks) to pause once the drops are in position and before they begin mixing. The default is zero.
* **--pause-after** INT (or **-pa**): The amount of time (measured in the number of clock ticks) to pause once the drops have finished mixing and before they begin walking to the output wells. The default is zero.

Looking at the code, it appears that --pause-before and --pause-after haven’t actually been implemented. They probably should be. They also should probably be changed to taking an amount of time rather than a number of ticks.

The current code has mixing sequences for two through 12 drops (both single-drop and fully-mixed). Other sizes (and tolerances other than 10%) can be easily found and added to the library.

For example:

python wombat.py mix 5 --shuttles 2

python wombat.py mix 8 --full --shuttles 2

dilute

The mix task dispenses one drop of reagent from well 2 and a sufficient number of drops of solvent from well 3 to ensure that the lead drop (or all drops if the --fold option is specified) is diluted to an n-x dilution (to within the specified tolerance).

The dilute task takes the same options as the mix task, but the positional *NUM-DROPS* parameter is replaced by a positional *FOLD* parameter. For example, a fold argument of 8 indicates that an 8x dilution, i.e., the equivalent of mixing one drop of reagent with seven drops of solvent.

Note that if the --full option is not specified, the dilution will likely be possible with fewer drops of solvent. (For example, an 8x dilution of a single drop can be accomplished with three drops of solvent.)

Note also that if the --full option is not specified, the fold parameter does not need to be an integer. For example, a single drop that would result from diluting two drops of reagent with three drops of solvent can be performed by

python wombat.py dulute 2.5

The current code has dilution sequences for 2x through 5.5x in 0.5x increments, 6x through 12x, 15x, 16x, 20x, 25x, 32x, 50x, and 100x. Full mixes are available from 2x through 10x. Other sizes (and tolerances other than 10%) can be easily found and added to the library.

Adding new tasks

If for some reason you want to add a task of your own, the code for the current ones can be found in mpam/src/mpam/exerciser\_tasks.py. I really don’t expect anybody else to understand this code at this point, though, so you’re probably better off just asking me.

If you do decide to give it a try, in addition to creating the subclass of Task, you have to modify JoeyExerciser.\_\_init\_\_() (in mpam/tools/joey.py) to add an instance of your new subclass.

Support

The Thylacine repository has an issues list that I actually do pay attention to, so if you run into a bug or have a feature request (e.g., a new task or option), feel free to add a new issue. If you mention me (@evan-kirshenbaum) in the text, I believe I’ll get notified.[[10]](#footnote-10)

This issues list is also where to look to see if I’m already planning on doing something. Most of the issues of interest will be tagged with one or more of

* in: Wombat exerciser tool
* in: board display GUI
* in: MPAM API

If you add a new issue, don’t worry about adding labels. I’ll add the appropriate ones. The only ones that could be useful are the first 5 (the red ones), which give me an indication of how important an issue is:

* 1 low
* 2 minor
* 3 medium
* 4 important
* 5 critical

1. Indeed, at the moment, the API for interacting with the full Joey board has not yet been defined, let alone implemented, so the Joey tool can only be run in disconnected mode. [↑](#footnote-ref-1)
2. I believe that this is codenamed “wallaby”, but I may be mistaken. [↑](#footnote-ref-2)
3. The actual MPAM was only sketched out, and the current API bears little resemblance to the initial sketch. At the moment, it’s probably best to think of this as a general Python API to DMF boards in general. [↑](#footnote-ref-3)
4. Note that if you use Cygwin, the current distributed version is 3.8.10. This will not work. [↑](#footnote-ref-4)
5. Strictly speaking, **tools** doesn’t need to be on the path if you run the programs while the **tools** directory is your current working directory. **src** is always required. [↑](#footnote-ref-5)
6. If Cygwin ever gets a sufficiently-modern Python and you use that, this will need to be in Unix format, I believe. [↑](#footnote-ref-6)
7. Or as close to simultaneously as the OpenDrop controller makes it. The desired states are sent to the controller in a single operation. [↑](#footnote-ref-7)
8. There is a feature request in the issues list for adding a pause button for this purpose (among others). [↑](#footnote-ref-8)
9. This is, alas, aspirational at this point. [↑](#footnote-ref-9)
10. If not, I will certainly notice the new issue soon. Or you could send me e-mail at [evan.kirshenbaum@hp.com](mailto:evan.kirshenbaum@hp.com). [↑](#footnote-ref-10)