Manual on InstructionWriter

Trevor Y. H. Ho
trevor.ho@ed.ac.uk
trevor.y.h.ho@gmail.com

February 2021

Contents

1 Introduction

This introduction is identical to the `README.md` file on the GitHub page `https://github.com/tyhho/OT-2_Protocols`. Please feel free to skip this part if you are directed from there.

The **InstructionWriter** is a small Python script to facilitate the programming of an OT-2 robot. It works in conjunction with protocols written in the Opentrons OT-2 Python Protocol API (hereafter referred to as OT-2 APIv2), therefore, familiarity with the API is necessary.

In a nutshell, the **InstructionWriter** takes an Excel file with samples location information and convert them into machine-parsable lines containing *VOLUMES* and *PATHS* information. The lines are then copied and pasted into a Python protocol which decodes the information and feed them into OT-2 API functions.

The **InstructionWriter** exists because neither the OT-2 APIv2 nor the Opentrons Protocol Designer[1] builds on the *what* goes into *what*. They are centered on *where* goes to *where*.

What does that mean?

Let us look at an example of resuspending lyophilized oligonucleotides with water (aka primer resuspension). Say we have 2 tubes of primers, labelled "oligo001" and "oligo002". We think in terms of:

1. **oligo001** resuspended with 324 µL of water
2. **oligo002** resuspended with 275 µL of water

Issue is, the robot and the API does not think in this way. They only care about the *PATHS* and the *VOLUMES* in a transfer. For example, the above transfers would be defined on a robot as:

1. Transfer 324 µL from Slot 1, A1 (water) to Slot 2, A1 (primer tube)
2. Transfer 275 µL from Slot 1, A1 to Slot 2, A2

As a result, users need to do this conversion of *what* into *where* in their heads or on papers. This is manageable if one is dealing with only a few transfers (say < 20), but once the protocol scales up it become mentally difficult or even impossible to track all the *PATHS* (Slot 1, A1 to Slot 2, A1) and *VOLUMES* (302 µL).

When that does happen, a common thing we bench-top experimentalist does is to create an Excel spreadsheet where we list the locations, volumes and labels of what goes into what. This does not make the mundane task more pleasant by any means, but it creates a trackable checklist so one can focus on one transfer at any given moment.

The Excel spreadsheet will look like that in Table 1 below.

So basically, we need something to convert the Excel-recorded information from Table 1, which are human-readable and concern *SAMPLES and VOLUMES*, into machine-parsable instructions, which concern *PATHS and VOLUMES*. And, this is exactly what the **InstructionWriter** does.

To do this, a **machine-line** was arbitrarily defined and it takes the form of:

```
{vol.}${source slot}_{source well}->{dest. slot}_{dest. well}
```

---

[1] The Protocol Designer does ask users to first define the liquids and their locations, but ultimately, users still define the transfer of every liquid by themselves, and the identity of the liquid becomes somewhat irrelevant.

well	primer name	water (μL)
A1	oligo001	324
A2	oligo002	275
A3	oligo003	196
A4	oligo004	225
A5	oligo005	243
A6	oligo006	293
B1	oligo007	246
B2	oligo008	256
B3	oligo009	275
B4	oligo010	266

Table 1: Typical Excel Spreadsheet for tracking pipetting. Note: this is NOT what the spreadsheet should look like in the Excel file for **InstructionWriter**.

where, the *VOLUME* is separated from the *PATH* by a dollar "$" symbol. The *SOURCE* and the *DESTINATION* is separated by an arrow "->", and within each source and destination, the *SLOT* on the deck and the *WELL NAME* is separated by an underscore "_".

Given so, assuming the source of water is placed on slot 1, well 1, and that all primer tubes on placed on slot 2, the first two converted *INSTRUCTIONS* would then look like:

```
1  324$1_A1->2_A1
2  275$1_A1->2_A2
```

To allow a Python script to decode the information, these instructions are wrapped as a list of strings, which can then be copied and pasted under a list. Then, all individual information can be extracted using str.split inside a for-loop. The code would look like the following:

```
1  inst_list = [
2      "324$1_A1->2_A1",
3      "275$1_A1->2_A2",
4      "196$1_A1->2_A3",
5      "225$1_A1->2_A4",
6      "243$1_A1->2_A5",
7      "293$1_A1->2_A6",
8      "246$1_A1->2_B1",
9      "256$1_A1->2_B2",
10     "275$1_A1->2_B3",
11     "266$1_A1->2_A4"
12     ]
13
14 for inst in inst_list:
15
16     # Decodes information from each instruction line
17     vol, path = inst.split('$')
18     vol = float(vol)
19     source, dest = path.split('->')
20     source_slot, source_well = source.split('_')
21     dest_slot, dest_well = dest.split('_')
22
23     # Execution of transfer
24     r_pipette.transfer(vol,
25         labware_items[source_slot].wells_by_name()[source_well],
26         labware_items[dest_slot].wells_by_name()[dest_well],
```

```
27          new_tip='always'
28              )
```

This allows medium throughput processes to be set up in ease. The **InstructionWriter**, however, is not meant to address the need for high throughput processes. In those cases, all samples should be serialized and the scale should on slots instead of wells. It will be easier to be handled by the Protocol Designer, or by programming the API in columns / plates.

1.1   Commenting out instructions

Sometimes when testing or executing a protocol, mistakes and errors can happen. And you need to reset the protocol and re-run it again. This can be frustrating if you already have transfers that were manually rescued. In this sense, skipping transfers is easy to do through these instructions. Users can just comment them out by adding a hashtag # in front of the instruction line.

```
1  inst_list = [
2      # "324$1_A1->2_A1",
3      # "275$1_A1->2_A2",
4      # "196$1_A1->2_A3",
5      # "225$1_A1->2_A4", # These instructions above will not be executed
6      "243$1_A1->2_A5",
7      "293$1_A1->2_A6",
8      "246$1_A1->2_B1",
9      "256$1_A1->2_B2",
10     "275$1_A1->2_B3",
11     "266$1_A1->2_A4"
12     ]
```

2   Executing the InstructionWriter

To run the **InstructionWriter**, users must install a Python 3 environment on their computer. This can be done via installation of Anaconda (`www.anaconda.com`).

The **InstructionWriter** can be call directly in command line prompt where the `InstructionWriter.py` file is located:

```
1  > python -m InstructionWriter {path to your excel file}.xlsx
```

Important: Make sure the Excel file is closed or a `PermissionError` will raise.

It is best to use a relative file path, e.g. `./folder/your_excel_file.xlsx`. The file extension of your Excel file must be `.xlsx`, not `.xls`. It can be UPPER or lower cases.

Alternatively, it can be imported as a module. The following script assumes the python or Jupyter notebook file is located in the same directory as `InstructionWriter.py`:

```
1  import InstructionWriter
2  instructions = InstructionWriter.write_instructions("
       path_to_your_excel_file.xlsx")
```

3 Formatting the input Excel file

So long as users format the Excel file properly, the **InstructionWriter** will convert the information properly.

3.1 Top-level (or file-level) organization

In the Excel file, each spreadsheet represents a slot, for either the *SOURCE* or the *DESTINATION*.
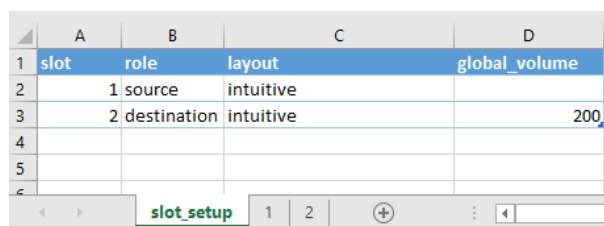
There is an additional spreadsheet, which **must be** named as `slot_setup`, and it tells the **InstructionWriter** which spreadsheet(s) are source slots and which other(s) are destination slots. It also tells the **InstructionWriter** how to interpret the *layout* in each spreadsheet.

The following rules must be observed:

- The source or destination sheets must be named as a number from 1 to 11. Do not leave any spaces or insert any characters before or after the number.

- Each spreadsheet must be either a source or a destination. Pipetting within the same slot is not supported.

- All spreadsheets can be arranged in any order.

- The `slot_setup` spreadsheet must exist in the file and its name must not be modified.

As such, the Excel file must have at least 3 spreadsheets for the **InstructionWriter** to properly function.

3.2 Formatting the "slot_setup" spreadsheet

Figure 1: Example `slot_setup` for intuitive to intuitive layout

The `slot_setup` spreadsheet has 4 columns:

1. `slot`

   Plain numbers from 1 - 11. The same slot number must not appear twice under this column

2. `role`

   Must be either `source` or `destination`

3. `layout`

   See more descriptions in Table 2 below.

4. `global volume`

   Only applicable to `destination` slots. If provided, any missing volumes in other layouts will use this `global volume` supplied. Mandated for `intuitive` layout if role = `destination`, optional for all other layouts.

In the `slot_setup` spreadsheet, the `role` and `layout` options must match in the following way:

role	layout
source	intuitive
	dataframe
destination	intuitive
	df_variable_sample
	df_variable_volume
	df_variable_sample_n_volume

Table 2: Compatible `role` and `layout` options

The **InstructionWriter** will throw an error in case of mismatch between the `role` and `layout` options.

3.3 Formatting the spreadsheets for slots

There are a few key things common to all spreadsheets for slots, regardless of layout.

- To reiterate, separate the sources and destinations onto different slots.

- Any source samples must not occur more than once across the entire deck.

- Forbidden characters

  Do not include any of the characters of the plus sign "+", the dollar sign "$" and the underscore "_" anywhere in the spreadsheet. Doing so will result in machine lines that cannot be parsed by the recommended Python script.

- Discouraged characters

  Avoid the characters the hashtag "#" and the dot "." in the sample name and elsewhere. This has not been rigorously tested but it is best to treat them as forbidden characters as well.

- Do not put empty "spaces" in wells that are not supposed to receive anything.

- For all destination slots, all information in the spreadsheet are processed row-by-row and from left to right. See Figure 2 for illustration.

- Make sure the actual layouts used match with the information supplied in the `slot_setup` spreadsheet. The **InstructionWriter** is not capable in auto-inferring or guessing the layouts on behalf of users.

- The **InstructionWriter** does not check if your layout corresponds to an actual labware. The "intuitive" layout puts some constraints on the rows and columns so users would be limited to some choices, but still in some cases some wells may not exist physically and there will be no checks on them.

- For all tabular layouts, i.e. `dataframe`, `df_variable_sample`, `df_variable_volume`, and `df_variable_sample_n_volume`, do not put down a well name (aka well ID) if nothing should go into that well. The **InstructionWriter** will not be able to handle it and will run into errors.

	A	B	C	D	E	F
1	row	1	2	3	4	5
2	A	1 →				
3	B	2 →				
4	C	3 →				

	A	B	C	D	E
1	well	sample1	vol1	sample2	vol2
2	A1	1 →			
3	A2	2 →			
4	A3	3 →			
5	A4	4 →			
6	A5	5 →			
7	A6	6 →			

Figure 2: Processing order of information in spreadsheets for slots

4 Layout options and their uses

4.1 intuitive

This layout allows users to input sample information in a view resembling the actual labware. It is user-friendly, and is great for source slots with a few samples. It is not ideal when used for destinations that have multiple samples to be dispensed into.

4.1.1 Layouts that can be recognized under "intuitive":

Figure 3: Options for the "intuitive" layout. Deviation from these options will result in errors.

Note that the column header "row" must be present for the first column and placed above the row index "A", and the table must be put on the top-left corner of the spreadsheet. Do not change the labels of the rows and columns.

4.1.2 "intuitive" for sources

Figure 4: Example "intuitive" layout for source slot, corresponding to the labware
`opentrons_24_tuberack_eppendorf_1.5ml_safelock_snapcap`

- Put one sample in each well only

8

4.1.3 "intuitive" for destinations

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	row	1	2	3	4	5	6	7	8	9	10	11	12
2	A	+a+b+c											
3	B		+dye	+dye	+dye		+dye	+dye	+dye		+dye	+dye	+dye
4	C		+dye		+dye			+dye					+dye
5	D		+dye		+dye			+dye					+dye
6	E		+dye		+dye			+dye			+dye	+dye	+dye
7	F		+dye		+dye			+dye			+dye		
8	G		+dye		+dye			+dye			+dye		
9	H		+dye	+dye	+dye			+dye			+dye	+dye	+dye
10													

slot_setup | 1 | **2** | ⊕

Figure 5: Example "intuitive" layout for destination slot, corresponding to the labware `corning_96_wellplate_360ul_flat`

- Each sample that goes into each well must be prefixed with a plus sign "+". It does not matter whether there are spaces between the plus signs and then previous/next sample name

- The parameter `global volume` must be provided from in the `slot_setup` spreadsheet. Failure to do so will result in error.

- Different volumes for different samples to go into the same well is not supported.

9

4.2 dataframe

This is just plain table form for putting down the sample names and their locations, for use on source slots only. There must be only two columns: `well` and `sample`. Do not add any extra columns except comment columns. Do not change the column headers.

	A	B
1	well	sample
2	A1	J23117
3	A2	J23100
4	A3	J23101
5	B1	B0030
6	B2	B0032
7	B3	B0034
8	C1	B0015
9	C2	L3S2P21
10	C3	T7term

Figure 6: Example "dataframe" layout for source slots

4.3 df_variable_sample

This a destination layout format where the volumes of each sample category is fixed, but the identities of the input sample vary from one destination well to another.

`df_variable_sample` can take an unlimited number of columns, provided that they are properly formatted.

The general format is as follow, assuming there are only 2 sample categories.

well	{sample category}_{volume}	{sample category }_{volume}
{well ID}	{sample name}	{sample name}
{well ID}	{sample name}	{sample name}
...	...	...

Table 3: General format for `df_variable_sample`

Figure 7: Example "df_variable_sample" layout for destination slots

- The sample category name and the volume are separated by an underscore "_".

- The sample category does not really matter. However, the user must provide a sample category name that is free of forbidden characters, as explained in subsection 3.3. Plus, it is a good practice to use meaningful descriptions for documentation.

- Volume input can have decimal places, i.e. they can be `float`.

- If the volume is not given in the column header, the **InstructionWriter** will use the `global volume` from the `slot_setup` spreadsheet if it is available. If it is not given, the volume will be set to 0. See example below in Figure 8

- There is no limit on the number of sample categories.

- It is possible to leave out a sample for a given sample category, even if the sample volume is given. The supposed "sample" will simply be skipped. See example below in Figure 8.

	A	B	C
1	well	primer1_	primer2_5
2	A1	primer001	
3	A3		primer003

Figure 8: Example "df_variable_sample" layout with empty cells and missing volumes. In this case, the **InstructionWriter** will look for `global volume` for "primer001" to dispense into well A1. It will ignore the empty cells for sample category "primer2" for well A1, and also for sample category "primer1" for well A3.

4.4 df_variable_volume

This a destination layout format where the set of samples going into the destination wells are fixed, but the volumes of the sample to be dispensed vary from one destination well to another.

`df_variable_volume` can take an unlimited number of columns.

The general format is as follow, assuming there are only 2 samples.

well	{sample}	{sample}
{well ID}	{volume}	{volume}
{well ID}	{volume}	{volume}
...	...	...

Table 4: General format for `df_variable_sample`

	A	B	C
1	well	# primer	water
2	A1	oligo001	324
3	A2	oligo002	275
4	A3	oligo003	196
5	A4	oligo004	225
6	A5	oligo005	243
7	A6	oligo006	293
8	B1	oligo007	246
9	B2	oligo008	256
10	B3	oligo009	275
11	B4	oligo010	266

Figure 9: Example "df_variable_volume" layout for destination slots. Note that the column "# primer" is a comment column and will not be processed. See section 5 for details.

- For any volumes entered, the sample name must be given. Not doing so will result in error.

- If a volume is left blank but the sample name is given in the column header, the **InstructionWriter** will use the `global volume` from the `slot_setup` spreadsheet if it is available. If it is not given, the volume will be set to 0.

- There is no limit on the number of sample columns.

4.5 df_variable_sample_n_volume

This destination layout most customizable, and can substitute the roles of `df_variable_sample` and `df_variable_volume`. For each destination wells, users can specify any number of "sample" + "volume" pairs, and each destination well can any number of such pairs.

The general format below assumes destination wells each receiving two different samples and different volumes.

well	{sample cat.}	{volume cat.}	{sample cat.}	{volume cat.}
{well ID}	{sample A name}	{sample A vol.}	{sample C name}	{sample C vol.}
{well ID}	{sample B name}	{sample B vol.}	{sample D name}	{sample D vol.}
...	...	...	...	...

Table 5: General format for `df_variable_sample_n_volume`

	A	B	C	D	E	F	G	H	I
1	well	# construct ID	promoter	promoter vol	RBS	RBS vol	terminator	ter vol	# remarks
2	A1	pTEST1	J23100	2	B0030	1	L3S2P21	1	
3	A2	pTEST2	J23101	3	B0030	1	L3S2P21	1	
4	A3	pTEST3	J23117	1	B0032	1.5	L3S2P21	1	
5	A4	pTEST4	J23100	2	B0032	1.5	L3S2P21	1	
6	A5	pTEST5	J23101	3			L3S2P21	1	should lead to 0 transformants
7	A6	pTEST6	J23117	1	B0034	2	L3S2P21	1	
8	A7	pTEST7	J23100	2	B0034	2	L3S2P21	1	
9	A8	pTEST8	J23101	3	B0034	2	L3S2P21	1	

Figure 10: Example "df_variable_sample_n_volume" layout for destination slots. Columns "# construct ID" and "# remarks" are comment columns and will not be processed. See section 5 for details.

- Unlike the other two destination dataframe layouts, the column headers in `df_variable_sample_n_volume`, except `well`, are merely placeholders. Users can put anything there and the information will not be processed. Having said that, from perspectives of documentation and avoiding errors, it is best to fill them with meaningful descriptions.

- Just to reiterate, keep the `well` column and do not make any changes to the column header.

- For every "sample" + "volume" combination, they must appear in pairs. The sample name must be placed on the left cell and and the volume must be placed on the right cell. An error will be thrown if only either the sample or the volume cell is given.

- The `global volume` parameter from the `slot_setp` spreadhsheet is not applicable to the `df_variable_sample_n_volume` layout. Every volume must be defined.

- It is possible that empty cells be inserted between pairs or even within pairs. It will work so long as the total number of filled cells per row, excluding the well location, is an even number. However, the practice of interrupting pairs of "sample" + volume is strongly discouraged.

- As seen in the example in Figure 10, it is possible to leave out some pairs. Each destination well can be assigned different numbers of "sample" + "volume" pairs, provided that they are in pairs.

- For `df_variable_sample_n_volume`, information is extracted row-by-row from left to right. This means that, there is actually no relationship between one row and the other, and samples belonging to the same category do not necessary need to group under the same column. This is still encouraged though from the perspective of documentation.

5 Comment columns

In all source or destination sheets, it is possible to insert comment columns for remarks and meta-information that are useful for the users but not the **InstructionWriter**. These columns must starts with the hashtag '#' and all information throughout the column, vertically, will not be processed.

The mechanism is simply that the **InstructionWriter** removes any empty columns or any comments columns before performing any downstream processes. Therefore, it is even possible to insert a comment column between "sample" and "volume" under the `df_variable_sample_n_volume` layout.

However, if a comment column is inserted, it must be located at the top level of the table, whichever layout is being used. Say, on the `intuitive` layout, inserting a comment column underneath the grid will result in an error. See Figure 11 below for what works and what doesn't.

Acceptable comment columns for "intuitive" layout

	A	B	C	D	E	F	G	H	I
1	row	1	2	3	4	5		# comment column here is fine	
2	A	water							
3	B		DAPG					random comment	
4	C								
5									
6								random comment 2	

Acceptable comment columns for "df_variable_sample_n_volume" layout

	A	B	C	D	E	F	G	H
1	# destination sample	well	sample1	# this comment column will be removed	vol1	sample2	vol2	# remarks
2	x1	A1	oligo001	c1	1	oligo002	4	r1
3	x2	A2	oligo001	c2	2	oligo003	5	r2
4	x3	A3	oligo001	c3	3	oligo004	6	r3

Unacceptable comment columns for "intuitive" layout

	A	B	C	D	E	F
1	row	1	2	3	4	5
2	A	water				
3	B		DAPG			
4	C					
5						
6			# comment here will give error			

Unacceptable comment columns for "df_variable_sample_n_volume" layout

	A	B	C	D
1	well	sample1		vol1
2	A1	oligo001		1
3	A2	oligo001	# putting this comment column here will result in error	2
4	A3	oligo001		3

Figure 11: Examples for acceptable and unacceptable comment columns, using `intuitive` and `df_variable_sample_n_volume` layouts for illustration.

6 Applications

This part is identical to the `README.md` file on the GitHub page `https://github.com/tyhho/OT-2_Protocols`.

Protocols I performed using the **InstructionWriter**:

1. Mixing different plasmids for co-transformation (`intuitive` − > `intuitive`)

2. Standardizing cell mass for whole-cell lysates preparation for Western Blot analysis (`intuitive` − > `df_variable_sample_n_volume`)

3. Cherry picking strains that showed desirable traits when screening libraries of mutated variants (`dataframe` − > (`df_variable_sample`)

4. Moving PCR products from 96-well PCR plate into DNA purification columns (`dataframe` − > `df_variable_sample`)

5. Migrating column purified DNA back to 96-well PCR for submission of Sanger sequencing (`dataframe` − > `df_variable_sample`)

6. Moving bacterial cultures from 96-well deep plate into microcentrifuge tubes for saving glycerol stocks (`intuitive` − > `intuitive`)

7 License

Licensed under MIT License.