Autoprotocol Documentation

Release 4.0.0

Transcriptic

November 28, 2017

Contents

CHAPTER 1

autoprotocol.protocol

protocol.Protocol

class autoprotocol.protocol.**Protocol**(*refs=None*, *instructions=None*)

A Protocol is a sequence of instructions to be executed, and a set of containers on which those instructions act.

Initially, a Protocol has an empty sequence of instructions and no referenced containers. To add a reference to a container, use the ref() method, which returns a Container.

```
p = Protocol()
my_plate = p.ref("my_plate", id="ct1xae8jabbe6",
                           cont_type="96-pcr", storage="cold_4")
```

To add instructions to the protocol, use the helper methods in this class

```
p.transfer(source=my_plate.well("A1"),
           dest=my_plate.well("B4"),
           volume="50:microliter")
p.thermocycle(my_plate, groups=[
               { "cycles": 1,
                 "steps": [
                   { "temperature": "95:celsius",
                     "duration": "1:hour"
                   }]
               }])
```

Autoprotocol Output:

```
{
  "refs": {
    "my_plate": {
      "id": "ct1xae8jabbe6",
      "store": {
        "where": "cold_4"
      }
    }
  },
  "instructions": [
    {
      "groups": [
        {
          "transfer": [
            {
              "volume": "50.0:microliter",
```

```
                    "to": "my_plate/15",
                    "from": "my_plate/0"
                  }
                ]
              }
            ],
            "op": "pipette"
          },
          {
            "volume": "10:microliter",
            "dataref": null,
            "object": "my_plate",
            "groups": [
              {
                "cycles": 1,
                "steps": [
                  {
                    "duration": "1:hour",
                    "temperature": "95:celsius"
                  }
                ]
              }
            ],
            "op": "thermocycle"
          }
        ]
      }
```

Protocol.container_type()

`Protocol.`**`container_type`**(*shortname*)

Convert a ContainerType shortname into a ContainerType object.

Parameters **`shortname`** (*str*) – String representing one of the ContainerTypes in the _CONTAINER_TYPES dictionary.

Returns Returns a Container type object corresponding to the shortname passed to the function. If a ContainerType object is passed, that same ContainerType is returned.

Return type *ContainerType*

Raises `ValueError` – If an unknown ContainerType shortname is passed as a parameter.

Protocol.ref()

`Protocol.`**`ref`**(*name*, *id=None*, *cont_type=None*, *storage=None*, *discard=None*, *cover=None*)

Add a Ref object to the dictionary of Refs associated with this protocol and return a Container with the id, container type and storage or discard conditions specified.

Example Usage:

```
p = Protocol()

# ref a new container (no id specified)
sample_ref_1 = p.ref("sample_plate_1",
                     cont_type="96-pcr",
                     discard=True)
```

```
# ref an existing container with a known id
sample_ref_2 = p.ref("sample_plate_2",
                     id="ct1cxae33lkj",
                     cont_type="96-pcr",
                     storage="ambient")
```

Autoprotocol Output:

```
{
  "refs": {
    "sample_plate_1": {
      "new": "96-pcr",
      "discard": true
    },
    "sample_plate_2": {
      "id": "ct1cxae33lkj",
      "store": {
        "where": "ambient"
      }
    }
  },
  "instructions": []
}
```

Parameters

- **name** (*str*) – name of the container/ref being created.
- **id** (*str*) – id of the container being created, from your organization's inventory on http://secure.transcriptic.com. Strings representing ids begin with "ct".
- **cont_type** (*str, ContainerType*) – container type of the Container object that will be generated.
- **storage** (*{"ambient", "cold_20", "cold_4", "warm_37"}, optional*) – temperature the container being referenced should be stored at after a run is completed. Either a storage condition must be specified or discard must be set to True.
- **discard** (*bool, optional*) – if no storage condition is specified and discard is set to True, the container being referenced will be discarded after a run.

Returns

container –

Container object generated from the id and container type provided.

Return type *Container*

Raises

- `RuntimeError` – If a container previously referenced in this protocol (existant in refs section) has the same name as the one specified.
- `RuntimeError` – If no container type is specified.
- `RuntimeError` – If no valid storage or discard condition is specified.

Protocol.append()

`Protocol.`**`append`**(*instructions*)

Append instruction(s) to the list of Instruction objects associated with this protocol. The other functions on Protocol() should be used in lieu of doing this directly.

Example Usage:

```
p = Protocol()
p.append(Incubate("sample_plate", "ambient", "1:hour"))
```

Autoprotocol Output:

```
"instructions": [
    {
      "duration": "1:hour",
      "where": "ambient",
      "object": "sample_plate",
      "shaking": false,
      "op": "incubate"
    }
]
```

Parameters **`instructions`** (*Instruction*) – Instruction object to be appended.

Protocol.as_dict()

`Protocol.`**`as_dict`**()

Return the entire protocol as a dictionary.

Example Usage:

```
from autoprotocol.protocol import Protocol
import json

p = Protocol()
sample_ref_2 = p.ref("sample_plate_2",
                     id="ct1cxae33lkj",
                     cont_type="96-pcr",
                     storage="ambient")
p.seal(sample_ref_2)
p.incubate(sample_ref_2, "warm_37", "20:minute")

print json.dumps(p.as_dict(), indent=2)
```

Autoprotocol Output:

```
{
  "refs": {
    "sample_plate_2": {
      "id": "ct1cxae33lkj",
      "store": {
        "where": "ambient"
      }
    }
  },
  "instructions": [
    {
```

```
            "object": "sample_plate_2",
            "op": "seal"
          },
          {
            "duration": "20:minute",
            "where": "warm_37",
            "object": "sample_plate_2",
            "shaking": false,
            "op": "incubate"
          }
        ]
      }
```

Returns dict with keys "refs" and "instructions" and optionally "time_constraints" and "outs", each of which contain the "refified" contents of their corresponding Protocol attribute.

Return type dict

Protocol.get_instruction_index()

`Protocol.`**`get_instruction_index`**`()`

Get index of the last appended instruction

Example Usage:

```
p = Protocol()
plate_1 = p.ref("plate_1", id=None, cont_type="96-flat",
                discard=True)

p.cover(plate_1)
time_point_1 = p.get_instruction_index()  # time_point_1 = 0
```

Raises `ValueError` – If an instruction index is less than 0

Returns Index of the preceding instruction

Return type int

Protocol.add_time_constraint()

`Protocol.`**`add_time_constraint`**`(`*from_dict*, *to_dict*, *less_than=None*, *more_than=None*, *mirror=False*`)`

Constraint the time between two instructions

Add time constraints from *from_dict* to *to_dict*. Time constraints guarantee that the time from the *from_dict* to the *to_dict* is less than or greater than some specified duration. Care should be taken when applying time constraints as constraints may make some protocols impossible to schedule or run.

Though autoprotocol orders instructions in a list, instructions do not need to be run in the order they are listed and instead depend on the preceding dependencies. Time constraints should be added with such limitations in mind.

Constraints are directional; use *mirror=True* if the time constraint should be added in both directions. Note that mirroring is only applied to the less_than constraint, as the more_than constraint implies both a minimum delay betweeen two timing points and also an explicit ordering between the two timing points.

Example Usage:

```
plate_1 = protocol.ref("plate_1", id=None, cont_type="96-flat",
                       discard=True)
plate_2 = protocol.ref("plate_2", id=None, cont_type="96-flat",
                       discard=True)

protocol.cover(plate_1)
time_point_1 = protocol.get_instruction_index()

protocol.cover(plate_2)
time_point_2 = protocol.get_instruction_index()

protocol.add_time_constraint(
    {"mark": plate_1, "state": "start"},
    {"mark": time_point_1, "state": "end"},
    less_than = "1:minute")
protocol.add_time_constraint(
    {"mark": time_point_2, "state": "start"},
    {"mark": time_point_1, "state": "start"},
    less_than = "1:minute", True)
```

Autoprotocol Output:

```
{
    "refs": {
        "plate_1": {
            "new": "96-flat",
            "discard": true
        },
        "plate_2": {
            "new": "96-flat",
            "discard": true
        }
    },
    "time_constraints": [
        {
            "to": {
                "instruction_end": 0
            },
            "less_than": "1.0:minute",
            "from": {
                "ref_start": "plate_1"
            }
        },
        {
            "to": {
                "instruction_start": 0
            },
            "less_than": "1.0:minute",
            "from": {
                "instruction_start": 1
            }
        },
        {
            "to": {
                "instruction_start": 1
            },
            "less_than": "1.0:minute",
            "from": {
```

```
                    "instruction_start": 0
                }
            }
        ],
        "instructions": [
            {
                "lid": "standard",
                "object": "plate_1",
                "op": "cover"
            },
            {
                "lid": "standard",
                "object": "plate_2",
                "op": "cover"
            }
        ]
    }
```

Parameters

- **from_dict** (*dict*) – Dictionary defining the initial time constraint condition. Composed of keys: “mark” and “state”

  mark: int or Container instruction index of container

  state: “start” or “end” specifies either the start or end of the “mark” point

- **to_dict** (*dict*) – Dictionary defining the end time constraint condition. Specified in the same format as from_dict
- **less_than** (*str, Unit*) – max time between from_dict and to_dict
- **more_than** (*str, Unit*) – min time between from_dict and to_dict
- **mirror** (*bool, optional*) – choice to mirror the from and to positions when time constraints should be added in both directions (only applies to the less_than constraint)

Raises

- `ValueError` – If an instruction mark is less than 0
- `TypeError` – If mark is not container or integer
- `TypeError` – If state not in [’start’, ‘end’]
- `KeyError` – If to_dict or from_dict does not contain ‘mark’
- `KeyError` – If to_dict or from_dict does not contain ‘state’
- `ValueError` – If time is less than ‘0:second’
- `RuntimeError` – If from_dict and to_dict are equal
- `RuntimeError` – If more_than is greater than less_than
- `RuntimeError` – If from_dict[”marker”] and to_dict[”marker”] are equal and from_dict[”state”] = “end”

Protocol.distribute()

Protocol.**distribute**(*source*, *dest*, *volume*, *allow_carryover=False*, *mix_before=False*, *mix_vol=None*, *repetitions=10*, *flowrate='100:microliter/second'*, *aspirate_speed=None*, *aspirate_source=None*, *dispense_speed=None*, *distribute_target=None*, *pre_buffer=None*, *disposal_vol=None*, *transit_vol=None*, *blowout_buffer=None*, *tip_type=None*, *new_group=False*)

Distribute liquid from source well(s) to destination wells(s).

Example Usage:

```
p = Protocol()
sample_plate = p.ref("sample_plate",
                     None,
                     "96-flat",
                     storage="warm_37")
sample_source = p.ref("sample_source",
                      "ct32kj234l21g",
                      "micro-1.5",
                      storage="cold_20")

p.distribute(sample_source.well(0),
             sample_plate.wells_from(0,8,columnwise=True),
             "200:microliter",
             mix_before=True,
             mix_vol="500:microliter",
             repetitions=20)
```

Autoprotocol Output:


```
"instructions": [
  {
    "groups": [
      {
        "distribute": {
          "to": [
            {
              "volume": "150.0:microliter",
              "well": "sample_plate/0"
            },
            {
              "volume": "150.0:microliter",
              "well": "sample_plate/12"
            },
            {
              "volume": "150.0:microliter",
              "well": "sample_plate/24"
            },
            {
              "volume": "150.0:microliter",
              "well": "sample_plate/36"
            },
            {
              "volume": "150.0:microliter",
              "well": "sample_plate/48"
            },
            {
              "volume": "150.0:microliter",
              "well": "sample_plate/60"
```

```
                },
                {
                  "volume": "150.0:microliter",
                  "well": "sample_plate/72"
                },
                {
                  "volume": "150.0:microliter",
                  "well": "sample_plate/84"
                }
              ],
              "from": "sample_source/0",
              "mix_before": {
                "volume": "500:microliter",
                "repetitions": 20,
                "speed": "100:microliter/second"
              }
            }
          }
        ],
        "op": "pipette"
      }
    ]
```

Parameters

- **source** (*Well, WellGroup*) – Well or wells to distribute liquid from. If passed as a WellGroup with set_volume() called on it, liquid will be automatically be drawn from the wells specified using the fill_wells function.
- **dest** (*Well, WellGroup*) – Well or wells to distribute liquid to.
- **volume** (*str, Unit, list*) – Volume of liquid to be distributed to each destination well. If a single string or unit is passed to represent the volume, that volume will be distributed to each destination well. If a list of volumes is provided, that volume will be distributed to the corresponding well in the WellGroup provided. The length of the volumes list must therefore match the number of wells in the destination WellGroup if destination wells are recieving different volumes.
- **allow_carryover** (*bool, optional*) – specify whether the same pipette tip can be used to aspirate more liquid from source wells after the previous volume aspirated has been depleted.
- **mix_before** (*bool, optional*) – Specify whether to mix the liquid in the destination well before liquid is transferred.
- **mix_vol** (*str, Unit, optional*) – Volume to aspirate and dispense in order to mix liquid in a wells before liquid is distributed.
- **repetitions** (*int, optional*) – Number of times to aspirate and dispense in order to mix liquid in a well before liquid is distributed.
- **flowrate** (*str, Unit, optional*) – Speed at which to mix liquid in well before liquid is distributed.
- **aspirate_speed** (*str, Unit, optional*) – Speed at which to aspirate liquid from source well. May not be specified if aspirate_source is also specified. By default this is the maximum aspiration speed, with the start speed being half of the speed specified.
- **aspirate_source** (*fn, optional*) – Can't be specified if aspirate_speed is also specified.

- **dispense_speed** (*str, Unit, optional*) – Speed at which to dispense liquid into the destination well. May not be specified if dispense_target is also specified.
- **distribute_target** (*fn, optional*) – A function that contains additional parameters for distributing to target wells including depth, dispense_speed, and calibrated volume. If this parameter is specified, the same parameters will be applied to every destination well. Will supersede dispense_speed parameters if also specified.
- **pre_buffer** (*str, Unit, optional*) – Volume of air aspirated before aspirating liquid.
- **disposal_vol** (*str, Unit, optional*) – Volume of extra liquid to aspirate that will be dispensed into trash afterwards.
- **transit_vol** (*str, Unit, optional*) – Volume of air aspirated after aspirating liquid to reduce presence of bubbles at pipette tip.
- **blowout_buffer** (*bool, optional*) – If true the operation will dispense the pre_buffer along with the dispense volume. Cannot be true if disposal_vol is specified.

Raises

- `RuntimeError` – If no mix volume is specified for the mix_before instruction.
- `ValueError` – If source and destination well(s) is/are not expressed as either Wells or WellGroups.

Protocol.transfer()

`Protocol.`**`transfer`**(*source*, *dest*, *volume*, *one_source=False*, *one_tip=False*, *aspirate_speed=None*, *dispense_speed=None*, *aspirate_source=None*, *dispense_target=None*, *pre_buffer=None*, *disposal_vol=None*, *transit_vol=None*, *blowout_buffer=None*, *tip_type=None*, *new_group=False*, ***mix_kwargs*)

Transfer liquid from one specific well to another. A new pipette tip is used between each transfer step unless the "one_tip" parameter is set to True.

Example Usage:

```
    p = Protocol()
    sample_plate = p.ref("sample_plate",
                         ct32kj234l21g,
                         "96-flat",
                         storage="warm_37")


    # a basic one-to-one transfer:
    p.transfer(sample_plate.well("B3"),
               sample_plate.well("C3"),
               "20:microliter")

    # using a basic transfer in a loop:
    for i in xrange(1, 12):
      p.transfer(sample_plate.well(i-1),
                 sample_plate.well(i),
                 "10:microliter")

    # transfer liquid from each well in the first column of a 96-well
    # plate to each well of the second column using a new tip and
    # a different volume each time:
    volumes = ["5:microliter", "10:microliter", "15:microliter",
               "20:microliter", "25:microliter", "30:microliter",
```

```
            "35:microliter", "40:microliter"]

    p.transfer(sample_plate.wells_from(0,8,columnwise=True),
               sample_plate.wells_from(1,8,columnwise=True),
               volumes)

    # transfer liquid from wells A1 and A2 (which both contain the same
    # source) into each of the following 10 wells:
    p.transfer(sample_plate.wells_from("A1", 2),
               sample_plate.wells_from("A3", 10),
               "10:microliter",
               one_source=True)

    # transfer liquid from wells containing the same source to multiple
    # other wells without discarding the tip in between:
    p.transfer(sample_plate.wells_from("A1", 2),
               sample_plate.wells_from("A3", 10),
               "10:microliter",
               one_source=True,
               one_tip=True)
```

Parameters

- **source** (*Well, WellGroup*) – Well or wells to transfer liquid from. If multiple source wells are supplied and one_source is set to True, liquid will be transfered from each source well specified as long as it contains sufficient volume. Otherwise, the number of source wells specified must match the number of destination wells specified and liquid will be transfered from each source well to its corresponding destination well.
- **dest** (*Well, WellGroup*) – Well or WellGroup to which to transfer liquid. The number of destination wells must match the number of source wells specified unless one_source is set to True.
- **volume** (*str, Unit, list*) – The volume(s) of liquid to be transferred from source wells to destination wells. Volume can be specified as a single string or Unit, or can be given as a list of volumes. The length of a list of volumes must match the number of destination wells given unless the same volume is to be transferred to each destination well.
- **one_source** (*bool, optional*) – Specify whether liquid is to be transferred to destination wells from a group of wells all containing the same substance.
- **one_tip** (*bool, optional*) – Specify whether all transfer steps will use the same tip or not.
- **mix_after** (*bool, optional*) – Specify whether to mix the liquid in the destination well after liquid is transferred.
- **mix_before** (*bool, optional*) – Specify whether to mix the liquid in the source well before liquid is transferred.
- **mix_vol** (*str, Unit, optional*) – Volume to aspirate and dispense in order to mix liquid in a wells before and/or after each transfer step.
- **repetitions** (*int, optional*) – Number of times to aspirate and dispense in order to mix liquid in well before and/or after each transfer step.
- **flowrate** (*str, Unit, optional*) – Speed at which to mix liquid in well before and/or after each transfer step.
- **aspirate_speed** (*str, Unit, optional*) – Speed at which to aspirate liquid from source

well. May not be specified if aspirate_source is also specified. By default this is the maximum aspiration speed, with the start speed being half of the speed specified.

- **dispense_speed** (*str, Unit, optional*) – Speed at which to dispense liquid into the destination well. May not be specified if dispense_target is also specified.
- **aspirate_source** (*fn, optional*) – Can't be specified if aspirate_speed is also specified.
- **dispense_target** (*fn, optional*) – Same but opposite of aspirate_source.
- **pre_buffer** (*str, Unit, optional*) – Volume of air aspirated before aspirating liquid.
- **disposal_vol** (*str, Unit, optional*) – Volume of extra liquid to aspirate that will be dispensed into trash afterwards.
- **transit_vol** (*str, Unit, optional*) – Volume of air aspirated after aspirating liquid to reduce presence of bubbles at pipette tip.
- **blowout_buffer** (*bool, optional*) – If true the operation will dispense the pre_buffer along with the dispense volume. Cannot be true if disposal_vol is specified.
- **tip_type** (*str, optional*) – Type of tip to be used for the transfer operation.
- **new_group** (*bool, optional*) –

Raises

- `RuntimeError` – If more than one volume is specified as a list but the list length does not match the number of destination wells given.
- `RuntimeError` – If transferring from WellGroup to WellGroup that have different number of wells and one_source is not True.

Protocol.acoustic_transfer()

`Protocol.`**`acoustic_transfer`**(*source*, *dest*, *volume*, *one_source=False*, *droplet_size='25:nanoliter'*)

Specify source and destination wells for transfering liquid via an acoustic liquid handler. Droplet size is usually device-specific.

Parameters

- **source** (*Well, WellGroup*) – Well or wells to transfer liquid from. If multiple source wells are supplied and one_source is set to True, liquid will be transfered from each source well specified as long as it contains sufficient volume. Otherwise, the number of source wells specified must match the number of destination wells specified and liquid will be transfered from each source well to its corresponding destination well.
- **dest** (*Well, WellGroup*) – Well or WellGroup to which to transfer liquid. The number of destination wells must match the number of source wells specified unless one_source is set to True.
- **volume** (*str, Unit, list*) – The volume(s) of liquid to be transferred from source wells to destination wells. Volume can be specified as a single string or Unit, or can be given as a list of volumes. The length of a list of volumes must match the number of destination wells given unless the same volume is to be transferred to each destination well.
- **one_source** (*bool, optional*) – Specify whether liquid is to be transferred to destination wells from a group of wells all containing the same substance.
- **droplet_size** (*str, Unit, optional*) – Volume representing a droplet_size. The volume of each *transfer* group should be a multiple of this volume.

Example Usage:

```
p.acoustic_transfer(
    echo.wells(0,1).set_volume("12:nanoliter"),
    plate.wells_from(0,5), "4:nanoliter", one_source=True)
```

Autoprotocol Output:


```
"instructions": [
{
  "groups": [
    {
      "transfer": [
        {
          "volume": "0.004:microliter",
          "to": "plate/0",
          "from": "echo_plate/0"
        },
        {
          "volume": "0.004:microliter",
          "to": "plate/1",
          "from": "echo_plate/0"
        },
        {
          "volume": "0.004:microliter",
          "to": "plate/2",
          "from": "echo_plate/0"
        },
        {
          "volume": "0.004:microliter",
          "to": "plate/3",
          "from": "echo_plate/1"
        },
        {
          "volume": "0.004:microliter",
          "to": "plate/4",
          "from": "echo_plate/1"
        }
      ]
    }
  ],
  "droplet_size": "25:microliter",
  "op": "acoustic_transfer"
}]
```


Protocol.consolidate()

Protocol.**consolidate**(*sources*, *dest*, *volumes*, *allow_carryover=False*, *mix_after=False*, *mix_vol=None*, *flowrate='100:microliter/second'*, *repetitions=10*, *aspirate_speed=None*, *dispense_speed=None*, *aspirate_source=None*, *dispense_target=None*, *pre_buffer=None*, *transit_vol=None*, *blowout_buffer=None*, *tip_type=None*, *new_group=False*)

Aspirates from each source well, in order, the volume specified, then dispenses the sum volume into the target well. Be aware that the same tip will be used to aspirate from all the source wells, so if you want to avoid contaminating any of them you should use a separate transfer group. Consolidate is limited by the maximum volume of the disposable tip. If the total volume you want to dispense into the target well exceeds the volume that will fit in one tip, you must either specify *allow_carryover* to allow the tip to carry on pipetting from the

source wells after it has touched the target well, or break up your operation into multiple groups with separate tips.

Parameters

- **sources** (*Well, WellGroup*) – Well or wells to transfer liquid from.
- **dest** (Well) – Well to which to transfer consolidated liquid.
- **volumes** (*str, Unit, list*) – The volume(s) of liquid to be transferred from source well(s) to destination well. Volume can be specified as a single string or Unit, or can be given as a list of volumes. The length of a list of volumes must match the number of source wells given.
- **mix_after** (*bool, optional*) – Specify whether to mix the liquid in the destination well after liquid is transferred.
- **mix_vol** (*str, Unit, optional*) – Volume to aspirate and dispense in order to mix liquid in a wells before and/or after each transfer step.
- **repetitions** (*int, optional*) – Number of times to aspirate and dispense in order to mix liquid in well before and/or after each transfer step.
- **flowrate** (*str, Unit, optional*) – Speed at which to mix liquid in well before and/or after each transfer step.
- **speed** (*aspirate*) – Speed at which to aspirate liquid from source well. May not be specified if aspirate_source is also specified. By default this is the maximum aspiration speed, with the start speed being half of the speed specified.
- **dispense_speed** (*str, Unit, optional*) – Speed at which to dispense liquid into the destination well. May not be specified if dispense_target is also specified.
- **aspirate_source** (*fn, optional*) – Options for aspirating liquid. Cannot be specified if aspirate_speed is also specified.
- **dispense_target** (*fn, optional*) – Options for dispensing liquid. Cannot be specified if dispense_speed is also specified.
- **pre_buffer** (*str, Unit, optional*) – Volume of air aspirated before aspirating liquid.
- **transit_vol** (*str, Unit, optional*) – Volume of air aspirated after aspirating liquid to reduce presence of bubbles at pipette tip.
- **blowout_buffer** (*bool, optional*) – If true the operation will dispense the pre_buffer along with the dispense volume cannot be true if disposal_vol is specified.

Raises

- `TypeError` – If supplying more than one destination well for consolidation.
- `ValueError` – If a volume list is supplied and the length does not match the number of source wells.

Protocol.dispense()

`Protocol.`**`dispense`**(*ref*, *reagent*, *columns*, *speed_percentage=None*, *is_resource_id=False*, *step_size='5:microliter'*, *x_cassette=None*)

Dispense specified reagent to specified columns.

Example Usage:

```
p = Protocol()
sample_plate = p.ref("sample_plate",
                     None,
                     "96-flat",
                     storage="warm_37")

p.dispense(sample_plate,
           "water",
           [{"column": 0, "volume": "10:microliter"},
            {"column": 1, "volume": "20:microliter"},
            {"column": 2, "volume": "30:microliter"},
            {"column": 3, "volume": "40:microliter"},
            {"column": 4, "volume": "50:microliter"},
            {"column": 5, "volume": "60:microliter"},
            {"column": 6, "volume": "70:microliter"},
            {"column": 7, "volume": "80:microliter"},
            {"column": 8, "volume": "90:microliter"},
            {"column": 9, "volume": "100:microliter"},
            {"column": 10, "volume": "110:microliter"},
            {"column": 11, "volume": "120:microliter"}
           ])
```

Autoprotocol Output:

```
"instructions": [
    {
        "reagent": "water",
        "object": "sample_plate",
        "columns": [
            {
                "column": 0,
                "volume": "10:microliter"
            },
            {
                "column": 1,
                "volume": "20:microliter"
            },
            {
                "column": 2,
                "volume": "30:microliter"
            },
            {
                "column": 3,
                "volume": "40:microliter"
            },
            {
                "column": 4,
                "volume": "50:microliter"
            },
            {
                "column": 5,
                "volume": "60:microliter"
            },
            {
                "column": 6,
                "volume": "70:microliter"
            },
            {
                "column": 7,
```

```
                    "volume": "80:microliter"
                },
                {
                    "column": 8,
                    "volume": "90:microliter"
                },
                {
                    "column": 9,
                    "volume": "100:microliter"
                },
                {
                    "column": 10,
                    "volume": "110:microliter"
                },
                {
                    "column": 11,
                    "volume": "120:microliter"
                }
            ],
            "op": "dispense"
        }
    ]
```

Parameters

- **ref** (Container) – Container for reagent to be dispensed to.
- **reagent** (*str, well*) – Reagent to be dispensed. Use a string to specify the name or resource_id (see below) of the reagent to be dispensed. Alternatively, use a well to specify that the dispense operation must be executed using a specific aliquot as the dispense source.
- **columns** (*list*) – Columns to be dispensed to, in the form of a list of dicts specifying the column number and the volume to be dispensed to that column. Columns are expressed as integers indexed from 0. [{"column": <column num>, "volume": <volume>}, ...]
- **speed_percentage** (*int, optional*) – Integer between 1 and 100 that represents the percentage of the maximum speed at which liquid is dispensed from the reagent dispenser.
- **is_resource_id** (*bool, optional*) – If true, interprets reagent as a resource ID
- **step_size** (*str, Unit, optional*) – Specifies that the dispense operation must be executed using a peristaltic pump with the given step size. Note that the volume dispensed in each column must be an integer multiple of the step_size. Currently, step_size must be either 5 uL or 0.5 uL. If set to None, will use vendor specified defaults.
- **x_cassette** (*str, optional*) – Specifies a specific cassette to be used with this instruction. Cassette will be checked against a list of allowed values. Each cassette has a pre-defined value for step_size, and certain cassettes may require human execution.

Protocol.dispense_full_plate()

Protocol.**dispense_full_plate**(*ref*, *reagent*, *volume*, *speed_percentage=None*, *is_resource_id=False*, *step_size='5:microliter'*, *x_cassette=None*)

Dispense the specified amount of the specified reagent to every well of a container using a reagent dispenser.

Example Usage:

```
p = Protocol()
sample_plate = p.ref("sample_plate",
                     None,
                     "96-flat",
                     storage="warm_37")

p.dispense_full_plate(sample_plate,
                      "water",
                      "100:microliter")
```

Autoprotocol Output:


```
"instructions": [
    {
      "reagent": "water",
      "object": "sample_plate",
      "columns": [
        {
          "column": 0,
          "volume": "100:microliter"
        },
        {
          "column": 1,
          "volume": "100:microliter"
        },
        {
          "column": 2,
          "volume": "100:microliter"
        },
        {
          "column": 3,
          "volume": "100:microliter"
        },
        {
          "column": 4,
          "volume": "100:microliter"
        },
        {
          "column": 5,
          "volume": "100:microliter"
        },
        {
          "column": 6,
          "volume": "100:microliter"
        },
        {
          "column": 7,
          "volume": "100:microliter"
        },
        {
          "column": 8,
          "volume": "100:microliter"
        },
        {
          "column": 9,
          "volume": "100:microliter"
        },
        {
          "column": 10,
```

```
                "volume": "100:microliter"
              },
              {
                "column": 11,
                "volume": "100:microliter"
              }
            ],
            "op": "dispense"
          }
      ]
```

Parameters

- **ref** (Container) – Container for reagent to be dispensed to.
- **reagent** (*str, well*) – Reagent to be dispensed. Use a string to specify the name or resource_id (see below) of the reagent to be dispensed. Alternatively, use a well to specify that the dispense operation must be executed using a specific aliquot as the dispense source.
- **volume** (*Unit, str*) – Volume of reagent to be dispensed to each well
- **speed_percentage** (*int, optional*) – Integer between 1 and 100 that represents the percentage of the maximum speed at which liquid is dispensed from the reagent dispenser.
- **is_resource_id** (*bool, optional*) – If true, interprets reagent as a resource ID
- **step_size** (*str, Unit, optional*) – Specifies that the dispense operation must be executed using a peristaltic pump with the given step size. Note that the volume dispensed in each column must be an integer multiple of the step_size. Currently, step_size must be either 5 uL or 0.5 uL. If set to None, will use vendor specified defaults.
- **x_cassette** (*str, optional*) – Specifies a specific cassette to be used with this instruction. Cassette will be checked against a list of allowed values. Each cassette has a pre-defined value for step_size, and certain cassettes may require human execution.

Protocol.stamp()

Protocol.**stamp**(*source_origin*, *dest_origin*, *volume*, *shape={'rows': 8, 'columns': 12}*, *mix_before=False*, *mix_after=False*, *mix_vol=None*, *repetitions=10*, *flowrate='100:microliter/second'*, *aspirate_speed=None*, *dispense_speed=None*, *aspirate_source=None*, *dispense_target=None*, *pre_buffer=None*, *disposal_vol=None*, *transit_vol=None*, *blowout_buffer=None*, *one_source=False*, *one_tip=False*, *new_group=False*)

Note: the way this method now works is significantly different to the way it has in previous versions, please make sure to read the documentation below and adjust existing scripts utilizing stamp() accordingly

A stamp instruction consists of a list of groups of transfers, each of which specifies from and to well references (ref/well_index) representing the top-left well or origin of a specified shape.

The volume field defines the volume of liquid that will be aspirated from every well of the shape specified starting at the from field and dispensed into the corresponding wells starting at the to field.

Currently, the shape field may only be a rectangle object defined by rows and columns attributes representing the number of contiguous tip rows and columns to transfer.

The shape parameter is optional and will default to a full 8 rows by 12 columns. The tip_layout field refers to the SBS compliant layout of tips, is optional, and will default to the layout of a 96 tip box.

The following plate types are currently supported: 96 and 384.

Example Usage:

```
p = Protocol()

plate_1_96 = p.ref("plate_1_96", None, "96-flat", discard=True)
plate_2_96 = p.ref("plate_2_96", None, "96-flat", discard=True)
plate_1_384 = p.ref("plate_1_384", None, "384-flat", discard=True)
plate_2_384 = p.ref("plate_2_384", None, "384-flat", discard=True)

# A full-plate transfer between two 96 or 384-well plates
p.stamp(plate_1_96, plate_2_96, "10:microliter")
p.stamp(plate_1_384, plate_2_384, "10:microliter")

# Defining shapes for selective stamping:
row_rectangle = dict(rows=1, columns=12)
two_column_rectangle = dict(rows=8, columns=2)

# A transfer from the G row to the H row of another 96-well plate
p.stamp(plate_1_96.well("G1"), plate_2_96.well("H1"),
"10:microliter", row_rectangle)

# A 2-column transfer from columns 1,2 of a 96-well plate to
#columns 2,4 of a 384-well plate
p.stamp(plate_1_96.well("A1"), plate_1_384.wells_from("A2", 2,
columnwise=True), "10:microliter", two_column_rectangle)

# A 2-row transfer from rows 1,2 of a 384-well plate to rows 2,3
#of a 96-well plate
p.stamp(plate_1_384.wells(["A1", "A2", "B1", "B2"]), plate_1_96.
wells(["B1", "B1", "C1", "C1"]), "10:microliter",
shape=row_rectangle)
```

Parameters

- **source_origin** (*Container, Well, WellGroup, List of Wells*) – Top-left well or wells where the rows/columns will be defined with respect to the source transfer. If a container is specified, stamp will be applied to all quadrants of the container.
- **dest_origin** (*Container, Well, WellGroup, List of Wells*) – Top-left well or wells where the rows/columns will be defined with respect to the destination transfer. If a container is specified, stamp will be applied to all quadrants of the container
- **volume** (*str, Unit, list*) – Volume(s) of liquid to move from source plate to destination plate. Volume can be specified as a single string or Unit, or can be given as a list of volumes. The length of a list of volumes must match the number of destination wells given unless the same volume is to be transferred to each destination well.
- **shape** (*dictionary, list, optional*) – The shape(s) parameter is optional and will default to a rectangle corresponding to a full 96-well plate (8 rows by 12 columns). The rows and columns will be defined wrt the specified origin. The length of a list of shapes must match the number of destination wells given unless the same shape is to be used for each destination well. If the length of shape is greater than 1, one_tip=False.

  Example

```
rectangle = {}
rectangle["rows"] = 8
rectangle["columns"] = 12
```

- **mix_after** (*bool, optional*) – Specify whether to mix the liquid in destination wells after liquid is transferred.
- **mix_before** (*bool, optional*) – Specify whether to mix the liquid in source wells before liquid is transferred.
- **mix_vol** (*str, Unit, optional*) – Volume to aspirate and dispense in order to mix liquid in wells before and/or after it is transfered.
- **repetitions** (*int, optional*) – Number of times to aspirate and dispense in order to mix liquid in wells before and/or after it is transfered.
- **flowrate** (*str, Unit, optional*) – Speed at which to mix liquid in well before and/or after each transfer step in units of "microliter/second".
- **dispense_speed** (*str, Unit, optional*) – Speed at which to dispense liquid into the destination well. May not be specified if dispense_target is also specified.
- **aspirate_source** (*fn, optional*) – Can't be specified if aspirate_speed is also specified.
- **dispense_target** (*fn, optional*) – Same but opposite of aspirate_source.
- **pre_buffer** (*str, Unit, optional*) – Volume of air aspirated before aspirating liquid.
- **disposal_vol** (*str, Unit, optional*) – Volume of extra liquid to aspirate that will be dispensed into trash afterwards.
- **transit_vol** (*str, Unit, optional*) – Volume of air aspirated after aspirating liquid to reduce presence of bubbles at pipette tip.
- **blowout_buffer** (*bool, optional*) – If true the operation will dispense the pre_buffer along with the dispense volume. Cannot be true if disposal_vol is specified.
- **one_source** (*bool, optional*) – Specify whether liquid is to be transferred to destination origins from a group of origins all containing the same substance. Volume of all wells in the shape must be equal to or greater than the volume in the origin well. Specifying origins with overlapping shapes can produce undesireable effects.
- **one_tip** (*bool, optional*) – Specify whether all transfer steps will use the same tip or not. If multiple different shapes are used, one_tip cannot be true.
- **new_group** (*bool, optional*) – Example

```
p.stamp(plate_1_96.well("A1"), plate_2_96.well("A1"),
"10:microliter")
p.stamp(plate_1_96.well("A1"), plate_2_96.well("A1"),
"10:microliter")
```

Autoprotocol Output:

```
"instructions": [
    {
      "groups": [
        {
          "transfer": [
            {
              "volume": "10.0:microliter",
              "to": "plate_2_96/0",
              "from": "plate_1_96/0"
            }
          ],
          "shape": {
```

```
                        "rows": 8,
                        "columns": 12
                      },
                      "tip_layout": 96
                    }
                  ],
                  "op": "stamp"
                },
                {
                  "groups": [
                    {
                      "transfer": [
                        {
                          "volume": "10.0:microliter",
                          "to": "plate_2_96/0",
                          "from": "plate_1_96/0"
                        }
                      ],
                      "shape": {
                        "rows": 8,
                        "columns": 12
                      },
                      "tip_layout": 96
                    }
                  ],
                  "op": "stamp"
                }
              ]
```

Protocol.illuminaseq()

`Protocol.`**`illuminaseq`**(*flowcell*, *lanes*, *sequencer*, *mode*, *index*, *library_size*, *dataref*, *cycles=None*)

Load aliquots into specified lanes for Illumina sequencing. The specified aliquots should already contain the appropriate mix for sequencing and require a library concentration reported in ng/uL.

Example Usage:

```
p = Protocol()
sample_wells = p.ref(
    "test_plate", None, "96-pcr", discard=True).wells_from(0, 8)

p.illuminaseq("PE",
              [
                  {"object": sample_wells[0], "library_concentration": 1.0},
                  {"object": sample_wells[1], "library_concentration": 5.32},
                  {"object": sample_wells[2], "library_concentration": 54},
                  {"object": sample_wells[3], "library_concentration": 20},
                  {"object": sample_wells[4], "library_concentration": 23},
                  {"object": sample_wells[5], "library_concentration": 23},
                  {"object": sample_wells[6], "library_concentration": 21},
                  {"object": sample_wells[7], "library_concentration": 62}
              ],
              "hiseq", "rapid", 'none', 250, "my_illumina")
```

Autoprotocol Output:

```
    "instructions": [
        {
          "dataref": "my_illumina",
          "index": "none",
          "lanes": [
            {
              "object": "test_plate/0",
              "library_concentration": 1.0
            },
            {
              "object": "test_plate/1",
              "library_concentration": 5.32
            },
            {
              "object": "test_plate/2",
              "library_concentration": 54
            },
            {
              "object": "test_plate/3",
              "library_concentration": 20
            },
            {
              "object": "test_plate/4",
              "library_concentration": 23
            },
            {
              "object": "test_plate/5",
              "library_concentration": 23
            },
            {
              "object": "test_plate/6",
              "library_concentration": 21
            },
            {
              "object": "test_plate/7",
              "library_concentration": 62
            }
          ],
          "flowcell": "PE",
          "mode": "mid",
          "sequencer": "hiseq",
          "library_size": 250,
          "op": "illumina_sequence"
        }
      ]
```

Parameters

- **flowcell** (*str*) – Flowcell designation: “SR” or ” “PE”
- **lanes** (*list of dicts*) –

```
              "lanes": [{
                    "object": aliquot, Well,
                    "library_concentration": decimal, // ng/uL
                  },
                  {...}]
```

- **sequencer** (*str*) – Sequencer designation: “miseq”, “hiseq” or “nextseq”
- **mode** (*str*) – Mode designation: “rapid”, “mid” or “high”
- **index** (*str*) – Index designation: “single”, “dual” or “none”
- **library_size** (*integer*) – Library size expressed as an integer of basepairs
- **dataref** (*str*) – Name of sequencing dataset that will be returned.

Raises

- TypeError: – If index and dataref are not of type str.
- TypeError: – If library_concentration is not a number.
- TypeError: – If library_size is not an integer.
- ValueError: – If flowcell, sequencer, mode, index are not of type a valid option.
- ValueError: – If number of lanes specified is more than the maximum lanes of the specified type of sequencer.

Protocol.sangerseq()

Protocol.**sangerseq**(*cont*, *wells*, *dataref*, *type='standard'*, *primer=None*)

Send the indicated wells of the container specified for Sanger sequencing. The specified wells should already contain the appropriate mix for sequencing, including primers and DNA according to the instructions provided by the vendor.

Example Usage:

```
p = Protocol()
sample_plate = p.ref("sample_plate",
                     None,
                     "96-flat",
                     storage="warm_37")

p.sangerseq(sample_plate,
            sample_plate.wells_from(0,5).indices(),
            "seq_data_022415")
```

Autoprotocol Output:

```
"instructions": [
    {
      "dataref": "seq_data_022415",
      "object": "sample_plate",
      "wells": [
        "A1",
        "A2",
        "A3",
        "A4",
        "A5"
      ],
      "op": "sanger_sequence"
    }
  ]
```

Parameters

- **cont** (*Container, str*) – Container with well(s) that contain material to be sequenced.

- **type** (*str*) – Type of sequencing reaction to take place ("standard" or "rca"), defaults to "standard"
- **wells** (*list, WellGroup, Well*) – WellGroup of wells to be measured or a list of well references in the form of ["A1", "B1", "C5", ...]
- **primer** (*container*) – Tube containing sufficient primer for all RCA reactions. This field will be ignored if you specify the sequencing type as "standard". Tube containing sufficient primer for all RCA reactions
- **dataref** (*str*) – Name of sequencing dataset that will be returned.

Protocol.mix()

`Protocol.`**`mix`**(*well*, *volume='50:microliter'*, *speed='100:microliter/second'*, *repetitions=10*, *one_tip=False*)

Mix specified well using a new pipette tip

Example Usage:

```
p = Protocol()
sample_source = p.ref("sample_source",
                      None,
                      "micro-1.5",
                      storage="cold_20")

p.mix(sample_source.well(0), volume="200:microliter",
      repetitions=25)
```

Autoprotocol Output:

```
"instructions": [
    {
      "groups": [
        {
          "mix": [
            {
              "volume": "200:microliter",
              "well": "sample_source/0",
              "repetitions": 25,
              "speed": "100:microliter/second"
            }
          ]
        }
      ],
      "op": "pipette"
    }
  ]
}
```

Parameters

- **well** (*Well, WellGroup, list of Wells*) – Well(s) to be mixed. If a WellGroup is passed, each well in the group will be mixed using the specified parameters.
- **volume** (*str, Unit, optional*) – volume of liquid to be aspirated and expelled during mixing
- **speed** (*str, Unit, optional*) – flowrate of liquid during mixing
- **repetitions** (*int, optional*) – number of times to aspirate and expell liquid during mixing

- **one_tip** (*bool*) – mix all wells with a single tip

Protocol.spin()

`Protocol.`**`spin`**(*ref*, *acceleration*, *duration*, *flow_direction=None*, *spin_direction=None*)

Apply acceleration to a container.

Example Usage:

```
p = Protocol()
sample_plate = p.ref("sample_plate",
                     None,
                     "96-flat",
                     storage="warm_37")

p.spin(sample_plate, "1000:g", "20:minute", flow_direction="outward)
```

Autoprotocol Output:

```
"instructions": [
    {
      "acceleration": "1000:g",
      "duration": "20:minute",
      "flow_direction": "outward",
      "spin_direction": [
        "cw",
        "ccw"
      ]
      "object": "sample_plate",
      "op": "spin"
    }
]
```

Parameters

- **ref** (Container) – The container to be centrifuged.
- **acceleration** (*str*) – Acceleration to be applied to the plate, in units of *g* or *meter/second^2*.
- **duration** (*str, Unit*) – Length of time that acceleration should be applied.
- **flow_direction** (*str*) – Specifies the direction contents will tend toward with respect to the container. Valid directions are “inward” and “outward”, default value is “inward”.
- **spin_direction** (*list of strings*) – A list of “cw” (clockwise), “cww” (counterclockwise). For each element in the list, the container will be spun in the stated direction for the set “acceleration” and “duration”. Default values are derived from the “flow_direction” parameter. If “flow_direction” is “outward”, then “spin_direction” defaults to [”cw”, “ccw”]. If “flow_direction” is “inward”, then “spin_direction” defaults to [”cw”].

Raises

- TypeError: – If ref to spin is not of type Container.
- TypeError: – If spin_direction or flow_direction are not properly formatted.
- ValueError: – If spin_direction or flow_direction do not have appropriate values.

Protocol.thermocycle()

Protocol.**thermocycle**(*ref*, *groups*, *volume='10:microliter'*, *dataref=None*, *dyes=None*, *melting_start=None*, *melting_end=None*, *melting_increment=None*, *melting_rate=None*)

Append a Thermocycle instruction to the list of instructions, with groups is a list of dicts in the form of:

```
"groups": [{
    "cycles": integer,
    "steps": [{
      "duration": duration,
      "temperature": temperature,
      "read": boolean // optional (default false)
    },{
      "duration": duration,
      "gradient": {
        "top": temperature,
        "bottom": temperature
      },
      "read": boolean // optional (default false)
    }]
}],
```

Thermocycle can also be used for either conventional or row-wise gradient PCR as well as qPCR. Refer to the examples below for details.

Example Usage:

To thermocycle a container according to the protocol:

- **1 cycle:**
  - 95 degrees for 5 minutes
- **30 cycles:**
  - 95 degrees for 30 seconds
  - 56 degrees for 20 seconds
  - 72 degrees for 30 seconds
- **1 cycle:**
  - 72 degrees for 10 minutes

```
p = Protocol()
sample_plate = p.ref("sample_plate",
                     None,
                     "96-pcr",
                     storage="warm_37")

# a plate must be sealed before it can be thermocycled
p.seal(sample_plate)

p.thermocycle(sample_plate,
              [
               {"cycles": 1,
                "steps": [{
                   "temperature": "95:celsius",
                   "duration": "5:minute",
                   }]
```

```
                },
                {"cycles": 30,
                    "steps": [
                        {"temperature": "95:celsius",
                         "duration": "30:second"},
                        {"temperature": "56:celsius",
                         "duration": "20:second"},
                        {"temperature": "72:celsius",
                         "duration": "20:second"}
                        ]
            },
                {"cycles": 1,
                    "steps": [
                    {"temperature": "72:celsius", "duration":"10:minute"}]
                }
            ])
```

Autoprotocol Output:

```
"instructions": [
{
  "object": "sample_plate",
  "op": "seal"
},
{
  "volume": "10:microliter",
  "dataref": null,
  "object": "sample_plate",
  "groups": [
    {
      "cycles": 1,
      "steps": [
        {
          "duration": "5:minute",
          "temperature": "95:celsius"
        }
      ]
    },
    {
      "cycles": 30,
      "steps": [
        {
          "duration": "30:second",
          "temperature": "95:celsius"
        },
        {
          "duration": "20:second",
          "temperature": "56:celsius"
        },
        {
          "duration": "20:second",
          "temperature": "72:celsius"
        }
      ]
    },
    {
      "cycles": 1,
      "steps": [
        {
```

```
            "duration": "10:minute",
            "temperature": "72:celsius"
          }
        ]
      }
    ],
    "op": "thermocycle"
}]
```

To gradient thermocycle a container according to the protocol:

- **1 cycle:**
  - 95 degrees for 5 minutes
- **30 cycles:**
  - 95 degrees for 30 seconds

    Top Row: * 55 degrees for 20 seconds Bottom Row: * 65 degrees for 20 seconds
  - 72 degrees for 30 seconds
- **1 cycle:**
  - 72 degrees for 10 minutes

```
p = Protocol()
sample_plate = p.ref("sample_plate",
                     None,
                     "96-pcr",
                     storage="warm_37")

# a plate must be sealed before it can be thermocycled
p.seal(sample_plate)

p.thermocycle(sample_plate,
              [
               {"cycles": 1,
                "steps": [{
                   "temperature": "95:celsius",
                   "duration": "5:minute",
                   }]
                },
                {"cycles": 30,
                    "steps": [
                      {
                        "duration": "30:second",
                        "temperature": "95:celsius"
                      },
                      {
                       "duration": "20:second",
                       "gradient": {
                          "top": "56:celsius",
                          "bottom": "58:celsius"
                        }
                      },
                      {
                        "duration": "20:second",
                        "temperature": "72:celsius"
                      }
```

```
                        ]
                },
                    {"cycles": 1,
                         "steps": [{"temperature": "72:celsius", "duration":"10:minute"}]
                    }
                ])
```

To conduct a qPCR, at least one dye type and the dataref field has to be specified. The example below uses SYBR dye and the following temperature profile:

- **1 cycle:**
  - 95 degrees for 3 minutes
- **40 cycles:**
  - 95 degrees for 10 seconds
  - 60 degrees for 30 seconds (Read during extension)

```
p = Protocol()
sample_plate = p.ref("sample_plate",
                     None,
                     "96-pcr",
                     storage="warm_37")

# a plate must be sealed before it can be thermocycled
p.seal(sample_plate)

p.thermocycle(sample_plate,
              [{"cycles": 1,
                "steps": [{
                    "temperature": "95:celsius",
                    "duration": "3:minute",
                    }]
                },
                {"cycles": 40,
                    "steps": [
                        {"temperature": "95:celsius",
                         "duration": "10:second",
                         "read": False},
                        {"temperature": "60:celsius",
                         "duration": "30:second",
                         "read": True},
                        ]
              }],
              dataref = "my_qpcr_data",
              dyes = {
                "SYBR": sample_plate.all_wells().indices()}
              )
```

Parameters

- **ref** (Container) – Container to be thermocycled.
- **groups** (*list of dicts*) – List of thermocycling instructions formatted as above
- **volume** (*str, Unit, optional*) – Volume contained in wells being thermocycled
- **dataref** (*str, optional*) – Name of dataref representing read data if performing qPCR
- **dyes** (*dict, optional*) – Dictionary mapping dye types to the wells they're used in

- **melting_start** (*str, Unit*) – Temperature at which to start the melting curve.
- **melting_end** (*str, Unit*) – Temperature at which to end the melting curve.
- **melting_increment** (*str, Unit*) – Temperature by which to increment the melting curve. Accepted increment values are between 0.1 and 9.9 degrees celsius.
- **melting_rate** (*str, Unit*) – Specifies the duration of each temperature step in the melting curve.

Raises

- AttributeError: – If groups are not properly formatted
- TypeError: – If ref to thermocycle is not of type Container.

Protocol.incubate()

`Protocol.`**`incubate`**(*ref*, *where*, *duration*, *shaking=False*, *co2=0*, *uncovered=False*, *target_temperature=None*, *shaking_params=None*)

Move plate to designated thermoisolater or ambient area for incubation for specified duration.

Example Usage:

```
p = Protocol()
sample_plate = p.ref("sample_plate",
                     None,
                     "96-pcr",
                     storage="warm_37")

# a plate must be sealed/covered before it can be incubated
p.seal(sample_plate)
p.incubate(sample_plate, "warm_37", "1:hour", shaking=True)
```

Autoprotocol Output:

```
"instructions": [
    {
       "object": "sample_plate",
       "op": "seal"
    },
    {
        "duration": "1:hour",
        "where": "warm_37",
        "object": "sample_plate",
        "shaking": true,
        "op": "incubate",
        "co2_percent": 0
    }
  ]
```

Parameters

- **ref** (*Ref, str*) – The container to be incubated
- **where** (*{"ambient", "warm_37", "cold_4", "cold_20", "cold_80"}*) – Temperature at which to incubate specified container
- **duration** (*Unit, str*) – Length of time to incubate container

- **shaking** (*bool, optional*) – Specify whether or not to shake container if available at the specified temperature
- **target_tempterature** (*Unit, str, optional*) – Specify a target temperature for a device (eg. an incubating block) to reach during the specified duration.
- **shaking_params** (*dict, optional*) – Specifify "path" and "frequency" of shaking parameters to be used with compatible devices (eg. thermoshakes)

Protocol.absorbance()

`Protocol.`**`absorbance`**(*ref*, *wells*, *wavelength*, *dataref*, *flashes=25*, *incubate_before=None*, *temperature=None*)

Read the absorbance for the indicated wavelength for the indicated wells. Append an Absorbance instruction to the list of instructions for this Protocol object.

Example Usage:

```
p = Protocol()
sample_plate = p.ref("sample_plate",
                     None,
                     "96-flat",
                     storage="warm_37")

p.absorbance(sample_plate, sample_plate.wells_from(0,12),
             "600:nanometer", "test_reading", flashes=50)
```

Autoprotocol Output:

```
"instructions": [
    {
      "dataref": "test_reading",
      "object": "sample_plate",
      "wells": [
        "A1",
        "A2",
        "A3",
        "A4",
        "A5",
        "A6",
        "A7",
        "A8",
        "A9",
        "A10",
        "A11",
        "A12"
      ],
      "num_flashes": 50,
      "wavelength": "600:nanometer",
      "op": "absorbance"
    }
  ]
```

Parameters

- **ref** (*str, Ref*) –
- **wells** (*list, WellGroup, Well*) – WellGroup of wells to be measured or a list of well references in the form of ["A1", "B1", "C5", ...]

- **wavelength** (*str, Unit*) – wavelength of light absorbance to be read for the indicated wells
- **dataref** (*str*) – name of this specific dataset of measured absorbances
- **flashes** (*int, optional*) –
- **temperature** (*str, Unit, optional*) – set temperature to heat plate reading chamber
- **incubate_before** (*dict, optional*) – incubation prior to reading if desired

```
{
  "shaking": {
    "amplitude": str, Unit
    "orbital": bool
    }
"duration": str, Unit
}
```

Protocol.fluorescence()

`Protocol.`**`fluorescence`**(*ref*, *wells*, *excitation*, *emission*, *dataref*, *flashes=25*, *temperature=None*, *gain=None*, *incubate_before=None*)

Read the fluoresence for the indicated wavelength for the indicated wells. Append a Fluorescence instruction to the list of instructions for this Protocol object.

Example Usage:

```
p = Protocol()
sample_plate = p.ref("sample_plate",
                     None,
                     "96-flat",
                     storage="warm_37")

p.fluorescence(sample_plate, sample_plate.wells_from(0,12),
               excitation="587:nanometer", emission="610:nanometer",
               dataref="test_reading")
```

Autoprotocol Output:

```
"instructions": [
    {
      "dataref": "test_reading",
      "excitation": "587:nanometer",
      "object": "sample_plate",
      "emission": "610:nanometer",
      "wells": [
        "A1",
        "A2",
        "A3",
        "A4",
        "A5",
        "A6",
        "A7",
        "A8",
        "A9",
        "A10",
        "A11",
        "A12"
      ],
```

```
        "num_flashes": 25,
        "op": "fluorescence"
      }
    ]
```

Parameters

- **ref** (*str, Container*) – Container to plate read.
- **wells** (*list, WellGroup, Well*) – WellGroup of wells to be measured or a list of well references in the form of [”A1”, “B1”, “C5”, ...]
- **excitation** (*str, Unit*) – Wavelength of light used to excite the wells indicated
- **emission** (*str, Unit*) – Wavelength of light to be measured for the indicated wells
- **dataref** (*str*) – Name of this specific dataset of measured absorbances
- **flashes** (*int, optional*) – Number of flashes.
- **temperature** (*str, Unit, optional*) – set temperature to heat plate reading chamber
- **gain** (*float, optional*) – float between 0 and 1, multiplier, gain=0.2 of maximum signal amplification
- **incubate_before** (*dict, optional*) – incubation prior to reading if desired

```
{
  "shaking": {
    "amplitude": str, Unit
    "orbital": bool
    }
"duration": str, Unit
}
```

Protocol.luminescence()

`Protocol.`**`luminescence`**(*ref*, *wells*, *dataref*, *incubate_before=None*, *temperature=None*)

Read luminescence of indicated wells.

Example Usage:

```
p = Protocol()
sample_plate = p.ref("sample_plate",
                     None,
                     "96-flat",
                     storage="warm_37")

p.luminescence(sample_plate, sample_plate.wells_from(0,12),
               "test_reading")
```

Autoprotocol Output:

```
"instructions": [
    {
      "dataref": "test_reading",
      "object": "sample_plate",
      "wells": [
        "A1",
        "A2",
```

```
            "A3",
            "A4",
            "A5",
            "A6",
            "A7",
            "A8",
            "A9",
            "A10",
            "A11",
            "A12"
          ],
          "op": "luminescence"
        }
      ]
```

Parameters

- **ref** (*str, Container*) – Container to plate read.
- **wells** (*list, WellGroup, Well*) – WellGroup of wells to be measured or a list of well references in the form of [”A1”, “B1”, “C5”, ...]
- **dataref** (*str*) – Name of this dataset of measured luminescence readings.
- **temperature** (*str, Unit, optional*) – set temperature to heat plate reading chamber
- **incubate_before** (*dict, optional*) – incubation prior to reading if desired

```
{
  "shaking": {
    "amplitude": str, Unit
    "orbital": bool
    }
"duration": str, Unit
}
```

Protocol.gel_separate()

`Protocol.`**`gel_separate`**(*wells*, *volume*, *matrix*, *ladder*, *duration*, *dataref*)

Separate nucleic acids on an agarose gel.

Example Usage:

```
p = Protocol()
sample_plate = p.ref("sample_plate",
                     None,
                     "96-flat",
                     storage="warm_37")

p.gel_separate(sample_plate.wells_from(0,12), "10:microliter",
               "agarose(8,0.8%)", "ladder1", "11:minute",
               "genotyping_030214")
```

Autoprotocol Output:

```
"instructions": [
    {
      "dataref": "genotyping_030214",
```

```
          "matrix": "agarose(8,0.8%)",
          "volume": "10:microliter",
          "ladder": "ladder1",
          "objects": [
            "sample_plate/0",
            "sample_plate/1",
            "sample_plate/2",
            "sample_plate/3",
            "sample_plate/4",
            "sample_plate/5",
            "sample_plate/6",
            "sample_plate/7",
            "sample_plate/8",
            "sample_plate/9",
            "sample_plate/10",
            "sample_plate/11"
          ],
          "duration": "11:minute",
          "op": "gel_separate"
        }
    ]
```

Parameters

- **wells** (*list, WellGroup, Well*) – List of wella or WellGroup containing wells to be separated on gel.
- **volume** (*str, Unit*) – Volume of liquid to be transferred from each well specified to a lane of the gel.
- **matrix** (*str*) – Matrix (gel) in which to gel separate samples
- **ladder** (*str*) – Ladder by which to measure separated fragment size
- **duration** (*str, Unit*) – Length of time to run current through gel.
- **dataref** (*str*) – Name of this set of gel separation results.

Protocol.gel_purify()

Protocol.**gel_purify**(*extracts*, *volume*, *matrix*, *ladder*, *dataref*)

Separate nucleic acids on an agarose gel and purify according to parameters. If gel extract lanes are not specified, they will be sequentially ordered and purified on as many gels as necessary.

Each element in extracts specifies a source loaded in a single lane of gel with a list of bands that will be purified from that lane. If the same source is to be run on separate lanes, a new dictionary must be added to extracts. It is also possible to add an element to extract with a source but without a list of bands. In that case, the source will be run in a lane without extraction.

Example Usage:

```
p = Protocol()
sample_wells = p.ref("test_plate", None, "96-pcr",
                     discard=True).wells_from(0, 8)
extract_wells = [p.ref("extract_" + str(i.index), None,
                       "micro-1.5", storage="cold_4").well(0)
                 for i in sample_wells]
```

```
extracts = [make_gel_extract_params(
                w,
                make_band_param(
                    "TE",
                    "5:microliter",
                    80,
                    79,
                    extract_wells[i]))
                for i, w in enumerate(sample_wells)]

p.gel_purify(extracts, "10:microliter",
             "size_select(8,0.8%)", "ladder1",
             "gel_purify_example")
```

Autoprotocol Output:

For extracts[0]

```
{
  "band_list": [
    {
      "band_size_range": {
        "max_bp": 80,
        "min_bp": 79
      },
      "destination": Well(Container(extract_0), 0, None),
      "elution_buffer": "TE",
      "elution_volume": "Unit(5.0, 'microliter')"
    }
  ],
  "gel": None,
  "lane": None,
  "source": Well(Container(test_plate), 0, None)
}
```

Parameters

- **extracts** (*List of dicts*) – Dictionary containing parameters for gel extraction, must be in the form of:

```
[
  {
  "band_list": [
    {
      "band_size_range": {
        "max_bp": int,
        "min_bp": int
      },
      "destination": Well,
      "elution_buffer": str,
      "elution_volume": Volume
    }
  ],
  "gel": int or None,
  "lane": int or None,
  "source": Well
  }
]
```

util.make_gel_extract_params() and util.make_band_param() can be used to create these dictionaries

- **band_list** (*list of dicts*) – List of bands to be extracted from the lane
- **band_size_range** (*dict*) – Dictionary for the size range of the band to be extracted
- **max_bp** (*int*) – Maximum size for the band
- **min_bp** (*int*) – Minimum size for the band
- **destination** (Well) – Well to place the extracted material
- **elution_buffer** (*str*) – Buffer to use to extract the band, commonly "water"
- **elution_volume** (*str, Unit*) – Volume of elution_buffer to extract the band into
- **gel** (*int*) – Integer identifier for the gel if using multiple gels
- **lane** (*int*) – Integer identifier for the lane of a gel to run the source
- **source** (Well) – Well from whcih to purify the material
- **volume** (*str, Unit*) – Volume of liquid to be transferred from each well specified to a lane of the gel.
- **matrix** (*str*) – Matrix (gel) in which to gel separate samples
- **ladder** (*str*) – Ladder by which to measure separated fragment size
- **dataref** (*str*) – Name of this set of gel separation results.

Raises

- RuntimeError: – If matrix is not properly formatted.
- AttributeError: – If extract parameters are not a list of dictionaries.
- KeyError: – If extract parameters do not contain the specified parameter keys.
- ValueError: – If min_bp is greater than max_bp.
- ValueError: – If extract destination is not of type Well.
- ValueError: – If extract elution volume is not of type Unit
- ValueError: – if extract elution volume is not greater than 0.
- RuntimeError: – If gel extract lanes are set for some but not all extract wells.
- RuntimeError: – If all samples do not fit on single gel type.
- TypeError: – If lane designated for gel extracts is not an integer.
- RuntimeError: – If designated lane index is outside lanes within the gel.
- RuntimeError: – If lanes not designated and number of extracts not equal to number of samples.

Protocol.seal()

`Protocol.`**`seal`**(*ref*, *type=None*)

Seal indicated container using the automated plate sealer.

Example Usage:

```
p = Protocol()
sample_plate = p.ref("sample_plate",
                     None,
                     "96-pcr",
                     storage="warm_37")

p.seal(sample_plate)
```

Autoprotocol Output:

```
"instructions": [
    {
      "object": "sample_plate",
      "type": "ultra-clear"
      "op": "seal"
    }
  ]
```

Parameters

- **ref** (Container) – Container to be sealed
- **type** (*str*) – Seal type to be used, such as “ultra-clear” or “foil”.

Raises

- `TypeError` – If ref is not of type Container.
- `RuntimeError` – If container type does not have *seal* capability.
- `RuntimeError` – If seal is not a valid seal type.
- `RuntimeError` – If container is already covered with a lid.

Protocol.unseal()

`Protocol.`**`unseal`**`(`*ref*`)`

Remove seal from indicated container using the automated plate unsealer.

Example Usage:

```
p = Protocol()
sample_plate = p.ref("sample_plate",
                     None,
                     "96-pcr",
                     storage="warm_37")
# a plate must be sealed to be unsealed
p.seal(sample_plate)

p.unseal(sample_plate)
```

Autoprotocol Output:

```
"instructions": [
    {
      "object": "sample_plate",
      "op": "seal",
      "type": "ultra-clear"
    },
    {
```

```
        "object": "sample_plate",
        "op": "unseal"
      }
    ]
```

Parameters **ref** (Container) – Container to be unsealed.

Raises

- TypeError – If ref is not of type Container.
- RuntimeError – If container is covered with a lid not a seal.

Protocol.cover()

Protocol.**cover**(*ref*, *lid=None*)

Place specified lid type on specified container

Example Usage:

```
p = Protocol()
sample_plate = p.ref("sample_plate",
                     None,
                     "96-flat",
                     storage="warm_37")
p.cover(sample_plate, lid="universal")
```

Autoprotocol Output:

```
"instructions": [
    {
      "lid": "universal",
      "object": "sample_plate",
      "op": "cover"
    }
  ]
```

Parameters

- **ref** (Container) – Container to be convered.
- **lid** (*str, optional*) – Type of lid to cover the container. Must be a valid lid type for the container type.

Raises

- TypeError – If ref is not of type Container.
- RuntimeError – If container type does not have *cover* capability.
- RuntimeError – If lid is not a valid lid type.
- RuntimeError – If container is already sealed with a seal.

Protocol.uncover()

Protocol.**uncover**(*ref*)

Remove lid from specified container

Example Usage:

```
p = Protocol()
sample_plate = p.ref("sample_plate",
                     None,
                     "96-flat",
                     storage="warm_37")
# a plate must have a cover to be uncovered
p.cover(sample_plate, lid="universal")

p.uncover(sample_plate)
```

Autoprotocol Output:

```
"instructions": [
    {
      "lid": "universal",
      "object": "sample_plate",
      "op": "cover"
    },
    {
      "object": "sample_plate",
      "op": "uncover"
    }
  ]
```

Parameters **ref** (Container) – Container to remove lid.

Raises

- TypeError – If ref is not of type Container.
- RuntimeError – If container is sealed with a seal not covered with a lid.

Protocol.flow_analyze()

Protocol.**flow_analyze**(*dataref*, *FSC*, *SSC*, *neg_controls*, *samples*, *colors=None*, *pos_controls=None*)

Perform flow cytometry. The instruction will be executed within the voltage range specified for each channel, optimized for the best sample separation/distribution that can be achieved within these limits. The vendor will specify the device that this instruction is executed on and which excitation and emission spectra are available. At least one negative control is required, which will be used to define data acquisition parameters as well as to determine any autofluorescent properties for the sample set. Additional negative positive control samples are optional. Positive control samples will be used to optimize single color signals and, if desired, to minimize bleed into other channels.

For each sample this instruction asks you to specify the *volume* and/or *captured_events*. Vendors might also require *captured_events* in case their device does not support volumetric sample intake. If both conditions are supported, the vendor will specify if data will be collected only until the first one is met or until both conditions are fulfilled.

Example Usage:

```
p = Protocol()
dataref = "test_ref"
FSC = {"voltage_range": {"low": "230:volt", "high": "280:volt"},
       "area": True, "height": True, "weight": False}
SSC = {"voltage_range": {"low": "230:volt", "high": "280:volt"},
       "area": True, "height": True, "weight": False}
neg_controls = {"well": "well0", "volume": "100:microliter",
```

```
                    "captured_events": 5, "channel": "channel0"}
    samples = [
        {"well": "well0", "volume": "100:microliter", "captured_events": 9}]

    p.flow_analyze(dataref, FSC, SSC, neg_controls,
                   samples, colors=None, pos_controls=None)
```

Autoprotocol Output:

```
    {
      "channels": {
        "FSC": {
          "voltage_range": {
            "high": "280:volt",
            "low": "230:volt"
          },
          "area": true,
          "height": true,
          "weight": false
        },
        "SSC": {
          "voltage_range": {
            "high": "280:volt",
            "low": "230:volt"
          },
          "area": true,
          "height": true,
          "weight": false
        }
      },
      "op": "flow_analyze",
      "negative_controls": {
        "channel": "channel0",
        "well": "well0",
        "volume": "100:microliter",
        "captured_events": 5
      },
      "dataref": "test_ref",
      "samples": [
        {
          "well": "well0",
          "volume": "100:microliter",
          "captured_events": 9
        }
      ]
    }
```

Parameters

- **dataref** (*str*) – Name of flow analysis dataset generated.
- **FSC** (*dict*) – Dictionary containing FSC channel parameters in the form of:

```
    {
      "voltage_range": {
        "low": "230:volt",
        "high": "280:volt"
        },
```

```
  "area": true,              //default: true
  "height": true,            //default: true
  "weight": false            //default: false
}
```

- **SSC** (*dict*) – Dictionary of SSC channel parameters in the form of:

```
{
  "voltage_range": {
    "low": <voltage>,
    "high": <voltage>"
    },
  "area": true,              //default: true
  "height": true,            //default: false
  "weight": false            //default: false
}
```

- **neg_controls** (*list of dicts*) – List of negative control wells in the form of:

```
{
    "well": well,
    "volume": volume,
    "captured_events": integer,     // optional, default infinity
    "channel": [channel_name]
}
```

  at least one negative control is required.

- **samples** (*list of dicts*) – List of samples in the form of:

```
{
    "well": well,
    "volume": volume,
    "captured_events": integer     // optional, default infinity
}
```

  at least one sample is required

- **colors** (*list of dicts, optional*) – Optional list of colors in the form of:

```
[{
  "name": "FitC",
  "emission_wavelength": "495:nanometer",
  "excitation_wavelength": "519:nanometer",
  "voltage_range": {
    "low": <voltage>,
    "high": <voltage>
  },
  "area": true,              //default: true
  "height": false,           //default: false
  "weight": false            //default: false
}, ... ]
```

- **pos_controls** (*list of dicts, optional*) – Optional list of positive control wells in the form of:

```
[{
    "well": well,
    "volume": volume,
    "captured_events": integer,     // optional, default infinity
    "channel": [channel_name],
    "minimize_bleed": [{            // optional
      "from": color,
      "to": [color]
    }, ...
]
```

Raises

- `TypeError` – If inputs are not of the correct type.
- `UnitError` – If unit inputs are not properly formatted.
- `AssertionError` – If required parameters are missing.
- `ValueError` – If volumes are not correctly formatted or present.

Protocol.oligosynthesize()

`Protocol.`**`oligosynthesize`**`(`*`oligos`*`)`

Specify a list of oligonucleotides to be synthesized and a destination for each product.

Example Usage:

```
oligo_1 = p.ref("oligo_1", None, "micro-1.5", discard=True)

p.oligosynthesize([{"sequence": "CATGGTCCCCTGCACAGG",
                    "destination": oligo_1.well(0),
                    "scale": "25nm",
                    "purification": "standard"}])
```

Autoprotocol Output:

```
"instructions": [
    {
      "oligos": [
        {
          "destination": "oligo_1/0",
          "sequence": "CATGGTCCCCTGCACAGG",
          "scale": "25nm",
          "purification": "standard"
        }
      ],
      "op": "oligosynthesize"
    }
  ]
```

Parameters **`oligos`** (*list of dicts*) – List of oligonucleotides to synthesize. Each dictionary should contain the oligo's sequence, destination, scale and purification

```
[
    {
      "destination": "my_plate/A1",
      "sequence": "GATCRYMKSWHBVDN",
```

```
                    // - standard IUPAC base codes
                    // - IDT also allows rX (RNA), mX (2' O-methyl RNA), and
                    //   X*/rX*/mX* (phosphorothioated)
                    // - they also allow inline annotations for modifications,
                    //   eg "GCGACTC/3Phos/" for a 3' phosphorylation
                    //   eg "aggg/iAzideN/cgcgc" for an internal modification
                  "scale": "25nm" | "100nm" | "250nm" | "1um",
                  "purification": "standard" | "page" | "hplc",
                    // default: standard
                },
                ...
            ]
```

Protocol.spread()

`Protocol.`**`spread`**(*source*, *dest*, *volume*)

Spread the specified volume of the source aliquot across the surface of the agar contained in the object container

Example Usage:

```
p = Protocol()

agar_plate = p.ref("agar_plate", None, "1-flat", discard=True)
bact = p.ref("bacteria", None, "micro-1.5", discard=True)

p.spread(bact.well(0), agar_plate.well(0), "55:microliter")
```

Autoprotocol Output:

```
{
  "refs": {
    "bacteria": {
      "new": "micro-1.5",
      "discard": true
    },
    "agar_plate": {
      "new": "1-flat",
      "discard": true
    }
  },
  "instructions": [
    {
      "volume": "55.0:microliter",
      "to": "agar_plate/0",
      "from": "bacteria/0",
      "op": "spread"
    }
  ]
}
```

Parameters

- **`source`** (Well) – Source of material to spread on agar
- **`dest`** (Well) – Reference to destination location (plate containing agar)
- **`volume`** (*str, Unit*) – Volume of source material to spread on agar

Protocol.autopick()

Protocol.**autopick**(*sources*, *dests*, *min_abort=0*, *criteria={}*, *dataref='autopick'*, *newpick=False*)

Pick colonies from the agar-containing location(s) specified in *sources* to the location(s) specified in *dests* in highest to lowest rank order until there are no more colonies available. If fewer than min_abort pickable colonies have been identified from the location(s) specified in *sources*, the run will stop and no further instructions will be executed.

Example Usage:

Autoprotocol Output:

Parameters

- **sources** (*Well, WellGroup, list of Wells*) – Reference wells containing agar and colonies to pick
- **dests** (*Well, WellGroup, list of Wells*) – List of destination(s) for picked colonies
- **criteria** (*dict*) – Dictionary of autopicking criteria.
- **min_abort** (*int, optional*) – Total number of colonies that must be detected in the aggregate list of *from* wells to avoid aborting the entire run.

Protocol.image_plate()

Protocol.**image_plate**(*ref*, *mode*, *dataref*)

Capture an image of the specified container.

Example Usage:

```
p = Protocol()

agar_plate = p.ref("agar_plate", None, "1-flat", discard=True)
bact = p.ref("bacteria", None, "micro-1.5", discard=True)

p.spread(bact.well(0), agar_plate.well(0), "55:microliter")
p.incubate(agar_plate, "warm_37", "18:hour")
p.image_plate(agar_plate, mode="top", dataref="my_plate_image_1")
```

Autoprotocol Output:

```
{
  "refs": {
    "bacteria": {
      "new": "micro-1.5",
      "discard": true
    },
    "agar_plate": {
      "new": "1-flat",
      "discard": true
    }
  },
  "instructions": [
    {
      "volume": "55.0:microliter",
      "to": "agar_plate/0",
      "from": "bacteria/0",
      "op": "spread"
    },
```

```
        {
          "where": "warm_37",
          "object": "agar_plate",
          "co2_percent": 0,
          "duration": "18:hour",
          "shaking": false,
          "op": "incubate"
        },
        {
          "dataref": "my_plate_image_1",
          "object": "agar_plate",
          "mode": "top",
          "op": "image_plate"
        }
      ]
    }
```

Parameters

- **ref** (*str, Container*) – Container to take image of
- **mode** (*str*) – Imaging mode (currently supported: "top")
- **dataref** (*str*) – Name of data reference of resulting image

Protocol.provision()

`Protocol.`**`provision`**(*resource_id*, *dests*, *volumes*)

Provision a commercial resource from a catalog into the specified destination well(s). A new tip is used for each destination well specified to avoid contamination.

Parameters

- **resource_id** (*str*) – Resource ID from catalog.
- **dests** (*Well, WellGroup, list of Wells*) – Destination(s) for specified resource.
- **volumes** (*str, Unit, list of str, list of Unit*) – Volume(s) to transfer of the resource to each destination well. If one volume of specified, each destination well recieve that volume of the resource. If destinations should recieve different volumes, each one should be specified explicitly in a list matching the order of the specified destinations.

Raises

- `TypeError` – If resource_id is not a string.
- `RuntimeError` – If length of the list of volumes specified does not match the number of destination wells specified.
- `TypeError` – If volume is not specified as a string or Unit (or a list of either)

Protocol.flash_freeze()

`Protocol.`**`flash_freeze`**(*container*, *duration*)

Flash freeze the contents of the specified container by submerging it in liquid nitrogen for the specified amount of time.

Example Usage:

```
p = Protocol()

sample = p.ref("liquid_sample", None, "micro-1.5", discard=True)
p.flash_freeze(sample, "25:second")
```

Autoprotocol Output:

```
{
  "refs": {
    "liquid_sample": {
      "new": "micro-1.5",
      "discard": true
    }
  },
  "instructions": [
    {
      "duration": "25:second",
      "object": "liquid_sample",
      "op": "flash_freeze"
    }
  ]
}
```

Parameters

- **container** (*Container, str*) – Container to be flash frozen.
- **duration** (*str, Unit*) – Duration to submerge specified container in liquid nitrogen.

Protocol.mag_incubate()

Protocol.**mag_incubate**(*head*, *container*, *duration*, *magnetize=False*, *tip_position=1.5*, *temperature=None*, *new_tip=False*, *new_instruction=False*)

Incubate the container for a set time with tips set at *tip_position*.

Example Usage:

```
p = Protocol()
plate = p.ref("plate_0", None, "96-pcr", storage="cold_20")

p.mag_incubate("96-pcr", plate, "30:minute", magnetize=False,
               tip_position=1.5, temperature=None, new_tip=False)
```

Autoprotocol Output:

```
"instructions": [
    {
      "groups": [
        [
          {
            "incubate": {
              "duration": "30:minute",
              "tip_position": 1.5,
              "object": "plate_0",
              "magnetize": false,
              "temperature": null
            }
          }
```

```
            ]
          ],
          "magnetic_head": "96-pcr",
          "op": "magnetic_transfer"
        }
      ]
```

Parameters

- **head** (*str*) – Magnetic head to use for the magnetic bead transfers
- **container** (Container) – Container to incubate beads
- **duration** (*str, Unit*) – Time for incubation
- **magnetize** (*bool*) – Specify whether to magnetize the tips
- **tip_position** (*float*) – Position relative to well height that tips are held
- **temperature** (*str, Unit*) – Temperature heat block is set at
- **new_tip** (*bool*) – Specify whether to use a new tip to complete the step
- **new_instruction** (*bool*) – Specify whether to create a new magnetic_transfer instruction

Protocol.mag_collect()

Protocol.**mag_collect**(*head*, *container*, *cycles*, *pause_duration*, *bottom_position=0.0*, *temperature=None*, *new_tip=False*, *new_instruction=False*)

Collect beads from a container by cycling magnetized tips in and out of the container with an optional pause at the bottom of the insertion.

Example Usage:

```
p = Protocol()
plate = p.ref("plate_0", None, "96-pcr", storage="cold_20")

p.mag_collect("96-pcr", plate, 5, "30:second", bottom_position=
              0.0, temperature=None, new_tip=False,
              new_instruction=False)
```

Autoprotocol Output:

```
"instructions": [
    {
      "groups": [
        [
          {
            "collect": {
              "bottom_position": 0.0,
              "object": "plate_0",
              "temperature": null,
              "cycles": 5,
              "pause_duration": "30:second"
            }
          }
        ]
      ],
      "magnetic_head": "96-pcr",
```

```
        "op": "magnetic_transfer"
      }
    ]
```

Parameters

- **head** (*str*) – Magnetic head to use for the magnetic bead transfers
- **container** (Container) – Container to incubate beads
- **cycles** (*int*) – Number of cycles to raise and lower tips
- **pause_duration** (*str, Unit*) – Time tips are paused in bottom position each cycle
- **bottom_position** (*float*) – Position relative to well height that tips are held during pause
- **temperature** (*str, Unit*) – Temperature heat block is set at
- **new_tip** (*bool*) – Specify whether to use a new tip to complete the step
- **new_instruction** (*bool*) – Specify whether to create a new magnetic_transfer instruction

Protocol.mag_dry()

`Protocol.`**`mag_dry`**(*head*, *container*, *duration*, *new_tip=False*, *new_instruction=False*)

Dry beads with magnetized tips above and outside a container for a set time.

Example Usage:

```
p = Protocol()
plate = p.ref("plate_0", None, "96-pcr", storage="cold_20")

p.mag_dry("96-pcr", plate, "30:minute", new_tip=False,
          new_instruction=False)
```

Autoprotocol Output:

```
"instructions": [
    {
      "groups": [
        [
          {
            "dry": {
              "duration": "30:minute",
              "object": "plate_0"
            }
          }
        ]
      ],
      "magnetic_head": "96-pcr",
      "op": "magnetic_transfer"
    }
  ]
```

Parameters

- **head** (*str*) – Magnetic head to use for the magnetic bead transfers
- **container** (Container) – Container to dry beads above

- **duration** (*str, Unit*) – Time for drying
- **new_tip** (*bool*) – Specify whether to use a new tip to complete the step
- **new_instruction** (*bool*) – Specify whether to create a new magnetic_transfer instruction

Protocol.mag_mix()

Protocol.**mag_mix**(*head*, *container*, *duration*, *frequency*, *center=0.5*, *amplitude=0.5*, *magnetize=False*, *temperature=None*, *new_tip=False*, *new_instruction=False*)

Mix beads in a container by cycling tips in and out of the container.

Example Usage:

```
p = Protocol()
plate = p.ref("plate_0", None, "96-pcr", storage="cold_20")

p.mag_mix("96-pcr", plate, "30:second", "60:hertz", center=0.75,
          amplitude=0.25, magnetize=True, temperature=None,
          new_tip=False, new_instruction=False)
```

Autoprotocol Output:

```
"instructions": [
    {
      "groups": [
        [
          {
            "mix": {
              "center": 0.75,
              "object": "plate_0",
              "frequency": "2:hertz",
              "amplitude": 0.25,
              "duration": "30:second",
              "magnetize": true,
              "temperature": null
            }
          }
        ]
      ],
      "magnetic_head": "96-pcr",
      "op": "magnetic_transfer"
    }
  ]
```

Parameters

- **head** (*str*) – Magnetic head to use for the magnetic bead transfers
- **container** (Container) – Container to incubate beads
- **duration** (*str, Unit*) – Total time for this sub-operation
- **frequency** (*str, Unit*) – Cycles per second (hertz) that tips are raised and lowered
- **center** (*float*) – Position relative to well height where oscillation is centered
- **amplitude** (*float*) – Distance relative to well height to oscillate around "center"
- **magnetize** (*bool*) – Specify whether to magnetize the tips

- **temperature** (*str, Unit*) – Temperature heat block is set at
- **new_tip** (*bool*) – Specify whether to use a new tip to complete the step
- **new_instruction** (*bool*) – Specify whether to create a new magnetic_transfer instruction

Protocol.mag_release()

Protocol.**mag_release**(*head*, *container*, *duration*, *frequency*, *center=0.5*, *amplitude=0.5*, *temperature=None*, *new_tip=False*, *new_instruction=False*)

Release beads into a container by cycling tips in and out of the container with tips unmagnetized.

Example Usage:

```
p = Protocol()
plate = p.ref("plate_0", None, "96-pcr", storage="cold_20")

p.mag_release("96-pcr", plate, "30:second", "60:hertz", center=0.75,
              amplitude=0.25, temperature=None, new_tip=False,
              new_instruction=False)
```

Autoprotocol Output:

```
"instructions": [
    {
      "groups": [
        [
          {
            "release": {
              "center": 0.75,
              "object": "plate_0",
              "frequency": "2:hertz",
              "amplitude": 0.25,
              "duration": "30:second",
              "temperature": null
            }
          }
        ]
      ],
      "magnetic_head": "96-pcr",
      "op": "magnetic_transfer"
    }
  ]
```

Parameters

- **head** (*str*) – Magnetic head to use for the magnetic bead transfers
- **container** (Container) – Container to incubate beads
- **duration** (*str, Unit*) – Total time for this sub-operation
- **frequency** (*str, Unit*) – Cycles per second (hertz) that tips are raised and lowered
- **center** (*float*) – Position relative to well height where oscillation is centered
- **amplitude** (*float*) – Distance relative to well height to oscillate around “center”
- **temperature** (*str, Unit*) – Temperature heat block is set at
- **new_tip** (*bool*) – Specify whether to use a new tip to complete the step

- **new_instruction** (*bool*) – Specify whether to create a new magnetic_transfer instruction

Protocol.measure_concentration()

Protocol.**measure_concentration**(*wells*, *dataref*, *measurement*, *volume='2:microliter'*)

Measure the concentration of DNA, ssDNA, RNA or protein in the specified volume of the source aliquots.

Example Usage:

```
p = Protocol()

test_plate = p.ref("test_plate", id=None, cont_type="96-flat",
    storage=None, discard=True)
p.measure_concentration(test_plate.wells_from(0, 3), "mc_test",
    "DNA")
p.measure_concentration(test_plate.wells_from(3, 3),
    dataref="mc_test2", measurement="protein",
    volume="4:microliter")
```

Autoprotocol Output:

```
{
  "refs": {
    "test_plate": {
      "new": "96-flat",
      "discard": true
    }
  },
  "instructions": [
     {
        "volume": "2.0:microliter",
        "dataref": "mc_test",
        "object": [
            "test_plate/0",
            "test_plate/1",
            "test_plate/2"
        ],
        "op": "measure_concentration",
        "measurement": "DNA"
    },
    ...
  ]
}
```

Parameters

- **wells** (*list, WellGroup, Well*) – WellGroup of wells to be measured
- **volume** (*str, Unit*) – Volume of sample required for analysis
- **dataref** (*str*) – Name of this specific dataset of measurements
- **measurement** (*str*) – Class of material to be measured. One of ["DNA", "ssDNA", "RNA", "protein"].

Protocol.measure_mass()

Protocol.**measure_mass**(*containers*, *dataref*)

Measure the mass of a list of containers.

Example Usage:

```
p = Protocol()

test_plate = p.ref("test_plate", id=None, cont_type="96-flat",
    storage=None, discard=True)
p.measure_mass([test_plate], "test_data")
```

Autoprotocol Output:

```
{
  "refs": {
    "test_plate": {
      "new": "96-flat",
      "discard": true
    }
  },
  "instructions": [
     {
        "dataref": "test_data",
        "object": [
            "test_plate
        ],
        "op": "measure_mass"
      }
  ]
}
```

Parameters

- **containers** (*list, Container*) – list of containers to be measured
- **dataref** (*str*) – Name of this specific dataset of measurements

Protocol.measure_volume()

Protocol.**measure_volume**(*wells*, *dataref*)

Measure the volume of each well in wells.

Example Usage:

```
p = Protocol()

test_plate = p.ref("test_plate", id=None, cont_type="96-flat",
    storage=None, discard=True)
p.measure_volume(test_plate.from_wells(0,2), "test_data")
```

Autoprotocol Output:

```
{
  "refs": {
    "test_plate": {
      "new": "96-flat",
      "discard": true
```

```
          }
      },
      "instructions": [
          {
              "dataref": "test_data",
              "object": [
                  "test_plate/0",
                  "test_plate/1"
              ],
              "op": "measure_volume"
          }
      ]
  }
```

Parameters

- **wells** (*list, well, WellGroup*) – list of wells to be measured
- **dataref** (*str*) – Name of this specific dataset of measurements

protocol.Ref

class autoprotocol.protocol.**Ref**(*name*, *opts*, *container*)
Link a ref name (string) to a Container instance.

CHAPTER 2

autoprotocol.container

container.Container

class autoprotocol.container.**Container**(*id*, *container_type*, *name=None*, *storage=None*, *cover=None*)

A reference to a specific physical container (e.g. a tube or 96-well microplate).

Every Container has an associated ContainerType, which defines the well count and arrangement, amongst other properties.

There are several methods on Container which present a convenient interface for defining subsets of wells on which to operate. These methods return a WellGroup.

Containers are usually declared using the Protocol.ref method.

Parameters

- **name** (*str*) – name of the container/ref being created.
- **id** (*string*) – Alphanumerical identifier for a Container.
- **container_type** (ContainerType) – ContainerType associated with a Container.

Container.well()

Container.**well**(*i*)

Return a Well object representing the well at the index specified of this Container.

Parameters **i** (*int, str*) – Well reference in the form of an integer (ex: 0) or human-readable string (ex: "A1").

Container.tube()

Container.**tube**()

Checks if container is tube and returns a Well respresenting the zeroth well.

Returns

Return type *Well*

Raises *AttributeError :* – If container is not tube

Container.wells()

Container.**wells**(**args*)
Return a WellGroup containing references to wells corresponding to the index or indices given.

Parameters **args** (*str, int, list*) – Reference or list of references to a well index either as an integer or a string.

Container.robotize()

Container.**robotize**(*well_ref*)
Return the integer representation of the well index given, based on the ContainerType of the Container.

Uses the robotize function from the ContainerType class. Refer to *ContainerType.robotize()* for more information.

Container.humanize()

Container.**humanize**(*well_ref*)
Return the human readable representation of the integer well index given based on the ContainerType of the Container.

Uses the humanize function from the ContainerType class. Refer to *ContainerType.humanize()* for more information.

Container.decompose()

Container.**decompose**(*well_ref*)
Return a tuple representing the column and row number of the well index given based on the ContainerType of the Container.

Uses the decompose function from the ContainerType class. Refer to *ContainerType.decompose()* for more information.

Container.all_wells()

Container.**all_wells**(*columnwise=False*)
Return a WellGroup representing all Wells belonging to this Container.

Parameters **columnwise** (*bool, optional*) – returns the WellGroup columnwise instead of row-wise (ordered by well index).

Container.quadrant()

Container.**quadrant**(*quad*)
Return a WellGroup of Wells corresponding to the selected quadrant of this Container.

This is only applicable to 384-well plates.

Parameters **quad** (*int*) – Specifies the quadrant number of the well (ex. 2)

Container.wells_from()

Container.**wells_from**(*start*, *num*, *columnwise=False*)
Return a WellGroup of Wells belonging to this Container starting from the index indicated (in integer or string form) and including the number of proceeding wells specified. Wells are counted from the starting well rowwise unless columnwise is True.

Parameters

- **start** (*Well, int, str*) – Starting well specified as a Well object, a human-readable well index or an integer well index.
- **num** (*int*) – Number of wells to include in the Wellgroup.
- **columnwise** (*bool, optional*) – Specifies whether the wells included should be counted columnwise instead of the default rowwise.

Container.inner_wells()

Container.**inner_wells**(*columnwise=False*)
Return a WellGroup of all wells on a plate excluding wells in the top and bottom rows and in the first and last columns.

Parameters **columnwise** (*bool, optional*) – returns the WellGroup columnwise instead of rowwise (ordered by well index).

Container.discard()

Container.**discard**()
Set the storage condition of a container to None and container to be discarded if ref in protocol.

Example

```
    p = Protocol()
    container = p.ref("new_container", cont_type="96-pcr",
                      storage="cold_20")
    p.incubate(c, "warm_37", "30:minute")
    container.discard()

    Autoprotocol generated:

    .. code-block:: json

       "refs": {
          "new_container": {
            "new": "96-pcr",
            "discard": true
          }
        }
```

Container.set_storage()

Container.**set_storage**(*storage*)
Set the storage condition of a container, will overwrite an existing storage condition, will remove discard True.

Parameters **storage** (*str*) – Storage condition.

Raises *TypeError :* – If storage condition not of type str.

container.Well

class autoprotocol.container.**Well**(*container*, *index*)
A Well object describes a single location within a container.

Do not construct a Well directly – retrieve it from the related Container object.

Parameters

- **container** (Container) – The Container this well belongs to.
- **index** (*integer*) – The index of this well within the container.
- **volume** (Unit) – Theoretical volume of this well.
- **properties** (*dict*) – Additional properties of this well represented as a dictionary.

Well.set_properties()

Well.**set_properties**(*properties*)
Set properties for a Well. Existing property dictionary will be completely overwritten with the new dictionary.

Parameters **properties** (*dict*) – Custom properties for a Well in dictionary form.

Well.add_properties()

Well.**add_properties**(*properties*)
Add properties to the properties attribute of a Well. If any key/value pairs are present in both the old and new dictionaries, they will be overwritten by the pairs in the new dictionary.

Parameters **properties** (*dict*) – Dictionary of properties to add to a Well.

Well.set_volume()

Well.**set_volume**(*vol*)
Set the theoretical volume of liquid in a Well.

Parameters **vol** (*str, Unit*) – Theoretical volume to indicate for a Well.

Well.set_name()

Well.**set_name**(*name*)
Set a name for this well for it to be included in a protocol's "outs" section

Parameters **name** (*str*) – Well name.

Well.humanize()

Well.**humanize**()
Return the human readable representation of the integer well index given based on the ContainerType of the Well.

Uses the humanize function from the ContainerType class. Refer to *ContainerType.humanize()* for more information.

Well.__repr__()

Well.**__repr__**()
Return a string representation of a Well.

container.WellGroup

class autoprotocol.container.**WellGroup**(*wells*)
A logical grouping of Wells.

Wells in a WellGroup do not necessarily need to be in the same container.

Parameters **wells** (*list*) – List of Well objects contained in this WellGroup.

WellGroup.set_properties()

WellGroup.**set_properties**(*properties*)
Set the same properties for each Well in a WellGroup.

Parameters **properties** (*dict*) – Dictionary of properties to set on Well(s).

WellGroup.set_volume()

WellGroup.**set_volume**(*vol*)
Set the volume of every well in the group to vol.

Parameters **vol** (*Unit, str*) – Theoretical volume of each well in the WellGroup.

WellGroup.set_group_name()

WellGroup.**set_group_name**(*name*)
Assigns a name to a WellGroup.

Parameters **name** (*str*) – WellGroup name

WellGroup.wells_with()

WellGroup.**wells_with**(*prop*, *val=None*)
Returns a wellgroup of wells with the specified property and value

Parameters

- **prop** (*str*) – the property you are searching for

- **val** (*str*) – the value assigned to the property

WellGroup.indices()

`WellGroup.`**`indices`**`()`
Return the indices of the wells in the group in human-readable form, given that all of the wells belong to the same container.

WellGroup.append()

`WellGroup.`**`append`**`(`*other*`)`
Append another well to this WellGroup.

Parameters **other** (Well) – Well to append to this WellGroup.

WellGroup.extend()

`WellGroup.`**`extend`**`(`*other*`)`
Extend this WellGroup with another WellGroup.

Parameters **other** (*WellGroup or list of Wells*) – WellGroup to extend this WellGroup.

WellGroup.pop()

`WellGroup.`**`pop`**`(`*index=-1*`)`
Removes and returns the last well in the wellgroup, unless an index is specified. If index is specified, the well at that index is removed from the wellgroup and returned.

Parameters **index** (*int*) – the index of the well you want to remove and return

WellGroup.insert()

`WellGroup.`**`insert`**`(`*i*, *well*`)`
Insert a well at a given position.

Parameters

- **i** (*int*) – index to insert the well at
- **well** (Well) – insert this well at the index

WellGroup.__getitem__()

`WellGroup.`**`__getitem__`**`(`*key*`)`
Return a specific Well from a WellGroup.

Parameters **key** (*int*) – Position in a WellGroup in robotized form.

WellGroup.__len__()

`WellGroup.`**`__len__`**`()`
Return the number of Wells in a WellGroup.

WellGroup.__repr__()

WellGroup.**__repr__**()
Return a string representation of a WellGroup.

WellGroup.__add__()

WellGroup.**__add__**(*other*)
Append a Well or Wells from another WellGroup to this WellGroup.

Parameters **other** (*Well, WellGroup.*) –

autoprotocol.container_type

container_type.ContainerType

class autoprotocol.container_type.**ContainerType**

The ContainerType class holds the capabilities and properties of a particular container type.

Parameters

- **name** (*str*) – Full name describing a ContainerType.
- **is_tube** (*bool*) – Indicates whether a ContainerType is a tube (container with one well).
- **well_count** (*int*) – Number of wells a ContainerType contains.
- **well_depth_mm** (*int*) – Depth of well(s) contained in a ContainerType in millimeters.
- **well_volume_ul** (*int*) – Maximum volume of well(s) contained in a ContainerType in microliters.
- **well_coating** (*str*) – Coating of well(s) in container (ex. collagen).
- **sterile** (*bool*) – Indicates whether a ContainerType is sterile.
- **cover_types** (*list*) – List of valid covers associated with a ContainerType.
- **seal_types** (*list*) – List of valid seals associated with a ContainerType.
- **capabilities** (*list*) –

  List of capabilities associated with a ContainerType (ex. [”spin”, “incubate”]).
- **shortname** (*str*) – Short name used to refer to a ContainerType.
- **col_count** (*int*) – Number of columns a ContainerType contains.
- **dead_volume_ul** (*int*) – Volume of liquid that cannot be aspirated from any given well of a ContainerType via liquid-handling.
- **safe_min_volume_ul** (*int*) – Minimum volume of liquid to ensure adequate volume for liquid-handling aspiration from any given well of a ContainerType.
- **true_max_vol_ul** (*int*) – Maximum volume of well(s) in microliters, often same value as well_volume_ul (maximum working volume), however, some ContainerType(s) can have a different value corresponding to a true maximum volume of a well (ex. echo compatible containers)
- **vendor** (*str*) – ContainerType commercial vendor, if available.
- **cat_no** (*str*) – ContainerType vendor catalog number, if available.

- **prioritize_seal_or_cover** (*str, optional*) – "seal" or "cover", determines whether to prioritize sealing or covering defaults to "seal"

ContainerType.robotize()

ContainerType.**robotize**(*well_ref*)

Return a robot-friendly well reference from a number of well reference formats.

Example Usage:

```
>>> p = Protocol()
>>> my_plate = p.ref("my_plate", cont_type="6-flat", discard=True)
>>> my_plate.robotize("A1")
0
>>> my_plate.robotize("5")
5
>>> my_plate.robotize(my_plate.well(3))
3
>>> my_plate.robotize(["A1", "A2"])
[0, 1]
```

Parameters **well_ref** (*str, int, Well, list[str or int or Well]*) – Well reference to be robotized in string, integer or Well object form. Also accepts lists of str, int or Well.

Returns **well_ref** – Single or list of Well references passed as rowwise integer (left-to-right, top-to-bottom, starting at 0 = A1).

Return type int, list

Raises

- TypeError – If well reference given is not an accepted type.
- ValueError – If well reference given exceeds container dimensions.

ContainerType.humanize()

ContainerType.**humanize**(*well_ref*)

Return the human readable form of a well index based on the well format of this ContainerType.

Example Usage:

```
>>> p = Protocol()
>>> my_plate = p.ref("my_plate", cont_type="6-flat", discard=True)
>>> my_plate.humanize(0)
'A1'
>>> my_plate.humanize(5)
'B3'
>>> my_plate.humanize('0')
'A1'
```

Parameters **well_ref** (*int, str, list[int or str]*) – Well reference to be humanized in integer or string form. If string is provided, it has to be parseable into an int. Also accepts lists of int or str

Returns **well_ref** – Well index passed as human-readable form.

Return type str

Raises

- `TypeError` – If well reference given is not an accepted type.
- `ValueError` – If well reference given exceeds container dimensions.

ContainerType.decompose()

`ContainerType.`**`decompose`**`(`*idx*`)`
Return the (col, row) corresponding to the given well index.

Parameters **`well_ref`** (*str, int*) – Well index in either human-readable or integer form.

Returns **well_ref** – tuple containing the column number and row number of the given well_ref.

Return type tuple

ContainerType.row_count()

`ContainerType.`**`row_count`**`()`
Return the number of rows of this ContainerType.

Container Types

384-flat

```
decompose(self, idx):
"""
Return the (col, row) corresponding to the given well index.

Parameters
----------
well_ref : str, int
    Well index in either human-readable or integer form.

Returns
-------
well_ref : tuple
    tuple containing the column number and row number of the given
    well_ref.

"""
if not isinstance(idx, (int, basestring, Well)):
```

384-pcr

```
idx = self.robotize(idx)
return (idx // self.col_count, idx % self.col_count)

row_count(self):
"""
Return the number of rows of this ContainerType.

"""
return self.well_count // self.col_count
```

```
ER_TYPES = {
-flat": ContainerType(
name="384-well UV flat-bottom plate",
well_count=384,
well_depth_mm=None,
well_volume_ul=Unit(90.0, "microliter"),
```

384-echo

```
is_tube=False,
cover_types=["standard", "universal"],
seal_types=None,
capabilities=["pipette", "spin", "absorbance",
              "fluorescence", "luminescence",
              "incubate", "gel_separate",
              "gel_purify", "cover", "stamp",
              "dispense"],
shortname="384-flat",
col_count=24,
dead_volume_ul=Unit(7, "microliter"),
safe_min_volume_ul=Unit(15, "microliter"),
vendor="Corning",
cat_no="3706"

-pcr": ContainerType(
```

384-flat-white-white-lv

```
well_depth_mm=None,
well_volume_ul=Unit(40.0, "microliter"),
well_coating=None,
sterile=None,
is_tube=False,
cover_types=None,
seal_types=["ultra-clear", "foil"],
capabilities=["pipette", "spin", "thermocycle",
              "incubate", "gel_separate",
              "gel_purify",
              "seal", "stamp", "dispense"],
shortname="384-pcr",
col_count=24,
dead_volume_ul=Unit(2, "microliter"),
safe_min_volume_ul=Unit(3, "microliter"),
vendor="Eppendorf",
cat_no="951020539"
```

384-flat-white-white-tc

```
well_count=384,
well_depth_mm=None,
well_volume_ul=Unit(65.0, "microliter"),
well_coating=None,
```

```
sterile=None,
is_tube=False,
cover_types=["universal"],
seal_types=["foil", "ultra-clear"],
capabilities=["pipette", "seal", "spin",
              "incubate", "stamp", "dispense",
              "cover"],
shortname="384-echo",
col_count=24,
dead_volume_ul=Unit(15, "microliter"),
safe_min_volume_ul=Unit(15, "microliter"),
true_max_vol_ul=Unit(135, "microliter"),
vendor="Labcyte",
cat_no="P-05525"
```

384-flat-clear-clear

```
name="384-well flat-bottom low volume plate",
well_count=384,
well_depth_mm=9.39,
well_volume_ul=Unit(40.0, "microliter"),
well_coating=None,
sterile=False,
is_tube=False,
cover_types=["standard", "universal"],
seal_types=None,
capabilities=["absorbance", "cover", "dispense",
              "fluorescence", "image_plate",
              "incubate", "luminescence",
              "pipette", "spin",
              "stamp", "uncover"],
shortname="384-flat-white-white-lv",
col_count=24,
dead_volume_ul=Unit(5, "microliter"),
safe_min_volume_ul=Unit(15, "microliter"),
```

384-v-clear-clear

```
well_count=6,
well_depth_mm=None,
well_volume_ul=Unit(5000.0, "microliter"),
well_coating=None,
sterile=False,
cover_types=["standard", "universal"],
seal_types=None,
capabilities=["cover", "incubate", "image_plate"],
shortname="6-flat",
is_tube=False,
col_count=3,
dead_volume_ul=Unit(400, "microliter"),
safe_min_volume_ul=Unit(600, "microliter"),
vendor="Eppendorf",
cat_no="30720016"
```

```
lat": ContainerType(
name="1-well flat-bottom plate",
```

384-round-clear-clear

```
well_volume_ul=Unit(80000.0, "microliter"),
well_coating=None,
sterile=False,
cover_types=["universal"],
seal_types=None,
capabilities=["cover", "incubate"],
shortname="1-flat",
is_tube=False,
col_count=1,
dead_volume_ul=Unit(36000, "microliter"),
safe_min_volume_ul=Unit(40000, "microliter"),
vendor="Fisher",
cat_no="267060"

lat-tc": ContainerType(
name="6-well TC treated plate",
well_count=6,
well_depth_mm=None,
```

384-flat-white-white-nbs

```
sterile=False,
cover_types=["standard", "universal"],
seal_types=None,
capabilities=["cover", "incubate", "image_plate"],
shortname="6-flat-tc",
is_tube=False,
col_count=3,
dead_volume_ul=Unit(400, "microliter"),
safe_min_volume_ul=Unit(600, "microliter"),
vendor="Eppendorf",
cat_no="30720113"

-sw96-hp": ContainerType(
name="96-well singlewell highprofile reservoir",
well_count=1,
well_depth_mm=None,
well_volume_ul=Unit(200.0, "milliliter"),
well_coating=None,
```

384-flat-white-white-optiplate

```
-flat-white-white-optiplate": ContainerType(
name="384-well flat-bottom polystyrene optimized plate",
well_count=384,
well_depth_mm=Unit(10.45, "millimeter"),
well_volume_ul=Unit(105.0, "microliter"),
well_coating=None,
```

```
sterile=False,
is_tube=False,
cover_types=["universal"],
seal_types=["ultra-clear", "foil"],
capabilities=["incubate", "seal", "image_plate",
              "miniprep_source", "maxiprep_source",
              "maxiprep_destination", "stamp", "dispense",
              "spin", "sanger_sequence", "miniprep_destination",
              "flash_freeze", "echo_dest", "cover",
              "fluorescence", "luminescence", "pipette",
              "uncover", "bluewash"],
shortname="384-flat-white-white-optiplate",
col_count=24,
dead_volume_ul=Unit(24, "microliter"),
safe_min_volume_ul=Unit(30, "microliter"),
vendor="PerkinElmer",
cat_no="6007299"
```

96-flat

```
-flat-white-white-tc": ContainerType(
name="384-well flat-bottom low flange plate",
well_count=384,
well_depth_mm=11.43,
well_volume_ul=Unit(80.0, "microliter"),
well_coating=None,
sterile=True,
is_tube=False,
cover_types=["standard", "universal"],
seal_types=None,
capabilities=["absorbance", "cover", "dispense",
              "fluorescence", "image_plate",
              "incubate", "luminescence",
              "pipette", "spin",
              "stamp", "uncover"],
shortname="384-flat-white-white-tc",
col_count=24,
```

96-flat-uv

```
vendor="Corning",
cat_no="3570"

-flat-clear-clear": ContainerType(
name="384-well fully clear high binding plate",
well_count=384,
well_depth_mm=11.43,
well_volume_ul=Unit(80.0, "microliter"),
well_coating="high bind",
sterile=False,
is_tube=False,
cover_types=["standard", "universal", "low_evaporation"],
seal_types=["ultra-clear", "foil"],
capabilities=["incubate", "seal", "image_plate",
              "stamp", "dispense", "spin",
```

```
                "absorbance", "cover",
                "fluorescence", "luminescence",
                "pipette", "uncover"],
```

96-flat-black-black-fluotrac-600

```
seal_types=["ultra-clear"],
capabilities=["incubate", "seal", "deseal", "stamp", "dispense",
                "spin", "cover", "pipette", "uncover",
                "bluewash", "mesoscale_sectors600"],
shortname="96-10-spot-vplex-m-pro-inflamm1-MSD",
col_count=12,
dead_volume_ul=Unit(25, "microliter"),
safe_min_volume_ul=Unit(65, "microliter"),
vendor="Mesoscale",
cat_no="K15048G"

4-spot-mMIP3a-MSD": ContainerType(
name="96-well 4-spot mouse MIP3a MSD plate",
well_count=96,
well_depth_mm=None,
well_volume_ul=Unit(340.0, "microliter"),
well_coating=None,
sterile=False,
is_tube=False,
cover_types=["low_evaporation", "standard", "universal"],
seal_types=["ultra-clear"],
capabilities=["incubate", "seal", "deseal", "stamp", "dispense",
```

96-flat-clear-clear-tc

```
name="1.5mL Microcentrifuge tube",
well_count=1,
well_depth_mm=None,
well_volume_ul=Unit(1500.0, "microliter"),
well_coating=None,
sterile=False,
cover_types=None,
seal_types=None,
capabilities=["pipette", "gel_separate",
                "gel_purify", "incubate", "spin"],
shortname="micro-1.5",
is_tube=True,
col_count=1,
dead_volume_ul=Unit(20, "microliter"),
safe_min_volume_ul=Unit(20, "microliter"),
vendor="USA Scientific",
cat_no="1615-5500"

lat": ContainerType(
```

96-pcr

```
dead_volume_ul=Unit(5, "microliter"),
safe_min_volume_ul=Unit(20, "microliter"),
vendor="Corning",
cat_no="3700"

flat": ContainerType(
name="96-well flat-bottom plate",
well_count=96,
well_depth_mm=None,
well_volume_ul=Unit(340.0, "microliter"),
well_coating=None,
sterile=False,
is_tube=False,
cover_types=["low_evaporation", "standard", "universal"],
seal_types=None,
capabilities=["pipette", "spin", "absorbance",
              "fluorescence", "luminescence",
```

96-deep

```
              "dispense"],
shortname="96-flat",
col_count=12,
dead_volume_ul=Unit(25, "microliter"),
safe_min_volume_ul=Unit(65, "microliter"),
vendor="Corning",
cat_no="3632"

flat-uv": ContainerType(
name="96-well flat-bottom UV transparent plate",
well_count=96,
well_depth_mm=None,
well_volume_ul=Unit(340.0, "microliter"),
well_coating=None,
sterile=False,
is_tube=False,
```

96-v-kf

```
capabilities=["pipette", "spin", "absorbance",
              "fluorescence", "luminescence",
              "incubate", "gel_separate",
              "gel_purify", "cover", "stamp",
              "dispense"],
shortname="96-flat-uv",
col_count=12,
dead_volume_ul=Unit(25, "microliter"),
safe_min_volume_ul=Unit(65, "microliter"),
vendor="Corning",
cat_no="3635"

pcr": ContainerType(
name="96-well PCR plate",
```

```
well_count=96,
well_depth_mm=None,
well_volume_ul=Unit(160.0, "microliter"),
well_coating=None,
```

96-deep-kf

```
cover_types=None,
seal_types=["ultra-clear", "foil"],
capabilities=["pipette", "sangerseq", "spin",
              "thermocycle", "incubate",
              "gel_separate", "gel_purify",
              "seal", "stamp", "dispense"],
shortname="96-pcr",
col_count=12,
dead_volume_ul=Unit(3, "microliter"),
safe_min_volume_ul=Unit(5, "microliter"),
vendor="Eppendorf",
cat_no="951020619"

deep": ContainerType(
name="96-well extended capacity plate",
well_count=96,
well_depth_mm=None,
well_volume_ul=Unit(2000.0, "microliter"),
```

24-deep

```
cover_types=["standard", "universal"],
seal_types=["breathable"],
prioritize_seal_or_cover="cover",
capabilities=["pipette", "incubate",
              "gel_separate", "gel_purify",
              "cover", "stamp", "dispense",
              "seal"],
shortname="96-deep",
is_tube=False,
col_count=12,
dead_volume_ul=Unit(5, "microliter"),
safe_min_volume_ul=Unit(30, "microliter"),
vendor="Corning",
cat_no="3961"

v-kf": ContainerType(
```

6-flat

```
is_tube=False,
col_count=12,
dead_volume_ul=Unit(50, "microliter"),
safe_min_volume_ul=Unit(50, "microliter"),
vendor="Fisher",
cat_no="22-387-031"
```

```
deep": ContainerType(
name="24-well extended capacity plate",
well_count=24,
well_depth_mm=None,
well_volume_ul=Unit(10000.0, "microliter"),
well_coating=None,
sterile=False,
```

6-flat-tc

```
well_coating=None,
sterile=False,
cover_types=None,
seal_types=None,
capabilities=["pipette", "gel_separate",
              "gel_purify", "incubate", "spin"],
shortname="micro-2.0",
is_tube=True,
col_count=1,
dead_volume_ul=Unit(5, "microliter"),
safe_min_volume_ul=Unit(40, "microliter"),
vendor="E&K Scientific",
cat_no="280200"
```

1-flat

```
capabilities=["pipette", "incubate",
              "gel_separate", "gel_purify",
              "stamp", "dispense", "seal"],
shortname="24-deep",
is_tube=False,
col_count=6,
dead_volume_ul=Unit(15, "microliter"),
safe_min_volume_ul=Unit(60, "microliter"),
vendor="E&K Scientific",
cat_no="EK-2053-S"

ro-2.0": ContainerType(
name="2mL Microcentrifuge tube",
well_count=1,
```

micro-2.0

```
well_depth_mm=None,
well_volume_ul=Unit(200.0, "microliter"),
well_coating=None,
sterile=False,
cover_types=["standard"],
seal_types=None,
capabilities=["pipette", "incubate",
              "gel_separate", "mag_dry",
              "mag_incubate", "mag_collect",
              "mag_release", "mag_mix",
```

```
              "cover", "stamp", "dispense"],
shortname="96-v-kf",
is_tube=False,
col_count=12,
dead_volume_ul=Unit(20, "microliter"),
```

micro-1.5

```
cat_no="22-387-030"

deep-kf": ContainerType(
name="96-well extended capacity King Fisher plate",
well_count=96,
well_depth_mm=None,
well_volume_ul=Unit(1000.0, "microliter"),
well_coating=None,
sterile=False,
cover_types=["standard"],
seal_types=None,
capabilities=["pipette", "incubate",
              "gel_separate", "mag_dry",
              "mag_incubate", "mag_collect",
              "mag_release", "mag_mix",
```

CHAPTER 4

autoprotocol support

autoprotocol.unit

unit.Unit

class `autoprotocol.unit.`**`Unit`**(*value*, *units=None*)

A representation of a measure of physical quantities such as length, mass, time and volume. Uses Pint's Quantity as a base class for implementing units and inherits functionalities such as conversions and proper unit arithmetic. Note that the magnitude is stored as a double-precision float, so there are inherent issues when dealing with extremely large/small numbers as well as numerical rounding for non-base 2 numbers.

Example

```
vol_1 = Unit(10, 'microliter')
vol_2 = Unit(10, 'liter')
print(vol_1 + vol_2)

time_1 = Unit(1, 'second')
speed_1 = vol_1/time_1
print (speed_1)
print (speed_1.to('liter/hour'))
```

Returns 10000010.0:microliter 10.0:microliter / second 0.036:liter / hour

Return type

Unit.fromstring()

static `Unit.`**`fromstring`**(*s*)

Convert a string representation of a unit into a Unit object.

Example

```
Unit.fromstring("10:microliter")
```

becomes

```
Unit(10, "microliter")
```

Parameters **s** (*str*) – String in the format of "value:unit"

autoprotocol.util

util.make_dottable_dict

class autoprotocol.util.**make_dottable_dict**

Enable dictionaries to be accessed using dot notation instead of bracket notation. This class should probably never be used.

Example

```
>>> d = {"foo": {
            "bar": {
                "bat": "Hello!"
                }
            }
        }

>>> print d["foo"]["bar"]["bat"]
Hello!

>>> d = make_dottable_dict(d)

>>> print d.foo.bar.bat
Hello!
```

Parameters **dict** (*dict*) – Dictionary to be made dottable.

util.deep_merge_params()

autoprotocol.util.**deep_merge_params**(*defaults*, *override*)

Merge two dictionaries while retaining common key-value pairs.

Parameters

- **defaults** (*dict*) – Default dictionary to compare with overrides.
- **override** (*dict*) – Dictionary containing additional keys and/or values to override those corresponding to keys in the defaults dicitonary.

util.incubate_params()

autoprotocol.util.**incubate_params**(*duration*, *shake_amplitude=None*, *shake_orbital=None*)

Create a dictionary with incubation parameters which can be used as input for instructions. Currenly supports plate reader instructions and could be extended for use with other instructions.

Parameters

- **shake_amplitude** (*str, Unit*) – amplitude of shaking between 1 and 6:millimeter
- **shake_orbital** (*bool*) – True for oribital and False for linear shaking
- **duration** (*str, Unit*) – time for shaking

util.make_band_param()

autoprotocol.util.**make_band_param**(*elution_buffer*, *elution_volume*, *max_bp*, *min_bp*, *destination*)

Support function to generate gel extraction parameters The *Protocol.gel_purify()* instruction requires band parameters for extraction, which this function will generate.

Parameters

- **elution_buffer** (*str*) – Elution buffer to use to retrieve band
- **elution_volume** (*str, Unit*) – Volume to elute band into
- **max_bp** (*int*) – Max basepairs of band
- **min_bp** (*int*) – Min basepairs of band
- **destination** (Well) – Well to place extracted band into

util.make_gel_extract_params()

autoprotocol.util.**make_gel_extract_params**(*source*, *band_list*, *lane=None*, *gel=None*)

Support function to generate gel extraction parameters The *Protocol.gel_purify()* instruction requires a list of extraction parameters, which this function helps to generate.

Parameters

- **source** (well) – Source well for the extraction
- **band_list** (*list, dict*) – List of bands to collect from the source (use make_band_param to make a band dictionary)
- **lane** (*int, optional*) – Lane to load and collect the source. If not set, lane will be auto-generated
- **gel** (*int, optional*) – Gel to load and collect the source. If not set, gel will be auto-generated

autoprotocol.harness

harness.run()

autoprotocol.harness.**run**(*fn*, *protocol_name=None*, *seal_after_run=True*)

Run the protocol specified by the function.

If protocol_name is passed, use preview parameters from the protocol with the matching "name" value in the manifest.json file to run the given function. Otherwise, take configuration JSON file from the command line and run the given function.

Parameters

- **fn** (*function*) – Function that generates Autoprotocol
- **protocol_name** (*str, optional*) – str matching the "name" value in the manifest.json file

- **seal_after_run** (*bool, optional*) – Implicitly add a seal/cover to all stored refs within the protocol using seal_on_store()

harness.seal_on_store()

autoprotocol.harness.**seal_on_store**(*protocol*)

Implicitly adds seal/cover instructions to the end of a run for containers that do not have a cover. Cover type applied defaults first to "seal" if its within the capabilities of the container type, otherwise to "cover".

Example Usage:

```
def example_method(protocol, params):
cont = params['container']
p.transfer(cont.well("A1"), cont.well("A2"), "10:microliter")
p.seal(cont)
p.unseal(cont)
p.cover(cont)
p.uncover(cont)
```

Autoprotocol Output:

```
{
  "refs": {
    "plate": {
      "new": "96-pcr",
      "store": {
        "where": "ambient"
      }
    }
  },
  "instructions": [
    {
      "groups": [
        {
          "transfer": [
            {
              "volume": "10.0:microliter",
              "to": "plate/1",
              "from": "plate/0"
            }
          ]
        }
      ],
      "op": "pipette"
    },
    {
      "object": "plate",
      "type": "ultra-clear",
      "op": "seal"
    },
    {
      "object": "plate",
      "op": "unseal"
    },
    {
      "lid": "universal",
      "object": "plate",
      "op": "cover"
```

```
        },
        {
          "object": "plate",
          "op": "uncover"
        },
        {
          "type": "ultra-clear",
          "object": "plate",
          "op": "seal"
        }
      ]
    }
```

harness.Manifest

class autoprotocol.harness.**Manifest**(*json*)

Object representation of a manifest.json file

Parameters **object** (*JSON object*) – A manifest.json file with the following format:

```
    {
      "format": "python",
      "license": "MIT",
      "description": "This is a protocol.",
      "protocols": [
        {
          "name": "SampleProtocol",
          "version": 1.0.0,
          "command_string": "python sample_protocol.py",
          "preview": {
            "refs":{},
            "parameters": {},
          "inputs": {},
          "dependencies": []
        }
      ]
    }
```

CHAPTER 5

Changelog

- : fix documentation typos
- : remove cover prior to mag steps where applicable
- : add new containers, true_max_vol_ul in _CONTAINER_TYPES
- : add prioritize_seal_or_cover allow priority selection
- : allow breathable seals on 96-deep and 24-deep
- : add PerkinElmer 384-well optiplate to container_type (cat# 6007299), *384-flat-white-white-optiplate*
- : add support for *more_than* in *add_time_constraint*
- : add ability to specify *x_cassette* for dispense and dispense_full_plate methods
- : add ability to specify a well as reagent source for dispense and dispense_full_plate methods
- : add *step_size* to dispense and dispense_full_plate methods
- : add shaking capabilities to *Protocol.incubate()*
- : add *ceil* and *floor* methods to *Unit*
- : convert test suite to py.test
- : docstring cleanup, linting
- : update default lid types for *384-echo*, *96-flat*, *96-flat-uv*, and *96-flat-clear-clear-tc*
- : update pint requirements, update error handling on UnitError
- : fix documentation typos
- : new plate types *384-v-clear-clear*, *384-round-clear-clear*, *384-flat-white-white-nbs*
- : allow incubation of containers at ambient without covers
- : prevent invalid incubate parameters in *Protocol.absorbance()*
- : respect incubate conditions where uncovered=True
- : fix *Well.set_properties()* so that it completely overwrites the existing properties dict
- : fix name of *384-round-clear-clear*
- : more descriptive error message in *Protocol.ref()*
- : add functions and tests to enable use of *–dye_test* flag
- : Container method: *Container.tube()*

- : new plate type *96-flat-clear-clear-tc*
- : Unit validations from str in *Protocol.flow_analyze()* instruction
- : update documention for *harness.seal_on_store()*
- : cover_types and seal_types to _CONTAINER_TYPES
- : validations of input types before cover check
- : new plate types *384-flat-clear-clear*, *384-flat-white-white-lv*, *384-flat-white-white-tc*
- : validations before implicit cover or seal
- : *WellGroup.extend()* can now take in a list of wells
- : WellGroup methods: *WellGroup.set_group_name()*, *WellGroup.pop()*, *WellGroup.insert()*, *WellGroup.wells_with()*
- : assertions and tests for *Protocol.flow_analyze()*
- : autocover before *Protocol.incubate()*
- : *is_resource_id* added to *Protocol.dispense()* and *Protocol.dispense_full_plate()* instructions
- : plate type *6-flat-tc* to ContainerType
- : unit conversion to microliters in *Protocol.dispense()* instruction
- : documentation
- : *Protocol.dispense()* instruction tests
- : using release for changelog and integration into readthedocs documentation
- : convert pipette operations to microliters
- #128: cover_types on *96-deep-kf* and *96-deep*
- #127: convert pipette operations to microliters
- : dispense_speed and distribute_target in *Protocol.distribute()* instruction
- : plate type *6-flat-tc* to ContainerType
- : auto-uncover before *Protocol.provision()* instructions
- : *WellGroup.extend()* can now take in a list of wells
- : WellGroup methods: *WellGroup.set_group_name()*, *WellGroup.pop()*, *WellGroup.insert()*, *WellGroup.wells_with()*
- : assertions and tests for *Protocol.flow_analyze()*
- : autocover before *Protocol.incubate()*
- : *is_resource_id* added to *Protocol.dispense()* and *Protocol.dispense_full_plate()* instructions
- : compatibility with py3 in *Protocol.flow_analyze()*
- : *Protocol.spin()* auto-cover
- : removed capability 'cover' from *96-pcr* and *384-pcr* plates
- : *Protocol.dispense()* instruction json outputs
- : documentation
- : new plate types *384-flat-clear-clear*, *384-flat-white-white-lv*, *384-flat-white-white-tc*
- : validations before implicit cover or seal

- : cover_types and seal_types to _CONTAINER_TYPES
- : validations of input types before cover check
- : string input types for source, destination wells for Instructions *Protocol.consolidate()*, *Protocol.autopick()*, *Protocol.mix()*
- : track plate cover status - Container objects now have a *cover* attribute, implicit plate unsealing or uncovering prior to steps that require the plate to be uncovered.
- : *Protocol.stamp()* separates row stamps with more than 2 containers
- : *Protocol.mix()* allows one_tip=True
- : *unit.Unit* specific error handling
- : *Protocol.illuminaseq()* allows cycle specification
- : *Protocol.add_time_constraint()* added
- : harness.py returns proper boolean for thermocycle types
- : *unit.Unit* specific error handling
- : thermocycle gradient steps in harness.py
- : *Protocol.mix()* allows one_tip=True
- : *Protocol.acoustic_transfer()* handling of droplet size
- : *Protocol.spin()* instruction takes directional parameters
- : *Protocol.gel_purify()* parameters improved
- : *Protocol.illuminaseq()* instruction
- : *Protocol.measure_volume()* instruction
- : *Protocol.measure_mass()* instruction
- : Compatibility of Unit for acceleration
- : fix harness to be python3 compatible
- : Concatenation of Well to WellGroup no longer returns None
- : WellGroup checks that all elements are wells
- : gel string in documentation
- : support for list input type for humanize and robotize (container and container_type)
- : *Protocol.gel_purify()* instruction to instruction.py and protocol.py
- : :ref:container-discard‘ and and *Container.set_storage()* methods for containers
- : csv-table input type to harness.py
- : magnetic transfer instructions to now pass relevant inputs through units
- : Unit support for *molar*
- : disclaimer to README.md on unit support
- : Unit(Unit(...)) now returns a Unit
- : checking for valid plate read incubate parameters
- : additional parameter, *gain*, to *Protocol.fluorescence()*
- : documentation for magnetic transfer instructions correctly uses hertz

- : adding magnetic transfer functions to documentation
- : support for a new instruction for *Protocol.measure_concentration()*
- : helper function in util.py to create incubation dictionaries
- : additional parameters to spectrophotometry instructions (*Protocol.absorbance()*, *Protocol.luminescence()*, *Protocol.fluorescence()*) to instruction.py and protocol.py
- : Updated Unit package to default to *Autoprotocol* format representation for temperature and speed units
- : Updated maximum tip capacity for a transfer operation to 900uL instead of 750uL
- : Updated handling of multiplication and division of Units of the same dimension to automatically resolve when possible
- : *unit.Unit* now uses Pint's Quantity as a base class
- : update *6-flat* well volumes
- : release versioning has been removed in favor of protocol versioniong in harness.py
- : kf container types *96-v-kf* and *96-deep-kf* in container_type.py
- : *magnetic_transfer* instruction to instruction.py and protocol.py
- : *container*+ input type to harness.py
- : Update container_test.py and container_type_test.py to include safe_min_volume_ul
- : default versioning in manifest_test.json
- : updated dead_volume_ul values in _CONTAINER_TYPES
- : safe_min_volume_ul in _CONTAINER_TYPES
- : *Protocol.stamp()* smartly calculates max_tip_volume using residual volumes
- : *Protocol.autopick()* now conforms to updated ASC (**not backwards compatible**)
- : Allow single Well reading for Absorbance, Fluorescence and Luminescence
- : *WellGroup.extend()* method to WellGroup
- : Include well properties in outs
- : *Protocol.transfer()* respects when *mix_after* or *mix_before* is explicitly False
- : Protocol.stamp() allows one_tip=True when steps use a *mix_vol* greater than "31:microliter" even if transferred volumes are not all greater than "31:microliter"
- : Protocol.plate_to_magblock() and Protocol.plate_from_magblock()
- : *Protocol.stamp()* has been reformatted to take groups of transfers. This allows for one_tip=True, one_source=True, and WellGroup source and destinations
- : one_tip = True transfers > 750:microliter are transferred with single tip
- : volume tracking for *Protocol.stamp()* ing to/from 384-well plates
- : functionality to harness.py for naming aliquots
- : UserError exception class for returning custom errors from within protocol scripts
- : more recursion in *make_dottable_dict*, a completely unnecessary function you shouldn't use
- : Better handling of default append=true behavior for *Protocol.stamp()*
- : Small bug for transfer with one_source=true fixed

- : Transfers with one_source true does not keep track of the value of volume less than 10^-12
- : *Protocol.stamp()* transfers are not combinable if they use different tip volume types
- : unit conversion from milliliters or nanoliters to microliters in *Well.set_volume()*, *Protocol.provision()*, *Protocol.transfer()*, and *Protocol.distribute()*
- : "outs" section of protocol. Use *Well.set_name()* to name an aliquot
- : name property on Well
- : volume tracking to *Protocol.stamp()* and associated helper functions in autoprotocol.util
- : Arguments to *Protocol.transfer()* for *mix_before* and *mix_after* are now part of **mix_kwargs** to allow for specifying separate parameters for mix_before and mix_after
- : Better error handling in harness.py and accompanying tests
- : Test for more complicated *transfer'ing with 'one_source=True*
- : manually change storage condition destiny of a Container
- : Protocol.store()
- : Storage attribute on Container
- : *Protocol.stamp()* now support selective (row-wise and column-wise) stamping (see docstring for details)
- : semantic versioning fail
- : Arguments to *Protocol.transfer()* for *mix_before* and *mix_after* are now part of **mix_kwargs** to allow for specifying separate parameters for mix_before and mix_after
- : Better error handling in harness.py and accompanying tests
- : Test for more complicated *transfer'ing with 'one_source=True*
- : manually change storage condition destiny of a Container
- : Protocol.store()
- : Storage attribute on Container
- : Error with *transfer'ing with 'one_source=True*
- : unit conversion from milliliters or nanoliters to microliters in *Well.set_volume()*, *Protocol.provision()*, *Protocol.transfer()*, and *Protocol.distribute()*
- : "outs" section of protocol. Use *Well.set_name()* to name an aliquot
- : name property on Well
- : volume tracking to *Protocol.stamp()* and associated helper functions in autoprotocol.util
- : Unit scalar multiplication
- : Error when *Protocol.transfer()* ing over 750uL
- : Error with *Protocol.provision()* ing to multiple wells of the same container
- : semantic versioning fail
- : *Protocol.stamp()* now utilizes the new Autoprotocol *stamp* instruction instead of *Protocol.transfer()*
- : *__repr__* override for Unit class
- : volume tracking to destination wells when using Protocol.dispense()
- : *Stamp* class in autoprotocol.instruction

- : better error handling for *Protocol.transfer()* and *Protocol.distribute()*
- : refactored Protocol methods: *Protocol.ref()*, *Protocol.consolidate()*, *Protocol.transfer()*, *Protocol.distribute()*
- : fixed indentation
- : warnings for *_mul_* and *_div_* scalar Unit operations
- : *Protocol.dispense_full_plate()*
- : *Protocol.dispense()* Instruction and accompanying Protocol method for using a reagent dispenser
- : aliquot++, integer, boolean input types to harness.py
- : properties attribute to *Well*, along with *Well.set_properties()* method
- : melting keyword variables and changes to conditionals in Thermocycle
- : default input value and group and group+ input types in *harness.py*
- : *Protocol.pipette()* is now a private method *_pipette()*
- : *Protocol.sangerseq()* Instruction and method
- : seal takes a "type" parameter that defaults to ultra-clear
- : more tests
- : *Protocol.stamp()* Protocol method for using the 96-channel liquid handler
- : Added *pipette_tools* module containing helper methods for the extra pipetting parameters
- : Additional keyword arguments for *Protocol.transfer()* and *Protocol.distribute()* to customize pipetting
- : *Protocol.oligosynthesize()* Instruction
- : *Protocol.autopick()* Instruction
- : *Protocol.spread()* Instruction
- : *Protocol.flow_analyze()* Instruction
- : co2 parameter in *Protocol.incubate()*
- : *6-flat* container type in *_CONTAINER_TYPES*
- : *96-flat-uv* container type in *_CONTAINER_TYPES*
- : *Container.quadrant()* returns a WellGroup of the 96 wells representing the quadrant passed
- : *Well.add_properties()*
- : Well.properties is an empty hash by default
- : At least some Python3 compatibility
- : *Protocol.stamp()* ing to or from multiple containers now requires that the source or dest variable be passed as a list of *[{"container": <container>, "quadrant": <quadrant>}, ...]*
- : *Protocol.gel_separate()* generates instructions taking wells and matrix type passed
- : tox for testing with multiple versions of python
- : *new_group* keyword parameter on *Protocol.transfer()* and *Protocol.distribute()* to manually break up *Pipette()* Instructions
- : Thermocycle input type in *harness.py*
- : specify *Wells* on a container using *container.wells(1,2,3)'or 'container.wells([1,2,3])*
- : More Python3 Compatibility

- : More Python3 Compatibility
- : Additional type-checks in various functions
- : *Protocol.provision()* Protocol method
- : support for *choice* input type in *harness.py*
- : allow transfer from multiple sources to one destination
- : brought back recursively transferring volumes over 900 microliters
- : *1-flat* plate type to *_CONTAINER_TYPES*
- : support for container names with slashes in them in *harness.py*
- : add *Protocol.consolidate()* Protocol method and accompanying tests
- : *ImagePlate()* class and *Protocol.image_plate()* Protocol method for taking images of containers
- : volume adjustment when *Protocol.spread()* ing
- : collapse *Protocol.provision()* instructions if they're acting on the same container
- : *Protocol.sangerseq()* now accepts a sequencing *type* of *"rca"* or *"standard"* (defaults to "standard")
- : *criteria* and *dataref* fields to *Protocol.autopick()*
- : *Protocol.flash_freeze()* Protocol method and Instruction
- : *Protocol.ref()* behavior when specifying the *id* of an existing container
- : type check in Container.wells
- : README.rst
- : a wild test appeared!
- : link to library documentation at readthedocs.org to README
- : autoprotocol and JSON output examples for almost everything in docs
- : improved documentation tree
- : documentation for *plate_to_mag_adapter* and *plate_from_mag_adapter* **subject to change in near future**
- : documentation punctuation and grammar
- : check that a well already exists in a WellGroup
- : Added folder for sublime text snippets
- : Protocol.serial_dilute_rowwise()
- : Protocol.thermocycle_ramp()
- : More Python3 Compatibility
- : Additional type-checks in various functions
- : *Protocol.provision()* Protocol method
- : support for *choice* input type in *harness.py*
- : allow transfer from multiple sources to one destination
- : brought back recursively transferring volumes over 900 microliters
- : *1-flat* plate type to *_CONTAINER_TYPES*
- : support for container names with slashes in them in *harness.py*

- : add *Protocol.consolidate()* Protocol method and accompanying tests
- : *ImagePlate()* class and *Protocol.image_plate()* Protocol method for taking images of containers
- : volume adjustment when *Protocol.spread()* ing
- : typo in *Protocol.sangerseq()* instruction
- : documentation punctuation and grammar
- : check that a well already exists in a WellGroup
- : Added folder for sublime text snippets
- : *Protocol.stamp()* ing to or from multiple containers now requires that the source or dest variable be passed as a list of *[{"container": <container>, "quadrant": <quadrant>}, ...]*
- : *Protocol.gel_separate()* generates instructions taking wells and matrix type passed
- : tox for testing with multiple versions of python
- : *new_group* keyword parameter on *Protocol.transfer()* and *Protocol.distribute()* to manually break up *Pipette()* Instructions
- : Thermocycle input type in *harness.py*
- : specify *Wells* on a container using *container.wells(1,2,3)'or 'container.wells([1,2,3])*
- : More Python3 Compatibility
- : Transfering liquid from *one_source* actually works now
- : references to specific reagents for *Protocol.dispense()*
- : documentation for *plate_to_mag_adapter* and *plate_from_mag_adapter* **subject to change in near future**
- : *Protocol.pipette()* is now a private method *_pipette()*
- : *Protocol.sangerseq()* Instruction and method
- : seal takes a "type" parameter that defaults to ultra-clear
- : more tests
- : *Protocol.stamp()* Protocol method for using the 96-channel liquid handler
- : Added *pipette_tools* module containing helper methods for the extra pipetting parameters
- : Additional keyword arguments for *Protocol.transfer()* and *Protocol.distribute()* to customize pipetting
- : *Protocol.oligosynthesize()* Instruction
- : *Protocol.autopick()* Instruction
- : *Protocol.spread()* Instruction
- : *Protocol.flow_analyze()* Instruction
- : co2 parameter in *Protocol.incubate()*
- : *6-flat* container type in *_CONTAINER_TYPES*
- : *96-flat-uv* container type in *_CONTAINER_TYPES*
- : *Container.quadrant()* returns a WellGroup of the 96 wells representing the quadrant passed
- : *Well.add_properties()*
- : Well.properties is an empty hash by default
- : At least some Python3 compatibility

- : *Protocol.gel_separate()* generates number of instructions needed for number of wells passed
- : recursion to deal with transferring over 900uL of liquid
- : references to specific matrices and ladders in *Protocol.gel_separate()*
- : refactoring of type checks in *unit.Unit*
- : improved documentation tree
- : melting keyword variables and changes to conditionals in Thermocycle
- : default input value and group and group+ input types in *harness.py*
- : a wild test appeared!
- : link to library documentation at readthedocs.org to README
- : autoprotocol and JSON output examples for almost everything in docs
- : warnings for *_mul_* and *_div_* scalar Unit operations
- : *Protocol.dispense_full_plate()*
- : *Protocol.dispense()* Instruction and accompanying Protocol method for using a reagent dispenser
- : aliquot++, integer, boolean input types to harness.py
- : properties attribute to *Well*, along with *Well.set_properties()* method
- : spelling of luminescence :(
- : *well_type* from *_CONTAINER_TYPES*
- : "speed" parameter in *Protocol.spin()* to "acceleration"
- : README.rst
- : "one_tip" option on *Protocol.transfer()*
- : volume tracking upon *Protocol.transfer()* and *Protocol.distribute()*
- : dead_volume_ul in _CONTAINER_TYPES
- : *WellGroup.indices()* returns a list of string well indices
- : 3-clause BSD license, contributor info
- : *Container.inner_wells()* method to exclude edges
- : harness.py for parameter conversion
- : static methods *Pipette.transfers()* and *Pipette._transferGroup()*
- : NumPy style docstrings for most methods
- : initializing ap-py

CHAPTER 6

Credits

Autoprotocol-python is currently maintained by:

- Vanessa Biggers - polarpine - vanessa@transcriptic.com
- Yang Choo - yangchoo - yangchoo@transcriptic.com
- Peter Lee - transcripticpll - peter@transcriptic.com
- Jim Culver - drjimmypants - jim@transcriptic.com
- Cornelia Scheitz - cojofra - cornelia.scheitz@gmail.com

See all Github contributors

For more information about Autoprotocol and its specification, visit autoprotocol.org

CHAPTER 7

License

Use the sidebar to navigate specific module documentation.

Autoprotocol is a standard way to express experiments in life science. The autoprotocol-python repository contains a python library for generating Autoprotocol.

CHAPTER 8

Installation

```
$ git clone https://github.com/autoprotocol/autoprotocol-python
$ cd autoprotocol-python
$ python setup.py install
```

or, alternatively:

```
$ pip install autoprotocol
```

CHAPTER 9

Building a Protocol

A basic protocol object has empty "refs" and "instructions" stanzas. Various helper methods in the Protocol class are then used to append instructions and refs to the object such as in the simple protocol below:

```
import json
from autoprotocol.protocol import Protocol

#instantiate new Protocol object
p = Protocol()

#append refs (containers) to Protocol object
bacteria = p.ref("bacteria", cont_type="96-pcr", storage="cold_4")
medium = p.ref("medium", cont_type="micro-1.5", storage="cold_4")
reaction_plate = p.ref("reaction_plate", cont_type="96-flat", storage="warm_37")

#distribute medium from 1.5mL tube to reaction wells
p.distribute(medium.well(0).set_volume("1000:microliter"), reaction_plate.wells_from(0,4), "190:micro
#transfer bacteria from source wells to reaction wells
p.transfer(bacteria.wells_from(0,4), reaction_plate.wells_from(0,4),
    ["10:microliter", "20:microliter", "30:microliter", "40:microliter"])
#incubate bacteria at 37 degrees for 5 hours
p.incubate(reaction_plate, "warm_37", "5:hour")
#read absorbance of the first four wells on the reaction plate at 600 nanometers
p.absorbance(reaction_plate, reaction_plate.wells_from(0,4).indices(), "600:nanometer",
    "OD600_reading_01092014")

print json.dumps(p.as_dict(), indent=2)
```

The script above produces the following autoprotocol:

```
{
  "refs": {
    "medium": {
      "new": "micro-1.5",
      "store": {
        "where": "cold_4"
      }
    },
    "bacteria": {
      "new": "96-pcr",
      "store": {
        "where": "cold_4"
      }
    },
```

```
    "reaction_plate": {
      "new": "96-flat",
      "store": {
        "where": "warm_37"
      }
    }
  },
  "instructions": [
    {
      "groups": [
        {
          "distribute": {
            "to": [
              {
                "volume": "190.0:microliter",
                "well": "reaction_plate/0"
              },
              {
                "volume": "190.0:microliter",
                "well": "reaction_plate/1"
              },
              {
                "volume": "190.0:microliter",
                "well": "reaction_plate/2"
              },
              {
                "volume": "190.0:microliter",
                "well": "reaction_plate/3"
              }
            ],
            "from": "medium/0"
          }
        },
        {
          "transfer": [
            {
              "volume": "10.0:microliter",
              "to": "reaction_plate/0",
              "from": "bacteria/0"
            }
          ]
        },
        {
          "transfer": [
            {
              "volume": "20.0:microliter",
              "to": "reaction_plate/1",
              "from": "bacteria/0"
            }
          ]
        },
        {
          "transfer": [
            {
              "volume": "30.0:microliter",
              "to": "reaction_plate/2",
              "from": "bacteria/0"
            }
```

```
          ]
        },
        {
          "transfer": [
            {
              "volume": "40.0:microliter",
              "to": "reaction_plate/3",
              "from": "bacteria/0"
            }
          ]
        }
      ],
      "op": "pipette"
    },
    {
      "duration": "5:hour",
      "where": "warm_37",
      "object": "reaction_plate",
      "shaking": false,
      "op": "incubate"
    },
    {
      "dataref": "OD600_reading_01092014",
      "object": "reaction_plate",
      "wells": [
        "A1",
        "A2",
        "A3",
        "A4"
      ],
      "num_flashes": 25,
      "wavelength": "600:nanometer",
      "op": "absorbance"
    }
  ]
}
```

CHAPTER 10

Contributing

The easiest way to contribute is to fork this repository and submit a pull request. You can also write an email to us if you want to discuss ideas or bugs.

- Vanessa Biggers: vanessa@transcriptic.com
- Max Hodak: max@transcriptic.com

autoprotocol-python is BSD licensed (see LICENSE). Before we can accept your pull request, we require that you sign a CLA (Contributor License Agreement) allowing us to distribute your work under the BSD license. Email one of the authors listed above for more details.

CHAPTER 11

Search the Docs

- genindex
- search

Symbols

A

C

D

E

F

G

H

I