Autoprotocol Documentation

Release 5.4.0

Transcriptic

Apr 19, 2019

Contents

Module containing the main *Protocol* object and associated functions

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

class autoprotocol.protocol.**Protocol**(*refs=None*, *instructions=None*, *propagate_properties=False*)
A Protocol is a sequence of instructions to be executed, and a set of containers on which those instructions act.

Parameters

- **refs** (*list(Ref)*) – Pre-existing refs that the protocol should be populated with.
- **instructions** (*list(Instruction)*) – Pre-existing instructions that the protocol should be populated with.
- **propagate_properties** (*bool, optional*) – Whether liquid handling operations should propagate aliquot properties from source to destination wells.

Examples

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": [
```

(continues on next page)

(continued from previous page)

```
        {
          "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"
    }
  ]
}
```

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.

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)
```

(continues on next page)

(continued from previous page)

```
# 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, optional*) – 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 or ContainerType*) – container type of the Container object that will be generated.
- **storage** (*Enum({"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.
- **cover** (*str, optional*) – name of the cover which will be on the container/ref

Returns Container object generated from the id and container type provided

Return type *Container*

Raises

- RuntimeError – If a container previously referenced in this protocol (existent 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.

add_time_constraint(*from_dict*, *to_dict*, *less_than=None*, *more_than=None*, *mirror=False*, *ideal=None*, *optimization_cost=None*)

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.

Ideal time constraints are sometimes helpful for ensuring that a certain set of operations happen within some specified time. This can be specified by using the *ideal* parameter. There is an optional *optimization_cost* parameter associated with *ideal* time constraints for specifying the penalization system used for calculating deviations from the *ideal* time. When left unspecified, the *optimization_cost* function defaults to linear. Please refer to the ASC for more details on how this is implemented.

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", mirror=True)

# Ideal time constraint
protocol.add_time_constraint(
    {"mark": time_point_1, "state": "start"},
    {"mark": time_point_2, "state": "end"},
    ideal = "30:second",
    optimization_cost = "squared")
```

Autoprotocol Output:

```
{
    "refs": {
        "plate_1": {
            "new": "96-flat",
            "discard": true
        },
        "plate_2": {
            "new": "96-flat",
            "discard": true
        }
```

(continues on next page)

(continued from previous page)


```
    },
    "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
            }
        },
        {
            "from": {
                "instruction_start": 0
            },
            "to": {
                "instruction_end": 1
            },
            "ideal": {
                "value": "5:minute",
                "optimization_cost": "squared"
            }
        }

    ],
    "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 or Unit, optional*) – max time between from_dict and to_dict
- **more_than** (*str or Unit, optional*) – 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)
- **ideal** (*str or Unit, optional*) – ideal time between from_dict and to_dict
- **optimization_cost** (*Enum({"linear", "squared", "exponential"}), optional*) – cost function used for calculating the penalty for missing the *ideal* timing

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']
- TypeError – If any of *ideal*, *more_than*, *less_than* is not a Unit of the 'time' dimension
- 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'
- ValueError – If *optimization_cost* is specified but *ideal* is not
- ValueError – If *more_than* is greater than *less_than*
- ValueError – If *ideal* is smaller than *more_than* or greater than *less_than*
- RuntimeError – If *from_dict* and *to_dict* are equal
- RuntimeError – If from_dict["marker"] and to_dict["marker"] are equal and from_dict["state"] = "end"

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

batch_containers(*containers*, *batch_in=True*, *batch_out=False*)

Batch containers such that they all enter or exit together.

Example Usage:

```
plate_1 = protocol.ref("p1", None, "96-pcr", storage="cold_4")
plate_2 = protocol.ref("p2", None, "96-pcr", storage="cold_4")

protocol.batch_containers([plate_1, plate_2])
```

Autoprotocol Output:

```
{
  "refs": {
    "p1": {
      "new": "96-pcr",
      "store": {
        "where": "cold_4"
      }
    },
    "p2": {
      "new": "96-pcr",
      "store": {
        "where": "cold_4"
      }
    }
  },
  "time_constraints": [
    {
      "from": {
        "ref_start": "p1"
      },
      "less_than": "0:second",
      "to": {
        "ref_start": "p2"
      }
    },
    {
      "from": {
        "ref_start": "p1"
      },
      "more_than": "0:second",
      "to": {
        "ref_start": "p2"
      }
    }
  ]
}
```

Parameters

- **containers** (*list(Container)*) – Containers to batch
- **batch_in** (*bool, optional*) – Batch the entry of containers, default True
- **batch_out** (*bool, optional*) – Batch the exit of containers, default False

Raises

- `TypeError` – If containers is not a list
- `TypeError` – If containers is not a list of Container object

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

store(*container*, *condition*)

Manually adjust the storage destiny for a container used within this protocol.

Parameters

- **container** (Container) – Container used within this protocol
- **condition** (*str*) – New storage destiny for the specified Container

Raises

- TypeError – If container argument is not a Container object
- RuntimeError – If the container passed is not already present in self.refs

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

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

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"
    }
]
```

Parameters

- **source** (*Well or WellGroup or list(Well)*) – Well or wells to transfer liquid from. If multiple source wells are supplied and one_source is set to True, liquid will be transferred 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 transferred from each source well to its corresponding destination well.
- **dest** (*Well or WellGroup or list(Well)*) – 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 or Unit or 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 or Unit, optional*) – Volume representing a droplet_size. The volume of each *transfer* group should be a multiple of this volume.

Returns Returns the *autoprotocol.instruction.AcousticTransfer* instruction created from the specified parameters

Return type *AcousticTransfer*

Raises

- TypeError – Incorrect input types, e.g. source/dest are not Well or WellGroup or list of Well
- RuntimeError – Incorrect length for source and destination
- RuntimeError – Transfer volume not being a multiple of droplet size
- RuntimeError – Insufficient volume in source wells

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},
```

(continues on next page)

(continued from previous page)

```
        {"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
            },
            {
                "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(dict)*) –

```
"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** (*int*) – Library size expressed as an integer of basepairs
- **dataref** (*str*) – Name of sequencing dataset that will be returned.
- **cycles** (*Enum({"read_1", "read_2", "index_1", "index_2"})*) – Parameter specific to Illuminaseq read-length or number of sequenced bases. Refer to the ASC for more details

Returns Returns the *autoprotocol.instruction.IlluminaSeq* instruction created from the specified parameters

Return type *IlluminaSeq*

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.
- KeyError – Invalid keys specified for cycles parameter

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 or str*) – Container with well(s) that contain material to be sequenced.
- **wells** (*list(Well) or WellGroup or Well*) – WellGroup of wells to be measured or a list of well references in the form of ["A1", "B1", "C5", ...]
- **dataref** (*str*) – Name of sequencing dataset that will be returned.
- **type** (*Enum({"standard", "rca"})*) – Sanger sequencing type
- **primer** (*Container, optional*) – 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

Returns Returns the *autoprotocol.instruction.SangerSeq* instruction created from the specified parameters

Return type *SangerSeq*

Raises

- RuntimeError – No primer location specified for rca sequencing type
- ValueError – Wells belong to more than one container
- TypeError – Invalid input type for wells

dispense(*ref*, *reagent*, *columns*, *is_resource_id=False*, *step_size='5:uL'*, *flowrate=None*, *nozzle_position=None*, *pre_dispense=None*, *shape=None*, *shake_after=None*)

Dispense specified reagent to specified columns.

Example Usage:

```
from autoprotocol.liquid_handle.liquid_handle_builders import *
from autoprotocol.instructions import Dispense
from autoprotocol import Protocol

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

(continues on next page)

(continued from previous page)

```
p.dispense(sample_plate,
           "water",
           Dispense.builders.columns(
               [Dispense.builders.column(0, "10:uL"),
                Dispense.builders.column(1, "20:uL"),
                Dispense.builders.column(2, "30:uL"),
                Dispense.builders.column(3, "40:uL"),
                Dispense.builders.column(4, "50:uL")
               ])
           )

p.dispense(
    sample_plate,
    "water",
    Dispense.builders.columns(
        [Dispense.builders.column(0, "10:uL")]
    ),
    Dispense.builders.nozzle_position(
        position_x=Unit("1:mm"),
        position_y=Unit("2:mm"),
        position_z=Unit("20:mm")
    ),
    shape_builder(
        rows=8, columns=1, format="SBS96"
    )
)
```

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"
            }
        ],
        "op": "dispense"
    },
```

(continues on next page)

(continued from previous page)

```
    {
        "reagent": "water",
        "object": "sample_plate",
        "columns": [
            {
                "column": 0,
                "volume": "10:microliter"
            }
        ],
        "nozzle_position" : {
            "position_x" : "1:millimeter",
            "position_y" : "2:millimeter",
            "position_z" : "20:millimeter"
        },
        "shape" : {
            "rows" : 8,
            "columns" : 1,
            "format" : "SBS96"
        }
        "op": "dispense"
    },
]
```

Parameters

- **ref** (Container) – Container for reagent to be dispensed to.
- **reagent** (*str or 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(dict("column": int, "volume": str/Unit))*) – Columns to be dispensed to, in the form of a list(dict) 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>}, . . .]
- **is_resource_id** (*bool, optional*) – If true, interprets reagent as a resource ID
- **step_size** (*str or 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.
- **flowrate** (*str or Unit, optional*) – The rate at which liquid is dispensed into the ref in units of volume/time.
- **nozzle_position** (*dict, optional*) – A dict represent nozzle offsets from the bottom middle of the plate's wells. see Dispense.builders.nozzle_position; specified as {"position_x": Unit, "position_y": Unit, "position_z": Unit}.
- **pre_dispense** (*str or Unit, optional*) – The volume of reagent to be dispensed per-nozzle into waste immediately prior to dispensing into the ref.
- **shape** (*dict, optional*) – The shape of the dispensing head to be used for the dispense. See liquid_handle_builders.shape_builder; specified as {"rows": int, "columns": int, "format": str} with format being a valid SBS format.

- **shake_after** (*dict, optional*) – Parameters that specify how a plate should be shaken at the very end of the instruction execution. See Dispense.builders.shake_after.

Returns Returns the *autoprotocol.instruction.Dispense* instruction created from the specified parameters

Return type *Dispense*

Raises

- TypeError – Invalid input types, e.g. ref is not of type Container
- ValueError – Columns specified is invalid for this container type
- ValueError – Invalid step-size given
- ValueError – Invalid pre-dispense volume

dispense_full_plate(*ref*, *reagent*, *volume*, *is_resource_id=False*, *step_size='5:uL'*, *flowrate=None*, *nozzle_position=None*, *pre_dispense=None*, *shape=None*, *shake_after=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"
            },
            {
```

(continues on next page)

(continued from previous page)

```
                "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 or* 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 *or str*) – Volume of reagent to be dispensed to each well
- **is_resource_id** (*bool, optional*) – If true, interprets reagent as a resource ID
- **step_size** (*str or* 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.
- **flowrate** (*str or* Unit*, optional*) – The rate at which liquid is dispensed into the ref in units of volume/time.
- **nozzle_position** (*dict, optional*) – A dict represent nozzle offsets from the bottom middle of the plate's wells. see Dispense.builders.nozzle_position; specified as

{"position_x": Unit, "position_y": Unit, "position_z": Unit}.

- **pre_dispense** (*str or Unit, optional*) – The volume of reagent to be dispensed per-nozzle into waste immediately prior to dispensing into the ref.
- **shape** (*dict, optional*) – The shape of the dispensing head to be used for the dispense. See liquid_handle_builders.shape_builder; specified as {"rows": int, "columns": int, "format": str} with format being a valid SBS format.
- **shake_after** (*dict, optional*) – Parameters that specify how a plate should be shaken at the very end of the instruction execution. See Dispense.builders.shake_after.

Returns Returns the *autoprotocol.instruction.Dispense* instruction created from the specified parameters

Return type *Dispense*

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 or 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(str)*) – 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”].

Returns Returns the *autoprotocol.instruction.Spin* instruction created from the specified parameters

Return type *Spin*

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.

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

Append a Thermocycle instruction to the list of instructions, with groups is a list(dict) 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

- **1 cycle:**
  - 4 degrees for 30 seconds
- all cycles: Lid temperature at 97 degrees

```
from instruction import Thermocycle

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,
    [
        Thermocycle.builders.group(
            steps=[
                Thermocycle.builders.step("95:celsius", "5:minute")
            ]
        ),
        Thermocycle.builders.group(
            steps=[
                Thermocycle.builders.step("95:celsius", "30:s"),
                Thermocycle.builders.step("56:celsius", "20:s"),
                Thermocycle.builders.step("72:celsius", "20:s"),
            ],
            cycles=30
        ),
        Thermocycle.builders.group(
            steps=[
                Thermocycle.builders.step("72:celsius", "10:minute")
            ]
        ),
        Thermocycle.builders.group(
            steps=[
                Thermocycle.builders.step("4:celsius", "30:s")
            ]
        )
    ],
    lid_temperature="97:celsius"
)
```

Autoprotocol Output:

```
"instructions": [
    {
        "object": "sample_plate",
        "op": "seal"
    },
    {
        "volume": "10:microliter",
        "dataref": null,
        "object": "sample_plate",
        "groups": [
```

(continues on next page)

(continued from previous page)

```
            {
                "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"
                    }
                ]
            },
            {
                "cycles": 1,
                "steps": [
                    {
                        "duration": "30:second",
                        "temperature": "4: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: * 65 degrees for 20 seconds Bottom Row: * 55 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,
    [
        Thermocycle.builders.group(
            steps=[
                Thermocycle.builders.step("95:celsius", "5:minute")
            ]
        ),
        Thermocycle.builders.group(
            steps=[
                Thermocycle.builders.step("95:celsius", "30:s"),
                Thermocycle.builders.step(
                    {"top": "65:celsius", "bottom": "55:celsius"},
                    "20:s"
                ),
                Thermocycle.builders.step("72:celsius", "20:s"),
            ],
            cycles=30
        ),
        Thermocycle.builders.group(
            steps=[
                Thermocycle.builders.step("72:celsius", "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",
```

(continues on next page)

(continued from previous page)

```
                    storage="warm_37")

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

p.thermocycle(
    sample_plate,
    [
        Thermocycle.builders.group(
            steps=[
                Thermocycle.builders.step("95:celsius", "3:minute")
            ]
        ),
        Thermocycle.builders.group(
            steps=[
                Thermocycle.builders.step(
                    "95:celsius",
                    "10:second",
                    read=False
                ),
                Thermocycle.builders.step(
                    "95:celsius",
                    "10:second",
                    read=True
                )
            ],
            cycles=40
        )
    ],
    dataref = "my_qpcr_data",
    dyes = {"SYBR": sample_plate.all_wells().indices()}
)
```

Parameters

- **ref** (Container) – Container to be thermocycled.
- **groups** (*list(dict)*) – List of thermocycling instructions formatted as above
- **volume** (*str or 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 or Unit, optional*) – Temperature at which to start the melting curve.
- **melting_end** (*str or Unit, optional*) – Temperature at which to end the melting curve.
- **melting_increment** (*str or Unit, optional*) – Temperature by which to increment the melting curve. Accepted increment values are between 0.1 and 9.9 degrees celsius.
- **melting_rate** (*str or Unit, optional*) – Specifies the duration of each temperature step in the melting curve.

- **lid_temperature** (*str or Unit, optional*) – Specifies the lid temperature throughout the duration of the thermocycling instruction

Returns Returns the *autoprotocol.instruction.Thermocycle* instruction created from the specified parameters

Return type *Thermocycle*

Raises

- AttributeError – If groups are not properly formatted
- TypeError – If ref to thermocycle is not of type Container.
- ValueError – Container specified cannot be thermocycled
- ValueError – Lid temperature is not within bounds

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 or str*) – The container to be incubated
- **where** (*Enum({"ambient", "warm_37", "cold_4", "cold_20", "cold_80"})*) – Temperature at which to incubate specified container
- **duration** (*Unit or str*) – Length of time to incubate container
- **shaking** (*bool, optional*) – Specify whether or not to shake container if available at the specified temperature

- **co2** (*Number, optional*) – Carbon dioxide percentage
- **uncovered** (*bool, optional*) – Specify whether the container should be uncovered during incubation
- **target_temperature** (*Unit or str, optional*) – Specify a target temperature for a device (eg. an incubating block) to reach during the specified duration.
- **shaking_params** (*dict, optional*) – Specify "path" and "frequency" of shaking parameters to be used with compatible devices (eg. thermoshakes)

Returns Returns the *autoprotocol.instruction.Incubate* instruction created from the specified parameters

Return type *Incubate*

Raises

- TypeError – Invalid input types given, e.g. ref is not of type Container
- RuntimeError – Incubating uncovered in a location which is shaking

absorbance(*ref*, *wells*, *wavelength*, *dataref*, *flashes=25*, *incubate_before=None*, *temperature=None*, *settle_time=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"
```

(continues on next page)

(continued from previous page)

```
    }
]
```

Parameters

- **ref** (*str or Container*) – Object to execute the absorbance read on
- **wells** (*list(Well) or WellGroup or Well*) – WellGroup of wells to be measured or a list of well references in the form of ["A1", "B1", "C5", ...]
- **wavelength** (*str or 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*) – number of flashes for the read
- **temperature** (*str or Unit, optional*) – set temperature to heat plate reading chamber
- **settle_time** (*Unit, optional*) – the time before the start of the measurement, defaults to vendor specifications
- **incubate_before** (*dict, optional*) – parameters for incubation before executing the plate read See Also `Absorbance.builders.incubate_params()`

Returns Returns the *autoprotocol.instruction.Absorbance* instruction created from the specified parameters

Return type *Absorbance*

Raises

- `TypeError` – Invalid input types, e.g. wells given is of type Well, WellGroup or list of wells
- `ValueError` – Wells specified are not from the same container
- `ValueError` – Settle time has to be greater than 0
- `UnitError` – Settle time is not of type Unit

fluorescence(*ref*, *wells*, *excitation*, *emission*, *dataref*, *flashes=25*, *temperature=None*, *gain=None*, *incubate_before=None*, *detection_mode=None*, *position_z=None*, *settle_time=None*, *lag_time=None*, *integration_time=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 or Container*) – Container to plate read.
- **wells** (*list(Well) or WellGroup or Well*) – WellGroup of wells to be measured or a list of well references in the form of [“A1”, “B1”, “C5”, . . .]
- **excitation** (*str or Unit*) – Wavelength of light used to excite the wells indicated
- **emission** (*str or Unit*) – Wavelength of light to be measured for the indicated wells
- **dataref** (*str*) – Name of this specific dataset of measured fluoresence
- **flashes** (*int, optional*) – Number of flashes.
- **temperature** (*str or 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*) – parameters for incubation before executing the plate read See Also `Fluorescence.builders.incubate_params()`
- **detection_mode** (*str, optional*) – set the detection mode of the optics, [“top”, “bottom”], defaults to vendor specified defaults.
- **position_z** (*dict, optional*) – distance from the optics to the surface of the plate transport, only valid for “top” detection_mode and vendor capabilities. Specified as either a set distance - “manual”, OR calculated from a WellGroup - “calculated_from_wells”. Only one position_z determination may be specified

```
position_z = {
    "manual": Unit
```

(continues on next page)

(continued from previous page)

```
    - OR -
    "calculated_from_wells": []
}
```

- **settle_time** (Unit, *optional*) – the time before the start of the measurement, defaults to vendor specifications
- **lag_time** (Unit, *optional*) – time between flashes and the start of the signal integration, defaults to vendor specifications
- **integration_time** (Unit, *optional*) – duration of the signal recording, per Well, defaults to vendor specifications

Examples

position_z:

```
position_z = {
    "calculated_from_wells": ["plate/A1", "plate/A2"]
}

-OR-

position_z = {
    "manual": "20:micrometer"
}
```

Returns Returns the *autoprotocol.instruction.Fluorescence* instruction created from the specified parameters

Return type *Fluorescence*

Raises

- TypeError – Invalid input types, e.g. wells given is of type Well, WellGroup or list of wells
- ValueError – Wells specified are not from the same container
- ValueError – Settle time, integration time or lag time has to be greater than 0
- UnitError – Settle time, integration time, lag time or position z is not of type Unit
- ValueError – Unknown value given for *detection_mode*
- ValueError – Position z specified for non-top detection mode
- KeyError – For position_z, only *manual* and *calculated_from_wells* is allowed
- NotImplementedError – Specifying *calculated_from_wells* as that has not been implemented yet

luminescence(*ref*, *wells*, *dataref*, *incubate_before=None*, *temperature=None*, *settle_time=None*, *integration_time=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 or Container*) – Container to plate read.
- **wells** (*list(Well) or WellGroup or 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 or Unit, optional*) – set temperature to heat plate reading chamber
- **settle_time** (*Unit, optional*) – the time before the start of the measurement, defaults to vendor specifications
- **incubate_before** (*dict, optional*) – parameters for incubation before executing the plate read See Also `Absorbance.builders.incubate_params()`
- **integration_time** (*Unit, optional*) – duration of the signal recording, per Well, defaults to vendor specifications

Returns Returns the *autoprotocol.instruction.Luminescence* instruction created from the specified parameters

Return type *Luminescence*

Raises

- `TypeError` – Invalid input types, e.g. wells given is of type Well, WellGroup or list of wells
- `ValueError` – Wells specified are not from the same container
- `ValueError` – Settle time or integration time has to be greater than 0
- `UnitError` – Settle time or integration time is not of type Unit

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(Well) or WellGroup or Well*) – List of wells or WellGroup containing wells to be separated on gel.
- **volume** (*str or 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 or Unit*) – Length of time to run current through gel.
- **dataref** (*str*) – Name of this set of gel separation results.

Returns Returns the *autoprotocol.instruction.GelSeparate* instruction created from the specified parameters

Return type *GelSeparate*

Raises

- TypeError – Invalid input types, e.g. wells given is of type Well, WellGroup or list of wells
- ValueError – Specifying more wells than the number of available lanes in the selected matrix

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,
```

(continues on next page)

(continued from previous page)

```
                "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(dict)*) – List of gel extraction parameters See Also `GelPurify.builders.extract()`
- **volume** (*str or 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.

Returns Returns the *autoprotocol.instruction.GelPurify* instruction created from the specified parameters

Return type *GelPurify*

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.

seal(*ref*, *type=None*, *mode=None*, *temperature=None*, *duration=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, mode="thermal", temperature="160:celsius")
```

Autoprotocol Output:

```
"instructions": [
    {
        "object": "sample_plate",
        "type": "ultra-clear",
        "mode": "thermal",
        "mode_params": {
            "temperature": "160:celsius"
        }
        "op": "seal"
    }
]
```

Parameters

- **ref** (Container) – Container to be sealed
- **type** (*str, optional*) – Seal type to be used, such as "ultra-clear" or "foil".
- **mode** (*str, optional*) – Sealing method to be used, such as "thermal" or "adhesive". Defaults to None, which is interpreted sensibly based on the execution environment.
- **temperature** (Unit *or str, optional*) – Temperature at which to melt the sealing film onto the ref. Only applicable to thermal sealing; not respected if the sealing mode is adhesive. If unspecified, thermal sealing temperature defaults correspond with manufacturer-recommended or internally-optimized values for the target container type. Applies only to thermal sealing.
- **duration** (Unit *or str, optional*) – Duration for which to press the (heated, if thermal) seal down on the ref. Defaults to manufacturer-recommended or internally- optimized seal times for the target container type. Currently applies only to thermal sealing.

Returns Returns the *autoprotocol.instruction.Seal* instruction created from the specified parameters

Return type *Seal*

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 the sealing mode is invalid, or incompatible with the given ref
- RuntimeError – If thermal sealing params (temperature and/or duration) are specified alongside an adhesive sealing mode.
- RuntimeError – If specified thermal sealing parameters are invalid
- RuntimeError – If container is already covered with a lid.

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.

Returns Returns the *autoprotocol.instruction.Unseal* instruction created from the specified parameters

Return type *Unseal*

Raises

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

cover (*ref*, *lid=None*, *retrieve_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",
```

(continues on next page)

(continued from previous page)

```
        "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.
- **retrieve_lid** (*bool, optional*) – Flag to retrieve lid from previously stored location (see uncover).

Returns Returns the *autoprotocol.instruction.Cover* instruction created from the specified parameters

Return type *Cover*

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.
- TypeError – If retrieve_lid is not a boolean.

uncover(*ref*, *store_lid=None*)

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.
- **store_lid** (*bool, optional*) – Flag to store the uncovered lid.

Returns Returns the *autoprotocol.instruction.Uncover* instruction created from the specified parameters

Return type *Uncover*

Raises

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

flow_cytometry(*dataref*, *samples*, *lasers*, *collection_conditions*, *width_threshold=None*, *window_extension=None*, *remove_coincident_events=None*)

A non-ambiguous set of parameters for performing flow cytometry.

Parameters

- **dataref** (*str*) – Name of dataset that will be returned.
- **samples** (*list(Well) or Well or WellGroup*) – Wells to be analyzed
- **lasers** (*list(dict)*) – See FlowCytometryBuilders.laser.
- **collection_conditions** (*dict*) – See FlowCytometryBuilders.collection_conditions.
- **width_threshold** (*int or float, optional*) – Threshold to determine width measurement.
- **window_extension** (*int or float, optional*) – Front and rear window extension.
- **remove_coincident_events** (*bool, optional*) – Remove coincident events.

Returns Returns a *autoprotocol.instruction.FlowCytometry* instruction created from the specified parameters.

Return type *FlowCytometry*

Raises

- TypeError – If *lasers* is not of type list.
- TypeError – If *samples* is not of type Well, list of Well, or WellGroup.
- TypeError – If *width_threshold* is not a number.
- TypeError – If *window_extension* is not a number.
- TypeError – If *remove_coincident_events* is not of type bool.

Examples

Example flow cytometry protocol

```
p = Protocol()
plate = p.ref("sample-plate", cont_type="384-flat", discard=True)

lasers = [FlowCytometry.builders.laser(
    excitation="405:nanometers",
    channels=[
        FlowCytometry.builders.channel(
            emission_filter=FlowCytometry.builders.emission_filter(
                channel_name="VL1",
                shortpass="415:nanometers",
                longpass="465:nanometers"
            ),
            detector_gain="10:millivolts"
        )
    ]
)]

collection_conds = FlowCytometry.builders.collection_conditions(
    acquisition_volume="5.0:ul",
    flowrate="12.5:ul/min",
    wait_time="10:seconds",
    mix_cycles=10,
    mix_volume="10:ul",
    rinse_cycles=10
)

p.flow_cytometry("flow-1234", plate.wells_from(0, 3), lasers,
                 collection_conds)
```

Autoprotocol Output:

```
{
  "op": "flow_cytometry",
  "dataref": "flow-1234",
  "samples": [
    "sample-plate/0",
    "sample-plate/1",
    "sample-plate/2"
  ],
  "lasers": [
    {
      "excitation": "405:nanometer",
      "channels": [
        {
          "emission_filter": {
            "channel_name": "VL1",
            "shortpass": "415:nanometer",
            "longpass": "465:nanometer"
          },
          "detector_gain": "10:millivolt"
        }
      ]
    }
  ],
  "collection_conditions": {
    "acquisition_volume": "5:microliter",
    "flowrate": "12.5:microliter/minute",
```

(continues on next page)

(continued from previous page)

```
    "stop_criteria": {
      "volume": "5:microliter"
    },
    "wait_time": "10:second",
    "mix_cycles": 10,
    "mix_volume": "10:microliter",
    "rinse_cycles": 10
  }
}
```

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,
```

(continues on next page)

(continued from previous page)

```
            "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(dict)*) – 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(dict)*) – 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(dict), 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(dict), optional*) – Optional list of positive control wells in the form of:

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

Returns Returns the *autoprotocol.instruction.FlowAnalyze* instruction created from the specified parameters

Return type *FlowAnalyze*

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.

`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(dict)*) – 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,
        //   e.g. "GCGACTC/3Phos/" for a 3' phosphorylation
        //   e.g. "aggg/iAzideN/cgcgc" for an
        //   internal modification
        "scale": "25nm" | "100nm" | "250nm" | "1um",
        "purification": "standard" | "page" | "hplc",
        // default: standard
    }, ...
]
```

Returns Returns the *autoprotocol.instruction.Oligosynthesize* instruction created from the specified parameters

Return type *Oligosynthesize*

autopick(*sources*, *dests*, *min_abort=0*, *criteria=None*, *dataref='autopick'*)
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 or WellGroup or list(Well)*) – Reference wells containing agar and colonies to pick
- **dests** (*Well or WellGroup or list(Well)*) – 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.
- **dataref** (*str*) – Name of dataset to save the picked colonies to

Returns Returns the *autoprotocol.instruction.Autopick* instruction created from the specified parameters

Return type *Autopick*

Raises

- TypeError – Invalid input types for sources and dests
- ValueError – Source wells are not all from the same container

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"
            }
          }
        ]
```

(continues on next page)

(continued from previous page)

```
      ],
      "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 or 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

Returns Returns the *autoprotocol.instruction.MagneticTransfer* instruction created from the specified parameters

Return type *MagneticTransfer*

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 or 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 or Unit, optional*) – 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

Returns Returns the *autoprotocol.instruction.MagneticTransfer* instruction created from the specified parameters

Return type *MagneticTransfer*

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,
                        "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 or 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 or 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

Returns Returns the *autoprotocol.instruction.MagneticTransfer* instruction created from the specified parameters

Return type *MagneticTransfer*

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 or Unit*) – Total time for this sub-operation
- **frequency** (*str or 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 or 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

Returns Returns the *autoprotocol.instruction.MagneticTransfer* instruction created from the specified parameters

Return type *MagneticTransfer*

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 or Unit*) – Total time for this sub-operation
- **frequency** (*str or 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 or 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

Returns Returns the *autoprotocol.instruction.MagneticTransfer* instruction created from the specified parameters

Return type *MagneticTransfer*

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"
    },
    {
```

(continues on next page)

(continued from previous page)

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

Returns Returns the *autoprotocol.instruction.ImagePlate* instruction created from the specified parameters

Return type *ImagePlate*

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 or WellGroup or list(Well)*) – Destination(s) for specified resource.
- **volumes** (*str or Unit or list(str) or list(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)
- ValueError – Volume to provision exceeds max capacity of well

Returns Returns the *autoprotocol.instruction.Provision* instruction created from the specified parameters

Return type *Provision*

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

Returns Returns the *autoprotocol.instruction.FlashFreeze* instruction created from the specified parameters

Return type *FlashFreeze*

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(Well) or WellGroup or Well*) – WellGroup of wells to be measured
- **volume** (*str or 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"].

Returns Returns the *autoprotocol.instruction.MeasureConcentration* instruction created from the specified parameters

Return type *MeasureConcentration*

Raises TypeError – *wells* specified is not of a valid input type

measure_mass(*container*, *dataref*)

Measure the mass of a container.

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
```

(continues on next page)

(continued from previous page)

```
        }
    },
    "instructions": [
        {
            "dataref": "test_data",
            "object": [
                "test_plate"
            ],
            "op": "measure_mass"
        }
    ]
}
```

Parameters

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

Returns Returns the *autoprotocol.instruction.MeasureMass* instruction created from the specified parameters

Return type *MeasureMass*

Raises TypeError – Input given is not of type Container

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

Returns Returns the *autoprotocol.instruction.MeasureVolume* instruction created from the specified parameters

Return type *MeasureVolume*

Raises TypeError – *wells* specified is not of a valid input type

count_cells(*wells*, *volume*, *dataref*, *labels=None*)

Count the number of cells in a sample that are positive/negative for a given set of labels.

Example Usage:

```
p = Protocol()

cell_suspension = p.ref(
    "cells_with_trypan_blue",
    id=None,
    cont_type="micro-1.5",
    discard=True
)
p.count_cells(
    cell_suspension.well(0),
    "10:microliter",
    "my_cell_count",
    ["trypan_blue"]
)
```

Autoprotocol Output:

```
{
    "refs": {
        "cells_with_trypan_blue": {
            "new": "micro-1.5",
            "discard": true
        }
    },
    "instructions": [
        {
            "dataref": "my_cell_count",
            "volume": "10:microliter",
            "wells": [
                "cells_with_trypan_blue/0"
            ],
            "labels": [
                "trypan_blue"
            ],
            "op": "count_cells"
        }
    ]
}
```

Parameters

- **wells** (*Well or list(Well) or WellGroup*) – List of wells that will be used for cell counting.
- **volume** (*Unit*) – Volume that should be consumed from each well for the purpose of cell counting.
- **dataref** (*str*) – Name of dataset that will be returned.
- **labels** (*list(string), optional*) – Cells will be scored for presence or absence of each label in this list. If staining is required to visualize these labels, they must be added before execution of this instruction.

Returns Returns the *autoprotocol.instruction.CountCells* instruction created from the specified parameters

Return type *CountCells*

Raises TypeError – *wells* specified is not of a valid input type

spectrophotometry(*dataref*, *obj*, *groups*, *interval=None*, *num_intervals=None*, *temperature=None*, *shake_before=None*)

Generates an instruction with one or more plate reading steps executed on a single plate with the same device. This could be executed once, or at a defined interval, across some total duration.

Example Usage:

```
p = Protocol()
read_plate = p.ref("read plate", cont_type="96-flat", discard=True)

groups = Spectrophotometry.builders.groups(
    [
        Spectrophotometry.builders.group(
            "absorbance",
            Spectrophotometry.builders.absorbance_mode_params(
                wells=read_plate.wells(0, 1),
                wavelength=["100:nanometer", "200:nanometer"],
                num_flashes=15,
                settle_time="1:second"
            )
        ),
        Spectrophotometry.builders.group(
            "fluorescence",
            Spectrophotometry.builders.fluorescence_mode_params(
                wells=read_plate.wells(0, 1),
                excitation=[
                    Spectrophotometry.builders.wavelength_selection(
                        ideal="650:nanometer"
                    )
                ],
                emission=[
                    Spectrophotometry.builders.wavelength_selection(
                        shortpass="600:nanometer",
                        longpass="700:nanometer"
                    )
                ],
                num_flashes=15,
                settle_time="1:second",
                lag_time="9:second",
                integration_time="2:second",
                gain=0.3,
```

(continues on next page)

(continued from previous page)

```
                read_position="top"
            )
        ),
        Spectrophotometry.builders.group(
            "luminescence",
            Spectrophotometry.builders.luminescence_mode_params(
                wells=read_plate.wells(0, 1),
                num_flashes=15,
                settle_time="1:second",
                integration_time="2:second",
                gain=0.3
            )
        ),
        Spectrophotometry.builders.group(
            "shake",
            Spectrophotometry.builders.shake_mode_params(
                duration="1:second",
                frequency="9:hertz",
                path="ccw_orbital",
                amplitude="1:mm"
            )
        ),
    ]
)

shake_before = Spectrophotometry.builders.shake_before(
    duration="10:minute",
    frequency="5:hertz",
    path="ccw_orbital",
    amplitude="1:mm"
)

p.spectrophotometry(
    dataref="test data",
    obj=read_plate,
    groups=groups,
    interval="10:minute",
    num_intervals=2,
    temperature="37:celsius",
    shake_before=shake_before
)
```

Autoprotocol Output:


```
{
  "op": "spectrophotometry",
  "dataref": "test data",
  "object": "read plate",
  "groups": [
    {
      "mode": "absorbance",
      "mode_params": {
        "wells": [
          "read plate/0",
          "read plate/1"
        ],
        "wavelength": [
```


(continues on next page)

(continued from previous page)


```
          "100:nanometer",
          "200:nanometer"
        ],
        "num_flashes": 15,
        "settle_time": "1:second"
      }
    },
    {
      "mode": "fluorescence",
      "mode_params": {
        "wells": [
          "read plate/0",
          "read plate/1"
        ],
        "excitation": [
          {
            "ideal": "650:nanometer"
          }
        ],
        "emission": [
          {
            "shortpass": "600:nanometer",
            "longpass": "700:nanometer"
          }
        ],
        "num_flashes": 15,
        "settle_time": "1:second",
        "lag_time": "9:second",
        "integration_time": "2:second",
        "gain": 0.3,
        "read_position": "top"
      }
    },
    {
      "mode": "luminescence",
      "mode_params": {
        "wells": [
          "read plate/0",
          "read plate/1"
        ],
        "num_flashes": 15,
        "settle_time": "1:second",
        "integration_time": "2:second",
        "gain": 0.3
      }
    },
    {
      "mode": "shake",
      "mode_params": {
        "duration": "1:second",
        "frequency": "9:hertz",
        "path": "ccw_orbital",
        "amplitude": "1:millimeter"
      }
    }
  ],
  "interval": "10:minute",
```


(continues on next page)

(continued from previous page)

```
  "num_intervals": 2,
  "temperature": "37:celsius",
  "shake_before": {
    "duration": "10:minute",
    "frequency": "5:hertz",
    "path": "ccw_orbital",
    "amplitude": "1:millimeter"
  }
}
```

Parameters

- **dataref** (*str*) – Name of the resultant dataset to be returned.
- **obj** (*Container or str*) – Container to be read.
- **groups** (*list*) – A list of groups generated by SpectrophotometryBuilders groups builders, any of absorbance_mode_params, fluorescence_mode_params, luminescence_mode_params, or shake_mode_params.
- **interval** (*Unit or str, optional*) – The time between each of the read intervals.
- **num_intervals** (*int, optional*) – The number of times that the groups should be executed.
- **temperature** (*Unit or str, optional*) – The temperature that the entire instruction should be executed at.
- **shake_before** (*dict, optional*) – A dict of params generated by SpectrophotometryBuilders.shake_before that dictates how the obj should be incubated with shaking before any of the groups are executed.

Returns Returns the *autoprotocol.instruction.Spectrophotometry* instruction created from the specified parameters

Return type *Spectrophotometry*

Raises

- TypeError – Invalid num_intervals specified, must be an int
- ValueError – No interval specified but shake groups specified with no duration

transfer(*source*, *destination*, *volume*, *rows=1*, *columns=1*, *source_liquid=<class 'autoprotocol.liquid_handle.liquid_class.LiquidClass'>*, *destination_liquid=<class 'autoprotocol.liquid_handle.liquid_class.LiquidClass'>*, *method=<class 'autoprotocol.liquid_handle.transfer.Transfer'>*, *one_tip=False*)

Generates LiquidHandle instructions between wells

Transfer liquid between specified pairs of source & destination wells.

Parameters

- **source** (*Well or WellGroup or list(Well)*) – Well(s) to transfer liquid from.
- **destination** (*Well or WellGroup or list(Well)*) – Well(s) to transfer liquid to.

- **volume** (*str or Unit or list(str) or list(Unit)*) – Volume(s) of liquid to be transferred from source wells to destination wells. The number of volumes specified must correspond to the number of destination wells.
- **rows** (*int, optional*) – Number of rows to be concurrently transferred
- **columns** (*int, optional*) – Number of columns to be concurrently transferred
- **source_liquid** (*LiquidClass or list(LiquidClass), optional*) – Type(s) of liquid contained in the source Well. This affects the aspirate and dispense behavior including the flowrates, liquid level detection thresholds, and physical movements.
- **destination_liquid** (*LiquidClass or list(LiquidClass), optional*) – Type(s) of liquid contained in the destination Well. This affects liquid level detection thresholds.
- **method** (*Transfer or list(Transfer), optional*) – Integrates with the specified source_liquid and destination_liquid to define a set of physical movements.
- **one_tip** (*bool, optional*) – If True then a single tip will be used for all operations

Returns Returns a list of *autoprotocol.instruction.LiquidHandle* instructions created from the specified parameters

Return type list(*LiquidHandle*)

Raises

- ValueError – if the specified parameters can't be interpreted as lists of equal length
- ValueError – if one_tip is true, but not all transfer methods have a tip_type

Examples

Transfer between two single wells

```
from autoprotocol import Protocol, Unit

p = Protocol()
source = p.ref("source", cont_type="384-flat", discard=True)
destination = p.ref(
    "destination", cont_type="394-pcr", discard=True
)
p.transfer(source.well(0), destination.well(1), "5:ul")
```

Sequential transfers between two groups of wells

```
sources = source.wells_from(0, 8, columnwise=True)
dests = destination.wells_from(1, 8, columnwise=True)
volumes = [Unit(x, "ul") for x in range(1, 9)]
p.transfer(sources, dests, volumes)
```

Concurrent transfers between two groups of wells

```
# single-column concurrent transfer
p.transfer(
    source.well(0), destination.well(1), "5:ul", rows=8
)

# 96-well concurrent transfer from the A1 to B2 quadrants
```

(continues on next page)

(continued from previous page)

```
p.transfer(
    source.well(0), destination.well(13), "5:ul", rows=8, columns=12
)

# 384-well concurrent transfer
p.transfer(
    source.well(0), destination.well(0), "5:ul", rows=16, columns=24
)
```

Transfer with extra parameters

```
from autoprotocol.liquid_handle import Transfer
from autoprotocol.instruction import LiquidHandle

p.transfer(
    source.well(0), destination.well(0), "5:ul",
    method=Transfer(
        mix_before=True,
        dispense_z=LiquidHandle.builders.position_z(
           reference="well_top"
        )
    )
)
```

Transfer using other built in Transfer methods

```
from autoprotocol.liquid_handle import DryWellTransfer

p.transfer(
    source.well(0), destination.well(1), "5:ul",
    method=DryWellTransfer
)
```

For examples of other more complicated behavior, see the documentation for LiquidHandleMethod.

See also:

`Transfer()` base LiquidHandleMethod for transfer operations

`mix`(*well*, *volume*, *rows=1*, *columns=1*, *liquid=<class 'autoprotocol.liquid_handle.liquid_class.LiquidClass'>*, *method=<class 'autoprotocol.liquid_handle.mix.Mix'>*, *one_tip=False*)
Generates LiquidHandle instructions within wells

Mix liquid in specified wells.

Parameters

- **`well`** (*`Well or WellGroup or list(Well)`*) – Well(s) to be mixed.
- **`volume`** (*`str or Unit or list(str) or list(Unit)`*) – Volume(s) of liquid to be mixed within the specified well(s). The number of volume(s) specified must correspond with the number of well(s).
- **`rows`** (*`int, optional`*) – Number of rows to be concurrently mixed
- **`columns`** (*`int, optional`*) – Number of columns to be concurrently mixed

- **liquid** (*LiquidClass or list(LiquidClass), optional*) – Type(s) of liquid contained in the Well(s). This affects the aspirate and dispense behavior including the flowrates, liquid level detection thresholds, and physical movements.
- **method** (*Mix or list(Mix), optional*) – Method(s) with which Integrates with the specified liquid to define a set of physical movements.
- **one_tip** (*bool, optional*) – If True then a single tip will be used for all operations

Returns Returns a list of *autoprotocol.instruction.LiquidHandle* instructions created from the specified parameters

Return type list(*LiquidHandle*)

Raises

- ValueError – if the specified parameters can't be interpreted as lists of equal length
- ValueError – if one_tip is true, but not all mix methods have a tip_type
- ValueError – if the specified volume is larger than the maximum tip capacity of the available liquid_handling devices for a given mix

Examples

Mix within a single well

```
from autoprotocol import Protocol, Unit

p = Protocol()
plate = p.ref("example_plate", cont_type="384-flat", discard=True)

p.mix(plate.well(0), "5:ul")
```

Sequential mixes within multiple wells

```
wells = plate.wells_from(0, 8, columnwise=True)
volumes = [Unit(x, "ul") for x in range(1, 9)]
p.mix(wells, volumes)
```

Concurrent mixes within multiple wells

```
# single-column concurrent mix
p.mix(plate.well(0), "5:ul", rows=8)

# 96-well concurrent mix in the A1 quadrant
p.mix(plate.well(0), "5:ul", rows=8, columns=12)

# 96-well concurrent mix in the A2 quadrant
p.mix(plate.well(1), "5:ul", rows=8, columns=12)

# 384-well concurrent mix
p.mix(plate.well(0), "5:ul", rows=16, columns=24)
```

Mix with extra parameters

```
from autoprotocol.liquid_handle import Mix
from autoprotocol.instruction import LiquidHandle

```

(continues on next page)

(continued from previous page)

```
p.mix(
    plate.well(0), "5:ul", rows=8,
    method=Mix(
        mix_params=LiquidHandle.builders.mix(

        )
    )
)
```

See also:

`Mix()` base LiquidHandleMethod for mix operations

`spread`(*source*, *dest*, *volume='50:microliter'*, *dispense_speed='20:microliter/second'*)

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

Uses a spiral pattern generated by a set of liquid_handle instructions.

Example Usage: .. code-block:: python

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")

Parameters

- **`source`** (`Well`) – Source of material to spread on agar
- **`dest`** (`Well`) – Reference to destination location (plate containing agar)
- **`volume`** (*`str or Unit, optional`*) – Volume of source material to spread on agar
- **`dispense_speed`** (*`str or Unit, optional`*) – Speed at which to dispense source aliquot across agar surface

Returns Returns a *`autoprotocol.instruction.LiquidHandle`* instruction created from the specified parameters

Return type *LiquidHandle*

Raises

- `TypeError` – If specified source is not of type Well
- `TypeError` – If specified destination is not of type Well

CHAPTER 1

autoprotocol.instruction module

Contains all the Autoprotocol Instruction objects

copyright 2018 by The Autoprotocol Development Team, see AUTHORS for more details.

license BSD, see LICENSE for more details

class autoprotocol.instruction.**Instruction**(*op*, *data*)
Base class for an instruction that is to later be encoded as JSON.

class autoprotocol.instruction.**MagneticTransfer**(*groups*, *magnetic_head*)
A magnetic_transfer instruction is constructed as a list of lists of groups, executed in order, where each group is a collect, release, dry, incubate, or mix sub-operation. These sub-operations control the behavior of tips which can be magnetized, and a heating platform. Groups in the same list of groups use the same tips.

Parameters

- **groups** (*list(list(dict))*) – dict in the groups should belong to one of the following categories:

  collect: Collects beads from the specified "object" by raising and lowering magnetized tips repeatedly with an optional pause at well bottom.

  release: Release beads from unmagnetized tips by oscillating the tips vertically into and out of the "object".

  dry: Dry beads on magnetized tips above and outside the "object".

  incubate: Incubate the "object".

  mix: Oscillate the tips into and out of the "object"

- **magnetic_head** (*str*) – Head-type used for this instruction

class autoprotocol.instruction.**Dispense**(*object*, *columns*, *reagent=None*, *resource_id=None*, *reagent_source=None*, *step_size=None*, *flowrate=None*, *nozzle_position=None*, *pre_dispense=None*, *shape=None*, *shake_after=None*)
Dispense specified reagent to specified columns. Only one of reagent, resource_id, and reagent_source can be

specified for a given instruction.

Parameters

- **object** (*Container or str*) – Container for reagent to be dispensed to.
- **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 indexed from 0. [{"column": <column num>, "volume": <volume>}, ...]
- **reagent** (*str, optional*) – Reagent to be dispensed.
- **resource_id** (*str, optional*) – Resource to be dispensed.
- **reagent_source** (*Well, optional*) – Aliquot to be dispensed from.
- **step_size** (*str or Unit, optional*) – Specifies that the dispense operation must be executed using a pump that has a dispensing resolution of step_size.
- **flowrate** (*str or Unit, optional*) – The rate at which the peristaltic pump should dispense in Units of flow rate, e.g. microliter/second.
- **nozzle_position** (*dict, optional*) – A dict represent nozzle offsets from the center of the bottom of the plate's well. see Dispense.builders.nozzle_position; specified as {"position_x": Unit, "position_y": Unit, "position_z": Unit}.
- **pre_dispense** (*str or Unit, optional*) – The volume of reagent to be dispensed per-nozzle into waste immediately prior to dispensing into the ref.
- **shape** (*dict, optional*) – The shape of the dispensing head to be used for the dispense. See liquid_handle_builders.shape_builder; specified as {"rows": int, "columns": int, "format": str} with format being a valid SBS format.
- **shake_after** (*dict, optional*) – Parameters that specify how a plate should be shaken at the very end of the instruction execution. {"duration": Unit, "frequency": Unit, "path": str, "amplitude": Unit}

class autoprotocol.instruction.**AcousticTransfer**(*groups*, *droplet_size*)

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

Parameters

- **groups** (*list(dict)*) – List of *transfer* groups in the form of:

```
{
    "transfer": [
        {
            "to": "foo/A1",
            "from": "bar/A1",
            "volume": "1:nl"
        }
    ]
}
```

- **droplet_size** (*str or Unit*) – Volume representing a droplet_size. The volume of each transfer should be a multiple of this volume.

class autoprotocol.instruction.**Spin**(*object*, *acceleration*, *duration*, *flow_direction=None*, *spin_direction=None*)

Apply the specified amount of acceleration to a plate using a centrifuge.

Parameters

- **object** (*Ref or str*) – Container to be centrifuged.
- **acceleration** (*str*) – Amount of acceleration to be applied to the container, expressed in units of "g" or "meter/second^2"
- **duration** (*str or Unit*) – Amount of time to apply acceleration.
- **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(str)*) – 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". If "flow_direction" is "outward", then "spin_direction" defaults to ["cw", "ccw"]. If "flow_direction" is "inward", then "spin_direction" defaults to ["cw"].

class autoprotocol.instruction.**Thermocycle**(*object*, *groups*, *volume='25:microliter'*, *dataref=None*, *dyes=None*, *melting=None*, *lid_temperature=None*)

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

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

To specify a melting curve, all four melting-relevant parameters must have a value.

Parameters

- **object** (*str or Ref*) – Container to be thermocycled
- **groups** (*list(dict)*) – List of thermocycling instructions formatted as above
- **volume** (*str or 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** (*dict*) – Melting parameters See Also `Thermocycle.builders.melting()`
- **lid_temperature** (*str or Unit*) – Specifies the lid temperature throughout the duration of the instruction

Raises

- `ValueError` – If one of dataref and dyes is specified but the other isn't
- `ValueError` – If melting curve parameters are specified but dyes isn't

class autoprotocol.instruction.**Incubate**(*object*, *where*, *duration*, *shaking=False*, *co2=0*, *target_temperature=None*, *shaking_params=None*)

Store a sample in a specific environment for a given duration. Once the duration has elapsed, the sample will be returned to the ambient environment until it is next used in an instruction.

Parameters

- **object** (*Ref or str*) – The container to be incubated
- **where** (*Enum({"ambient", "warm_37", "cold_4", "cold_20", "cold_80"})*) – Temperature at which to incubate specified container
- **duration** (*Unit or str*) – Length of time to incubate container
- **shaking** (*bool, optional*) – Specify whether or not to shake container if available at the specified temperature
- **target_temperature** (*Unit or str, optional*) – Specify a target temperature for a device (eg. an incubating block) to reach during the specified duration.
- **shaking_params** (*dict, optional*) – Specify "path" and "frequency" of shaking parameters to be used with compatible devices (eg. thermoshakes)
- **co2** (*Number, optional*) – Carbon dioxide percentage

class autoprotocol.instruction.**IlluminaSeq**(*flowcell*, *lanes*, *sequencer*, *mode*, *index*, *library_size*, *dataref*, *cycles*)

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.

Parameters

- **flowcell** (*str*) – Flowcell designation: "SR" or " "PE"
- **lanes** (*list(dict)*) –

```
"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.
- **cycles** (*Enum({"read_1", "read_2", "index_1", "index_2"})*) – Parameter specific to Illuminaseq read-length or number of sequenced bases. Refer to the ASC for more details

class autoprotocol.instruction.**SangerSeq**(*object*, *wells*, *dataref*, *type*, *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.

Parameters

- **object** (*Container or str*) – Container with well(s) that contain material to be sequenced.
- **wells** (*list(str)*) – Well indices of the container that contain appropriate materials to be sent for sequencing.
- **dataref** (*str*) – Name of sequencing dataset that will be returned.
- **type** (*Enum({"standard", "rca"})*) – Sanger sequencing type
- **primer** (*Container, optional*) – 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

class autoprotocol.instruction.**GelSeparate**(*objects*, *volume*, *matrix*, *ladder*, *duration*, *dataref*)

Separate nucleic acids on an agarose gel.

Parameters

- **objects** (*list or WellGroup or Well*) – List of wells or WellGroup containing wells to be separated on gel.
- **volume** (*str or 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 or Unit*) – Length of time to run current through gel.
- **dataref** (*str*) – Name of this set of gel separation results.

class autoprotocol.instruction.**GelPurify**(*objects*, *volume*, *matrix*, *ladder*, *dataref*, *extract*)

Separate nucleic acids on an agarose gel and purify.

Parameters

- **objects** (*list or WellGroup*) – WellGroup of wells to be purified
- **volume** (*str or Unit*) – Volume of sample required for analysis
- **dataref** (*str*) – Name of this specific dataset of measurements
- **matrix** (*str*) – Agarose concentration and number of wells on gel used for separation
- **ladder** (*str*) – Size range of ladder to be used to compare band size to
- **dataref** – Name of dataset containing fragment sizes returned
- **extract** (*list(dict)*) –

```
"extract": [{
      "elution_volume": volume,
      "elution_buffer": string, "water" | "TE",
      "lane": int,
      "band_size_range": {
        "min_bp": int,
        "max_bp": int,
      },
      "destination": well
    },
    {...}]
```

class autoprotocol.instruction.**Absorbance**(*object*, *wells*, *wavelength*, *dataref*, *flashes=25*, *incubate_before=None*, *temperature=None*, *settle_time=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.

Parameters

- **object** (*str or Ref*) – Object to execute the absorbance read on
- **wells** (*list(Well) or WellGroup*) – WellGroup of wells to be measured or a list of well references in the form of ["A1", "B1", "C5", ...]
- **wavelength** (*str or 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*) – number of flashes for the read
- **incubate_before** (*dict, optional*) – incubation prior to reading if desired

  shaking: dict, optional

  shake parameters if desired

  amplitude: str or Unit amplitude of shaking between 1 and 6:millimeter

  orbital: bool True for oribital and False for linear shaking

  duration: str, Unit, optional time prior to plate reading
- **temperature** (*str or Unit, optional*) – set temperature to heat plate reading chamber
- **settle_time** (*str or Unit, optional*) – time to pause before each well read

class autoprotocol.instruction.**Fluorescence**(*object*, *wells*, *excitation*, *emission*, *dataref*, *flashes=25*, *incubate_before=None*, *temperature=None*, *gain=None*, *detection_mode=None*, *position_z=None*, *settle_time=None*, *lag_time=None*, *integration_time=None*)

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

Parameters

- **object** (*str or Container*) – object to execute the fluorescence read on
- **wells** (*list(Well) or WellGroup*) – WellGroup of wells to be measured or a list of well references in the form of ["A1", "B1", "C5", ...]
- **excitation** (*str or Unit*) – wavelength of light used to excite the wells indicated
- **emission** (*str or 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 for this read
- **incubate_before** (*dict, optional*) – incubation prior to reading if desired

  shaking: dict, optional

  shake parameters if desired

amplitude: str or Unit amplitude of shaking between 1 and 6:millimeter

orbital: bool True for oribital and False for linear shaking

duration: str, Unit, optional time prior to plate reading

- **temperature** (*str or Unit, optional*) – set temperature to heat plate reading chamber
- **gain** (*float, optional*) – float between 0 and 1, multiplier of maximum signal amplification
- **detection_mode** (*str, optional*) – set the detection mode of the optics, ["top", "bottom"], defaults to vendor specified defaults.
- **position_z** (*dict, optional*) – distance from the optics to the surface of the plate transport, only valid for "top" detection_mode and vendor capabilities. Specified as either a set distance - "manual", OR calculated from a WellGroup - "calculated_from_wells". Only one position_z determination may be specified

```
position_z = {
    "manual": Unit
    - OR -
    "calculated_from_wells": []
}
```

- **manual** (*str, Unit, optional*) – parameter available within "position_z" to set the distance from the optics to the plate transport.
- **calculated_from_wells** (*list, WellGroup, Well, optional*) – parameter available within "position_z" to set the distance from the optics to the plate transport. If specified, the average optimal (maximal signal) distance will be chosen from the list of wells and applied to all measurements.
- **settle_time** (*Unit, optional*) – the time before the start of the measurement, defaults to vendor specifications
- **lag_time** (*Unit, optional*) – time between flashes and the start of the signal integration, defaults to vendor specifications
- **integration_time** (*Unit, optional*) – duration of the signal recording, per Well, defaults to vendor specifications

class autoprotocol.instruction.**Luminescence**(*object*, *wells*, *dataref*, *incubate_before=None*, *temperature=None*, *settle_time=None*, *integration_time=None*)

Read luminesence of indicated wells

Parameters

- **object** (*str or Container*) – object to execute the luminescence read on
- **wells** (*list or WellGroup*) – WellGroup or list of wells to be measured
- **dataref** (*str*) – name which dataset will be saved under
- **incubate_before** (*dict, optional*) – incubation prior to reading if desired

shaking: dict, optional

shake parameters if desired

amplitude: str or Unit amplitude of shaking between 1 and 6:millimeter

orbital: bool True for oribital and False for linear shaking

duration: str, Unit, optional time prior to plate reading

- **temperature** (*str or Unit, optional*) – set temperature to heat plate reading chamber
- **settle_time** (*str or Unit, optional*) – time to pause before each well read
- **integration_time** (*Unit, optional*) – duration of the signal recording, per Well, defaults to vendor specifications

class autoprotocol.instruction.**Seal**(*object*, *type='ultra-clear'*, *mode=None*, *mode_params=None*)

Seal indicated container using the automated plate sealer.

Parameters

- **object** (*Ref or str*) – Container to be sealed
- **type** (*str, optional*) – Seal type to be used (optional)
- **mode** (*str, optional*) – Method used to seal plate (optional). "thermal" or "adhesive"
- **mode_params** (*dict, optional*) – Thermal sealing parameters

  temperature [str, optional] Temperature to seal plate at

  duration [str, optional] Duration for which to apply heated sealing plate onto ref

class autoprotocol.instruction.**Unseal**(*object*)

Remove seal from indicated container using the automated plate unsealer.

Parameters **object** (*Ref or str*) – Container to be unsealed

class autoprotocol.instruction.**Cover**(*object*, *lid='standard'*, *retrieve_lid=None*)

Place specified lid type on specified container

Parameters

- **object** (*str*) – Container to be covered
- **lid** (*Enum({'standard', 'universal', 'low_evaporation'}), optional*) – Type of lid to cover container with
- **retrieve_lid** (*bool*) – Flag to retrieve lid from stored location

class autoprotocol.instruction.**Uncover**(*object*, *store_lid=None*)

Remove lid from specified container

Parameters

- **object** (*str*) – Container to remove lid from
- **store_lid** (*bool*) – Flag to store the uncovered lid

class autoprotocol.instruction.**FlowCytometry**(*dataref*, *samples*, *lasers*, *collection_conditions*, *width_threshold=None*, *window_extension=None*, *remove_coincident_events=None*)

This instruction provides a non-ambiguous set of parameters for the performance of flow cytometry.

Parameters

- **dataref** (*str*) – Name of dataset that will be returned.
- **samples** (*list(Well) or Well or WellGroup*) – Wells to be analyzed
- **lasers** (*list(dict)*) – See FlowCytometryBuilders.laser.

- **collection_conditions** (*dict*) – See FlowCytometryBuilders.collection_conditions.
- **width_threshold** (*int or float, optional*) – Threshold to determine width measurement.
- **window_extension** (*int or float, optional*) – Front and rear window extension.
- **remove_coincident_events** (*bool, optional*) – Remove coincident events. Defaults to false.

class autoprotocol.instruction.**FlowAnalyze**(*dataref*, *FSC*, *SSC*, *negative_controls*, *samples*, *colors=None*, *positive_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:

Autoprotocol Output:

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
}
```

- **negative_controls** (*list(dict)*) – 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(dict)*) – 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(dict), 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
}]
```

positive_controls [list(dict), 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]
    }]
}]
```

class autoprotocol.instruction.**Oligosynthesize**(*oligos*)

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
```

(continues on next page)

(continued from previous page)

```
        // - 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
    },
    ...
]
```

class autoprotocol.instruction.**Autopick**(*groups*, *criteria*, *dataref*)

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.

Parameters

- **groups** (*list(dict)*) – Groups of colonies to pick and where to transport them to
- **criteria** (*dict*) – Dictionary of autopicking criteria.
- **dataref** (*str*) – Name of dataset to save the picked colonies to

class autoprotocol.instruction.**ImagePlate**(*object*, *mode*, *dataref*)

Capture an image of the specified container.

Parameters

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

class autoprotocol.instruction.**Provision**(*resource_id*, *dests*)

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** (*list(dict)*) – Destination(s) for specified resource, together with volume information

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)

class autoprotocol.instruction.**FlashFreeze**(*object*, *duration*)

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

Parameters

- **object** (Container *or* *str*) – Container to be flash frozen.

- **duration** (*str or Unit*) – Duration to submerge specified container in liquid nitrogen.

class autoprotocol.instruction.**MeasureConcentration**(*object*, *volume*, *dataref*, *measurement*)

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

Parameters

- **object** (*list or WellGroup*) – WellGroup of wells to be measured
- **volume** (*str or 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"].

class autoprotocol.instruction.**MeasureMass**(*object*, *dataref*)

Measure the mass of containers

Parameters

- **object** (Container) – Container ref
- **dataref** (*str*) – Name of the data for the measurement

class autoprotocol.instruction.**MeasureVolume**(*object*, *dataref*)

Measure the mass of containers

Parameters

- **object** (*list(Container)*) – list of containers
- **dataref** (*str*) – Name of the data for the measurement

class autoprotocol.instruction.**CountCells**(*wells*, *volume*, *dataref*, *labels=None*)

Count the number of cells in a sample that are positive/negative for a given set of labels.

Parameters

- **wells** (WellGroup) – List of wells that will be used for cell counting.
- **volume** (Unit) – Volume that should be consumed from each well for the purpose of cell counting.
- **dataref** (*str*) – Name of dataset that will be returned.
- **labels** (*list(string), optional*) – Cells will be scored for presence or absence of each label in this list. If staining is required to visualize these labels, they must be added before execution of this instruction.

class autoprotocol.instruction.**Spectrophotometry**(*dataref*, *object*, *groups*, *interval=None*, *num_intervals=None*, *temperature=None*, *shake_before=None*)

Execute a Spectrophotometry plate read on the obj.

Parameters

- **dataref** (*str*) – Name of the resultant dataset to be returned.
- **object** (*Container or str*) – Container to be read.
- **groups** (*list*) – A list of groups generated by SpectrophotometryBuilders groups builders, any of absorbance_mode_params, fluorescence_mode_params, luminescence_mode_params, or shake_mode_params.

- **interval** (*Unit or str, optional*) – The time between each of the read intervals.
- **num_intervals** (*int, optional*) – The number of times that the groups should be executed.
- **temperature** (*Unit or str, optional*) – The temperature that the entire instruction should be executed at.
- **shake_before** (*dict, optional*) – A dict of params generated by SpectrophotometryBuilders.shake_before that dictates how the obj should be incubated with shaking before any of the groups are executed.

class autoprotocol.instruction.**LiquidHandle**(*locations*, *shape=None*, *mode=None*, *mode_params=None*)

Manipulates liquids within locations

A liquid handle instruction is constructed as a list of locations, where each location consists of the well location and the tip transports carried out within the well.

Each liquid handle instruction corresponds to a single tip or set of tips.

Parameters

- **locations** (*list(dict)*) – See Also `LiquidHandle.builders.location()`
- **shape** (*dict, optional*) – See Also `LiquidHandle.builders.shape()`
- **mode** (*str, optional*) – the liquid handling mode
- **mode_params** (*dict, optional*) – See Also `LiquidHandle.builders.instruction_mode_params()`

CHAPTER 2

autoprotocol.container

2.1 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

- **id** (*str, optional*) – Alphanumerical identifier for a Container.
- **container_type** (ContainerType) – ContainerType associated with a Container.
- **name** (*str, optional*) – name of the container/ref being created.
- **storage** (*str, optional*) – name of the storage condition.
- **cover** (*str, optional*) – name of the cover on the container.

Raises AttributeError – Invalid cover-type given

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").

Returns Well for given reference

Return type *Well*

Raises TypeError – index given is not of the right type

well_from_coordinates(*row*, *column*)
Gets the well at 0-indexed position (row, column) within the container. The origin is in the top left corner.

Parameters

- **row** (*int*) – The 0-indexed row index of the well to be fetched
- **column** (*int*) – The 0-indexed column index of the well to be fetched

Returns The well at position (row, column)

Return type *Well*

tube()
Checks if container is tube and returns a Well representing the zeroth well.

Returns Zeroth well of tube

Return type *Well*

Raises AttributeError – If container is not tube

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.

Returns Wells from specified references

Return type *WellGroup*

Raises TypeError – Well reference is not of a valid input type

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.

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.

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.

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

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

Returns WellGroup of all Wells in Container

Return type *WellGroup*

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).

Returns WellGroup of inner wells

Return type *WellGroup*

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 or int or 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.

Returns WellGroup of selected wells

Return type *WellGroup*

Raises TypeError – Incorrect input types, e.g. *num* has to be of type int

is_sealed()
Check if Container is sealed.

is_covered()
Check if Container is covered.

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

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

Returns WellGroup of wells for the specified quadrant

Return type *WellGroup*

Raises ValueError – Invalid quadrant specified for this Container type

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.

Returns Container with modified storage condition

Return type *Container*

Raises TypeError – If storage condition not of type str.

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
      }
    }
```

wells_from_shape(*origin*, *shape*)

Gets a WellGroup that originates from the *origin* and is distributed across the container in *shape*. This group has a Well for each index in *range(shape["rows"] * shape["columns"])*.

In cases where the container dimensions are smaller than the shape format's dimensions the returned WellGroup will reference some wells multiple times. This is analogous to an SBS96-formatted liquid handler acting with multiple tips in each well of an SBS24-formatted plate.

Parameters

- **origin** (*int or str*) – The index of the top left corner origin of the shape
- **shape** (*dict*) – See Also Instruction.builders.shape

Returns The group of wells distributed in *shape* from the *origin*

Return type *WellGroup*

Raises ValueError – if the shape exceeds the extents of the container

__repr__()

Return a string representation of a Container using the specified name. (ex. Container('my_plate'))

2.2 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** (*int*) – The index of this well within the container.

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.

Returns Well with modified properties

Return type *Well*

add_properties(*properties*)
Add properties to the properties attribute of a Well.

If any property with the same key already exists for the Well then:

- if both old and new properties are lists then append the new property
- otherwise overwrite the old property with the new one

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

Returns Well with modified properties

Return type *Well*

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

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

Returns Well with modified volume

Return type *Well*

Raises

- TypeError – Incorrect input-type given
- ValueError – Volume set exceeds maximum well volume

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.

Returns Well with modified name

Return type *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.

Returns Index of well in Container (in human readable form)

Return type str

available_volume()
Returns the available volume of a Well. This is calculated as nominal volume - container_type dead volume

Returns Volume in well

Return type *Unit*(volume)

Raises RuntimeError – Well has no volume

__repr__()
Return a string representation of a Well.

2.3 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.

Raises TypeError – Wells is not of the right input type

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

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

Returns WellGroup with modified properties

Return type *WellGroup*

add_properties(*properties*)
Add the same properties for each Well in a WellGroup.

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

Returns WellGroup with modified properties

Return type *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.

Returns WellGroup with modified volume

Return type *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.

Returns List of humanized indices from this WellGroup

Return type list(str)

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

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

Returns WellGroup with appended well

Return type *WellGroup*

Raises TypeError – other is not of type Well

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

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

Returns WellGroup extended with specified WellGroup

Return type *WellGroup*

Raises TypeError – Input WellGroup is not of the right type

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

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

Returns Name of wellgroup

Return type str

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, optional*) – the value assigned to the property

Returns WellGroup with modified properties

Return type *WellGroup*

Raises TypeError – property or value defined does not have right input type

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, optional*) – the index of the well you want to remove and return

Returns Well with selected index from WellGroup

Return type *Well*

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

Returns WellGroup with inserted wells

Return type *WellGroup*

Raises TypeError – index or well defined does not have right input type

__setitem__(*key*, *item*)
Set a specific Well in a WellGroup.

Parameters

- **key** (*int*) – Position in a WellGroup in robotized form.
- **item** (Well) – Well or WellGroup to be added

Raises TypeError – Item specified is not of type *Well*

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

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

Returns Specified well from given key

Return type *Well*

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

__repr__()
Return a string representation of a WellGroup.

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

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

Returns WellGroup with appended wells

Return type *WellGroup*

Raises TypeError – Input given is not of type Well or WellGroup

CHAPTER 3

autoprotocol.container_type

3.1 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** (*float*) – Depth of well(s) contained in a ContainerType in millimeters.
- **well_volume_ul** (Unit) – 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** (Unit) – Volume of liquid that cannot be aspirated from any given well of a ContainerType via liquid-handling.

- **safe_min_volume_ul** (Unit) – Minimum volume of liquid to ensure adequate volume for liquid-handling aspiration from any given well of a ContainerType.
- **true_max_vol_ul** (Unit, *optional*) – 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, optional*) – ContainerType commercial vendor, if available.
- **cat_no** (*str, optional*) – 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"

well_from_coordinates(*row*, *column*)

Gets the well at 0-indexed position (row, column) within the container. The origin is in the top left corner.

Parameters

- **row** (*int*) – The 0-indexed row index of the well to be fetched
- **column** (*int*) – The 0-indexed column index of the well to be fetched

Returns The robotized index of the well at at position (row, column)

Return type Int

Raises

- ValueError – if the specified row is outside the bounds of the container_type
- ValueError – if the specified column is outside the bounds of the container_type

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 Single or list of Well references passed as row-wise integer (left-to-right, top-to-bottom, starting at 0 = A1).

Return type int or list

Raises

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

- `ValueError` – If well reference given is in an invalid format.

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.

decompose(*idx*)

Return the (col, row) corresponding to the given well index.

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

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

Return type tuple

Raises `TypeError` – Index given is not of the right parameter type

row_count()

Return the number of rows of this ContainerType.

3.2 Container Types

autoprotocol.container_type.**FLAT384 = ContainerType(name='384-well UV flat-bottom plate', i**

autoprotocol.container_type.**PCR384 = ContainerType(name='384-well PCR plate', is_tube=False**

autoprotocol.container_type.**ECHO384 = ContainerType(name='384-well Echo plate', is_tube=Fal**

autoprotocol.container_type.**FLAT384WHITELV = ContainerType(name='384-well flat-bottom low v**

autoprotocol.container_type.**FLAT384WHITETC = ContainerType(name='384-well flat-bottom low i**

autoprotocol.container_type.**FLAT384CLEAR = ContainerType(name='384-well fully clear high bi**

autoprotocol.container_type.**FLAT96 = ContainerType(name='96-well flat-bottom plate', is_tub**

autoprotocol.container_type.**FLAT96UV = ContainerType(name='96-well flat-bottom UV transpare**

autoprotocol.container_type.**PCR96 = ContainerType(name='96-well PCR plate', is_tube=False,**
autoprotocol.container_type.**DEEP96 = ContainerType(name='96-well extended capacity plate',**
autoprotocol.container_type.**V96KF = ContainerType(name='96-well v-bottom King Fisher plate**
autoprotocol.container_type.**DEEP96KF = ContainerType(name='96-well extended capacity King**
autoprotocol.container_type.**DEEP24 = ContainerType(name='24-well extended capacity plate',**
autoprotocol.container_type.**MICRO2 = ContainerType(name='2mL Microcentrifuge tube', is_tube**
autoprotocol.container_type.**MICRO15 = ContainerType(name='1.5mL Microcentrifuge tube', is_t**
autoprotocol.container_type.**FLAT6 = ContainerType(name='6-well cell culture plate', is_tube**
autoprotocol.container_type.**FLAT1 = ContainerType(name='1-well flat-bottom plate', is_tube=**
autoprotocol.container_type.**FLAT6TC = ContainerType(name='6-well TC treated plate', is_tube**
autoprotocol.container_type.**RESSW96HP = ContainerType(name='96-well singlewell highprofile**
autoprotocol.container_type.**RESMW8HP = ContainerType(name='8-row multiwell highprofile rese**
autoprotocol.container_type.**RESMW12HP = ContainerType(name='12-column multiwell highprofile**
autoprotocol.container_type.**FLAT96CLEARTC = ContainerType(name='96-well flat-bottom TC trea**
autoprotocol.container_type.**V384CLEAR = ContainerType(name='384-well v-bottom polypropylene**
autoprotocol.container_type.**ROUND384CLEAR = ContainerType(name='384-well round-bottom plate**
autoprotocol.container_type.**RESSW384LP = ContainerType(name='384-well singlewell lowprofile**
autoprotocol.container_type.**ECHO384LDV = ContainerType(name='384-well Echo low dead volume**
autoprotocol.container_type.**ECHO384LDVPLUS = ContainerType(name='384-well Echo low dead vol**

CHAPTER 4

autoprotocol.builders

Builders Module containing builders, which help build inputs for Instruction parameters

copyright 2018 by The Autoprotocol Development Team, see AUTHORS for more details.

license BSD, see LICENSE for more details

4.1 Summary

These builder methods are used to generate and validate complex data structures used in Autoprotocol specification. Each of them is capable of using their own output as input. Therefore these builders are also used as inline checks in Protocol methods.

Notes

Generally these builders should not be called from this file directly. They're more easily accessible by referencing a specific Instruction's builders attribute (e.g. *Spectrophotometry.Builders.mode_params*).

See also:

Instruction Instructions corresponding to each of the builders

class autoprotocol.builders.**InstructionBuilders**
General builders that apply to multiple instructions

shape(*rows=1*, *columns=1*, *format=None*)
Helper function for building a shape dictionary

Parameters

- **rows** (*int, optional*) – Number of rows to be concurrently transferred
- **columns** (*int, optional*) – Number of columns to be concurrently transferred

- **format** (*str, optional*) – Plate format in String form. e.g. "SBS96" or "SBS384"

Returns shape parameters

Return type dict

Raises

- TypeError – If rows/columns aren't ints
- ValueError – If an invalid row/column count is given
- ValueError – If an invalid shape is given
- ValueError – If rows/columns are greater than what is allowed for the format

class autoprotocol.builders.**ThermocycleBuilders**

These builders are meant for helping to construct the *groups* argument in the *Protocol.thermocycle* method

dyes(***kwargs*)

Helper function for creating a dye parameter

Parameters ****kwargs** (*dict(str: int or list(int))*) – A mapping from a dye (str) to the index of a well

Returns A thermocycling dye to well mapping

Return type dict

Raises

- ValueError – If any of the specified dyes are not valid
- ValueError – If wells is not an int, str, list(int), or list(str)

dyes_from_well_map(*well_map*)

Helper function for creating a dye parameter from a well_map

Take a map of wells to the dyes it contains and returns a map of dyes to the list of wells that contain it.

Parameters **well_map** (*dict(well, str)*) – A thermocycling well to dye mapping

Returns A thermocycling dye to well mapping

Return type dict

See also:

Thermocycle.Builders.dyes() standard constructor for the dyes parameter

static melting(*start=None*, *end=None*, *increment=None*, *rate=None*)

Helper function for creating melting parameters

Generates melt curve parameters for Thermocycle Instructions.

Parameters

- **start** (*str or Unit*) – The starting temperature for the melt curve
- **end** (*str or Unit*) – The ending temperature for the melt curve
- **increment** (*str or Unit*) – The temperature increment of the melt curve
- **rate** (*str or Unit*) – The duration the individual increments

Returns A thermocycling melt curve specification

Return type dict

Raises `ValueError` – If some, but not all melt curve parameters are specified

group(*steps*, *cycles=1*)

Helper function for creating a thermocycle group, which is a series of steps repeated for the number of cycles

Parameters

- **steps** (*list(ThermocycleBuilders.step)*) – Steps to be carried out. At least one step has to be specified. See *ThermocycleBuilders.step* for more information
- **cycles** (*int, optional*) – Number of cycles to repeat the specified steps. Defaults to 1

Returns A thermocycling group

Return type dict

Raises

- `TypeError` – Invalid input types, i.e. *cycles* is not of type int and *steps* is not of type list
- `ValueError` – *cycles* is not positive
- `ValueError` – *steps* does not contain any elements

static step(*temperature*, *duration*, *read=None*)

Helper function for creating a thermocycle step.

Parameters

- **temperature** (*Unit or dict(str, Unit)*) – Block temperature which the contents should be thermocycled at.

  If a gradient thermocycle is desired, specifying a dict with "top" and "bottom" keys will control the desired temperature at the top and bottom rows of the block, creating a gradient along the column.

  ..code-block:: python

    temperature = {"top": "50:celsius", "bottom": "45:celsius"}

- **duration** (*str or Unit*) – Duration where the specified temperature parameters will be applied
- **read** (*Boolean, optional*) – Determines if a read at wavelengths specified by the dyes in the parent *thermocycle* instruction will be enabled for this particular step. Useful for qPCR applications.

Returns A thermocycling step

Return type dict

Raises

- `TypeError` – Invalid input types, e.g. *read* is not of type bool
- `ValueError` – Invalid format specified for *temperature* dict
- `ValueError` – Duration is not greater than 0 second

shape(*rows=1*, *columns=1*, *format=None*)

Helper function for building a shape dictionary

Parameters

- **rows** (*int, optional*) – Number of rows to be concurrently transferred
- **columns** (*int, optional*) – Number of columns to be concurrently transferred
- **format** (*str, optional*) – Plate format in String form. e.g. "SBS96" or "SBS384"

Returns shape parameters

Return type dict

Raises

- `TypeError` – If rows/columns aren't ints
- `ValueError` – If an invalid row/column count is given
- `ValueError` – If an invalid shape is given
- `ValueError` – If rows/columns are greater than what is allowed for the format

class autoprotocol.builders.**DispenseBuilders**

These builders are meant for helping to construct arguments in the *Protocol.dispense* method.

static nozzle_position(*position_x=None*, *position_y=None*, *position_z=None*)

Generates a validated nozzle_position parameter.

Parameters

- **position_x** (Unit, *optional*) –
- **position_y** (Unit, *optional*) –
- **position_z** (Unit, *optional*) –

Returns Dictionary of nozzle position parameters

Return type dict

static column(*column*, *volume*)

Generates a validated column parameter.

Parameters

- **column** (*int*) –
- **volume** (*str*, Unit) –

Returns Column parameter of type {"column": int, "volume": Unit}

Return type dict

columns(*columns*)

Generates a validated columns parameter.

Parameters **columns** (*list({"column": int, "volume": str, Unit})*) –

Returns List of columns of type ({"column": int, "volume": str, Unit})

Return type list

Raises

- `ValueError` – No *column* specified for columns
- `ValueError` – Non-unique column indices

shake_after(*duration*, *frequency=None*, *path=None*, *amplitude=None*)
Generates a validated shake_after parameter.

Parameters

- **duration** (Unit, *str*) –
- **frequency** (Unit, *str*, *optional*) –
- **path** (*str*, *optional*) –
- **amplitude** (Unit, *str*, *optional*) –

Returns Shake after dictionary of type {"duration": Unit, "frequency": Unit, "path": str, "amplitude": Unit}

Return type dict

Raises ValueError – Invalid shake path specified

shape(*rows=1*, *columns=1*, *format=None*)
Helper function for building a shape dictionary

Parameters

- **rows** (*int*, *optional*) – Number of rows to be concurrently transferred
- **columns** (*int*, *optional*) – Number of columns to be concurrently transferred
- **format** (*str*, *optional*) – Plate format in String form. e.g. "SBS96" or "SBS384"

Returns shape parameters

Return type dict

Raises

- TypeError – If rows/columns aren't ints
- ValueError – If an invalid row/column count is given
- ValueError – If an invalid shape is given
- ValueError – If rows/columns are greater than what is allowed for the format

class autoprotocol.builders.**SpectrophotometryBuilders**
These builders are meant for helping to construct arguments for the *Spectrophotometry* instruction.

static wavelength_selection(*shortpass=None*, *longpass=None*, *ideal=None*)
Generates a representation of a wavelength selection by either filters (using shortpass/longpass) or monochromators (using ideal)

Parameters

- **shortpass** (Unit, *str*, *optional*) –
- **longpass** (Unit, *str*, *optional*) –
- **ideal** (Unit, *str*, *optional*) –

Returns Wavelength selection parameters.

Return type dict

groups(*groups*)

Parameters **groups** (*list(dict)*) – A list of spectrophotometry groups.

Returns A list of spectrophotometry groups.

Return type list(dict)

group(*mode*, *mode_params*)

Parameters

- **mode** (*str*) – A string representation of a valid spectrophotometry mode.
- **mode_params** (*dict*) – A dict of mode_params corresponding to the mode.

Returns A spectrophotometry group.

Return type dict

Raises ValueError – Invalid mode specified

absorbance_mode_params(*wells*, *wavelength*, *num_flashes=None*, *settle_time=None*, *read_position=None*, *position_z=None*)

Parameters

- **wells** (*iterable(Well) or WellGroup*) – Wells to be read.
- **wavelength** (*Unit or str*) – The wavelengths at which to make absorbance measurements.
- **num_flashes** (*int, optional*) – The number of discrete reads to be taken and then averaged.
- **settle_time** (*Unit or str, optional*) – The time to wait between moving to a well and reading it.
- **read_position** (*Enum("top", "bottom"), optional*) – The position of the probe relative to the plate for the read
- **position_z** (*dict, optional*) – This should be specified with either *position_z_manual* or *position_z_calculated*

Returns Formatted mode_params for an absorbance mode.

Return type dict

Raises

- TypeError – Invalid type specified for input parameters, e.g. *num_flashes* not of type int
- ValueError – Invalid wells specified

fluorescence_mode_params(*wells*, *excitation*, *emission*, *num_flashes=None*, *settle_time=None*, *lag_time=None*, *integration_time=None*, *gain=None*, *read_position=None*, *position_z=None*)

Parameters

- **wells** (*iterable(Well) or WellGroup*) – Wells to be read.
- **excitation** (*list(dict)*) – A list of SpectrophotometryBuilders.wavelength_selection to determine the wavelegnth(s) of excitation light used.
- **emission** (*list(dict)*) – A list of SpectrophotometryBuilders.wavelength_selection to determine the wavelegnth(s) of emission light used.

- **num_flashes** (*int, optional*) – The number of discrete reads to be taken and then combined.
- **settle_time** (*Unit or str, optional*) – The time to wait between moving to a well and reading it.
- **lag_time** (*Unit or str, optional*) – The time to wait between excitation and reading.
- **integration_time** (*Unit or str, optional*) – Time over which the data should be collected and integrated.
- **gain** (*int, optional*) – The amount of gain to be applied to the readings.
- **read_position** (*str, optional*) – The position from which the wells should be read.
- **position_z** (*dict, optional*) – This should be specified with either *position_z_manual* or *position_z_calculated*

Returns Formatted mode_params for a fluorescence mode.

Return type dict

Raises

- TypeError – Invalid input types, e.g. settle_time is not of type Unit(second)
- ValueError – Invalid wells specified
- ValueError – Gain is not between 0 and 1

luminescence_mode_params(*wells*, *num_flashes=None*, *settle_time=None*, *integration_time=None*, *gain=None*, *read_position=None*, *position_z=None*)

Parameters

- **wells** (*iterable(Well) or WellGroup*) – Wells to be read.
- **num_flashes** (*int, optional*) – The number of discrete reads to be taken and then combined.
- **settle_time** (*Unit or str, optional*) – The time to wait between moving to a well and reading it.
- **integration_time** (*Unit or str, optional*) – Time over which the data should be collected and integrated.
- **gain** (*int, optional*) – The amount of gain to be applied to the readings.
- **read_position** (*Enum("top", "bottom"), optional*) – The position of the probe relative to the plate for the read
- **position_z** (*dict, optional*) – This should be specified with either *position_z_manual* or *position_z_calculated*

Returns Formatted mode_params for a luminescence mode.

Return type dict

Raises

- TypeError – Invalid input types, e.g. settle_time is not of type Unit(second)
- ValueError – Gain is not between 0 and 1

shake_mode_params(*duration=None*, *frequency=None*, *path=None*, *amplitude=None*)

Parameters

- **duration** (*Unit or str, optional*) – The duration of the shaking incubation, if not specified then the incubate will last until the end of read interval.
- **frequency** (*Unit or str, optional*) – The frequency of the shaking motion.
- **path** (*str, optional*) – The name of a shake path. See the spectrophotometry ASC for diagrams of different shake paths.
- **amplitude** (*Unit or str, optional*) – The amplitude of the shaking motion.

Returns Formatted mode_params for a shake mode.

Return type dict

shake_before(*duration*, *frequency=None*, *path=None*, *amplitude=None*)

Parameters

- **duration** (*Unit or str*) – The duration of the shaking incubation.
- **frequency** (*Unit or str, optional*) – The frequency of the shaking motion.
- **path** (*str, optional*) – The name of a shake path. See the spectrophotometry ASC for diagrams of different shake paths.
- **amplitude** (*Unit or str, optional*) – The amplitude of the shaking motion.

Returns Formatted mode_params for a shake mode.

Return type dict

position_z_manual(*reference=None*, *displacement=None*)

Helper for building position_z parameters for a manual position_z configuration

Parameters

- **reference** (*str, optional*) – Must be one of "plate_top", "plate_bottom", "well_top", "well_bottom"
- **displacement** (*Unit or str, optional*) – Displacement from reference position. Negative would refer to the *well_top* to *well_bottom* direction, while positive would refer to the opposite direction.

Returns position_z parameters for a Spectrophotometry instruction

Return type dict

Raises

- `ValueError` – If reference was not in the allowed list
- `ValueError` – If invalid displacement was provided

position_z_calculated(*wells*, *heuristic=None*)

Helper for building position_z parameters for a calculated position_z configuration

Parameters

- **wells** (*list(Well)*) – List of wells to calculate the z-position from

- **heuristic** (*str, optional*) – Must be one of "max_mean_read_without_saturation" or "closest_distance_without_saturation". Please refer to ASC-041 for the full explanation

Returns position_z parameters for a Spectrophotometry instruction

Return type dict

Raises

- ValueError – If a list of wells is not provided
- ValueError – If an invalid heuristic is specified

shape(*rows=1*, *columns=1*, *format=None*)

Helper function for building a shape dictionary

Parameters

- **rows** (*int, optional*) – Number of rows to be concurrently transferred
- **columns** (*int, optional*) – Number of columns to be concurrently transferred
- **format** (*str, optional*) – Plate format in String form. e.g. "SBS96" or "SBS384"

Returns shape parameters

Return type dict

Raises

- TypeError – If rows/columns aren't ints
- ValueError – If an invalid row/column count is given
- ValueError – If an invalid shape is given
- ValueError – If rows/columns are greater than what is allowed for the format

class autoprotocol.builders.**LiquidHandleBuilders**

Builders for LiquidHandle Instructions

location(*location=None*, *transports=None*)

Helper for building locations

Parameters

- **location** (Well *or str, optional*) – Location refers to the well location where the transports will be carried out
- **transports** (*list(dict), optional*) – Transports refer to the list of transports that will be carried out in the specified location See Also LiquidHandle.builders.transport

Returns location parameters for a LiquidHandle instruction

Return type dict

Raises

- TypeError – If locations aren't str/well
- ValueError – If transports are specified, but empty

transport(*volume=None*, *pump_override_volume=None*, *flowrate=None*, *delay_time=None*, *mode_params=None*)

Helper for building transports

Parameters

- **volume** (*Unit or str, optional*) – Volume to be aspirated/dispensed. Positive volume -> Dispense. Negative -> Aspirate
- **pump_override_volume** (*Unit or str, optional*) – Calibrated volume, volume which the pump will move
- **flowrate** (*dict, optional*) – Flowrate parameters See Also LiquidHandle.builders.flowrate
- **delay_time** (*Unit or str, optional*) – Time spent waiting after executing mandrel and pump movement
- **mode_params** (*dict, optional*) – Mode parameters See Also LiquidHandle.builders.mode_params

Returns transport parameters for a LiquidHandle instruction

Return type dict

static flowrate(*target*, *initial=None*, *cutoff=None*, *acceleration=None*, *deceleration=None*)

Helper for building flowrates

Parameters

- **target** (*Unit or str*) – Target flowrate
- **initial** (*Unit or str, optional*) – Initial flowrate
- **cutoff** (*Unit or str, optional*) – Cutoff flowrate
- **acceleration** (*Unit or str, optional*) – Volumetric acceleration for initial to target (in ul/s^2)
- **deceleration** (*Unit or str, optional*) – Volumetric deceleration for target to cutoff (in ul/s^2)

Returns flowrate parameters for a LiquidHandle instruction

Return type dict

mode_params(*liquid_class=None*, *position_x=None*, *position_y=None*, *position_z=None*, *tip_position=None*)

Helper for building transport mode_params

Mode params contain information about tip positioning and the liquid being manipulated

Parameters

- **liquid_class** (*Enum({"default", "air"}), optional*) – The name of the liquid class to be handled. This affects how vendors handle populating liquid handling defaults.
- **position_x** (*dict, optional*) – Target relative x-position of tip in well. See Also LiquidHandle.builders.position_xy
- **position_y** (*dict, optional*) – Target relative y-position of tip in well. See Also LiquidHandle.builders.position_xy
- **position_z** (*dict, optional*) – Target relative z-position of tip in well. See Also LiquidHandle.builders.position_z
- **tip_position** (*dict, optional*) – A dict of positions x, y, and z. Should only be specified if none of the other tip position parameters have been specified.

Returns mode_params for a LiquidHandle instruction

Return type dict

Raises

- `ValueError` – If liquid_class is not in the allowed list
- `ValueError` – If both position_x|y|z and tip_position are specified

static move_rate(*target=None*, *acceleration=None*)
Helper for building move_rates

Parameters

- **target** (*Unit or str, optional*) – Target velocity. Must be in units of
- **acceleration** (*Unit or str, optional*) – Acceleration. Must be in units of

Returns move_rate parameters for a LiquidHandle instruction

Return type dict

position_xy(*position=None*, *move_rate=None*)
Helper for building position_x and position_y parameters

Parameters

- **position** (*Numeric, optional*) – Target relative x/y-position of tip in well in unit square coordinates.
- **move_rate** (*dict, optional*) – The rate at which the tip moves in the well See Also LiquidHandle.builders.move_rate

Returns position_xy parameters for a LiquidHandle instruction

Return type dict

Raises

- `TypeError` – If position is non-numeric
- `ValueError` – If position is not in range

position_z(*reference=None*, *offset=None*, *move_rate=None*, *detection_method=None*, *detection_threshold=None*, *detection_duration=None*, *detection_fallback=None*, *detection=None*)
Helper for building position_z parameters

Parameters

- **reference** (*str, optional*) –

  Must be one of "well_top", "well_bottom", "liquid_surface",
  "preceding_position"

- **offset** (*Unit or str, optional*) – Offset from reference position
- **move_rate** (*dict, optional*) – Controls the rate at which the tip moves in the well See Also LiquidHandle.builders.move_rate
- **detection_method** (*str, optional*) – Must be one of "tracked", "pressure", "capacitance"
- **detection_threshold** (*Unit or str, optional*) – The threshold which must be crossed before a positive reading is registered. This is applicable for capacitance and pressure detection methods

- **detection_duration** (*Unit or str, optional*) – The contiguous duration where the threshold must be crossed before a positive reading is registered. This is applicable for pressure detection methods
- **detection_fallback** (*dict, optional*) – Fallback option which will be used if sensing fails See Also LiquidHandle.builders.position_z
- **detection** (*dict, optional*) – A dict of detection parameters. Should only be specified if none of the other detection parameters have been specified.

Returns position_z parameters for a LiquidHandle instruction

Return type dict

Raises

- `ValueError` – If reference was not in the allowed list
- `ValueError` – If both detection_method|duration|threshold|fallback and detection are specified
- `ValueError` – If detection_method is not in the allowed list
- `ValueError` – If detection parameters were specified, but the reference position doesn't support detection

static instruction_mode_params(*tip_type=None*)

Helper for building instruction mode_params

Parameters **tip_type** (*str, optional*) – the string representation ofa tip_type See Also tip_type.py

Returns mode_params for a LiquidHandle instruction

Return type dict

mix(*volume*, *repetitions*, *initial_z*, *asp_flowrate=None*, *dsp_flowrate=None*)

Helper for building mix params for Transfer LiquidHandleMethods

Parameters

- **volume** (*Unit or str*) – the volume of the mix step
- **repetitions** (*int*) – the number of times that the mix should be repeated
- **initial_z** (*dict*) – the position that the tip should move to prior to mixing See Also LiquidHandle.builders.position_z
- **asp_flowrate** (*dict, optional*) – the flowrate of the aspiration portions of the mix See Also LiquidHandle.builders.flowrate
- **dsp_flowrate** (*dict, optional*) – the flowrate of the dispense portions of the mix See Also LiquidHandle.builders.flowrate

Returns mix parameters for a LiquidHandleMethod

Return type dict

Raises `TypeError` – If repetitions is not an int

blowout(*volume*, *initial_z*, *flowrate=None*)

Helper for building blowout params for LiquidHandleMethods

Parameters

- **volume** (*Unit or str*) – the volume of the blowout step

- **initial_z** (*dict*) – the position that the tip should move to prior to blowing out See Also LiquidHandle.builders.position_z
- **flowrate** (*dict, optional*) – the flowrate of the blowout See Also LiquidHandle.builders.flowrate

Returns blowout params for a LiquidHandleMethod

Return type dict

shape(*rows=1*, *columns=1*, *format=None*)
Helper function for building a shape dictionary

Parameters

- **rows** (*int, optional*) – Number of rows to be concurrently transferred
- **columns** (*int, optional*) – Number of columns to be concurrently transferred
- **format** (*str, optional*) – Plate format in String form. e.g. "SBS96" or "SBS384"

Returns shape parameters

Return type dict

Raises

- `TypeError` – If rows/columns aren't ints
- `ValueError` – If an invalid row/column count is given
- `ValueError` – If an invalid shape is given
- `ValueError` – If rows/columns are greater than what is allowed for the format

class autoprotocol.builders.**PlateReaderBuilders**
Helpers for building parameters for plate reading instructions

incubate_params(*duration*, *shake_amplitude=None*, *shake_orbital=None*, *shaking=None*)
Create a dictionary with incubation parameters which can be used as input for instructions. Currently supports plate reader instructions and could be extended for use with other instructions.

Parameters

- **duration** (*str or Unit*) – the duration to shake the plate for
- **shake_amplitude** (*str or Unit, optional*) – amplitude of shaking between 1 and 6:millimeter
- **shake_orbital** (*bool, optional*) – True for orbital and False for linear shaking
- **shaking** (*dict, optional*) – A dict of amplitude and orbital: should only be specified if none of the other tip shake parameters have been specified. Dictionary of incubate parameters

Returns plate reader incubate_params

Return type dict

Raises

- `ValueError` – if shake *duration* is not positive
- `ValueError` – if only one of shake_amplitude or shake_orbital is set
- `TypeError` – if *shake_orbital* is not a bool

- ValueError – if *shake_amplitude* is not positive

shape(*rows=1*, *columns=1*, *format=None*)
Helper function for building a shape dictionary

Parameters

- **rows** (*int, optional*) – Number of rows to be concurrently transferred
- **columns** (*int, optional*) – Number of columns to be concurrently transferred
- **format** (*str, optional*) – Plate format in String form. e.g. "SBS96" or "SBS384"

Returns shape parameters

Return type dict

Raises

- TypeError – If rows/columns aren't ints
- ValueError – If an invalid row/column count is given
- ValueError – If an invalid shape is given
- ValueError – If rows/columns are greater than what is allowed for the format

class autoprotocol.builders.**GelPurifyBuilders**
Helpers for building GelPurify instructions

extract(*source*, *band_list*, *lane=None*, *gel=None*)
Helper for building extract params for gel_purify

Parameters

- **source** (Well) – The Well that contains the sample be purified
- **band_list** (*list(dict)*) – A list of bands to be extracted from the source sample
- **lane** (*int, optional*) – The gel lane for the source sample to be run on
- **gel** (*int, optional*) – The number of the gel if using multiple gels

Returns gel_purify extract parameters

Return type dict

Raises TypeError – If source is not a Well

band(*elution_buffer*, *elution_volume*, *destination*, *min_bp=None*, *max_bp=None*, *band_size_range=None*)
Helper for building band params for gel_purify

Parameters

- **elution_buffer** (*str*) – The type of elution buffer to be used
- **elution_volume** (*str or Unit*) – The volume of sample to be eluted
- **destination** (Well) – The Well the extracted samples should be eluted into
- **min_bp** (*int, optional*) – The minimum size sample to be removed.
- **max_bp** (*int, optional*) – The maximum size sample to be removed.
- **band_size_range** (*dict, optional*) – A dict of band size parameters. Should only be specified if none of the other band size parameters have been specified.

Returns gel_purify band parameters

Return type dict

Raises

- `TypeError` – If destination is not a Well
- `TypeError` – If *min_bp* is not an int
- `TypeError` – If *max_bp* is not an int
- `ValueError` – If *min_bp* is not less than *max_bp*

shape(*rows=1*, *columns=1*, *format=None*)

Helper function for building a shape dictionary

Parameters

- **rows** (*int, optional*) – Number of rows to be concurrently transferred
- **columns** (*int, optional*) – Number of columns to be concurrently transferred
- **format** (*str, optional*) – Plate format in String form. e.g. "SBS96" or "SBS384"

Returns shape parameters

Return type dict

Raises

- `TypeError` – If rows/columns aren't ints
- `ValueError` – If an invalid row/column count is given
- `ValueError` – If an invalid shape is given
- `ValueError` – If rows/columns are greater than what is allowed for the format

class autoprotocol.builders.**MagneticTransferBuilders**

Helpers for building MagneticTransfer instruction parameters

static mag_dry(*object*, *duration*)

Helper for building mag_dry sub operations for MagneticTransfer

Parameters

- **object** (Container) – The Container to be operated on
- **duration** (*str or* Unit) – The duration of the operation

Returns mag_dry parameters

Return type dict

Raises `TypeError` – If *object* is not a Container

See also:

`Protocol.mag_dry()`

static mag_incubate(*object*, *duration*, *magnetize*, *tip_position*, *temperature=None*)

Helper for building mag_incubate sub operations for MagneticTransfer

Parameters

- **object** (Container) – The Container to be operated on
- **duration** (*str or* Unit) – The duration of the operation

- **magnetize** (*bool*) – Whether the magnetic head should be engaged during the operation
- **tip_position** (*float*) – Position relative to well height that magnetic head is held
- **temperature** (*str or Unit, optional*) – The temperature of the operation

Returns mag_incubate parameters

Return type dict

Raises

- TypeError – If *object* is not a Container
- TypeError – If *magnetize* is not a bool
- ValueError – If *tip_position* is not a positive number

See also:

Protocol.mag_incubate()

static mag_collect(*object*, *cycles*, *pause_duration*, *bottom_position=None*, *temperature=None*)
Helper for building mag_collect sub operations for MagneticTransfer

Parameters

- **object** (Container) – The Container to be operated on
- **cycles** (*int*) – The number of times the operation should be repeated
- **pause_duration** (*str or Unit*) – The delay time between each repetition of the operation
- **bottom_position** (*float, optional*) – Position relative to well height where the magnetic head pauses
- **temperature** (*str or Unit, optional*) – The temperature of the operation

Returns mag_collect parameters

Return type dict

Raises

- TypeError – If *object* is not a Container
- TypeError – If *cycles* is not an int
- ValueError – If *bottom_position* is not a positive number

See also:

Protocol.mag_collect()

static mag_release(*object*, *duration*, *frequency*, *center=None*, *amplitude=None*, *temperature=None*)
Helper for building mag_release sub operations for MagneticTransfer

Parameters

- **object** (Container) – The Container to be operated on
- **duration** (*str or Unit*) – The duration of the operation
- **frequency** (*str or Unit*) – The frequency of the magnetic head during the operation

- **center** (*float, optional*) – Position relative to well height where oscillation is centered
- **amplitude** (*float, optional*) – Distance relative to well height to oscillate around *center*
- **temperature** (*str or Unit, optional*) – The temperature of the operation

Returns mag_release parameters

Return type dict

Raises

- TypeError – If *object* is not a Container
- ValueError – If *center* is less than 0
- ValueError – If *amplitude* is greater than center

See also:

Protocol.mag_release()

static mag_mix(*object*, *duration*, *frequency*, *center=None*, *amplitude=None*, *magnetize=None*, *temperature=None*)

Helper for building mag_mix sub operations for MagneticTransfer

Parameters

- **object** (Container) – The Container to be operated on
- **duration** (*str or Unit*) – The duration of the operation
- **frequency** (*str or Unit, optional*) – The frequency of the magnetic head during the operation
- **center** (*float, optional*) – Position relative to well height where oscillation is centered
- **amplitude** (*float, optional*) – Distance relative to well height to oscillate around *center*
- **magnetize** (*bool, optional*) – Whether the magnetic head should be engaged during the operation
- **temperature** (*str or Unit, optional*) – The temperature of the operation

Returns mag_mix parameters

Return type dict

Raises

- TypeError – If *object* is not a Container
- ValueError – If *center* is less than 0
- ValueError – If *amplitude* is greater than center
- TypeError – If *magnetize* is not a bool

See also:

Protocol.mag_mix()

shape(*rows=1*, *columns=1*, *format=None*)

Helper function for building a shape dictionary

Parameters

- **rows** (*int, optional*) – Number of rows to be concurrently transferred
- **columns** (*int, optional*) – Number of columns to be concurrently transferred
- **format** (*str, optional*) – Plate format in String form. e.g. "SBS96" or "SBS384"

Returns shape parameters

Return type dict

Raises

- TypeError – If rows/columns aren't ints
- ValueError – If an invalid row/column count is given
- ValueError – If an invalid shape is given
- ValueError – If rows/columns are greater than what is allowed for the format

class autoprotocol.builders.**FlowCytometryBuilders**

Builders for FlowCytometry instructions.

laser(*excitation*, *channels*, *power=None*, *area_scaling_factor=None*)

Generates a dict of laser parameters.

Parameters

- **excitation** (Unit *or* str) – Excitation wavelength.
- **channels** (*list(dict)*) – See *FlowCytometryBuilders.channel()*.
- **power** (Unit *or* str, *optional*) – Laser power.
- **area_scaling_factor** (*Number, optional*) – Value to scale height and area equivalently.

Raises

- TypeError – If *channels* is not a list of dict.
- TypeError – If *channels* is not a list of dict.
- TypeError – If *area_scaling_factor* is not a number.

Returns A dict of laser parameters.

Return type dict

channel(*emission_filter*, *detector_gain*, *measurements=None*, *trigger_threshold=None*, *trigger_logic=None*)

Generates a dict of channel parameters.

Parameters

- **emission_filter** (*dict*) – See *FlowCytometryBuilders.emission_filter()*.
- **detector_gain** (Unit *or* str) – Detector gain.
- **measurements** (*dict, optional*) – Pulse properties to record. See *FlowCytometryBuilders.measurements()*.
- **trigger_threshold** (*int, optional*) – Channel intensity threshold. Events below this threshold.

- **trigger_logic** (*Enum({"and", "or"}), optional*) – Operator used to combine threshold.

Raises

- TypeError – If *trigger_threshold* is not of type int.
- ValueError – If *trigger_logic* is not one of {"and", "or"}.

Returns A dict of channel parameters.

Return type dict

emission_filter(*channel_name*, *shortpass=None*, *longpass=None*)

Generates a dict of emission filter parameters.

Parameters

- **channel_name** (*str*) – Specifies the channel name.
- **shortpass** (*Unit or str*) – Shortpass filter wavelength.
- **longpass** (*Unit or str*) – Longpass filter wavelength.

Raises ValueError – If values for longpass or shortpass are provided and channel_name is "FSC" or "SSC".

Returns A dict of emission_filter params.

Return type dict

static measurements(*area=None*, *height=None*, *width=None*)

Generates a dict of measurements parameters.

Parameters

- **area** (*bool, optional*) – Area measurement.
- **height** (*bool, optional*) – Height measurement.
- **width** (*bool, optional*) – Width measurement.

Raises TypeError – If any of *area* | *height* | *width* are not of type bool.

Returns A dict of measurements params.

Return type dict

collection_conditions(*acquisition_volume*, *flowrate*, *wait_time*, *mix_cycles*, *mix_volume*, *rinse_cycles*, *stop_criteria=None*)

Generates a dict of collection_conditions parameters.

Parameters

- **acquisition_volume** (*Unit or str*) – Acquisition volume.
- **flowrate** (*Unit or str*) – Flow rate.
- **wait_time** (*Unit or str*) – Waiting time.
- **mix_cycles** (*int*) – Number of mixing cycles before acquisition.
- **mix_volume** (*Unit or str*) – Mixing volume.
- **rinse_cycles** (*int*) – Number of rinsing cycles.
- **stop_criteria** (*dict, optional*) – See *FlowCytometryBuilders.stop_criteria()*.

Raises

- TypeError – If *rinse_cycles* is not of type int.
- TypeError – If *mix_cycles* is not of type int.

Returns A dict of *collection_condition* parameters.

Return type dict

static stop_criteria(*volume=None*, *events=None*, *time=None*)
Generates a dict of stop_criteria parameters.

Parameters

- **volume** (*Unit or str, optional*) – Stopping volume.
- **events** (*int, optional*) – Number of events to trigger stop.
- **time** (*Unit or str, optional*) – Stopping time.

Raises TypeError – If *events* is not of type int.

Returns A dict of *stop_criteria* params.

Return type dict

shape(*rows=1*, *columns=1*, *format=None*)
Helper function for building a shape dictionary

Parameters

- **rows** (*int, optional*) – Number of rows to be concurrently transferred
- **columns** (*int, optional*) – Number of columns to be concurrently transferred
- **format** (*str, optional*) – Plate format in String form. e.g. "SBS96" or "SBS384"

Returns shape parameters

Return type dict

Raises

- TypeError – If rows/columns aren't ints
- ValueError – If an invalid row/column count is given
- ValueError – If an invalid shape is given
- ValueError – If rows/columns are greater than what is allowed for the format

CHAPTER 5

autoprotocol.liquid_handle

5.1 liquid_handle.liquid_handle_method

LiquidHandleMethod

Base class for generating complex liquid handling behavior.

5.1.1 Summary

LiquidHandleMethods are passed as arguments to Protocol methods along with LiquidClasses to specify complex series of liquid handling behaviors.

Notes

Methods in this file should not be used directly, but are intended to be extended by other methods depending on desired behavior.

When creating a vendor-specific library it's likely desirable to monkey patch *LiquidHandleMethod._get_tip_types* to reference TipTypes that the vendor supports.

class autoprotocol.liquid_handle.liquid_handle_method.**LiquidHandleMethod**(*tip_type=None*, *blowout=True*)

Base LiquidHandleMethod

General framework for liquid handling abstractions and helpers for building a series of liquid_handle transports.

_shape
the SBS shape and number of rows and columns of the liquid_handle

Type dict

_transports
tracks transports to be added to the LiquidHandle instruction

Type list

Notes

There is a hierarchy of logic to all LiquidHandleMethods that abstracts a complex set of liquid handling behavior into smaller, discrete steps.

For step *x* (aspirate, dispense, mix) and parameter *y* (e.g. blowout):

- **Protocol method:**
  - calls LiquidHandleMethod._'x'_transports
- **LiquidHandleMethod._'x'_transports method:**
  - clears the _transports list
  - walks through all _transport methods including _transport_'y'
  - returns the _transports lists
- **LiquidHandleMethod._transport_'y' method:**
  - checks parameter *y* in addition to the default_'y' method
  - possibly generates a series of transports based on the two values
  - calls lower level helper methods
- **LiquidHandleMethod lower level helper methods:**
  - generate transports and append them to _transports

Examples

For specifying a single, global liquid handling behavior across all volumes the easiest way is to specify parameters when instantiating a LiquidHandleMethod.

```
from autoprotocol import Unit
from autoprotocol.instruction import LiquidHandle
from autoprotocol.liquid_handle import LiquidHandleMethod

lhm = LiquidHandleMethod(
    blowout=LiquidHandle.builders.blowout(volume=Unit(10, "uL"))
)
```

For behavior that relies on more liquid handling parameters or even defines new behavior you can define your own LiquidHandleMethod.

```
from autoprotocol import Unit
from autoprotocol.instruction import LiquidHandle
from autoprotocol.liquid_handle import LiquidHandleMethod

class NewLHM(LiquidHandleMethod):
    def default_blowout(self, volume):
        if volume < Unit(10, "uL"):
            blowout_volume = Unit(1, "uL")
        else:
            blowout_volume = Unit(10, "uL")
        return LiquidHandle.builders.blowout(
```

(continues on next page)

(continued from previous page)

```
            volume=blowout_volume
        )
```

See also:

`Transfer` method for handling liquid between two locations

`Mix` method for handling liquid within locations

`LiquidClass` contain properties that are intrinsic to specific liquids

`Protocol` contains methods that accept LiquidHandleMethods as arguments

`default_blowout`(*volume*)
Default blowout behavior

Parameters **`volume`** (`Unit`) –

Returns blowout_params

Return type dict

See also:

`blowout()` holds any user specified blowout parameters

`_transport_blowout()` generates the actual blowout transports

`static default_lld_position_z`(*liquid*)
Default lld position_z

Returns position_z for sensing the liquid surface

Return type dict

`static default_tracked_position_z`()
Default tracked position_z

Returns position_z for tracking the liquid surface

Return type dict

`static default_well_bottom_position_z`()
Default well bottom position_z

Returns position_z for the well bottom

Return type dict

`static default_well_top_position_z`()
Default well top position_z

Returns position_z for the well top

Return type dict

5.2 liquid_handle.liquid_class

LiquidClass

Base class for defining the portions of liquid handling behavior that are intrinsic to specific types of liquids.

class autoprotocol.liquid_handle.liquid_class.**LiquidClass**(*calibrated_volume=None*, *aspirate_flowrate=None*, *dispense_flowrate=None*, *delay_time=None*, *clld_threshold=None*, *plld_threshold=None*)

Contains properties intrinsic to individual LiquidClasses

name
the name of the liquid_class may be used by vendors to generate more sensible defaults for unspecified behavior

Type str

volume_calibration_curve
a calibration curve describing the relationship between tip_type, volume bins, and volume calibration parameters See Also VolumeCalibration

Type dict(str, *VolumeCalibration*)

aspirate_flowrate_calibration_curve
a calibration curve describing the relationship between tip_type, volume bins, and aspirate flowrate calibration parameters See Also VolumeCalibration

Type dict(str, *VolumeCalibration*)

dispense_flowrate_calibration_curve
a calibration curve describing the relationship between tip_type, volume bins, and dispense flowrate calibration parameters See Also VolumeCalibration

Type dict(str, *VolumeCalibration*)

_safe_volume_multiplier
a multiplier used by LiquidHandleMethods to estimate safe pump buffers for volume calibration without any prior knowledge about tip_type See Also LiquidHandleMethod._estimate_calibrated_volume

Type Numeric

Examples

For specifying a single, global liquid handling behavior across all volumes the easiest way is to specify parameters when instantiating a LiquidClass. If the following LiquidClass is specified then the pump_override_volume will always be set to 10:uL and the flowrate for all aspirate steps will have a target of 10:uL/s, regardless of the stated volume to be transferred.

```
from autoprotocol import Unit
from autoprotocol.instruction import LiquidHandle
from autoprotocol.liquid_handle import LiquidClass

lc = LiquidClass(
    aspirate_flowrate=LiquidHandle.builders.flowrate(
        target=Unit(10, "ul/s")
    ),
    calibrated_volume=Unit(10, "uL")
)
```

For behavior that differs between volumes you can define your own LiquidClass.

```
from autoprotocol import Unit
from autoprotocol.instruction import LiquidHandle
from autoprotocol.liquid_handle.liquid_class import (
    LiquidClass, VolumeCalibration, VolumeCalibrationBin
)

vol_curve = {
    "generic_1_50": VolumeCalibration(
        (Unit(5, "uL"), VolumeCalibrationBin(
            slope=1.1, intercept=Unit(0.1, "uL")
        )),
        (Unit(10, "uL"), VolumeCalibrationBin(
            slope=0.9, intercept=Unit(0.2, "uL")
        ))
    )
}
asp_flow_curve = {
    "generic_1_50": VolumeCalibration(
        (Unit(5, "uL"), LiquidHandle.builders.flowrate(
            target=Unit(50, "uL/s")
        )),
        (Unit(15, "uL"), LiquidHandle.builders.flowrate(
            target=Unit(200, "uL/s")
        ))
    )
}

class NewLC(LiquidClass):
    def __init__(self, *args, **kwargs):
        super(NewLC, self).__init__(*args, **kwargs)
        self.volume_calibration_curve = vol_curve
        self.aspirate_flowrate_calibration_curve = asp_flow_curve
```

See also:

VolumeCalibration used to specify calibration_curves

LiquidHandleMethod used to specify liquid handling movement behavior

Protocol.transfer accepts LiquidClass arguments to determine behavior

Protocol.mix accepts a LiquidClass argument to determine behavior

class autoprotocol.liquid_handle.liquid_class.**VolumeCalibrationBin**

Wrapper for slope and intercept parameters for linear fitting Holds information required to calibrate a volume for liquid handle step assuming a linear relationship between volume and calibrated volume.

static __new__(*cls*, *slope*, *intercept*)

Parameters

- **slope** (*Number*) – The slope of the linear fit volume calibration function.
- **intercept** (Unit) – The intercept of the linear fit volume calibration function.

Returns an object used for linear fitting volumes within a bin

Return type *VolumeCalibrationBin*

Raises TypeError – if slope is not a number

calibrate_volume(*volume*)
Calibrates the volume using slope and intercept

Parameters **volume** (Unit) – the volume to be calibrated

Returns calibrated volume

Return type *Unit*

count()
Return number of occurrences of value.

index()
Return first index of value.

Raises ValueError if the value is not present.

intercept
Alias for field number 1

slope
Alias for field number 0

class autoprotocol.liquid_handle.liquid_class.**VolumeCalibration**(**args*)
Wrapper for a volume-binned calibration curve A data structure that represents a calibration curve for either volumes or flowrates that are binned by upper bounded volume ranges.

binned_calibration_for_volume(*volume*)
Gets the smallest suitable bin in the calibration curve Finds the smallest point on the calibration curve that has a bin that's greater than or equal to the size of the specified value.

Parameters **volume** (*Unit or int or float*) – the value to be binned

Returns target_bin

Return type dict

Raises RuntimeError – No suitably large calibration bin

5.3 liquid_handle.transfer

Transfer LiquidHandleMethod

Base LiquidHandleMethod used by Protocol.transfer to generate a series of movements between pairs of wells.

class autoprotocol.liquid_handle.transfer.**Transfer**(*tip_type=None*, *blowout=True*, *prime=True*, *transit=True*, *mix_before=False*, *mix_after=True*, *aspirate_z=None*, *dispense_z=None*)
LiquidHandleMethod for generating transfers between pairs of wells

LiquidHandleMethod for transferring volume from one well to another.

_source_liquid
used to determine calibration, flowrates, and sensing thresholds

Type *LiquidClass*

_destination_liquid
used to determine calibration, flowrates, and sensing thresholds

Type *LiquidClass*

Notes

The primary entry points that for this class are:

- _aspirate_transports : generates transports for a source location
- _dispense_transports : generates transports for a destination location

See also:

LiquidHandleMethod base LiquidHandleMethod with reused functionality

Protocol.transfer the standard interface for interacting with Transfer

default_blowout(*volume*)
Default blowout behavior

Parameters **volume** (Unit) –

Returns blowout_params

Return type dict

See also:

blowout() holds any user specified blowout parameters

_transport_blowout() generates the actual blowout transports

default_mix_before(*volume*)
Default mix_before parameters

Parameters **volume** (Unit) –

Returns mix_before params

Return type dict

See also:

mix_before() holds any user defined mix_before parameters

_transport_mix() generates the actual mix_before transports

default_aspirate_z(*volume*)
Default aspirate_z parameters

Parameters **volume** (Unit) –

Returns aspirate position_z

Return type dict

See also:

aspirate_z() holds any user defined aspirate_z parameters

_transport_aspirate_target_volume() generates actual aspirate transports

default_prime(*volume*)
Default prime volume

Parameters **volume** (Unit) –

Returns priming volume

Return type *Unit*

See also:

prime() holds any user defined prime volume

_transport_aspirate_target_volume() generates actual aspirate transports

default_transit (*volume*)
Default transit volume

Parameters **volume** (Unit) –

Returns transit volume

Return type *Unit*

See also:

transit() holds any user defined transit volume

_transport_aspirate_transit() generates the actual transit transports

_transport_dispense_transit() generates the actual transit transports

default_dispense_z (*volume*)
Default aspirate_z parameters

Parameters **volume** (Unit) –

Returns dispense position_z

Return type dict

See also:

dispense_z() holds any user defined dispense_z parameters

_transport_dispense_target_volume() generates actual dispense transports

default_mix_after (*volume*)
Default mix_after parameters

Parameters **volume** (Unit) –

Returns mix_after params

Return type dict

See also:

mix_after() holds any user defined mix_after parameters

_transport_mix() generates the actual mix_after transports

static default_lld_position_z (*liquid*)
Default lld position_z

Returns position_z for sensing the liquid surface

Return type dict

static default_tracked_position_z()
Default tracked position_z

Returns position_z for tracking the liquid surface

Return type dict

static default_well_bottom_position_z()
Default well bottom position_z

Returns position_z for the well bottom

Return type dict

static default_well_top_position_z()
Default well top position_z

Returns position_z for the well top

Return type dict

class autoprotocol.liquid_handle.transfer.**DryWellTransfer**(*tip_type=None*, *blowout=True*, *prime=True*, *transit=True*, *mix_before=False*, *mix_after=False*, *aspirate_z=None*, *dispense_z=None*)
Dispenses while tracking liquid without mix_after

default_dispense_z(*volume*)
Default aspirate_z parameters

Parameters **volume** (Unit) –

Returns dispense position_z

Return type dict

See also:

dispense_z() holds any user defined dispense_z parameters

_transport_dispense_target_volume() generates actual dispense transports

default_aspirate_z(*volume*)
Default aspirate_z parameters

Parameters **volume** (Unit) –

Returns aspirate position_z

Return type dict

See also:

aspirate_z() holds any user defined aspirate_z parameters

_transport_aspirate_target_volume() generates actual aspirate transports

default_blowout(*volume*)
Default blowout behavior

Parameters **volume** (Unit) –

Returns blowout_params

Return type dict

See also:

blowout() holds any user specified blowout parameters

_transport_blowout() generates the actual blowout transports

static default_lld_position_z(*liquid*)
Default lld position_z

Returns position_z for sensing the liquid surface

Return type dict

default_mix_after(*volume*)
Default mix_after parameters

Parameters **volume** (Unit) –

Returns mix_after params

Return type dict

See also:

mix_after() holds any user defined mix_after parameters

_transport_mix() generates the actual mix_after transports

default_mix_before(*volume*)
Default mix_before parameters

Parameters **volume** (Unit) –

Returns mix_before params

Return type dict

See also:

mix_before() holds any user defined mix_before parameters

_transport_mix() generates the actual mix_before transports

default_prime(*volume*)
Default prime volume

Parameters **volume** (Unit) –

Returns priming volume

Return type *Unit*

See also:

prime() holds any user defined prime volume

_transport_aspirate_target_volume() generates actual aspirate transports

static default_tracked_position_z()
Default tracked position_z

Returns position_z for tracking the liquid surface

Return type dict

default_transit(*volume*)
Default transit volume

Parameters **volume** (Unit) –

Returns transit volume

Return type *Unit*

See also:

transit() holds any user defined transit volume

_transport_aspirate_transit() generates the actual transit transports

_transport_dispense_transit() generates the actual transit transports

static default_well_bottom_position_z()
Default well bottom position_z

Returns position_z for the well bottom

Return type dict

static default_well_top_position_z()
Default well top position_z

Returns position_z for the well top

Return type dict

class autoprotocol.liquid_handle.transfer.**PreMixBlowoutTransfer**(*tip_type=None*, *blowout=True*, *prime=True*, *transit=True*, *mix_before=False*, *mix_after=True*, *aspirate_z=None*, *dispense_z=None*, *pre_mix_blowout=True*)
Adds an additional blowout before the mix_after step

default_pre_mix_blowout(*volume*)
Default pre_mix_blowout parameters

Parameters **volume** (Unit) –

Returns pre_mix_blowout params

Return type dict

See also:

pre_mix_blowout() holds any user defined pre_mix_blowout parameters

_transport_pre_mix_blowout() generates the actual blowout transports

default_aspirate_z(*volume*)
Default aspirate_z parameters

Parameters **volume** (Unit) –

Returns aspirate position_z

Return type dict

See also:

aspirate_z() holds any user defined aspirate_z parameters

_transport_aspirate_target_volume() generates actual aspirate transports

default_blowout(*volume*)
Default blowout behavior

Parameters **volume** (Unit) –

Returns blowout_params

Return type dict

See also:

blowout() holds any user specified blowout parameters

_transport_blowout() generates the actual blowout transports

default_dispense_z(*volume*)
Default aspirate_z parameters

Parameters **volume** (Unit) –

Returns dispense position_z

Return type dict

See also:

dispense_z() holds any user defined dispense_z parameters

_transport_dispense_target_volume() generates actual dispense transports

static default_lld_position_z(*liquid*)
Default lld position_z

Returns position_z for sensing the liquid surface

Return type dict

default_mix_after(*volume*)
Default mix_after parameters

Parameters **volume** (Unit) –

Returns mix_after params

Return type dict

See also:

mix_after() holds any user defined mix_after parameters

_transport_mix() generates the actual mix_after transports

default_mix_before(*volume*)
Default mix_before parameters

Parameters **volume** (Unit) –

Returns mix_before params

Return type dict

See also:

mix_before() holds any user defined mix_before parameters

_transport_mix() generates the actual mix_before transports

default_prime(*volume*)
Default prime volume

Parameters **volume** (Unit) –

Returns priming volume

Return type *Unit*

See also:

prime() holds any user defined prime volume

_transport_aspirate_target_volume() generates actual aspirate transports

static default_tracked_position_z()
Default tracked position_z

Returns position_z for tracking the liquid surface

Return type dict

default_transit(*volume*)
Default transit volume

Parameters **volume** (Unit) –

Returns transit volume

Return type *Unit*

See also:

transit() holds any user defined transit volume

_transport_aspirate_transit() generates the actual transit transports

_transport_dispense_transit() generates the actual transit transports

static default_well_bottom_position_z()
Default well bottom position_z

Returns position_z for the well bottom

Return type dict

static default_well_top_position_z()
Default well top position_z

Returns position_z for the well top

Return type dict

5.4 liquid_handle.mix

Mix LiquidHandleMethod

Base LiquidHandleMethod used by Protocol.mix to generate a series of movements within individual wells.

class autoprotocol.liquid_handle.mix.**Mix**(*tip_type=None*, *blowout=True*, *repetitions=None*, *position_z=None*)

LiquidHandleMethod for generating transfers within wells

LiquidHandleMethod for moving volume within a single well.

_liquid
used to determine calibration, flowrates, and sensing thresholds

Type *LiquidClass*

Notes

The primary entry points that for this class are:

- _mix_transports : generates transports within a single location

---

See also:

LiquidHandleMethod base LiquidHandleMethod with reused functionality

Protocol.mix the standard interface for interacting with Mix

default_blowout(*volume*)
Default blowout behavior

Parameters **volume** (Unit) –

Returns blowout_params

Return type dict

See also:

blowout() holds any user specified blowout parameters

_transport_blowout() generates the actual blowout transports

default_repetitions(*volume*)
Default mix repetitions

Parameters **volume** (Unit) –

Returns number of mix repetitions

Return type int

See also:

repetitions() holds any user defined repetition parameters

_transport_mix() generates the actual mix transports

default_position_z(*volume*)
Default position_z

Parameters **volume** (Unit) –

Returns mix position_z

Return type dict

See also:

position_z() holds any user defined position_z parameters

_transport_mix() generates the actual mix transports

static default_lld_position_z(*liquid*)
Default lld position_z

Returns position_z for sensing the liquid surface

Return type dict

static default_tracked_position_z()
Default tracked position_z

Returns position_z for tracking the liquid surface

Return type dict

static default_well_bottom_position_z()
Default well bottom position_z

Returns position_z for the well bottom

Return type dict

static default_well_top_position_z()
Default well top position_z

Returns position_z for the well top

Return type dict

5.5 liquid_handle.tip_type

Generic tip type and device class mappings for LiquidHandleMethods

CHAPTER 6

autoprotocol support

6.1 autoprotocol.unit

6.1.1 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

unit object

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

Return type *Unit*

__str__(*ndigits=12*)

Parameters **ndigits** (*int, optional*) – Number of decimal places to round to, useful for numerical precision reasons

Returns This rounds the string presentation to 12 decimal places by default to account for the majority of numerical precision issues

Return type str

magnitude
Quantity's magnitude. Long form for *m*

ceil()
Equivalent of math.ceil(Unit) for python 2 compatibility

Returns ceil of Unit

Return type *Unit*

floor()
Equivalent of math.floor(Unit) for python 2 compatibility

Returns floor of Unit

Return type *Unit*

round(*ndigits*)
Equivalent of round(Unit) for python 2 compatibility

Parameters **ndigits** (*int*) – number of decimal places to be rounded to

Returns rounded unit

Return type *Unit*

6.2 autoprotocol.util

6.2.1 util.is_valid_well()

autoprotocol.util.**is_valid_well**(*well*)
Checks if an input is of type Well, WellGroup or list of type Well.

Example Usage:

```
if not is_valid_well(source):
    raise TypeError("Source must be of type Well, list of Wells, or "
                    "WellGroup.")
```

Parameters **well** (*Well or WellGroup or list(Well)*) – Parameter to validate is type Well, WellGroup, list of Wells.

Returns Returns True if param is of type Well, WellGroup or list of type Well.

Return type bool

6.2.2 util.parse_unit()

autoprotocol.util.**parse_unit**(*unit*, *accepted_unit=None*)

Parses and checks unit provided and ensures its of valid type and dimensionality.

Note that this also checks against the dimensionality of the *accepted_unit*. I.e. *parse_unit("1:s", "minute")* will return True.

Raises type errors if the Unit provided is invalid.

Parameters

- **unit** (*Unit or str*) – Input to be checked
- **accepted_unit** (*Unit or str or list(Unit) or list(str), optional*) – Dimensionality of unit should match against the accepted unit(s).

Examples

```
parse_unit("1:ul", "1:ml")
parse_unit("1:ul", "ml")
parse_unit("1:ul", ["ml", "kg"])
```

Returns Parsed and checked unit

Return type *Unit*

Raises TypeError – Error when input does not match expected type or dimensionality

6.3 autoprotocol.harness

6.3.1 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()

6.3.2 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"
```


(continues on next page)

(continued from previous page)

```
    }
  ]
}
```

6.3.3 harness.Manifest

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

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 7

Changelog

- #193: Add support for well properties with non-string values in *wells_with*
- #190: Make Well.add_properties extend the original instead of replacing it if both values are lists
- #191: Add initial cover state to ref opts (ASC-042)
- #185: Allow *384-flat-white-clear* containers to be sealed with *ultra-clear* seals
- #188: Add support for non-string aliquot property values as long as they're JSON-serializable
- #188: Add *Container* utils for selecting wells
- #188: Add *Protocol* flag to propagate aliquot properties when liquid handling
- #184: Improve CI pipeline and fix lint warnings for new versions of pylint
- #187: Remove Phabricator URI from .arcconfig
- #182: fix *WellGroup* missing equality method
- #183: fix *ThermocycleBuilders.dyes* to reference ints instead of Wells
- #186: Fix well volume math when liquid handling in python2 and add missing seal capability for *384-flat-white-clear*
- #180: add support for *read_position* and *position_z* to *spectrophotometry* (ASC-041)
- #177: update *Incubate* instruction and corresponding protocol method *co2* parameter docstrings and add type check
- #178: fixed passing through of store_lid field in *p.uncover*
- #176: increment version
- #175: fix transfer failing to assign *tip_type* with calibrated transfers that require splitting
- #161: shift *op* as an official attribute of Instruction
- #161: add ideal time constraints which can be specified by *add_time_constraint* (ASC-037)
- #161: add *batch_containers*, for controlling containers entering/exiting together

- #161: modify *acoustic_transfer* to no longer proactively group consecutive instructions. Please use *WellGroup* explicitly instead
- #161: add *util.parse_unit*, a helper for parsing and checking an unit input
- #161: add *util.check_unit*, a helper for checking the units in bounds
- #161: all protocol methods now return the Instruction
- #161: add parameters to *p.dispense*, including *flowrate*, *nozzle_position*, *step_size*, *reagent_source*, *dispense_speed*, *pre_dispense*, *shape*, *shake_after* options (ASC-027, ASC-029, ASC-036, ASC-039)
- #161: add *lid_temperature* to *p.thermocycle* (ASC-035)
- #161: add *settle_time* to *p.absorbance* (ASC-026)
- #161: add parameters to *p.fluorescence*, including *detection_mode*, *position_z*, *settle_time*, *lag_time*, *integration_time* (ASC-026)
- #161: add parameters to *p.luminescence*, including *settle_time*, *integration_time* (ASC-026)
- #161: add parameters to *p.seal*, including *mode*, *temperature*, *duration* (ASC-034)
- #161: add *retrieve_lid* to *p.cover* (ASC-040)
- #161: add *store_lid* to *p.uncover* (ASC-040)
- #161: change *measure_mass* instruction to take in a single container instead (ASC-030)
- #161: add *count_cells* instruction (ASC-033)
- #161: add *spectrophotometry* instruction (ASC-038)
- #161: added builtin support for *ceil* and *floor* and changed py2 compatibility *Unit.floor* and *Unit.ceil* methods to use them
- #161: converted all Unit internals to use Decimals in place of other Numbers
- #161: deprecate *util.make_dottable_dict* and *util.deep_merge_params*
- #161: deprecate *newpick* in *p.autopick*
- #161: deprecate support for generating multiple GelSeparate instructions using *p.gel_separate*
- #161: deprecate support for *p.append* in favor of *p._append_and_return*
- #163: deprecated Pipette, Stamp, Consolidate, Distribute, and Spread instructions
- #163: deprecated the *p.consolidate* and *p.distribute* protocol methods
- #163: replaced the internals of *p.spread* with a new implementation that generates a liquid_handle instruction
- #163: replaced *p.stamp* & *p.transfer* with a new implementation of *p.transfer* that generates a liquid_handle instruction
- #163: add LiquidHandleMethods and corresponding protocol methods to represent generic liquid handling abstractions
- #163: add liquid_handle instruction (ASC-032)
- #168: improved pruning of empty data structures from 'Instruction.data' field
- #166: add 384-well flat-bottom polystyrene plate containerType
- #165: deprecate *Instruction.json* method for now as most instructions contain non-JSON-serializable objects
- #165: update instruction serialization to use a new _as_AST method as op is no longer included in Instruction data

- #165: add __repr__ methods to Autoprotocol Python objects
- #170: port existing checkers to builders format
- #170: deprecate unused utils including *euclidean_distance*, *quad_ind_to_num*, and *quad_num_to_ind*
- #170: protect liquid_handle-related utils until they can be made more general-purpose
- #174: use more sensible default z positions for pre_buffer and blowout in *LiquidHandleMethod*
- #172: add new *FlowCytometry* instruction and corresponding protocol method
- #167: properly handle *transfer* with tip_type and no volume calibration
- #174: fix broken PreMixBlowoutTransfer that used outdated logic
- #160: change default linter to pylint and update tox
- #161: cleaned up references of *Unit.fromstring* and *Unit._magnitude*
- #162: fix and update docstrings so that sphinx can be executed with no warnings
- #164: update *docs/requirements.txt* for rtd to build properly
- #169: add CONTRIBUTING.rst, cleaned up README.md, and ported it to rst
- : 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), *container-type-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*
- : remove cover prior to mag steps where applicable
- : fix documentation typos
- : convert test suite to py.test
- : docstring cleanup, linting
- : update default lid types for *container-type-384-echo*, *container-type-96-flat*, *container-type-96-flat-uv*, and *container-type-96-flat-clear-clear-tc*
- : update pint requirements, update error handling on UnitError
- : new plate types *container-type-384-v-clear-clear*, *container-type-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 *container-type-384-round-clear-clear*
- : more descriptive error message in ref protocol
- : add functions and tests to enable use of *–dye_test* flag
- : Container method: *container-tube*
- : new plate type *container-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 *container-type-384-flat-clear-clear*, *container-type-384-flat-white-white-lv*, *container-type-384-flat-white-white-tc*
- : validations before implicit cover or seal
- : WellGroup.extend(wells) can now take in a list of wells
- : WellGroup methods: *wellgroup-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 *container-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 *container-type-96-deep-kf* and *container-type-96-deep*
- #127: convert pipette operations to microliters
- : dispense_speed and distribute_target in *protocol-distribute* instruction
- : plate type *container-type-6-flat-tc* to ContainerType
- : auto-uncover before *protocol-provision* instructions
- : WellGroup.extend(wells) can now take in a list of wells
- : WellGroup methods: *wellgroup-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 *container-type-96-pcr* and *container-type-384-pcr* plates

- : *protocol-dispense* instruction json outputs
- : documentation
- : new plate types *container-type-384-flat-clear-clear*, *container-type-384-flat-white-white-lv*, *container-type-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 *container-type-6-flat* well volumes
- : release versioning has been removed in favor of protocol versioniong in harness.py
- : kf container types *container-type-96-v-kf* and *container-type-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
- : *container-type-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
- : *container-type-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 8

Contributing

8.1 Licensing

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 or support@transcriptic.com for more details.

8.2 Features and Bugs

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

8.3 Package Structure

8.3.1 Protocol

- Primary user interface for Autoprotocol Python
- Represent high level abstractions around instructions, refs, and constraints
- Have more situational checks than those in `Builders` or `Instruction`
- Have simple arguments that are as flat as possible, ideally with no nesting
- **Don't necessarily have a 1:1 mapping to an `Instruction`**
  - a single call may generate multiple `Instruction` instances
  - a complicated, modal `Instruction` may have multiple corresponding `Protocol` methods
  - significantly complex instructions (e.g. `LiquidHandle`) may require parametrization with user-configurable class instances to avoid overloading the `Protocol` method with too many arguments

8.3.2 Builders

- Constructors for nested `Instruction` parameters
- Assigned to the `builders` attribute of their corresponding `Instruction`
- Only contain checks that are valid for all instances of their corresponding `Instruction`
- **Check the relationship between parameters**
  - a modal `Instruction` generally has `mode_params` that depend on the specified mode (e.g. `LiquidHandle` and `Spectrophotometry.groups`)
  - parameters like `shape` are very interdependent, and only certain combinations of `rows`, `columns`, and `format` are physically possible

8.3.3 Instruction

- Code analogue of an Autoprotocol Instruction; constructs Instruction JSON
- `__init__` parameters mirror structure of Autoprotocol Instruction
- Only validate the type, structure, and extent of their inputs

CHAPTER 9

Credits

Autoprotocol-Python is currently maintained by:

- Vanessa Biggers - polarpine - vanessa@transcriptic.com
- Yang Choo - yangchoo - yang@transcriptic.com
- Jim Culver - drjimmypants - jim@transcriptic.com
- Donald Dalton - dbdalton2000 - donald@transcriptic.com
- Varun Kanwar - VarunKanwar - varun@transcriptic.com
- Peter Lee - pleaderlee - peter@transcriptic.com
- Rhys Ormond - rhysormond - rhys@transcriptic.com

See all Github contributors

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

CHAPTER 10

License

Use the sidebar to navigate specific module documentation.

CHAPTER 11

Autoprotocol Python Library

Autoprotocol is the standard way to express experiments in life science. This repository contains a python library for generating Autoprotocol.

11.1 Installation

To work from the latest stable release:

```
pip install autoprotocol
```

check the the releases for more information about the changes that will be downloaded.

Alternatively to get more up-to-date features:

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

check the changelog for information about features included on *master* but not yet released.

11.2 Building a Protocol

A basic protocol is written by declaring `Protocol.ref` objects and acting on them with `Protocol.instruction` methods.

```python
import json
from autoprotocol.protocol import Protocol

# instantiate a protocol object
p = Protocol()

# generate a ref
# specify where it comes from and how it should be handled when the Protocol is done
plate = p.ref("test pcr plate", id=None, cont_type="96-pcr", discard=True)

# generate seal and spin instructions that act on the ref
# some parameters are explicitly specified and others are left to vendor defaults
p.seal(
    ref=plate,
    type="foil",
    mode="thermal",
    temperature="165:celsius",
    duration="1.5:seconds"
)
p.spin(
    ref=plate,
    acceleration="1000:g",
    duration="1:minute"
)

# serialize the protocol as Autoprotocol JSON
print(json.dumps(p.as_dict(), indent=2))
```

which prints

```
{
  "instructions": [
    {
      "op": "seal",
      "object": "test pcr plate",
      "type": "foil",
      "mode": "thermal",
      "mode_params": {
        "temperature": "165:celsius",
        "duration": "1.5:second"
      }
    },
    {
      "op": "spin",
      "object": "test pcr plate",
      "acceleration": "1000:g",
      "duration": "1:minute"
    }
  ],
  "refs": {
    "test pcr plate": {
      "new": "96-pcr",
      "discard": true
    }
  }
}
```

11.3 Extras

Select SublimeText snippets are included with this repository. To use them copy the `autoprotocol-python SublimeText Snippet` folder to your local Sublime `/Packages/User` directory.

11.4 Documentation

For more information, see the documentation.

11.5 Contributing

For more information, see CONTRIBUTING.

11.6 Search the Docs

- genindex
- search

Python Module Index

a

Index

Symbols

A

B

C

D

E

M

N

O

P

Q

R

S

T

U

V

W