S4-Clarity Documentation

Release 1.0

Semaphore Solutions

Apr 06, 2021

Contents

Used in numerous labs around the world, the S4-Clarity library provides an easy-to-use integration with the BaseSpace Clarity

- Classes representing familiar Clarity API entities that provide read-write access to most properties.
- Helper methods that simplify common operations.
- Base classes for scripts that integrate with Clarity: EPPs, DSAs, and shell scripts.
- Utilities that help with Clarity-related tasks, such as managing config slices, or automating the completion of a Step.

The S4-Clarity library lets developers interact with the Clarity API in fewer lines of code. With HTTP and XML boilerplate out of the way, you'll have your integration built in no time at all.

```
from s4.clarity.scripts import TriggeredStepEPP

LibraryVolume = 2.0
MolWeightBasePair = 660 * 1e6  # micrograms / mol
AssumedBasePairs = 400.0
TargetMolarity = 4.0
Overage = 4


class Normalization (TriggeredStepEPP):
    def on_record_details_enter(self):
        self.prefetch(self.PREFETCH_INPUTS, self.PREFETCH_OUTPUTS)

        for iomap in self.step.details.iomaps:
            library_concentration = iomap.input["Concentration"]
            library_molarity = library_concentration / (AssumedBasePairs *␣
↪MolWeightBasePair)
            iomap.output["Concentration"] = library_concentration
            iomap.output["Molarity (nM)"] = library_molarity
            iomap.output["Library Vol (uL)"] = LibraryVolume
            iomap.output["Tris HCl (uL)"] = LibraryVolume * (library_molarity /␣
↪TargetMolarity - 1)

        self.lims.artifacts.batch_update(self.step.details.outputs)


if __name__ == "__main__":
    Normalization.main()
```

CHAPTER 1

User Guide

1.1 User Guide

Here find some examples of how you might put the S4-Clarity library to use.

1.1.1 EPP Scripts

Example StepEPPs

Generate CSV

Generate a simple CSV file and attach it to a step.

This EPP extends *s4.clarity.scripts.StepEPP* and is meant to be initiated from a *Record Details* button.

Record Details Button

```
bash -c "/opt/gls/clarity/customextensions/env/bin/python
/opt/gls/clarity/customextensions/examples/generate_csv.py -u {username} -p {password}
-l {compoundOutputFileLuid0} --step-uri {stepURI} --fileId {compoundOutputFileLuid1}
--fileName 'example.csv' --artifactUDFName Concentration"
```

```
# Copyright 2019 Semaphore Solutions, Inc.
# ---------------------------------------------------------------------------

import logging
import argparse
from csv import DictWriter

from s4.clarity.scripts import StepEPP

HEADER_INPUT_NAME = "Input Name"
```

(continues on next page)

(continued from previous page)

```
HEADER_INPUT_ID = "Input Id"
HEADER_OUTPUT_NAME = "Output Name"
HEADER_OUTPUT_ID = "Output Id"

log = logging.getLogger(__name__)


class GenerateCSV(StepEPP):

    @classmethod
    def add_arguments(cls, argparser):
        # type: (argparse) -> None
        super(GenerateCSV, cls).add_arguments(argparser)
        argparser.add_argument("--fileId",
                               help="{compoundOutputFileLuids} token",
                               required=True)
        argparser.add_argument("--fileName",
                               type=str,
                               help="File name with extension",
                               default="epp_generated.csv")
        argparser.add_argument("--artifactUDFName",
                               type=str,
                               help="Name of the UDF to include in file",
                               required=True)

    def run(self):
        csv_file = self.step.open_resultfile(self.options.fileName, 'w', limsid=self.
↪options.fileId)

        csv_writer = DictWriter(csv_file, fieldnames=[
            HEADER_INPUT_NAME,
            HEADER_INPUT_ID,
            HEADER_OUTPUT_NAME,
            HEADER_OUTPUT_ID,
            self.options.artifactUDFName
        ])

        csv_writer.writeheader()

        for input_analyte, output_analytes in self.step.details.input_keyed_lookup.
↪items():
            for output_analyte in output_analytes:
                udf_value = output_analyte.get(self.options.artifactUDFName, "Empty")

                row = {
                    HEADER_INPUT_NAME: input_analyte.name,
                    HEADER_INPUT_ID: input_analyte.limsid,
                    HEADER_OUTPUT_NAME: output_analyte.name,
                    HEADER_OUTPUT_ID: output_analyte.limsid,
                    self.options.artifactUDFName: udf_value
                }

                log.info("Writing the following line: %s" % row)

                csv_writer.writerow(row)

        csv_file.commit()
```

(continues on next page)

(continued from previous page)

```
if __name__ == "__main__":
    GenerateCSV.main()
```

Example TriggeredStepEPPs

QC Set Next Step

Sets the *next step* action for each output analyte.

This EPP extends *s4.clarity.scripts.TriggeredStepEPP* and is meant to be triggered on the transition into the *Next Steps* screen, and again on the *End of Step* transition.

Record Details Exit transition:

```
bash -c "/opt/gls/clarity/customextensions/env/bin/python
/opt/gls/clarity/customextensions/examples/qc_set_next_step.py -u {username} -p
↪{password}
-l {compoundOutputFileLuid0} --step-uri {stepURI} --action RecordDetailsExit"
```

End of Step transition:

```
bash -c "/opt/gls/clarity/customextensions/env/bin/python
/opt/gls/clarity/customextensions/examples/qc_set_next_step.py -u {username} -p
↪{password}
-l {compoundOutputFileLuid0} --step-uri {stepURI} --action EndOfStep"
```

```
# Copyright 2019 Semaphore Solutions, Inc.
# ---------------------------------------------------------------------------

import logging

from s4.clarity.artifact import Artifact
from s4.clarity.scripts import TriggeredStepEPP

log = logging.getLogger(__name__)


class QCSetNextStep(TriggeredStepEPP):

    def should_repeat_step(self, input_analyte):
        # type: (Artifact) -> bool

        # QC flag is set on output result file artifacts
        output_measurements = self.step.details.input_keyed_lookup[input_analyte]

        # If QC failed for any replicate of the input it should repeat
        return any(output.qc_failed() for output in output_measurements)

    def on_record_details_exit(self):
        """
        Set the next step actions for the user to inspect.
        """
        for analyte, action in self.step.actions.artifact_actions.items():
```

(continues on next page)

(continued from previous page)

```
            if self.should_repeat_step(analyte):
                log.info("Setting Analyte '%s' (%s) to repeat step." % (analyte.name,␣
↪analyte.limsid))
                action.repeat()
            else:
                action.next_step()

        self.step.actions.commit()

    def on_end_of_step(self):
        """
         Ensure analytes repeat the step but do not overwrite other user selections.
        """

        # As this is a QC step it is the inputs that are moving to the next step.
        for input_analyte, action in self.step.actions.artifact_actions.items():

            if self.should_repeat_step(input_analyte):
                log.info("Setting Analyte '%s' (%s) to repeat step." % (input_analyte.
↪name, input_analyte.limsid))
                action.repeat()

        self.step.actions.commit()


if __name__ == "__main__":
    QCSetNextStep.main()
```

Create Pools of Two

Groups the step's input analytes into pools of two.

This EPP extends *s4.clarity.scripts.TriggeredStepEPP* and is meant to be triggered on the transition into *Pooling*.

Pooling Enter transition:

```
bash -c "/opt/gls/clarity/customextensions/env/bin/python
/opt/gls/clarity/customextensions/examples/create_pools_of_two.py -u {username} -p
↪{password}
-l {compoundOutputFileLuid0} --step-uri {stepURI} --action PoolingEnter"
```

```
# Copyright 2019 Semaphore Solutions, Inc.
# ---------------------------------------------------------------------------

from s4.clarity.scripts import TriggeredStepEPP
from s4.clarity.scripts import UserMessageException


class CreatePoolsOfTwo(TriggeredStepEPP):

    def validate_even_number_of_inputs(self):
        num_input_analytes = len(self.step.pooling.available_inputs)
```

(continues on next page)

(continued from previous page)

```
        if num_input_analytes % 2:
            raise UserMessageException("Step must be started with an even number of␣
↪analytes.")

    def validate_single_input_replicate_created(self):
        for available_input in self.step.pooling.available_inputs:
            if available_input.replicates > 1:
                raise UserMessageException("No more than one replicate per analyte␣
↪allowed in this step")

    def on_pooling_enter(self):
        self.validate_single_input_replicate_created()
        self.validate_even_number_of_inputs()

        inputs = [a.input for a in self.step.pooling.available_inputs]

        for i in range(0, len(inputs) - 1, 2):
            pool_inputs = [
                inputs[i],
                inputs[i + 1]
            ]

            pool_name = "%s_%s" % (pool_inputs[0].limsid, pool_inputs[1].limsid)

            self.step.pooling.create_pool(pool_name, pool_inputs)

        self.step.pooling.commit()


if __name__ == "__main__":
    CreatePoolsOfTwo.main()
```

Example DerivedSampleAutomations

Set UDF Value

Assigns a user provided value to the analyte UDF specified for all selected analytes.

This EPP extends *s4.clarity.scripts.DerivedSampleAutomation* and is meant to be triggered from the projects dashboard.

Automation Configuration

```
bash -c "/opt/gls/clarity/customextensions/env/bin/python
/opt/gls/clarity/customextensions/examples/set_udf_value.py -u {username} -p
↪{password}
-a '{baseURI}v2' -d {derivedSampleLuids}  --udfName '{userinput:UDF_Name}' --udfValue
↪'{userinput:UDF_Value}'"
```

```
# Copyright 2019 Semaphore Solutions, Inc.
# ---------------------------------------------------------------------------

import argparse

from s4.clarity.scripts import DerivedSampleAutomation
```

(continues on next page)

(continued from previous page)

```
class SetUDFValue(DerivedSampleAutomation):

    @classmethod
    def add_arguments(cls, argparser):
        # type: (argparse) -> None
        super(SetUDFValue, cls).add_arguments(argparser)
        argparser.add_argument("--udfName",
                               type=str,
                               help="The name of the analyte UDF",
                               required=True)
        argparser.add_argument("--udfValue",
                               type=str,
                               help="The new value for the UDF",
                               required=True)

    def process_derived_samples(self):
        for artifact in self.artifacts:
            artifact[self.options.udfName] = self.options.udfValue

        self.lims.artifacts.batch_update(self.artifacts)

        return "Successfully set UDF '%s' to '%s' for %s derived samples" % \
               (self.options.udfName, self.options.udfValue, len(self.artifacts))


if __name__ == "__main__":
    SetUDFValue.main()
```

1.1.2 Shell Scripts

Accession Clarity Sample

Accessions a new sample into Clarity using the provided container name and project.

This script extends *s4.clarity.scripts.ShellScript* and is meant to be executed from the command line.

Example Invocation

```
python ./accession_clarity_sample.py -u <user name> -p <password> -r https://<clarity_
↪server>/api/v2
--sampleName <Sample Name> --projectName <Project Name> --containerName <Container␣
↪Name>
```

```
# Copyright 2019 Semaphore Solutions, Inc.
# ---------------------------------------------------------------------------

import logging
import argparse

from s4.clarity.scripts import ShellScript

log = logging.getLogger(__name__)
```

(continues on next page)

(continued from previous page)

```
class AccessionClaritySample(ShellScript):

    @classmethod
    def add_arguments(cls, argparser):
        # type: (argparse) -> None
        super(AccessionClaritySample, cls).add_arguments(argparser)
        argparser.add_argument("--sampleName",
                               type=str,
                               help="The name of the sample to create",
                               required=True)
        argparser.add_argument("--projectName",
                               type=str,
                               help="The name of an existing Clarity project",
                               required=True)
        argparser.add_argument("--containerName",
                               type=str,
                               help="The name of the sample container",
                               required=True)

    def run(self, *args):
        projects = self.lims.projects.query(name=self.options.projectName)
        if not projects:
            raise Exception("Project '%s' does not exist in Clarity" % self.options.
↪projectName)

        project = projects[0]

        tube_type = self.lims.container_types.get_by_name("Tube")
        container = self.lims.containers.new(container_type=tube_type, name=self.
↪options.containerName)
        container = self.lims.containers.add(container)

        sample = self.lims.samples.new(name=self.options.sampleName, project=project)
        sample.set_location_coords(container, 1, 1)
        self.lims.samples.add(sample)

        log.info("Sample '%s' successfully accessioned in Clarity" % self.options.
↪sampleName)


if __name__ == "__main__":
    AccessionClaritySample.main()
```

1.1.3 Workflow Testing

Workflow Run

Runs two samples through a three step protocol.

Example Invocation

```
python ./workflow_run.py -u <user name> -p <password> -r https://<clarity_server>/api/
↪v2
```

```
# Copyright 2019 Semaphore Solutions, Inc.
#
# Assumes a single protocol workflow that consists of three steps:
# - QC
# - Pooling
# - Standard
# ---------------------------------------------------------------------------

import logging

from s4.clarity.configuration import Workflow
from s4.clarity.container import Container
from s4.clarity.project import Project
from s4.clarity.sample import Sample
from s4.clarity.scripts import WorkflowTest
from s4.clarity.steputils.placement_utils import auto_place_artifacts
from s4.clarity.steputils.step_runner import StepRunner

log = logging.getLogger(__name__)

NAME_PROTOCOL = "Testing Protocol"


class QCStepRunner(StepRunner):
    def __init__(self, lims):
        super(QCStepRunner, self).__init__(lims, NAME_PROTOCOL, "QC Step")

    def record_details(self):
        # Set the QC flag on all output measurements
        for output in self.step.details.outputs:
            output.qc = True

        self.lims.artifacts.batch_update(self.step.details.outputs)

    def next_steps(self):
        self.step.actions.all_next_step()
        self.step.actions.commit()


class PoolingStepRunner(StepRunner):
    def __init__(self, lims):
        super(PoolingStepRunner, self).__init__(lims, NAME_PROTOCOL, "Pooling Step")

    def pooling(self):
        # Pool all inputs together
        self.step.pooling.create_pool("The Pool", self.step.details.inputs)
        self.step.pooling.commit()

    def record_details(self):
        # Set a step UDF named "Status"
        self.step.details['Status'] = "Pooling Complete"
        self.step.details.commit()

    def next_steps(self):
        self.step.actions.all_next_step()
        self.step.actions.commit()
```

(continues on next page)

(continued from previous page)

```
class StandardStepRunner(StepRunner):
    def __init__(self, lims):
        super(StandardStepRunner, self).__init__(lims, NAME_PROTOCOL, "Standard Step")

    def placement(self):
        auto_place_artifacts(self.step, self.step.details.outputs)

    def record_details(self):
        # Add all required reagents
        self.add_default_reagents()

        # Set the value of an analyte UDF on all of the outputs.
        for output in self.step.details.outputs:
            output["Inspected By"] = self.step.process.technician.last_name

        self.lims.artifacts.batch_update(self.step.details.outputs)

    def next_steps(self):
        self.step.actions.all_next_step()
        self.step.actions.commit()


class ExampleWorkflowTest(WorkflowTest):
    PROJECT_OWNER_USER_NAME = "admin"
    WORKFLOW_NAME = "Testing Workflow"

    def get_or_create_project(self, project_name):
        # type: (str) -> Project
        projects = self.lims.projects.query(name=project_name)
        if len(projects) > 0:
            log.info("Using existing project %s" % project_name)
            return projects[0]

        project = self.lims.projects.new(name=project_name)
        project.researcher = self.lims.researchers.query(username=self.PROJECT_OWNER_
↪USER_NAME)[0]

        # Submit the project back to Clarity
        self.lims.projects.add(project)
        log.info("Created project %s" % project_name)
        return project

    def get_workflow(self):
        # type: () -> Workflow
        workflow = self.lims.workflows.get_by_name(self.WORKFLOW_NAME)
        if workflow is None:
            raise Exception("Workflow '%s' does not exist in Clarity.")

        return workflow

    def create_tube(self):
        # type: () -> Container
        tube_type = self.lims.container_types.query(name="Tube")[0]
        container = self.lims.containers.new(container_type=tube_type)

        return self.lims.containers.add(container)
```

(continues on next page)

(continued from previous page)

```
    def create_sample(self, name):
        # type: (str) -> Sample
        project = self.get_or_create_project("Today's Project")
        sample = self.lims.samples.new(name=name, project=project)

        # Set the sample container
        container = self.create_tube()
        sample.set_location_coords(container, 1, 1)

        return self.lims.samples.add(sample)

    def run(self, *args):
        log.info("Accessioning samples")
        samples = [
            self.create_sample('Jane'),
            self.create_sample('Bob')
        ]

        log.info("Routing sample to beginning of workflow '%s'", self.WORKFLOW_NAME)
        workflow = self.get_workflow()
        workflow.enqueue([s.artifact for s in samples])

        log.info("Running sample through workflow '%s'", self.WORKFLOW_NAME)
        input_uris = [s.artifact.uri for s in samples]

        qc_step_runner = QCStepRunner(self.lims)
        qc_step_runner.run(inputuris=input_uris)

        pooling_step_runner = PoolingStepRunner(self.lims)
        pooling_step_runner.run(previousstep=qc_step_runner.step)

        StandardStepRunner(self.lims).run(previousstep=pooling_step_runner.step)

        log.info("Sample successfully pushed through workflow '%s'", self.WORKFLOW_
↪NAME)


if __name__ == "__main__":
    ExampleWorkflowTest.main()
```

CHAPTER 2

API Details

2.1 Clarity Elements

Listed here are the definitions for classes that mirror the XML entities used by the Clarity API.

2.1.1 Artifact

class s4.clarity.artifact.**Artifact**(*lims*, *uri=None*, *xml_root=None*, *name=None*, *limsid=None*)

Bases: *s4.clarity._internal.FieldsMixin*, *s4.clarity.ClarityElement*

Reference: https://www.genologics.com/files/permanent/API/latest/data_art.html#artifact

container

From "location.container". For XML value "location.value", use Python property `.location_value`.

Type *Container*

control_type

Type *ControlType*

demux

Provides access to the 'demux' endpoint added in Clarity 5.1.

Using this property with an earlier version of Clarity will result in a 404 Not Found error.

Type *ArtifactDemux*

file

Type *File*

is_control

Type bool

location_value
The value of the XML property 'location/value'

Type str

open_file(*mode*, *only_write_locally=False*, *name=None*)

Parameters

- **mode** (*str*) – 'r', 'r+', 'w', 'a', 'rb', 'r+b', 'wb', 'ab'. NOTE: 'r+' sets initial file position to the beginning, 'a' sets it to the end.
- **only_write_locally** (*bool*) – if true, don't upload this file to Clarity.
- **name** (*str*) – The name that will be used if you are creating a new file.

Return type *File*

output_type
The value of the XML property 'output-type'

Type str

parent_process
The linked *Process* from the 'parent-process' subnode

Type *Process*

parent_step

Type *Step*

qc
Whether QC is marked as PASSED or FAILED on the artifact

Clarity Value	Bool Value
PASSED	True
FAILED	False
UNKNOWN	None

Type bool

qc_failed()

Return type bool

qc_passed()

Return type bool

queued_stages

Type set[*Stage*]

reagent_label_name

Type str

reagent_label_names

Type list[str]

reagent_labels
ReagentLabel items from the 'reagent-label' subnodes

Type list[ReagentLabel]

sample

Type *Sample*

samples

Type list[*Sample*]

set_qc_flag(*value*)
The *qc* property should be used in favor of this method.

Parameters **value** (*bool*) – *True* if PASSED, *False* if FAILED, *None* to unset.

type
The value of the XML property 'type'

Type str

workflow_stages
WorkflowStageHistory items from the 'workflow-stage' subnodes

Type list[*WorkflowStageHistory*]

2.1.2 Artifact Action

class s4.clarity.step.**ArtifactAction**(*lims*, *step*, *xml_root*)

action
The value of the XML property 'action'

Type str

artifact_uri
The value of the XML property 'artifact-uri'

Type str

complete_and_repeat(*step_uri*)
Sets the Next Step property for the artifact specified by artifact_uri to 'Complete and Repeat' with the specified next step uri.

mark_protocol_complete()

next_step(*step_uri=None*)
Set the next step to continue to, or mark the protocol as complete if there is no next step.

If step_uri is not provided, artifact will: - Continue to the first transition defined for this step, if any transitions are defined. - Mark the protocol as complete, if there are no transitions (the protocol is done).

remove_from_workflow()
Sets the Next Step property for the artifact specified by artifact_uri to 'Remove From Workflow'

repeat()
Sets the Next Step property for the artifact specified by artifact_uri to 'Repeat Step'

review()
Sets the Next Step property for the artifact specified by artifact_uri to 'Request Manager Review'

rework(*step_uri*)
Sets the Next Step property for the artifact specified by artifact_uri to 'Rework from an earlier step' with the specified next step uri.

rework_step_uri
The value of the XML property 'rework-step-uri'

Type str

step_uri
The value of the XML property 'step-uri'

Type str

2.1.3 Artifact Demux

class s4.clarity.artifact.**ArtifactDemux**(*lims*, *uri=None*, *xml_root=None*, *name=None*, *limsid=None*)

Bases: *s4.clarity.ClarityElement*

Corresponds to the 'demux' type in the 'artifact' namespace of the Clarity API.

artifact
The linked *Artifact* from the 'artifact' subnode

Type *Artifact*

demux
The element *DemuxDetails* from subnode 'demux'

Type *DemuxDetails*

2.1.4 Automation

class s4.clarity.configuration.**Automation**(*lims*, *uri=None*, *xml_root=None*, *name=None*, *limsid=None*)

channel
The value of the XML property 'channel'

Type str

command_string
The value of the XML property 'string'

Type str

context
The value of the XML property 'context'

Type str

name
The value of the XML property 'name'

Type str

process_types
The linked *ProcessType* objects from the 'process-type' subnodes

Type list[*ProcessType*]

2.1.5 Available Input

class s4.clarity.step.**AvailableInput**(*lims*, *xml_root*)

input

Type *Artifact*

replicates
The value of the XML property 'replicates'

Type number

2.1.6 Clarity Element

class s4.clarity.**ClarityElement**(*lims*, *uri=None*, *xml_root=None*, *name=None*, *limsid=None*)

Variables

- **xml_root** (*ETree.Element*) –
- **uri** (*str|None*) –
- **lims** (LIMS) –

commit()
Shorthand for put_and_parse().

invalidate()
Clear the local cache, forcing a reload next time the element is used.

is_fully_retrieved()

Return type bool

limsid

Type str

name
The name of the element instance in Clarity.

Type str

post_and_parse(*alternate_uri=None*)
POST the current state of this object to a REST endpoint, then parse the response into this object.

Parameters **alternate_uri** (*str*) – Will be used instead of self.uri if provided.

Raises

- **requests.exceptions.HTTPError** – If there are communication problems.
- ***ClarityException*** – If Clarity returns an exception as XML.

put_and_parse(*alternate_uri=None*)
PUT the current state of this object to a REST endpoint, then parse the response into this object.

Parameters **alternate_uri** – Will be used instead of self.uri if provided.

Raises

- **requests.exceptions.HTTPError** – If there are communication problems.
- ***ClarityException*** – If Clarity returns an exception as XML.

refresh()
Retrieve fresh element representation from the API.

xml_root

Type ETree.Element

2.1.7 Clarity Exception

class s4.clarity.**ClarityException**(*msg*)
Bases: Exception

Exceptions that are returned as XML from Clarity LIMS.

static is_authentication_error(*response*)

static is_redirect(*response*)

static is_response_exception(*response*)

classmethod raise_if_present(*response*, *data=None*, *username=None*)

classmethod raise_on_exception(*response*, *data=None*)

2.1.8 Container

class s4.clarity.container.**Container**(*lims*, *uri=None*, *xml_root=None*, *name=None*, *limsid=None*)
Bases: *s4.clarity._internal.FieldsMixin*, *s4.clarity.ClarityElement*

artifact_at(*well*)

Parameters **well** (*str*) – String matching "Y:X" where Y is a column index and X is a row index. The string may use letters or numbers depending on the container type.

Return type *Artifact* or None

container_type
The linked *ContainerType* from the 'type' subnode

Type *ContainerType*

occupied_wells
The value of the XML property 'occupied-wells'

Type number

placements
Dict of string "Y:X" -> Artifacts.

Type dict[str, *Artifact*]

state
The value of the XML property 'state'

Type str

type_name
Read-only shortcut to containertype name, which we know without doing another GET.

Type str

2.1.9 Container Dimension

class s4.clarity.container.**ContainerDimension**(*lims*, *xml_root=None*)
Bases: s4.clarity._internal.element.WrappedXml

as_index(*label*)

as_label(*index*)

dimension_range
List of the labels for the given dimension (row or column)

Returns list[int]|list[str]

is_alpha
The value of the XML property 'is-alpha'

Type bool

offset
The value of the XML property 'offset'

Type number

size
The value of the XML property 'size'

Type number

2.1.10 Container Type

class s4.clarity.container.**ContainerType**(*lims*, *uri=None*, *xml_root=None*, *name=None*, *limsid=None*)

column_order_wells()
Wells in column order, e.g., A:1, A:2, ... Unavailable wells are omitted

Return type list[str]

is_tube
The value of the XML property 'is-tube'

Type bool

rc_to_well(*rc*)
Converts a zero based index of the row and column to a Clarity well position.

Example:

```
(1, 3) -> 'B:4'
```

Parameters **rc** (*tuple[int]*) – The zero based index of the row and the column.

Returns A Clarity formatted well position

Return type str

row_order_wells()
Wells in row order, e.g., A:1, B:1, ... Unavailable wells are omitted

Return type list[str]

total_capacity

Type int

unavailable_wells

Type set[str]

well_to_rc(*well*)
Converts a Clarity well position to the zero based index of the row and column.

Example:

```
'B:4' -> (1, 3)
```

Parameters **well** (*str*) – A Clarity formatted well position

Returns The zero based index of the row and the column.

Return type tuple[int]

x_dimension
The element *ContainerDimension* from subnode 'x-dimension'

Type *ContainerDimension*

x_dimension_range()

Deprecated use `ContainerType.x_dimension.dimension_range` instead

y_dimension
The element *ContainerDimension* from subnode 'y-dimension'

Type *ContainerDimension*

y_dimension_range()

Deprecated use `ContainerType.y_dimension.dimension_range` instead

2.1.11 Control Type

class s4.clarity.control_type.**ControlType**(*lims*, *uri=None*, *xml_root=None*, *name=None*, *limsid=None*)

Bases: *s4.clarity.ClarityElement*

archived
The value of the XML property 'archived'

Type bool

catalogue_number
The value of the XML property 'catalogue-number'

Type str

concentration
The value of the XML property 'concentration'

Type str

single_step
The value of the XML property 'single-step'

Type bool

supplier
The value of the XML property 'supplier'

Type str

website
The value of the XML property 'website'

Type str

2.1.12 Demux Artifact

class s4.clarity.artifact.**DemuxArtifact**(*lims*, *xml_root=None*)
Bases: s4.clarity._internal.element.WrappedXml

Corresponds to the 'demux-artifact' type in the 'artifact' namespace of the Clarity API.

artifact
The linked *Artifact* from the '.' subnode

Type *Artifact*

demux
The element *DemuxDetails* from subnode 'demux'

Some versions of Clarity do not provide this element when the *DemuxArtifact* is a pool of artifacts that are all from the same submitted sample. In this case, if more than one artifact is labeled, the pool *Artifact*'s demux will be provided, but if only one artifact is labeled, the pool's demux will not be provided.

Type *DemuxDetails*

reagent_labels
ReagentLabel items from the 'reagent-label' subnodes

Type list[ReagentLabel]

samples
The linked *Sample* objects from the './samples/sample' subnodes

Type list[*Sample*]

2.1.13 Demux Details

class s4.clarity.artifact.**DemuxDetails**(*lims*, *xml_root=None*)
Bases: s4.clarity._internal.element.WrappedXml

Corresponds to the 'demux-details' type in the 'artifact' namespace of the Clarity API.

demux_artifacts
DemuxArtifact items from the 'artifact' subnodes

Type list[*DemuxArtifact*]

pool_step
The linked *Step* from the './pool-step' subnode

Type *Step*

2.1.14 Element Factory

class s4.clarity.**ElementFactory**(*lims*, *element_class*, *batch_flags=None*, *request_path=None*, *name_attribute='name'*)

Provides access to a Clarity API endpoint. Implements conversion between XML and ClarityElement as well as caching and network services.

Parameters

- **request_path** (*str*) – for example, '/configuration/workflows'. when not specified, uses '/<plural of element name>'.
- **name_attribute** (*str*) – if not "name", provide this to adjust behaviour of 'get_by_name'.

add(*element*)

Add an element to the Factory's internal cache and persist it back to Clarity.

Return type *ClarityElement*

all(*prefetch=True*)

Queries Clarity for all ClarityElements associated with the Factory.

Parameters **prefetch** – Force load full content for each element.

Returns List of ClarityElements returned by Clarity.

batch_create(*elements*)

Creates new records in Clarity for each element and returns these new records as ClarityElements. If this operation can be performed in a single network operation it will be.

Parameters **elements** – A list of new ClarityElements that have not been persisted to Clarity yet.

Returns New ClarityElement records from Clarity, created with the data supplied to the method.

Raises ***ClarityException*** – if Clarity returns an exception as XML

batch_fetch(*elements*)

Updates the content of all ClarityElements with the current state from Clarity. Syntactic sugar for batch_get([e.uri for e in elements])

Returns A list of the elements returned by the query.

batch_get(*uris*, *prefetch=True*)

Queries Clarity for a list of uris described by their REST API endpoint. If this query can be made as a single request it will be done that way.

Parameters

- **uris** – A List of uris
- **prefetch** – Force load full content for each element.

Returns A list of the elements returned by the query.

batch_get_from_limsids(*limsids*)

Return a list of ClarityElements for a given list of limsids

Parameters **limsids** – A list of Clarity limsids

Returns A list of the elements returned by the query.

batch_invalidate(*elements*)

Clears the current local state for all elements. :param elements: The ClarityElements that are to have their current state cleared.

batch_refresh(*elements*)
Loads the current state of the elements from Clarity. Any changes made to these artifacts that has not been pushed to Clarity will be lost. :param elements: All ClarityElements to update from Clarity.

batch_tag

batch_update(*elements*)
Persists the ClarityElements back to Clarity. Will preform this action as a single query if possible.

Parameters **elements** – All ClarityElements to save the state of.

Raises ***ClarityException*** – if Clarity returns an exception as XML

can_batch_create()
Indicates if Clarity will allow batch record creation.

can_batch_get()
Indicates if Clarity will allow batch get requests.

can_batch_update()
Indicates if Clarity will allow batch updates.

can_query()
Indicates if Clarity will allow the user to submit queries.

delete(*element*)
Delete an element from the Factory's internal cache and delete it from Clarity.

from_limsid(*limsid*, *force_full_get=False*)
Returns the ClarityElement with the specified limsid.

from_link_node(*xml_node*)
Will return the ClarityElement described by the link node.

Link nodes are any xml element with the following attributes <element uri='...' name='...' limsid='...' />

from_link_nodes(*xml_nodes*)
Will return the ClarityElements described by the link nodes.

Link nodes are any xml element with the following attributes <element uri='...' name='...' limsid='...' />

get(*uri*, *force_full_get=False*, *name=None*, *limsid=None*)
Returns the cached ClarityElement described by the provide uri. If the element does not exist a new cache entry will be created with the provided name and limsid. If force_full_get is true, and the object is not fully retrieved it will be refreshed.

get_by_name(*name*)
Queries for a ClarityElement that is described by the unique name. An exception is raised if there is no match or more than one match.

Raises

- **NoMatchingElement** – if no match
- **MultipleMatchingElements** – if multiple matches

new(***kwargs*)
Create a new ClarityElement pre-populated with the provided values. This object has yet to be persisted to Clarity.

Parameters **kwargs** – Key/Value list of attribute name/value pairs to initialize the element with.

Returns A new ClarityElement, pre-populated with provided values.

post(*element*)
Posts the current state of the ClarityElement back to Clarity.

query(*prefetch=True*, ***params*)
Queries Clarity for ClarityElements associated with the Factory. The query will be made with the provided parameters encoded in the url. For the specific parameters to pass and the expected values please see the Clarity REST API.

Some of the expected parameters contain the '-' character, in which case the dictionary syntax of this call will need to be used.

Inline parameter names:

```
query(singlevaluename='single value', multivaluename=['A', 'B', 'C'])
```

Dictionary of parameters:

```
query(prefetch=True, ** {
    'single-value-name': 'single value',
    'multi-value-name': ['A', 'B', 'C']
})
```

Parameters

- **params** – Query parameters to pass to clarity.
- **prefetch** – Force load full content for each element.

Returns A list of the elements returned by the query.

query_uris(***params*)
For backwards compatibility, use query() instead. Does a query and returns the URIs of the results.

Parameters **params** – Query parameters to pass to clarity.

2.1.15 Fields Mixin

class s4.clarity._internal.**FieldsMixin**(*lims*, *uri=None*, *xml_root=None*, *name=None*, *limsid=None*)

fields

Type dict[str, object]

get(*name*, *default=None*)
Get a UDF, if it exists. (Non-exception version of []).

Parameters **default** – returned if the item is not present

Return type str or int or bool or datetime.datetime or float

get_formatted_number_string(*name*, *default=None*)
Get a Numeric UDF formatted to the correct precision, if the UDF exists.

Parameters **default** (*str*) – returned if the item is not present

Return type str

get_raw(*name*, *default=None*)
Get a UDF as a string, if it exists.

Parameters **default** – returned if the item is not present

Return type str

get_udf_config(*name*)
Get the underlying UDF configuration associated with the field :param name: name of the field :rtype: s4.clarity.configuration.Udf

xml_root

Type ETree.Element

2.1.16 File

class s4.clarity.file.**File**(*lims*, *uri=None*, *xml_root=None*, *name=None*, *limsid=None*)
Bases: *s4.clarity.ClarityElement*

This is a file in Clarity. It is also a Python file (more or less). You can read, write, and do everything else you can normally do with a Python file. NOTE: nothing will be committed to Clarity until you call close, or commit.

attached_to
The value of the XML property 'attached-to'

Type str

close()
Commit the file and close the data stream.

commit()
Shorthand for put_and_parse().

content_location
The value of the XML property 'content-location'

Type str

data

Returns The file data IO stream.

Return type io.IOBase

flush()

getvalue()

is_binary_mode

Type bool

is_published
The value of the XML property 'is-published'

Type bool

isatty()

name
The value of the XML property 'original-location'

Type str

classmethod new_empty(*attachment_point_element*, *name=None*)
Create a new empty *File*.

Parameters

- **attachment_point_element** (ClarityElement) – An element to attach the file to.
- **name** (*str*) – A name for the file.

Return type *File*

classmethod new_from_local(*attachment_point_element*, *local_file_path*, *mode='r+b'*)
Create a new *File* from a local file.

Parameters

- **attachment_point_element** (ClarityElement) – An element to attach the file to.
- **local_file_path** (*str*) – Path to the local file.
- **mode** (*str*) – Mode to open the file with.

Return type *File*

pipe_to(*target_file_object*)

Raises **FileNotFoundException** – if the file does not exist in Clarity.

read(*n=-1*)

readable()

readline(*length=None*)

readlines(*sizehint=0*)

replace_and_commit(*stream*, *name*, *content_type='text/plain'*)

replace_and_commit_from_local(*local_file_path*, *content_type='text/plain'*, *mode='r+b'*)

seek(*pos*, *mode=0*)

seek_to_end()

seekable()

tell()

truncate(*size=None*)

writable()

write(*s*)

writelines(*iterable*)

2.1.17 IO Map

class s4.clarity.iomaps.**IOMap**(*inputs*, *outputs*)

Variables

- **inputs** – list[Artifact]
- **outputs** – list[Artifact]

input

Type *Artifact*

Raises **Exception** – If there are multiple input artifacts

output

Type *Artifact*

Raises **Exception** – If there are multiple output artifacts

2.1.18 Lab

class s4.clarity.lab.**Lab**(*lims*, *uri=None*, *xml_root=None*, *name=None*, *limsid=None*)

Bases: *s4.clarity._internal.FieldsMixin*, *s4.clarity.ClarityElement*

2.1.19 LIMS

class s4.clarity.**LIMS**(*root_uri*, *username*, *password*, *dry_run=False*, *insecure=False*, *log_requests=False*, *timeout=None*)

Parameters

- **root_uri** (*str*) – Location of the clarity server e.g. (https://<clarity server>/api/v2/)
- **username** (*str*) – Clarity User Name
- **password** (*str*) – Clarity Password
- **dry_run** (*bool*) – If true, no destructive requests will be made to the Clarity API. Default false.
- **insecure** (*bool*) – Disables SSL validation. Default false.
- **timeout** (*int*) – Number of seconds to wait for connections and for reads from the Clarity API. Default None, which is no timeout.

Variables

- **steps** (ElementFactory) – Factory for *s4.clarity.step.Step*
- **samples** (ElementFactory) – Factory for *s4.clarity.sample.Sample*
- **artifacts** (ElementFactory) – Factory for *s4.clarity.artifact.Artifact*
- **files** (ElementFactory) – Factory for *s4.clarity.file.File*
- **containers** (ElementFactory) – Factory for *s4.clarity.container.Container*
- **projects** (ElementFactory) – Factory for *s4.clarity.project.Project*
- **workflows** (ElementFactory) – Factory for s4.clarity.configuration.workflow.Workflow
- **protocols** (ElementFactory) – Factory for s4.clarity.configuration.protocol.Protocol
- **process_types** (ElementFactory) – Factory for s4.clarity.configuration.process_type.ProcessType

- **process_templates** (ElementFactory) – Factory for s4.clarity.configuration.process_type.ProcessTemplate
- **processes** (ElementFactory) – Factory for *s4.clarity.process.Process*
- **researchers** (ElementFactory) – Factory for *s4.clarity.researcher.Researcher*
- **roles** (ElementFactory) – Factory for *s4.clarity.role.Role*
- **permissions** (ElementFactory) – Factory for *s4.clarity.permission.Permission*

DEFAULT_TIMEOUT = None

artifact(*limsid*)

Return type *Artifact*

artifact_from_uri(*uri*)

Return type *Artifact*

factory_for(*element_type*)

Return type *ElementFactory*

properties

Type dict

raw_request(*method*, *uri*, ***kwargs*)

Raises ***ClarityException*** – if Clarity returns an exception as XML

Return type requests.Response

request(*method*, *uri*, *xml_root=None*)

Return type ETree.Element

Raises ***ClarityException*** – if Clarity returns an exception as XML

sample(*limsid*)

Return type *Sample*

stage_from_uri(*uri*)

Return type *Stage*

step(*limsid*)

Return type *Step*

step_from_uri(*uri*)

Return type *Step*

stepconfiguration_from_uri(*uri*)

Return type *StepConfiguration*

2.1.20 Output Reagent

class s4.clarity.step.**OutputReagent**(*step*, *node*)

reagent_label

Type str

2.1.21 Permission

class s4.clarity.permission.**Permission**(*lims*, *uri=None*, *xml_root=None*, *name=None*, *limsid=None*)

Bases: *s4.clarity.ClarityElement*

action
The value of the XML property 'action'

Type str

description
The value of the XML property 'description'

Type str

2.1.22 Placement

class s4.clarity.step.**Placement**(*lims*, *xml_root=None*)

artifact
The linked *Artifact* from the '.' subnode

Type *Artifact*

container
The linked *Container* from the 'location/container' subnode

Type *Container*

location_value
The value of the XML property 'location/value'

Type str

2.1.23 Pool

class s4.clarity.step.**Pool**(*lims*, *xml_root=None*)

inputs

Type list[*Artifact*]

name

Type str

output

Type *Artifact*

2.1.24 Process

class s4.clarity.process.**Process**(**args*, ***kwargs*)

Bases: s4.clarity.iomaps.IOMapsMixin, *s4.clarity._internal.FieldsMixin*, *s4.clarity.ClarityElement*

Variables

- **iomaps** (*list[IOMap]*) –
- **inputs** (*list[Artifact]*) –
- **outputs** (*list[Artifact]*) –
- **shared_outputs** (*list[Artifact]*) –
- **uri** (*str|None*) –
- **lims** (LIMS) –

commit()

Shorthand for put_and_parse().

get(*name*, *default=None*)

Get a UDF, if it exists. (Non-exception version of []).

Parameters **default** – returned if the item is not present

Return type str or int or bool or datetime.datetime or float

get_formatted_number_string(*name*, *default=None*)

Get a Numeric UDF formatted to the correct precision, if the UDF exists.

Parameters **default** (*str*) – returned if the item is not present

Return type str

get_raw(*name*, *default=None*)

Get a UDF as a string, if it exists.

Parameters **default** – returned if the item is not present

Return type str

get_udf_config(*name*)

Get the underlying UDF configuration associated with the field :param name: name of the field :rtype: s4.clarity.configuration.Udf

invalidate()

Clear the local cache, forcing a reload next time the element is used.

iomaps_input_keyed()

Returns a mapping of input -> outputs.

Return type dict[*Artifact*, list[*Artifact*]]

iomaps_output_keyed()

Returns a mapping of output -> inputs.

Return type dict[*Artifact*, list[*Artifact*]]

refresh()

Retrieve fresh element representation from the API.

fields

Type dict[str, object]

limsid

Type str

process_type

Type *ProcessType*

technician
The linked *Researcher* from the 'technician' subnode

Type *Researcher*

2.1.25 Process Template

class s4.clarity.configuration.**ProcessTemplate**(*lims*, *uri=None*, *xml_root=None*, *name=None*, *limsid=None*)

Bases: *s4.clarity._internal.FieldsMixin*, *s4.clarity.ClarityElement*

is_default
The value of the XML property 'is-default'

Type bool

type
The linked *ProcessType* from the 'type' subnode

Type *ProcessType*

2.1.26 Process Type

class s4.clarity.configuration.**ProcessType**(*lims*, *uri=None*, *xml_root=None*, *name=None*, *limsid=None*)

Bases: *s4.clarity.ClarityElement*

get_parameter(*parameter_name*)

Parameters **parameter_name** (*str*) – the name of the parameter to retrieve

Return type dict

inputs
Retrieves the value of the property 'process-input'

Type list[dict]

outputs
Retrieves the value of the property 'process-output'

Type list[dict]

parameters
Retrieves the value of the property 'parameter'

Type list[dict]

2.1.27 Project

class s4.clarity.project.**Project**(*lims*, *uri=None*, *xml_root=None*, *name=None*, *limsid=None*)

Bases: *s4.clarity._internal.FieldsMixin*, *s4.clarity.ClarityElement*

close_date

The value of the XML property 'close-date'

Type datetime.date

invoice_date

The value of the XML property 'invoice-date'

Type datetime.date

open_date

The value of the XML property 'open-date'

Type datetime.date

researcher

The linked *Researcher* from the 'researcher' subnode

Type *Researcher*

2.1.28 Protocol

class s4.clarity.configuration.**Protocol**(*lims*, *uri=None*, *xml_root=None*, *name=None*, *limsid=None*)

Bases: *s4.clarity.ClarityElement*

index

The value of the XML property 'index'

Type number

number_of_steps

Type int

properties

The value of the XML property 'protocol-properties'

Type str

step(*name*)

Return type *StepConfiguration* or None

step_from_id(*stepid*)

Return type *StepConfiguration* or None

steps

Type list[*StepConfiguration*]

2.1.29 Protocol Step Field

class s4.clarity.configuration.**ProtocolStepField**(*lims*, *xml_root=None*)

attach_to
The value of the XML property 'attach-to'

Type str

name
The value of the XML property 'name'

Type str

style
The value of the XML property 'style'

Type str

2.1.30 Queue

class s4.clarity.queue.**Queue**(*lims*, *uri=None*, *xml_root=None*, *name=None*, *limsid=None*)
Bases: *s4.clarity.ClarityElement*

query(*prefetch=True*, ***params*)

queued_artifacts

2.1.31 Reagent Kit

class s4.clarity.reagent_kit.**ReagentKit**(*lims*, *uri=None*, *xml_root=None*, *name=None*, *limsid=None*)
Bases: *s4.clarity.ClarityElement*

archived
The value of the XML property 'archived'

Type bool

catalogue_number
The value of the XML property 'catalogue-number'

Type str

name
The value of the XML property 'name'

Type str

related_reagent_lots

supplier
The value of the XML property 'lot-supplier'

Type str

website
The value of the XML property 'website'

Type str

2.1.32 Reagent Lot

class s4.clarity.reagent_lot.**ReagentLot**(*lims*, *uri=None*, *xml_root=None*, *name=None*, *limsid=None*)

Bases: *s4.clarity.ClarityElement*

created_date
The value of the XML property 'created-date'

Type datetime.date

expiry_date
The value of the XML property 'expiry-date'

Type datetime.date

last_modified_date
The value of the XML property 'last-modified-date'

Type datetime.date

lot_number
The value of the XML property 'lot-number'

Type str

name
The value of the XML property 'name'

Type str

notes
The value of the XML property 'notes'

Type str

reagent_kit
The linked *ReagentKit* from the 'reagent-kit' subnode

Type *ReagentKit*

status
The value of the XML property 'status'

Type str

storage_location
The value of the XML property 'storage-location'

Type str

usage_count
The value of the XML property 'usage-count'

Type number

2.1.33 Reagent Type

class s4.clarity.reagent_type.**ReagentType**(*lims*, *uri=None*, *xml_root=None*, *name=None*, *limsid=None*)

Bases: *s4.clarity.ClarityElement*

attributes
The value of the XML property 'special-type'

Type str

name
The name of the element instance in Clarity.

Type str

reagent_category
The value of the XML property 'reagent-category'

Type str

special_type

Type str

2.1.34 Researcher

class s4.clarity.researcher.**Researcher**(*lims*, *uri=None*, *xml_root=None*, *name=None*, *limsid=None*)
Bases: *s4.clarity._internal.FieldsMixin*, *s4.clarity.ClarityElement*

add_role(*new_role*)

email
The value of the XML property 'email'

Type str

first_name
The value of the XML property 'first-name'

Type str

initials
The value of the XML property 'initials'

Type str

lab
The linked *Lab* from the 'lab' subnode

Type *Lab*

last_name
The value of the XML property 'last-name'

Type str

password
The value of the XML property 'credentials/password'

Type str

remove_role(*role*)

roles
The linked *Role* objects from the 'credentials/role' subnodes

Type list[*Role*]

username
The value of the XML property 'credentials/username'

Type str

2.1.35 Role

class s4.clarity.role.**Role**(*lims*, *uri=None*, *xml_root=None*, *name=None*, *limsid=None*)
Bases: *s4.clarity.ClarityElement*

permissions
The linked *Permission* objects from the 'permissions/permission' subnodes

Type list[*Permission*]

researchers

Type list[*Researcher*]

2.1.36 Router

class s4.clarity.routing.**Router**(*lims*)
Class allowing routing of multiple artifacts to a given workflow or stage

assign(*workflow_or_stage_uri*, *artifact_or_artifacts*)
Stages an artifact or multiple artifacts to be assigned to a workflow_or_stage_uri.

Parameters **workflow_or_stage_uri** (*str*) – The uri of either a workflow or workflow-stage. If a workflow uri is provided, the artifacts will be queued to the first stage. Otherwise, they will be queued to the specific workflow-stage.

clear()
Clears the routing node and the routing dict.

commit()
Generates the routing XML for workflow/stage assignment/unassignment and posts it.

remove(*artifact_or_artifacts*)
Remove given artifact or artifacts from the routing dict. No error is raised if the artifact is not found.

unassign(*workflow_or_stage_uri*, *artifact_or_artifacts*)
Stages an artifact or multiple artifacts to be unassigned from a workflow_or_stage_uri.

Parameters **workflow_or_stage_uri** (*str*) – The uri of either a workflow or workflow-stage. If a workflow uri is provided, the artifacts will be removed from any stages of that workflow. Otherwise, they will be removed from the specified workflow stage.

2.1.37 Sample

class s4.clarity.sample.**Sample**(*lims*, *uri=None*, *xml_root=None*, *name=None*, *limsid=None*)
Bases: *s4.clarity._internal.FieldsMixin*, *s4.clarity.ClarityElement*

artifact
The linked *Artifact* from the 'artifact' subnode

Type *Artifact*

date_completed
The value of the XML property 'date-completed'

Type datetime.date

date_received
The value of the XML property 'date-received'

Type datetime.date

is_control

Type bool

project
The linked *Project* from the 'project' subnode

Type *Project*

set_location(*container*, *row*, *col*)
Sets this artifact's location (usually for sample creation) with the given row and column, in the given container.

Parameters

- **container** (*s4.clarity.Container*) – The Sample's container
- **row** (*str or int*) – The well position row.
- **col** (*str or int*) – The well position column

Deprecated Use set_location_coords or set_location_well

set_location_coords(*container*, *row*, *col*)
Sets this artifact's location (usually for sample creation) with the given row and column, in the given container.

Equivalent to set_location_well with the string "<row>:<col>".

Parameters

- **container** (*s4.clarity.Container*) – The Sample's container
- **row** (*str or int*) – The well position row.
- **col** (*str or int*) – The well position column

set_location_well(*container*, *well*)
" Sets this artifact's location (usually for sample creation) with the given well location, in the given container.

Parameters

- **container** (*s4.clarity.Container*) – The Sample's container
- **well** (*str*) – The well position in the form "<row>:<col>"

2.1.38 Stage

class s4.clarity.configuration.**Stage**(*lims*, *uri=None*, *xml_root=None*, *name=None*, *limsid=None*)
Bases: *s4.clarity.ClarityElement*

enqueue(*artifact_or_artifacts*)
Add one or more artifacts to the stage's queue

Parameters **artifact_or_artifacts** (*s4.clarity.Artifact | list[s4.clarity.Artifact]*) – The artifact(s) to enqueue

index
The value of the XML property 'index'

Type str

protocol

Type *Protocol*

remove(*artifact_or_artifacts*)
Remove one or more sample artifacts from the stage

Parameters **artifact_or_artifacts** (*s4.clarity.Artifact | list[s4.clarity.Artifact]*) – The artifact(s) to enqueue

step

Type *Step*

workflow

Type *Workflow*

2.1.39 Step

class s4.clarity.step.**Step**(*lims*, *uri=None*, *xml_root=None*, *name=None*, *limsid=None*)
Bases: *s4.clarity.ClarityElement*

actions

Type *StepActions*

advance()
Advances the current step to the next screen.

automatic_next_step

Type *Step*

available_programs

Type *StepTrigger*

configuration

Type *StepConfiguration*

current_state

Type str

date_completed
The value of the XML property 'date-completed'

Type datetime

date_started
The value of the XML property 'date-started'

Type datetime

details

Type *StepDetails*

fields

Raises **NotImplementedError** – Steps don't have fields. Use step.details.

name
The value of the XML property 'configuration'

Type str

open_resultfile(*name*, *mode*, *only_write_locally=False*, *limsid=None*)

Return type s4.clarity.File

placements

Type *StepPlacements*

pooling

Type *StepPools*

process

Type *Process*

program_status

Type *StepProgramStatus*

reagent_lots

Type *StepReagentLots*

reagents

Type *StepReagents*

wait_for_epp()

Polls Clarity, blocking until the currently running EPP is done.

Raises **EppException** – When EPP execution fails.

Returns Zero

Return type int

2.1.40 Step Actions

class s4.clarity.step.**StepActions**(*lims*, *uri=None*, *xml_root=None*, *name=None*, *limsid=None*)

Bases: *s4.clarity.ClarityElement*

all_next_step(*step_uri=None*)

Set all artifacts actions to either next step, or mark protocol as complete.

If step_uri is not provided, artifacts will:

- Continue to the first transition defined for this step, if any transitions are defined.
- Mark the protocol as complete, if there are no transitions.

artifact_actions

A dictionary of ArtifactActions for this step, keyed by artifact.

Return type dict[*Artifact*, *ArtifactAction*]

escalated_artifacts

The linked *Artifact* objects from the './escalation/escalated-artifacts/escalated-artifact' subnodes

Type list[*Artifact*]

escalation_author

The linked *Researcher* from the './escalation/request/author' subnode

Type *Researcher*

escalation_date
The value of the XML property './escalation/request/date'

Type datetime

escalation_reviewer
The linked *Researcher* from the './escalation/request/reviewer' subnode

Type *Researcher*

next_actions
Retrieves the value of the property 'next-actions/next-action'

Type list[dict]

step
The linked *Step* from the 'step' subnode

Type *Step*

2.1.41 Step Configuration

class s4.clarity.configuration.**StepConfiguration**(*protocol*, *node*)
Bases: *s4.clarity.ClarityElement*

permitted_control_types

Type *ControlType*

post_and_parse(*alternate_uri=None*)
POST the current state of this object to a REST endpoint, then parse the response into this object.

Parameters **alternate_uri** (*str*) – Will be used instead of self.uri if provided.

Raises

- **requests.exceptions.HTTPError** – If there are communication problems.
- ***ClarityException*** – If Clarity returns an exception as XML.

process_type

Type *ProcessType*

properties
The value of the XML property 'step-properties'

Type str

protocol_step_index
The value of the XML property 'protocol-step-index'

Type number

put_and_parse(*alternate_uri=None*)
PUT the current state of this object to a REST endpoint, then parse the response into this object.

Parameters **alternate_uri** – Will be used instead of self.uri if provided.

Raises

- **requests.exceptions.HTTPError** – If there are communication problems.
- ***ClarityException*** – If Clarity returns an exception as XML.

queue

Type *Queue*

queue_fields
ProtocolStepField items from the 'queue-field' subnodes

Type list[*ProtocolStepField*]

refresh()

Raises **Exception** – Unable to refresh step directly, use protocol

required_reagent_kits

Type *ReagentKit*

sample_fields
ProtocolStepField items from the 'sample-field' subnodes

Type list[*ProtocolStepField*]

step_fields
ProtocolStepField items from the 'step-field' subnodes

Type list[*ProtocolStepField*]

transitions
Retrieves the value of the property 'transitions/transition'

Type list[dict]

triggers
Retrieves the value of the property 'epp-triggers/epp-trigger'

Type list[dict]

2.1.42 Step Details

class s4.clarity.step.**StepDetails**(*step*, **args*, ***kwargs*)
Bases: s4.clarity.iomaps.IOMapsMixin, *s4.clarity._internal.FieldsMixin*, *s4.clarity.ClarityElement*

Variables

- **step** (Step) –
- **iomaps** (*list[IOMap]*) –
- **inputs** (*list[Artifact]*) –
- **outputs** (*list[Artifact]*) –
- **shared_outputs** (*list[Artifact]*) –
- **uri** (*str|None*) –
- **lims** (LIMS) –

commit()
Shorthand for put_and_parse().

get(*name*, *default=None*)
Get a UDF, if it exists. (Non-exception version of []).

Parameters **default** – returned if the item is not present

Return type str or int or bool or datetime.datetime or float

get_formatted_number_string(*name*, *default=None*)
Get a Numeric UDF formatted to the correct precision, if the UDF exists.

Parameters **default** (*str*) – returned if the item is not present

Return type str

get_raw(*name*, *default=None*)
Get a UDF as a string, if it exists.

Parameters **default** – returned if the item is not present

Return type str

get_udf_config(*name*)
Get the underlying UDF configuration associated with the field :param name: name of the field :rtype: s4.clarity.configuration.Udf

invalidate()
Clear the local cache, forcing a reload next time the element is used.

iomaps_input_keyed()

Returns a mapping of input -> outputs.

Return type dict[*Artifact*, list[*Artifact*]]

iomaps_output_keyed()

Returns a mapping of output -> inputs.

Return type dict[*Artifact*, list[*Artifact*]]

refresh()
Retrieve fresh element representation from the API.

fields

Type dict[str, object]

limsid

Type str

name
The value of the XML property 'configuration'

Type str

sorted_input_sample_iomaps()
return list of sample iomaps sorted by sample name

2.1.43 Step Factory

class s4.clarity.**StepFactory**(*lims*, *element_class*, *batch_flags=None*, *request_path=None*, *name_attribute='name'*)
Bases: *s4.clarity.ElementFactory*

Parameters

- **request_path** (*str*) – for example, '/configuration/workflows'. when not specified, uses '/<plural of element name>'.

- **name_attribute** (*str*) – if not "name", provide this to adjust behaviour of 'get_by_name'.

from_link_node(*xml_node*)
Override so that we can accept a process link or a step link. Requires that node include a limsid

Return type *ClarityElement*

get_by_name(*protocol_name*, *step_name*)
Queries for a ClarityElement that is described by the unique name. An exception is raised if there is no match or more than one match.

Raises

- **NoMatchingElement** – if no match
- **MultipleMatchingElements** – if multiple matches

2.1.44 Step Placements

class s4.clarity.step.**StepPlacements**(*lims*, *uri=None*, *xml_root=None*, *name=None*, *limsid=None*)
Bases: *s4.clarity.ClarityElement*

add_selected_container(*new_container*)
Adds a new container to the step placement's list of selected containers.

clear_placements()
Clears all previous artifact placements recorded to this step. This is often called before starting automated placement to ensure that artifacts are not placed twice.

clear_selected_containers()
Clears the list of selected containers associated with the step. This can be used to remove containers that are automatically created for the step when they are not used.

commit()
Push placement changes back to Clarity.

create_placement(*artifact*, *container*, *well_string*)
Place the provided artifact, in the provided container at the location described by the well_string.

Parameters

- **artifact** – The artifact to place.
- **container** – The container that will hold the artifact.
- **well_string** – The location on the plate to place the artifact

create_placement_with_no_location(*artifact*)
Samples that are part of a process, but have been removed need to be included with out a location in some cases because Clarity will hold the old spot, which may now be used by another sample.

placements
Placement items from the 'output-placement' subnodes

Type list[*Placement*]

selected_containers

Type list[*Container*]

step
The linked *Step* from the 'step' subnode

Type *Step*

2.1.45 Step Pools

class s4.clarity.step.**StepPools**(*step*, *lims*, *uri*)
Bases: *s4.clarity.ClarityElement*

available_inputs

Type list[*AvailableInput*]

create_pool(*name*, *samples*)

pools

Type list[*Pool*]

2.1.46 Step Program Status

class s4.clarity.step.**StepProgramStatus**(*lims*, *uri=None*, *xml_root=None*, *name=None*, *limsid=None*)
Bases: *s4.clarity.ClarityElement*

Manage the status of the currently executing step. By setting a message to the step status a message box will be displayed to the user at the end of the step.

This will only work in the End of Step EPP script. Calling it from any other event hook will be rejected by Clarity.

The AI node will set the status to RUNNING, but this is not allow the API to set this value.

message
The value of the XML property 'message'

Type str

report_error(*message*)

report_ok(*message*)

report_warning(*message*)

status
The value of the XML property 'status'

Type str

step
The linked *Step* from the 'step' subnode

Type *Step*

2.1.47 Step Reagents

class s4.clarity.step.**StepReagents**(*step*, *lims*, *uri*)
Bases: *s4.clarity.ClarityElement*

output_reagents

Type list(*OutputReagent*)

reagent_category
The value of the XML property 'reagent-category'

Type str

2.1.48 Step Reagent Lots

class s4.clarity.step.**StepReagentLots**(*step*, *lims*, *uri*)
Bases: *s4.clarity.ClarityElement*

add_reagent_lots(*elements*)

Parameters **elements** (*list[ReagentLot]*) – Add reagent lots to the step

reagent_lots

Type list(*ReagentLot*)

2.1.49 Step Trigger

class s4.clarity.step.**StepTrigger**(*step*, *uri*)

fire()

name

Type str

2.1.50 UDF

class s4.clarity.configuration.**Udf**(*lims*, *uri=None*, *xml_root=None*, *name=None*, *limsid=None*)
Bases: *s4.clarity.ClarityElement*

add_preset(*new_preset_value*)
Add a new preset value to the end of the list. Ignores values that are already present.

Parameters **new_preset_value** (*str|unicode|int|float|datetime.date|bool*) – the preset value to add, with a type appropriate to the UDF. The value is not validated to be the correct type.

allow_non_preset_values
The value of the XML property 'allow-non-preset-values'

Type bool

attach_to_category
The value of the XML property 'attach-to-category'

Type str

attach_to_name
The value of the XML property 'attach-to-name'

Type str

field_type
The value of the XML property 'type'

Type str

first_preset_is_default_value
The value of the XML property 'first-preset-is-default-value'

Type bool

is_controlled_vocabulary
The value of the XML property 'is-controlled-vocabulary'

Type bool

is_deviation
The value of the XML property 'is-deviation'

Type bool

is_editable
The value of the XML property 'is-editable'

Type bool

is_required
The value of the XML property 'is-required'

Type bool

max_value
The value of the XML property 'max-value'

Type number

min_value
The value of the XML property 'min-value'

Type number

precision
The value of the XML property 'precision'

Type number

presets

Type list

remove_preset(*preset_value*)
Remove a preset value from the list.

Parameters **preset_value** (*str|unicode|int|float|datetime.date|bool*) – the preset value to remove, with a type appropriate to the UDF. The value is not validated to be the correct type.

set_default_preset(*default_preset_value*)
Sets a preset value as the default (puts first in the list). Adds value if it isn't already preset.

Parameters **default_preset_value** (*str|unicode|int|float|datetime.date|bool*) – the new default preset value, with a type appropriate to the UDF. The value is not validated to be the correct type.

Raises **Exception** – if the udf's first-preset-is-default property is currently false

show_in_lablink
The value of the XML property 'show-in-lablink'

Type bool

show_in_tables
The value of the XML property 'show-in-tables'

Type bool

2.1.51 UDF Factory

class s4.clarity.**UdfFactory**(*lims*, *element_class*, *batch_flags=None*, *request_path=None*, *name_attribute='name'*)

Bases: *s4.clarity.ElementFactory*

Parameters

- **request_path** (*str*) – for example, '/configuration/workflows'. when not specified, uses '/<plural of element name>'.
- **name_attribute** (*str*) – if not "name", provide this to adjust behaviour of 'get_by_name'.

get_by_name(*udf_name*, *attach_to_key*)

Parameters **attach_to_key** (*(str, str)*) – tuple comprising the element's attach-to-name and attach-to-category properties

Returns s4.clarity.configuration.Udf

Raises **NoMatchingElement** – if no match

2.1.52 Workflow

class s4.clarity.configuration.**Workflow**(*lims*, *uri=None*, *xml_root=None*, *name=None*, *limsid=None*)

Bases: *s4.clarity.ClarityElement*

ACTIVE_STATUS = 'ACTIVE'

ARCHIVED_STATUS = 'ARCHIVED'

PENDING_STATUS = 'PENDING'

enqueue(*artifact_or_artifacts*)
Add one or more artifacts to the start of the workflow

Type artifact_or_artifacts: Artifact | list[*Artifact*]

is_active

Type bool

is_archived

Type bool

is_pending

Type bool

protocols

Param prefetch: set to False if you don't want an automatic batch_get.

Type prefetch: bool

Return type list[*Protocol*]

remove(*artifact_or_artifacts*)
Remove one or more artifacts from the workflow

Type artifact_or_artifacts: Artifact | list[*Artifact*]

stage_from_id(*stageid*)

Return type *Stage* or None

stages

Return type list[*Stage*]

status
The value of the XML property 'status'

Type str

2.1.53 Workflow Stage History

class s4.clarity.artifact.**WorkflowStageHistory**(*lims*, *xml_root=None*)

name
The value of the XML property 'name'

Type str

stage
The linked *Stage* from the '.' subnode

Type *Stage*

status
The value of the XML property 'status'

Type str

uri
The value of the XML property 'uri'

Type str

2.2 Scripts

Listed here are the definitions of the various base classes of scripts.

2.2.1 Derived Sample Automation Script

class s4.clarity.scripts.**DerivedSampleAutomation**(*options*)
Bases: *s4.clarity.scripts.GenericScript*

A script run from the Project Dashboard screen.

Variables

- **lims** (LIMS) – The Clarity object to perform operations against.
- **artifacts** (*list[Artifact]*) – The list of Artifacts that the script applies to, loaded from the provided LIMS Ids.

Parameters **options** (*map*) – A map of the values of the supplied command line arguments. The default keys available are: *username*, *password*, *api_uri*, and *derived_sample_ids*.

Usage:

Implement process_derived_samples(), which must return a string to display success status to the user.

Optionally: add_arguments(argparser) # To add more arguments. Don't forget to call super.

Add to the end of your file:

if __name__ == '__main__': YourSubClass.main()

Example Clarity automation string. Contains an example of additional user input that would require an override of add_arguments to add the -x arg. *Note* that all userinput args are strings:

```
python <script_name>.py -u {username} -p {password} -a
'{baseURI}v2' -d {derivedSampleLuids} -x {userinput:input_x}
```

classmethod add_arguments(*argparser*)

Configures the Argument Parser to read command line input and store it in the self.options object.

This method can be overrode in sub classes to provide extra arguments specific to the script being written. When this is done it is important to call the parent class add_arguemnts methods so that all arguments are included.

Parameters **parser** (*argparse.ArgumentParser*) – The ArgumentParser that will be used to process the command line.

process_derived_samples()

Implement this to perform the work required. Method *must* return a summary success string, as it's used to display the user-facing message on script completion.

Returns Message to report success to the user

Return type str

Raise Exception to report failure

run()

The run method is called once the arguments are parsed and the logging is started. It will return an exit code indicating the success or failure of the process to run. Exit Codes: http://www.tldp.org/LDP/abs/html/exitcodes.html

Returns The Exit code

2.2.2 Generic Script

class s4.clarity.scripts.**GenericScript**(*options*)

GenericScript is the base abstract class for all of our script classes. The class provides common logging, argument parsing and entry point for all other script types.

The GenericScript provides the main() method as an entry point for all scripts. Because your implementing class will be the main entry point for the program it will need to call the main method.

The following code will make sure that the script is only run _if_ the file is the main python entry point:

if __name__ == '__main__': YourSubClass.main()

For more information on this see:

https://docs.python.org/3/library/__main__.html https://stackoverflow.com/questions/419163/what-does-if-name-main-do

Command line arguments can be added by overriding the add_arguments method. Remember to call the parent implementation of add_arguments so that all arguments are included.

Example method adding an extra command line argument:

```
@classmethod
def add_arguments(cls, argparser):
    super(YourSubClass, cls).add_arguments(argparser)
    argparser.add_argument(
        '-t', '--thisparam', type=str, help='Something helpful', required=True
    )
```

The values will be available in the self.options object.

An example of printing the value passed in as –thisparam print self.options.thisparam

This class works with the python logging system to gather log information and save it to a file. The files can be stored as html or plain text.

To use this logging system in your own files declare the logging object at the top of each file:

log = logging.getLogger(__name__)

The log object can then be used to log information that the GenericScript will save.

log.info("Low priority info to log.") log.warning("Warnings to log") log.error("Error conditions logged")

To add functionality to a class derived from GenericScript the run() method must be overrode in the child class.

Creates a new instance of this class and saves the command line options

Parameters **options** (*argparse.Namespace*) – The parsed command line options

TEXT_LOG_FORMAT = '%(asctime)s\t%(levelname)s\t%(name)s: %(message)s'

classmethod add_arguments(*parser*)

Configures the Argument Parser to read command line input and store it in the self.options object.

This method can be overrode in sub classes to provide extra arguments specific to the script being written. When this is done it is important to call the parent class add_arguemnts methods so that all arguments are included.

Parameters **parser** (*argparse.ArgumentParser*) – The ArgumentParser that will be used to process the command line.

final_summary = ''

static loggingpreamble(*obfuscated_options*)

classmethod main()

The entry point for all scripts. This method will exit() the program upon completion.

open_log_output_stream()

override this for more complicated logging output.

Return type io.IOBase

run(**args*)

The run method is called once the arguments are parsed and the logging is started. It will return an exit code indicating the success or failure of the process to run. Exit Codes: http://www.tldp.org/LDP/abs/html/exitcodes.html

Returns The Exit code

2.2.3 Shell Script

class s4.clarity.scripts.**ShellScript**(*options*)
Bases: *s4.clarity.scripts.GenericScript*

ShellScript provides the framework for a basic shell script that will communicate with Clarity. It provides all of the functionality of a GenericScript with the addition of a LIMS object.

Creates a new instance of this class and saves the command line options

Parameters **options** (*argparse.Namespace*) – The parsed command line options

classmethod add_arguments(*parser*)
Add command line arguments to be parsed.

Parameters **parser** (*argparse.ArgumentParser*) – The ArgumentParser that will be used to process the command line.

2.2.4 StepEpp

class s4.clarity.scripts.**StepEPP**(*options*)
Bases: *s4.clarity.scripts.GenericScript*

This class forms the base of the scripting for Step EPP scripts run from Clarity. It will provide access to a LIMS object as well as the Step data for the currently running step.

If the log file name that gets passed in over the command line is the limsId clarity assigned to an file it will be automatically uploaded at the end of the step.

Variables

- **step** (s4.clarity.step.Step) – The Clarity Step to execute the script against.
- **lims** (LIMS) – The Clarity LIMS object to use for API operations.

Creates a new StepEPP object and initializes the local LIMS and Step objects.

Parameters **options** – Parsed command line options

PREFETCH_INPUTS = 'inputs'

PREFETCH_OUTPUTS = 'outputs'

PREFETCH_SAMPLES = 'samples'

classmethod add_arguments(*argparser*)
Configures the Argument Parser to read command line input and store it in the self.options object.

This method can be overrode in sub classes to provide extra arguments specific to the script being written. When this is done it is important to call the parent class add_arguemnts methods so that all arguments are included.

Parameters **parser** (*argparse.ArgumentParser*) – The ArgumentParser that will be used to process the command line.

static display(**args*)

inputs
Shortcut for self.step.details.inputs.

Return type list[*Artifact*]

iomaps
Shortcut for self.step.details.iomaps.

Return type list[*IOMap*]

open_log_output_stream()
Use step's logfile.

outputs
Shortcut for self.step.details.outputs.

Return type list[*Artifact*]

prefetch(**categories*)
Fetch values for input artifacts, output artifacts, or samples in a batch.

Input and output samples are always an identical set, so 'samples' will fetch both.

Note: when only samples are selected, input artifacts will also be fetched. To change this behaviour, supply both 'outputs' and 'samples' in the categories list.

Parameters **categories** (*str|list[str]*) – List of any number of the strings: 'inputs', 'outputs', 'samples'.

Returns a list of all fetched objects

Return type list[*ClarityElement*]

2.2.5 TriggeredStepEpp

class s4.clarity.scripts.**TriggeredStepEPP**(*options*)
Bases: *s4.clarity.scripts.StepEPP*

TriggeredStepEPP acts as an EPP with multiple entry points, to allow the developer to group all scripts associated with a step together in a single file. A script implementing this class is intended to be called multiple times in the same step at different stages, with the Action parameter determining which method is called.

Choices for the Action parameter are automatically generated from all class methods starting with `on_`. The Action value is generated by taking the method name, trimming the `on_`, and transforming the rest of the name to Pascal-case. For example, an EPP String containing `-a TestThing` would attempt to execute a method named `on_test_thing`.

Usage: `python <script.py> -u {username} -p {password} --step-uri {stepURI:v2} -l {compoundOutputFileLuid#} -a <Action>`

Our suggested implementation when creating method names is to mirror Clarity's language for scripts triggered on step transitions, and having button-triggered scripts follow the button label, as shown here:

Action Parameter	EPP Method Name
BeginningOfStep	on_beginning_of_step
EndOfStep	on_end_of_step
PlacementEnter	on_placement_enter
PlacementExit	on_placement_exit
PoolingEnter	on_pooling_enter
PoolingExit	on_pooling_exit
AddReagentsEnter	on_add_reagents_enter
AddReagentsExit	on_add_reagents_exit
RecordDetailsEnter	on_record_details_enter
RecordDetailsExit	on_record_details_exit
CalculateQc	on_calculate_qc (Record Details buttons example)

Ultimately, though, as long as the on_ rules for method naming is followed, the pattern used in your implementation is up to you.

Creates a new StepEPP object and initializes the local LIMS and Step objects.

Parameters **options** – Parsed command line options

classmethod add_arguments(*argparser*)
Configures the Argument Parser to read command line input and store it in the self.options object.

This method can be overrode in sub classes to provide extra arguments specific to the script being written. When this is done it is important to call the parent class add_arguemnts methods so that all arguments are included.

Parameters **parser** (*argparse.ArgumentParser*) – The ArgumentParser that will be used to process the command line.

classmethod add_triggered_step_actions()

run()
The run method is called once the arguments are parsed and the logging is started. It will return an exit code indicating the success or failure of the process to run. Exit Codes: http://www.tldp.org/LDP/abs/html/exitcodes.html

Returns The Exit code

triggered_step_actions = {}

2.2.6 UserMessageException

class s4.clarity.scripts.**UserMessageException**
Bases: Exception

Stops the currently running EPP and displays a message box to the user. The message box is like other exceptions but will not display a stack trace.

2.2.7 Workflow Test

class s4.clarity.scripts.**WorkflowTest**(*options*)
Bases: *s4.clarity.scripts.ShellScript*

Creates a new instance of this class and saves the command line options

Parameters **options** (*argparse.Namespace*) – The parsed command line options

classmethod add_arguments(*argparser*)
Add command line arguments to be parsed.

Parameters **parser** (*argparse.ArgumentParser*) – The ArgumentParser that will be used to process the command line.

2.3 Utils

2.3.1 Artifact Ancestry

s4.clarity.utils.artifact_ancestry.**get_parent_artifacts**(*lims*, *artifacts*)
Helper method to get the parent artifacts keyed to the supplied artifacts

Parameters

- **lims** (LIMS) –
- **artifacts** (*list[Artifact]*) – The artifacts to get parent artifacts for

Return type dict[*Artifact*, list[*Artifact*]]

s4.clarity.utils.artifact_ancestry.**get_udfs_from_artifacts_or_ancestors**(*lims*, *artifacts_to_get_udf_from*, *required_udfs=None*, *optional_udfs=None*)

Walks the genealogy for each artifact in the artifacts_to_get_udf_from list and gets the value for udf_name from the supplied artifact, or its first available ancestor that has a value for the UDF. NOTE: The method will stop the search upon reaching any pooling step.

Parameters

- **lims** (LIMS) –
- **artifacts_to_get_udf_from** (*list[Artifact]*) – the list of artifacts whose ancestors should be inspected for the udf. Passed down recursively until all artifacts have been satisfied.
- **required_udfs** (*list[str]*) – The list of UDFs that *must* be found. Exception will be raised otherwise.
- **optional_udfs** (*list[str]*) – The list of UDFs that *can* be found, but do not need to be.

Return type dict[s4.clarity.Artifact, dict[str, str]]

Raises ***UserMessageException*** – if values can not be retrieved for all required_udfs for all of the provided artifacts

2.3.2 Dates

s4.clarity.utils.**date_to_str**(*dt*)

Return type str

s4.clarity.utils.**datetime_to_str**(*dt*)

Return type str

s4.clarity.utils.**str_to_date**(*string*)

Return type datetime.date

s4.clarity.utils.**str_to_datetime**(*string*)

Return type datetime

2.3.3 Sorting

s4.clarity.utils.**standard_sort_key**(*s*)

2.4 Step Utils

2.4.1 Actions

s4.clarity.steputils.actions.**get_artifact_workflow_stages_for_current_step**(*step*, *artifact*)

s4.clarity.steputils.actions.**get_current_workflow_stages**(*step*, *artifacts*)

Given artifacts in a currently running step, finds their current workflow stages.

Returns a dict mapping workflow stages to lists of artifacts which are currently in them.

Return type dict[*Stage*, list[*Artifact*]]

s4.clarity.steputils.actions.**route_to_next_protocol**(*step*, *artifacts_to_route*)

Queues the given artifacts directly to the first step of the next protocol. NOTE: Artifacts *must* be in-progress in the current step, or an exception will be thrown.

s4.clarity.steputils.actions.**route_to_stage_by_name**(*step*, *artifacts_to_route*, *target_stage_name*, *name_matches_base_name=<function <lambda>>*)

Queues the given artifacts to the first stage in the artifact's workflow with the given name. NOTE: Artifacts *must* be in-progress in the current step, or an exception will be thrown. Optionally takes a name comparison function to use. Defaults to exact name matching.

s4.clarity.steputils.actions.**set_next_actions**(*epp*, *default_action=None*, *controls_action=None*, *failed_qc_action=None*, *action_func=None*)

Parameters

- **failed_qc_action** – applied to any sample or control that has failed qc.
- **controls_action** – applied to controls. if this is a qc step, only controls which have passed.
- **action_func** (*(s4.clarity.Artifact) -> str*) – called with each artifact; must return an action (string). if either failed_qc_action or controls_action are set, and also action_func is set, action_func will only be called for artifacts which are not caught by those actions. if action_func is None, or returns None for an artifact, the default is used.
- **default_action** – if None, an appropriate action is calculated (e.g. next step, or complete protocol.)

2.4.2 Copy UDFs

s4.clarity.steputils.copyudfs.**copy_from_input_to_output**(*step*, *udf_names*)

Copies a set of UDFs from the inputs of a step to its outputs. - Supply a list of UDFs if the source and destination names are the same. - Supply a dictionary (source name to destination name) if they differ.

If the UDF is not present on the source, it is skipped.

Will throw an exception if there are more than one input per output.

s4.clarity.steputils.copyudfs.**copy_from_input_to_sample**(*step*, *udf_names*)
Copies a set of UDFs from the inputs of a step to each input's sample. - Supply a list of UDFs if the source and destination names are the same. - Supply a dictionary (source name to destination name) if they differ.

If the UDF is not present on the source, it is skipped.

s4.clarity.steputils.copyudfs.**copy_from_output_to_input**(*step*, *udf_names*)
Copies a set of UDFs from the outputs of a step to its inputs, one to one. - Supply a list of UDFs if the source and destination names are the same. - Supply a dictionary (source name to destination name) if they differ.

If the UDF is not present on the source, it is skipped.

Will throw an exception if there are more than one input per output or more than one output per input.

s4.clarity.steputils.copyudfs.**copy_from_output_to_sample**(*step*, *udf_names*)
Copies a set of UDFs from the outputs of a step to each output's sample. - Supply a list of UDFs if the source and destination names are the same. - Supply a dictionary (source name to destination name) if they differ.

If the UDF is not present on the source, it is skipped.

s4.clarity.steputils.copyudfs.**copy_from_sample_to_input**(*step*, *udf_names*)
Copies a set of UDFs to the inputs of a step from each input's sample. - Supply a list of UDFs if the source and destination names are the same. - Supply a dictionary (source name to destination name) if they differ.

If the UDF is not present on the source, it is skipped.

s4.clarity.steputils.copyudfs.**copy_from_sample_to_output**(*step*, *udf_names*)
Copies a set of UDFs to the outputs of a step from each output's sample. - Supply a list of UDFs if the source and destination names are the same. - Supply a dictionary (source name to destination name) if they differ.

If the UDF is not present on the source, it is skipped.

2.4.3 Files

s4.clarity.steputils.file_utils.**save_file_to_path**(*step*, *target_directory*, *limsid*, *file_name=None*)
Saves step output files to the specified directory.

Parameters

- **step** – The step which contains the files to save.
- **target_directory** – The directory path to save files to.
- **limsid** – The lims id of the file to save.
- **file_name** – Optional file name to use when saving the file to disk. It allows renaming on save.

2.4.4 Placements

s4.clarity.steputils.placement_utils.**auto_place_artifacts**(*step*, *artifacts*, *order='row'*)
Places the artifacts provided, in the order provided, to selected_containers in the step.

s4.clarity.steputils.placement_utils.**column_order_sort_keys**(*artifact*)
Provide container position sort-keys for the sorted function based on column-major order

Usage example:

```
sorted_outputs = sorted(self.outputs, key=row_order_sort_keys)
```

Return type tuple(str|int, str|int)

s4.clarity.steputils.placement_utils.**place_plate_to_plate_match_wells**(*step*, *input_container*, *output_container*)
Places samples in the input_container in the output_container at the same well location. Plates do not have to be the same dimensions, but artifacts placed at invalid wells will not be accepted by Clarity.

To submit these changes you will need to call step.placements.post_and_parse() after. :param step: The step that the placement is being done for :param input_container: Container with artifacts to place in the output container :param output_container: Container that will be populated with artifacts.

s4.clarity.steputils.placement_utils.**row_order_sort_keys**(*artifact*)
Provide container position sort-keys for the sorted function based on row-major order

Usage example:

```
sorted_outputs = sorted(self.outputs, key=row_order_sort_keys)
```

Return type tuple(str|int, str|int)

2.4.5 Step Average

s4.clarity.steputils.step_average_utils.**calculatecv**(*lst*)

s4.clarity.steputils.step_average_utils.**compute_average**(*epp*, *sourceudf*, *averageudf*, *excludeudf*, *cvudf*, *outputdilutionudf=None*, *inputdilutionudf=None*)

s4.clarity.steputils.step_average_utils.**discard_outliers**(*epp*, *sourceudf*, *excludeudf*, *cvthreshold*, *madthreshold*)

s4.clarity.steputils.step_average_utils.**find_dilution_factor**(*iomap*, *outputdilutionudf*, *inputdilutionudf*)

s4.clarity.steputils.step_average_utils.**median**(*lst*)

2.4.6 Step Runner

class s4.clarity.steputils.step_runner.**StepRunner**(*lims*, *protocolname*, *stepconfigname*, *usequeuedinputs=True*, *numberofinputs=4*)

StepRunner is the abstract class that provides base functionality to automatically execute a step in Clarity. Sub classes can implement 'virtual' methods to take custom action at each screen in the step.

add_default_reagents()

For every required reagent kit in the step, the first active lot will be selected. If there are no active lots it will be omitted.

add_reagents_step()

arranging()

fire_script(*script_name*)

get_controls_from_control_names(*candidate_control_names*)

get control els controls from array of control_names :raise Exception: If any control names not permitted or non-existent

get_previous_steps_outputs(*previous_step*, *get_all=False*)

By default, filters previous step's actioned artifacts which either continued to this step, or ended a protocol. If get_all=True, doesn't filter.

Returns list of artifact URIs that match the conditions

Return type list[str]

get_started_step(*previous_step*, *project_name*)

load_existing_step(*step_limsid*)

Loads an existing step from Clarity into the StepRunner. :param step_limsid: The LIMS ID of an existing, but not completed step.

load_new_step(*input_uri_list=None*, *previous_step=None*, *controls=None*, *container_type=None*, *reagent_type=None*, *control_names=None*)

Starts a new step to be used by the StepRunner.

Parameters

- **input_uri_list** – A list of artifacts that will be use to run the step
- **previous_step** – A step that has already run, the outputs of which will be used as the inputuris
- **controls** – The list of controls that will be included in this step
- **container_type** – The container that output artifacts will be placed in. If none is supplied the default will be used.
- **reagent_type** – The name of the reagent category to use for the step. If none is supplied the default will be used.
- **control_names** – If no controls are provided, this list of control names will be used to look up controls and include them in the step

next_steps()

placement()

pooling()

post_tube_to_tube(*container_type_name='Tube'*)
Create single-well (tube) containers, and place the outputs in them.

This works around a 'bug' in Clarity where, when a step is run through the API, Clarity will not automatically move artifacts from input tubes to output tubes.

It is necessary if you are automating a step that has tube inputs and tube outputs. Override the record_details() method in your sub class and call this method as the first action in it.

@param container_type_name: the name of the container type. Must be single-well, but not necessarily "Tube"

prefetch_artifacts()
Loads the current state in Clarity of all inputs and outputs.

record_details()

replicates_for_control(*control*)

replicates_for_inputuri(*input_uri*)

run(*inputuris=None*, *previousstep=None*, *controls=None*, *containertype=None*, *reagenttype=None*, *steplimsid=None*, *control_names=None*)
Runs a step from its current state to completion.

Parameters

- **inputuris** – A list of artifacts that will be use to run the step
- **previousstep** – A step that has already run, the outputs of which will be used as the inputuris
- **controls** – The list of controls that will be included in this step
- **containertype** – The container that output artifacts will be placed in. If none is supplied the default will be used.
- **reagenttype** – The name of the reagent category to use for the step. If none is supplied the default will be used.
- **steplimsid** – The LIMS ID of an existing, but not completed step. If this is provided all other arguments will be ignored
- **control_names** – If no controls are provided, this list of control names will be used to look up controls and include them in the step

run_to_state(*goal_state_name*)
Execute sequential stages in the step until the goal state is reached.

Parameters **goal_state_name** – The name of the state to run to, name must be in the list ALL_STEP_STATES list

set_artifact_udf_as_user(*artifact*, *udf_name*, *value*, *stop_on_error=True*)
Set an Artifact UDF after first validating that it's both visible on the step, and editable.

Parameters

- **artifact** (*s4.clarity.Artifact*) – An Artifact (Analyte or ResultFile) in a running step.
- **udf_name** – Name of the udf
- **value** – Value to set
- **stop_on_error** – Whether to raise an Exception if the user couldn't set the value. Otherwise, the method will log an error and continue

Raises **Exception** – If stop_on_error is True, and there is any reason that would prevent the user from setting the UDF in the Clarity UI

set_step_udf_as_user(*udf_name*, *value*, *stop_on_error=True*)
Set a Step UDF after first validating that it's both visible on the step, and editable.

Parameters

- **udf_name** – Name of the udf
- **value** – Value to set
- **stop_on_error** – Whether to raise an Exception if the user couldn't set the value. Otherwise, the method will log an error and continue

Raises **Exception** – If stop_on_error is True, and there is any reason that would prevent the user from setting the UDF in the Clarity UI

step_config

Type *StepConfiguration*

exception s4.clarity.steputils.step_runner.**StepRunnerException**

Python Module Index

S

Index

A

B

C

D

E

F

G

I

L

M

N

O

P

Q

R

S

T

U

W

X

Y