Luna client library

# Overview

Luna client library helps users to use machine learning API services published through Luna service without directly dealing with http requests.

# Use cases

1. User creates Luna client providing base url and subscription key
2. User lists operations
3. User calls a real time endpoint using Luna client library and get result
4. User calls an async operation and get operation id
5. User gets operation status with a specified operation id
6. User gets operation output (in JSON format) with a specified operation id
7. User list all the operations
8. User chain different operations and create a workflow

# Syntax

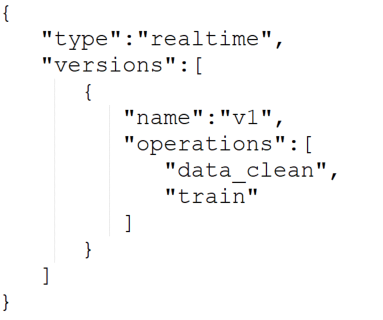
# Create luna client

luna\_client = LunaClient(base\_url, api\_name, subscription\_key, default\_api\_version)

# Realtime Operations

# List operation versions and names in JSON format

operations = luna\_client.list\_operations([api\_version])

example:   


# Call real time endpoint, operation output returned as JSON

result = luna\_client.<operation\_name>(user\_input, [api\_verison])

example:   


# Async (Pipeline) Operations

# Submit async operation, an Async Operation is returned

operation = luna\_client.<operation\_name>(user\_input, [api\_verison])

example:   


# Get operation status

operation.get\_status()

luna\_client.get\_operation\_status(operation)

luna\_client.get\_operation\_status(operation.operation\_id)

# Get operation output

operation.get\_output()

luna\_client.get\_operation\_output(operation)

luna\_client.get\_operation\_output(operation.operation\_id)

# Create workflow (sync and async)

workflow = luna\_client.create\_workflow(["data\_cleansing", "predict"], user\_input, api\_version)

workflow.validate()

workflow.run()

exec = workflow.run\_async()

exec.wait\_for\_completion()

exec.is\_complete()

exec.get\_status()

exec.get\_output()

# Functions

## LunaClient(base\_url, api\_name, subscription\_key, [optional]default\_api\_version)

Initialize a Luna client. It will

1. call the list operations API to get all the operations and validate the inputs
2. create functions dynamically for each operation

## list\_operations([optional]api\_version)

List available operations. The output format:

## luna\_client.<operation\_name>(user\_input, [optional]api\_verison)

Call the realtime or pipeline operation. If the default api version is not specified when creating the client, the api\_version is required.

It will call the API for corresponding operation and return the result (realtime) or an operation object (pipeline)

## luna\_client.get\_operation\_status(operation)

## luna\_client.get\_operation\_status(operation\_id)

Get the operation status. User can either pass in an operation object or just operation id

## luna\_client.get\_operation\_output(operation)

## luna\_client.get\_operation\_output(operation\_id)

Get the operation output. User can either pass in an operation object or just operation id.

## Operation class

The operation class has following properties:

Operation name: the name of the operation

Operation Id: the operation id returned by the API call

Status: the status returned by the submit or query API call

startTime: the startTime returned by the API call

endTime: the endTime returned by the API call

The operation class has following functions:

Get\_status() : get the status of current operation

Get\_output(): get output of current operation. Throw exception if the operation is not in Completed state

## Workflow

User can create a workflow to chain and schedule multiple operations.

workflow = luna\_client.create\_workflow(["data\_cleansing", "predict"], user\_input, [optional]api\_version)

workflow.validate()

workflow.run()

exec = workflow.run\_async()

exec.get\_status()

exec.await()

exec.get\_output()

User creates the workflow by providing the name of operations in a string array, the user input for the first operation and api version. User can validate the workflow (mainly operation names) before start the workflow.

When user start the workflow, the client library will call the operations in sequence, and pass in the output of the operation to the successor as input. If user calls the run() function, it will block the process until the workflow finished. If the user calls the run\_async() function, it will return an Execution object which contains a Task. User can query status of the workflow and the output when it is finished. User can call await() function to block until the workflow finished.