LogicMonitor DATA SDK Python

Release 0.0.9.beta2

Logicmonitor

Sep 18, 2023

CONTENTS:

This Python Library for ingesting the metrics into the LogicMonitor Platform

CHAPTER

ONE

PUSHMETRICS - METRICS INGESTION

1.1 Overview

LogicMonitor's Push Metrics feature allows you to send metrics directly to the LogicMonitor platform via a dedicated API, removing the need to route the data through a LogicMonitor Collector. Once ingested, these metrics are presented alongside all other metrics gathered via LogicMonitor, providing a single pane of glass for metric monitoring and alerting.

More details are available on support site

The `logicmonitor_data_sdk` module provides

- *`logicmonitor_data_sdk.api.metrics`*: a HTTP Api client for ingesting the metrics data.

1.2 Requirements.

Python 3.4+

1.3 Installation

1.3.1 pip install

Install from PyPI.

```
pip install logicmonitor_data_sdk
```

Then import the package:

```
import logicmonitor_data_sdk
```

1.4 Getting Started

Please follow the *Installation*.

1.4.1 Simple Example

The following example will create the new resource "SampleDevice" and data source called "PushMetricsDS". Please specified the valid company name, user id and user keys.

```
"""
=======
Copyright, 2021, LogicMonitor, Inc.
This Source Code Form is subject to the terms of the
Mozilla Public License, v. 2.0. If a copy of the MPL
was not distributed with this file, You can obtain
one at https://mozilla.org/MPL/2.0/.
=======
"""

# LogicMonitor metric data model is as below
#
# Company
#  |--- Resource (like device/service. Ex: VM)
#  |--- Data Source   (Ex. CPU)
#         |--- Instance (of a Data Source on a resource. Ex. CPU-1)
#               |--- Data Point (the metric which is being monitored. Ex. %Used)
#                     |- <Time> : <Metric Value>
#                     |- <Time> : <Metric Value>
#                     |...
import time
from random import random

import logicmonitor_data_sdk
from logicmonitor_data_sdk.api.metrics import Metrics
from logicmonitor_data_sdk.api.response_interface import ResponseInterface
from logicmonitor_data_sdk.api_client import ApiClient
from logicmonitor_data_sdk.models import DataSource, Resource, DataSourceInstance,␣
↪DataPoint
from example import system_properties

# Configure SDK with Account and access information
# On your LogicMonitor portal, create API token (LMv1) for user and get
# Access Id and Access Key
configuration = logicmonitor_data_sdk.Configuration(company='your_company',
                                                    id='API_ACCESS_ID',
                                                    key='API_ACCESS_KEY')


class MyResponse(ResponseInterface):
    """
    Sample callback to handle the response from the REST endpoints
    """
```

(continues on next page)

(continued from previous page)

```
    def success_callback(self, request, response, status, request_id):
        # logging.info("%s: %s: %s", response, status, request_id)
        print(response, status, request_id)

    def error_callback(self, request, response, status, request_id, reason):
        # logging.error("%s: %s: %s %s", response, status, reason, request_id)
        print(response, status, reason, request_id)


# Create api handle for Metrics use case (we also support Logs)
api_client = ApiClient(configuration=configuration)
metric_api = Metrics(batch=False, interval=10, response_callback=MyResponse(), api_
↪client=api_client)
return_val = metric_api.send_metrics(
    resource=Resource(
        ids={"system.hostname": "SampleDevice"},  # Core Properties of the Resource
        create=True,  # Auto-create resource if does not exist
        name="SampleDevice",  # Name of the resource
        properties=system_properties.get_system_info()),  # Additional Properties␣
↪[Optional]
    datasource=DataSource(
        name="SampleDS"),  # Name of data source is must. Rest optional
    instance=DataSourceInstance(
        name="SampleInstance"),  # Name of instance is must. Rest optional
    datapoint=DataPoint(
        name="SampleDataPoint"),  # The metric
    values={str(int(time.time())): str(random())}  # Values at specific time(s)
)
print("Return Value = ", return_val)
```

1.4.2 Detail Example - Disk Monitoring

Run below a working example for submitting the disk metrics to your LM account. This script will monitor the Usage, Free and Total of the disk at every 5 sec interval.

```
"""
=======
Copyright, 2021, LogicMonitor, Inc.
This Source Code Form is subject to the terms of the
Mozilla Public License, v. 2.0. If a copy of the MPL
was not distributed with this file, You can obtain
one at https://mozilla.org/MPL/2.0/.
=======
"""

import logging
import os
import time

import psutil as psutil
```

(continues on next page)

(continued from previous page)

```
import logicmonitor_data_sdk
from logicmonitor_data_sdk.api.metrics import Metrics
from logicmonitor_data_sdk.api.response_interface import ResponseInterface
from logicmonitor_data_sdk.models import Resource, DataSource, DataPoint, \
    DataSourceInstance

logger = logging.getLogger('lmdata.api')
logger.setLevel(logging.INFO)

configuration = logicmonitor_data_sdk.Configuration()
# For debug log, set the value to True
configuration.debug = False


class MyResponse(ResponseInterface):
    """
    Sample callback to handle the response from the REST endpoints
    """

    def success_callback(self, request, response, status, request_id):
        logger.info("%s: %s: %s", response, status, request_id)

    def error_callback(self, request, response, status, request_id, reason):
        logger.error("%s: %s: %s %s", response, status, reason, request_id)


def MetricRequest():
    """
    Main function to get the CPU values using `psutil` and send to Metrics REST endpoint
    """
    device_name = os.uname()[1]
    resource = Resource(ids={'system.displayname': device_name}, name=device_name,
                        create=True)
    datasource = DataSource(name="DiskUsingSDK")
    datapoints = ['total', 'used', 'free']
    metric_api = Metrics(batch=True, interval=30, response_callback=MyResponse())
    while True:
        partitions = psutil.disk_partitions()
        for p in partitions:
            # Using the device as instance name. We can use the mountpoint as well.

            instance_name = p.device
            usage = psutil.disk_usage(instance_name)._asdict()

            # Create the instance object for every device. Name should not have the
            # special characters so replacing it with the '-'.
            instance = DataSourceInstance(name=instance_name.replace('/', '-'),
                                          display_name=instance_name)
            for one_datapoint in datapoints:
                datapoint = DataPoint(name=one_datapoint)
                values = {str(int(time.time())): str(usage[one_datapoint])}
```

(continues on next page)

(continued from previous page)

```
                metric_api.send_metrics(resource=resource,
                                        datasource=datasource,
                                        instance=instance,
                                        datapoint=datapoint,
                                        values=values)
        time.sleep(5)


if __name__ == "__main__":
    MetricRequest()
```

Then run the program as:

```
pip install psutil
LM_COMPANY=<ACOUNT_NAME> LM_ACCESS_ID=<ID> LM_ACCESS_KEY='<KEY>' python disk_metrics.py
```

1.4.3 Simple Example - Logs

Run below script to send a log to Logicmonitor.

```
"""
=======
Copyright, 2021, LogicMonitor, Inc.
This Source Code Form is subject to the terms of the
Mozilla Public License, v. 2.0. If a copy of the MPL
was not distributed with this file, You can obtain
one at https://mozilla.org/MPL/2.0/.
=======
"""

# Sample Program to send Logs to LogicMonitor Platform
import logicmonitor_data_sdk
from logicmonitor_data_sdk.api.logs import Logs
from logicmonitor_data_sdk.models import Resource
from logicmonitor_data_sdk.api.response_interface import ResponseInterface
from example import system_properties


class MyResponse(ResponseInterface):
    """
    Sample callback to handle the response from the REST endpoints
    """

    def success_callback(self, request, response, status, request_id):
        # logging.info("%s: %s: %s", response, status, request_id)
        print(response, status, request_id)

    def error_callback(self, request, response, status, request_id, reason):
        # logging.error("%s: %s: %s %s", response, status, reason, request_id)
        print(response, status, reason, request_id)
```

(continues on next page)

(continued from previous page)

```
# Initialize LM SDK and provide required authentication parameters
# On LM Portal, create 'API Token' for the user to get access Id and access Key
configuration = logicmonitor_data_sdk.Configuration(company='your_company',
                                                    id='API access id',
                                                    key='API access key')

# The resource which is already present on LM Platform. Use a unique property to match
# the resource and send log for that.
resource = Resource(ids={"system.hostname": 'your_system'}, properties=system_properties.
↪get_system_info())

# Create an api handle for sending the logs
# "batch" would club logs for 8MB size or 30 Sec - whichever is earlier. Its default is
↪"True".
log_api = Logs(batch=False)

return_value = log_api.send_logs(resource=resource, msg="this is sample log")

print(return_value)
```

then run the script as :

```
pip install psutil
python log_non_batch.py
```

1.5 Configuration

SDK must be configured with logicmonitor_data_sdk.Configuration(). The account name, an API key and its id are required.

Mozilla Public License, v. 2.0. If a copy of the MPL was not distributed with this file, You can obtain one at https://mozilla.org/MPL/2.0/. =======

class logicmonitor_data_sdk.configuration.**Configuration**(***kwargs*)

This model is used to defining the configuration.

Parameters

- **company** (str) – The account name. If it is not provided then we will use the 'LM_COMPANY' environment variable.
- **authentication** (dict of *id* and *key*) – LogicMonitor supports verious types of the authentication. This variable will be used to specify the authentication key. If it is not provided then 'LM_ACCESS_ID' and 'LM_ACCESS_KEY' environment variable will be used to find the id and key.
- **id** (str) – The access token id. If it is not provided then we will use the 'LM_ACCESS_ID' environment variable or authentication variable.
- **key** (str) – The access token key. If it is not provided then we will use the 'LM_ACCESS_KEY' environment variable or authentication variable.

Examples: >>> import logicmonitor_data_sdk >>> # Or use 'id' and 'key' variables to specify the access token. >>> conf = logicmonitor_data_sdk.Configuration(company="ACCOUNT_NAME", id='API_ACCESS_ID', key= 'API_ACCESS_KEY')

property async_req

The async request.

Parameters
value – enable async request string.

Type
bool

property debug

Debug status

Parameters
value – The debug status, True or False.

Type
bool

property logger_file

The logger file.

If the logger_file is None, then add stream handler and remove file handler. Otherwise, add file handler and remove stream handler.

Parameters
value – The logger_file path.

Type
str

property logger_format

The logger format.

The logger_formatter will be updated when sets logger_format.

Parameters
value – The format string.

Type
str

ret_flags()

Returns the status of the BearerTokenFlag

Returns
A list with first element as bearer_flag

to_debug_report()

Gets the essential information for debugging.

Returns
The report for debugging.

1.6 API Calls

All URIs are relative to *https://<account_name>.logicmonitor.com/rest*

1.6.1 Usage

Be sure to initialize the client using *Configuration* and then use *Metrics Ingestion API*.

1.6.2 Metrics Ingestion API

Mozilla Public License, v. 2.0. If a copy of the MPL was not distributed with this file, You can obtain one at https://mozilla.org/MPL/2.0/. =======

class logicmonitor_data_sdk.api.metrics.**Metrics**(*batch=True*, *interval=10*, *response_callback=None*, *api_client=None*)

This API client is for ingesting the metrics in LogicMonitor. :param batch: Enable the batching support. :type batch: `bool` :param interval: Batching flush interval. If batching is enabled then after that second we will flush the data to REST endpoint. :type interval: `int` :param response_callback: Callback for response handling. :type response_callback: `logicmonitor_data_sdk.api.response_interface.ResonseInterface` :param api_client: The RAW HTTP REST client. :type api_client: `logicmonitor_data_sdk.api_client.ApiClient`

Examples

```
>>> from logicmonitor_data_sdk.api.metrics import Metrics
>>> from logicmonitor_data_sdk.configuration import Configuration
>>> conf = Configuration(company="ACCOUNT_NAME", id='API_ACCESS_ID', key='API_
↪ACCESS_KEY')
>>> # Create the Metrics client with batching support and flush interval as 30 sec.
>>> metricsApi = Metrics(batch=True, interval=10)
```

---

send_metrics(***kwargs*)

This send_metrics method is used to send the metrics to rest endpoint. :param resource: The Resource object. :type resource: *logicmonitor_data_sdk.models.resource.Resource* :param datasource: The datasource object. :type datasource: *logicmonitor_data_sdk.models.datasource.DataSource* :param instance: The instance object. :type instance: *logicmonitor_data_sdk.models.datasource_instance.DataSourceInstance* :param datapoint: The datapoint object. :type datapoint: *logicmonitor_data_sdk.models.datapoint.DataPoint* :param values: The values dictionary. :type values: `dict`

Returns

If in *Metrics* batching is enabled then None Otherwise the REST response will be return.

Examples

```
>>> import time
>>> from logicmonitor_data_sdk.api.metrics import Metrics
>>> from logicmonitor_data_sdk.configuration import Configuration
>>> from logicmonitor_data_sdk.models.resource import Resource
>>> from logicmonitor_data_sdk.models.datasource import DataSource
```

(continues on next page)

(continued from previous page)

```
>>> from logicmonitor_data_sdk.models.datasource_instance import␣
↪DataSourceInstance
>>> from logicmonitor_data_sdk.models.datapoint import DataPoint
>>>
>>> conf = Configuration(company="ACCOUNT_NAME", id='API_ACCESS_ID', key='API_
↪ACCESS_KEY')
>>> # Create the Metrics client with batching disabled
>>> metric_api = Metrics(batch=False)
>>> # Create the Resource object using the 'system.deviceId' properties.
>>> resource = Resource(ids={"system.hostname": "SampleDevice"}, create=True,␣
↪name="SampleDevice", properties={'using.sdk': 'true'})
>>> # Create the LMDataSource object for CPU monitoring
>>> ds = DataSource(name="CPU")
>>> # Create the DataSourceInstance object for CPU-0 instance monitoring
>>> instance = DataSourceInstance(name="CPU-0")
>>> # Create the DataPoint object for cpu-time
>>> dp = DataPoint(name='cpu_time', aggregation_type='sum')
>>> metric_api.send_metrics(resource=resource, datasource=ds, instance=instance,
↪ datapoint=dp, values={ time.time() : '23'})
```

1.6.3 Logs Ingestion API

Mozilla Public License, v. 2.0. If a copy of the MPL was not distributed with this file, You can obtain one at https://mozilla.org/MPL/2.0/. =======

class logicmonitor_data_sdk.api.logs.**Logs**(*batch=True*, *interval=10*, *response_callback=None*, *api_client=None*)

This API client is for ingesting the logs in LogicMonitor.

Parameters

- **batch** (bool) – Enable the batching support.
- **interval** (int) – Batching flush interval. If batching is enabled then after that second we will flush the data to REST endpoint.
- **response_callback** (logicmonitor_data_sdk.api.response_interface.ResonseInterface) – Callback for response handling.
- **api_client** (logicmonitor_data_sdk.api_client.ApiClient) – The RAW HTTP REST client.

Examples

```
>>> from logicmonitor_data_sdk.api.logs import Logs
>>> from logicmonitor_data_sdk.configuration import Configuration
>>> conf = Configuration(company="ACCOUNT_NAME", id='API_ACCESS_ID', key='API_
↪ACCESS_KEY')
>>> # Create the Logs client with batching support and flush interval as 30 sec.
>>> logsAPi = Logs(batch=True, interval=10)
```

send_logs(***kwargs*)

This send_logs method is used to sending the logs to rest endpoint.

Parameters

- **resource** (*logicmonitor_data_sdk.models.resource.Resource*) – The Resource object.
- **msg** (str) – The log message. e.g. msg = "this is sample log msg".
- **timestamp** (str or int, Optional) – The timestamp when the event occurred. Supported date formats are ISO8601 and Unix Epoch (in secs, ms, ns).
- **metadata** (dict,Optional) – Metadata which can be used for defining logsource and other properties.

Returns

If in *Logs* batching is enabled then None Otherwise the REST response will be return.

Examples

```
>>> import time
>>> from logicmonitor_data_sdk.api.logs import Logs
>>> from logicmonitor_data_sdk.configuration import Configuration
>>> from logicmonitor_data_sdk.models.resource import Resource
>>>
>>> conf = Configuration(company="ACCOUNT_NAME", id= 'API_ACCESS_ID', key= 'API_
↪ACCESS_KEY')
>>> # Create the Log client with batching enable
>>> log_api = Logs() # By default batching is enabled with interval of 30 sec.
>>> # Create the Resource object using the 'system.hostname' properties.
>>> resource = Resource(ids={"system.hostname": "SampleDevice"}, name=
↪"SampleDevice", properties={'using.sdk': 'true'})
>>> log_api.send_logs(resource=resource, msg = "this is a sample log")
```

1.7 Models

1.7.1 Resource

Mozilla Public License, v. 2.0. If a copy of the MPL was not distributed with this file, You can obtain one at https://mozilla.org/MPL/2.0/. =======

class logicmonitor_data_sdk.models.resource.**Resource**(*ids*, *name='None'*, *description=None*, *properties=None*, *create=False*)

This model is used to define the resource.

Parameters

- **ids** (dict) – An array of existing resource properties that will be used to identify the resource. See Managing Resources that Ingest Push Metrics for information on the types of properties that can be used. If no resource is matched and the create parameter is set to TRUE, a new resource is created with these specified resource IDs set on it. If the system.displayname and/or system.hostname property is included as resource IDs, they will be used as host name and display name respectively in the resulting resource.

- **name** (str) – Resource unique name. Only considered when creating a new resource.
- **properties** (dict of str, optional) – New properties for resource. Updates to existing resource properties are not considered. Depending on the property name, we will convert these properties into system, auto, or custom properties.
- **description** (str, optional) – Resource description. Only considered when creating a new resource.
- **create** (bool, optional) – Do you want to create the resource.

Examples

```
>>> from logicmonitor_data_sdk.models.resource import Resource
>>> # Create the Resource object using the 'system.deviceId' properties.
>>> resource = Resource(ids={'system.deviceId' : '1234'}, name='DeviceName',␣
↪create=False)
```

property create

Gets the create flag.

Returns
create flag.

Return type
bool

property description

Resource description. Only considered when creating a new resource.

Returns
The description of this Resource.

Return type
str

property ids

An array of existing resource properties that will be used to identify the resource. See Managing Resources that Ingest Push Metrics for information on the types of properties that can be used. If no resource is matched and the create parameter is set to TRUE, a new resource is created with these specified resource IDs set on it. If the system.displayname and/or system.hostname property is included as resource IDs, they will be used as host name and display name respectively in the resulting resource.

Returns
The ids of this Resource.

Return type
dict

property name

Resource unique name. Only considered when creating a new resource.

Returns
The name of this Resource.

Return type
str

property properties

New properties for resource. Updates to existing resource properties are not considered. Depending on the property name, we will convert these properties into system, auto, or custom properties.

Returns
The properties of this Resource.

Return type
dict

1.7.2 DataSource

Mozilla Public License, v. 2.0. If a copy of the MPL was not distributed with this file, You can obtain one at https://mozilla.org/MPL/2.0/. =======

class logicmonitor_data_sdk.models.datasource.**DataSource**(*name*, *display_name=None*, *group=None*, *id=None*, *singleInstanceDS=False*)

This model is used to defining the datasource object.

Parameters

- **name** (str) – DataSource unique name. Used to match an existing DataSource. If no existing DataSource matches the name provided here, a new DataSource is created with this name.
- **display_name** (str, optional) – DataSource display name. Only considered when creating a new DataSource.
- **group** (str, optional) – DataSource group name. Only considered when DataSource does not already belong to a group. Used to organize the DataSource within a DataSource group. If no existing DataSource group matches, a new group is created with this name and the DataSource is organized under the new group.
- **id** (int, optional) – DataSource unique ID. Used only to match an existing DataSource. If no existing DataSource matches the provided ID, an error results.
- **singleInstanceDS** (boolean,optional) – DataSource singleInstance attribute. When set to true, only one DataSource Instance can be created.

Examples

```
>>> from logicmonitor_data_sdk.models.datasource import DataSource
>>> # Create the DataSource object for CPU monitoring
>>> ds = DataSource(name='CPU')
```

property display_name

DataSource display name. Only considered when creating a new DataSource.

Returns
The display_name of this DataSource.

Return type
str

property group

DataSource group name. Only considered when DataSource does not already belong to a group. Used to organize the DataSource within a DataSource group. If no existing DataSource group matches, a new group is created with this name and the DataSource is organized under the new group.

Returns
The group of this DataSource.

Return type
str

property id

DataSource unique ID. Used only to match an existing DataSource. If no existing DataSource matches the provided ID, an error results.

Returns
The id of this DataSource.

Return type
int

property name

DataSource unique name. Used to match an existing DataSource. If no existing DataSource matches the name provided here, a new DataSource is created with this name.

Returns
The data_source of this DataSource.

Return type
str

property singleInstanceDS

DataSource singleInstanceDS attribute. When set to true, only one DataSource Instance can be created.

Returns
The value of the singleInstanceDS attribute: true or false

Return type
boolean

1.7.3 DataSourceInstance

Mozilla Public License, v. 2.0. If a copy of the MPL was not distributed with this file, You can obtain one at https://mozilla.org/MPL/2.0/. =======

class logicmonitor_data_sdk.models.datasource_instance.**DataSourceInstance**(*name*, *description=None*, *display_name=None*, *instance_id=None*, *properties=None*)

This model is used to defining the datasource object.

Parameters

- **name** (str) – Instance name. If no existing instance matches, a new instance is created with this name.
- **display_name** (str, optional) – Instance display name. Only considered when creating a new instance.
- **properties** (dict of str, optional) – New properties for instance. Updates to existing instance properties are not considered. Depending on the property name, we will convert these properties into system, auto, or custom properties.

Examples

```
>>> from logicmonitor_data_sdk.models.datasource_instance import DataSourceInstance
>>> # Create the DataSourceInstance object for CPU-0 instance monitoring
>>> instance = DataSourceInstance(name='CPU-0')
```

property display_name

Instance display name. Only considered when creating a new instance.

Parameters
display_name – The display_name of this DataSourceInstance.

Type
str

property instance_id

Instance ID. If no existing instance matches, a new instance is created with this id. Either Instance Name or Instance Id is Mandatory

Returns
The id of this DataSourceInstance.

Return type
int

property name

Instance name. If no existing instance matches, a new instance is created with this name.

Returns
The name of this DataSourceInstance.

Return type
str

property properties

New properties for instance. Updates to existing instance properties are not considered. Depending on the property name, we will convert these properties into system, auto, or custom properties.

Returns
The properties of this DataSourceInstance.

Return type
MapStringString

1.7.4 DataPoint

Mozilla Public License, v. 2.0. If a copy of the MPL was not distributed with this file, You can obtain one at https://mozilla.org/MPL/2.0/. =======

class logicmonitor_data_sdk.models.datapoint.**DataPoint**(*name*, *aggregation_type=None*, *description=None*, *type=None*, *percentile=None*)

This model is used to defining the datapoint object.

Parameters

- **name** (str) – Datapoint name. If no existing datapoint matches for specified DataSource, a new datapoint is created with this name.
- **aggregation_type** (str, optional) – The aggregation method, if any, that should be used if data is pushed in sub-minute intervals. Allowed options are "sum", "average" ,"percentile" and "none"(default) where "none" would take last value for that minute. Only considered when creating a new datapoint. See the About the Push Metrics REST API section of this guide for more information on datapoint value aggregation intervals.
- **description** (str, optional) – Datapoint description. Only considered when creating a new datapoint.
- **type** (str, optional) – Metric type as a number in string format. Allowed options are "guage" (default) and "counter". Only considered when creating a new datapoint.
- **percentile** (int, optional) – One of the Aggregation Type. Only set when aggregation_type is set as "percentile"

Examples

```
>>> from logicmonitor_data_sdk.models.datapoint import DataPoint
>>> # Create the DataPoint object for cpu_time
>>> dp = DataPoint(name='cpu_time', aggregation_type='sum')
```

property aggregation_type

The aggregation method, if any, that should be used if data is pushed in sub-minute intervals. Aloowed values are 'sum', 'average' , 'percentile' and 'none'(default). Only considered when creating a new datapoint.

Returns
The type of this DataPoint.

Return type
str

property description

Datapoint description. Only considered when creating a new datapoint.

Returns
The description of this DataPoint.

Return type
str

property name

Datapoint name. If no existing datapoint matches for specified DataSource, a new datapoint is created with this name.

Returns
The name of this DataPoint.

Return type
str

property percentile

One of the Aggregation Type. Only set when aggregation_type is set as "percentile".

Returns
The percentile of that datapoint between limit [0-100]

Return type
int

property type

Metric type (guage or counter) as a number in string format. Only considered when creating a new datapoint.

Returns
The type of this DataPoint.

Return type
str

1.7.5 ResponseInterface

Mozilla Public License, v. 2.0. If a copy of the MPL was not distributed with this file, You can obtain one at https://mozilla.org/MPL/2.0/. =======

class logicmonitor_data_sdk.api.response_interface.**ResponseInterface**

This is the callback interface for handling the response. End user can create his own class using this one to get the response status.

classmethod error_callback(*request*, *response*, *status*, *request_id*, *reason*)

This callback gets invoked for any error or exception from the end REST endpoint.

Parameters

- **request** (dict of str) – The json payload send to REST endpoint.
- **response** (dict of str) – Response received from the REST endpoint.
- **status** (int) – HTTP status code.
- **request_id** (str) – Unique request id generated by Rest endpoint.
- **reason** (str) – The reason for error.

classmethod success_callback(*request*, *response*, *status*, *request_id*)

This callback gets invoked for successful response from the end REST endpoint.

Parameters

- **request** (dict of str) – The json payload send to REST endpoint.
- **response** (dict of str) – Response received from the REST endpoint.
- **status** (int) – HTTP status code.
- **request_id** (str) – Unique request id generated by Rest endpoint.

1.7.6 Get in Touch

If you have questions in general, reach out to our support@logicmonitor.com

PYTHON MODULE INDEX

INDEX

N

P

R

S

T