events-protocol

Release 0.3.2

Guiabolso

Oct 25, 2022

CONTENTS

Library to be a Client and Server using Guiabolso's Events Protocol Specification, the especification of this implementation can be found here

CHAPTER

ONE

INSTALLATION

To install stable version, just download it from PyPI:

```
pip install events-protocol
```

To install from source code execute the following command:

```
pip install git+https://github.com/GuiaBolso/events-protocol-python#egg=events-
↪protocol
```

CHAPTER

TWO

BASIC USAGE

2.1 Client

The essencial information to send an event is:

Field	Description
url	URL that was exposed from Event Server
name	Event name registred on Event Server source
version	Event's version
payload	Payload with necessery event information

With this informatons we can send an event.

Instantiate the client:

```
from events_protocol.client import EventClient

client = EventClient(url="http://example.com/events/")
```

Send event:

```
# Exemplo passando apenas as informações essenciais
response = client.send_event(
    name="event:example",
    version=1,
    payload={
        "example": "example"
    },
)
```

Or you can send the event passing all of the informatons:

```
response = client.send_event(
    name="event:example",
    version=1,
    id="9230c47c-3bcf-11ea-b77f-2e728ce88125",
    flow_id="a47830ca-3bcf-11ea-a232-2e728ce88125",
    payload={
        "example": "example"
    },
    identity={
        "userId": "USER_ID",
    },
```

(continues on next page)

(continued from previous page)

```
    metadata={
        "date": "00-00-0000",
    },
    timeout=1000,
)
```

2.2 Server

An Event Server is composed by:

Component	Description
handler	Class that will receive and process the event
register	Class that will register the handler to Event Discovery
EventSchema	Event Schema will be accepted in the *payload* attribute

Example:

```
from events_protocol.server.handler.event_handler_registry import EventRegister
from events_protocol.core.builder import EventBuilder, Event
from events_protocol.core.model.base import CamelPydanticMixin
from events_protocol.core.model.event import Event, ResponseEvent
from events_protocol.server.handler.event_handler import EventHandler
from events_protocol.server.parser.event_processor import EventProcessor


class MyEventSchema(CamelPydanticMixin):
    example: str


class MyHandler(EventHandler):
    _SCHEMA = MyEventSchema

    @classmethod
    def handle(cls, event: Event) -> ResponseEvent:
        payload = cls.parse_event(event)
        response = {"MyEventPayload": payload.example}
        return EventBuilder.response_for(event, response)


class MyEventRegister(EventRegister):
    event_name = "get:event:example"
    event_version = 1
    event_handler = MyHandler


MyEventRegister.register_event()

event_input = Event(
    name="get:event:example",
    version=1,
    id="9230c47c-3bcf-11ea-b77f-2e728ce88125",
    flow_id="a47830ca-3bcf-11ea-a232-2e728ce88125",
    payload={
```

(continues on next page)

(continued from previous page)

```
        "example": "example"
    },
    identity={
        "userId": "USER_ID",
    },
    metadata={
        "date": "00-00-0000",
    },
)
input_body = event_input.to_json()

response = EventProcessor.process_event(input_body)
```

2.3 Content

2.3.1 Client

The essencial information to send an event is:

Field	Description
url	URL that was exposed from Event Server
name	Event name registred on Event Server source
version	Event's version
payload	Payload with necessery event information

With this informatons we can send an event.

Instantiate the client:

```
from events_protocol.client import EventClient

client = EventClient(url="http://example.com/events/")
```

Send event:

```
# Exemplo passando apenas as informações essenciais
response = client.send_event(
    name="event:example",
    version=1,
    payload={
        "example": "example"
    },
)
```

Or you can send the event passing all of the informatons:

```
response = client.send_event(
    name="event:example",
    version=1,
    id="9230c47c-3bcf-11ea-b77f-2e728ce88125",
```

(continues on next page)

(continued from previous page)

```
    flow_id="a47830ca-3bcf-11ea-a232-2e728ce88125",
    payload={
        "example": "example"
    },
    identity={
        "userId": "USER_ID",
    },
    metadata={
        "date": "00-00-0000",
    },
    timeout=1000,
)
```

2.3.2 Server

An Event Server is composed by:

Component	Description
handler	Class that will receive and process the event
register	Class that will register the handler to Event Discovery
EventSchema	Event Schema will be accepted in the *payload* attribute

Example:

```
from events_protocol.server.handler.event_handler_registry import EventRegister
from events_protocol.core.builder import EventBuilder, Event
from events_protocol.core.model.base import CamelPydanticMixin
from events_protocol.core.model.event import Event, ResponseEvent
from events_protocol.server.handler.event_handler import EventHandler
from events_protocol.server.parser.event_processor import EventProcessor


class MyEventSchema(CamelPydanticMixin):
    example: str


class MyHandler(EventHandler):
    _SCHEMA = MyEventSchema

    @classmethod
    def handle(cls, event: Event) -> ResponseEvent:
        payload = cls.parse_event(event)
        response = {"MyEventPayload": payload.example}
        return EventBuilder.response_for(event, response)


class MyEventRegister(EventRegister):
    event_name = "get:event:example"
    event_version = 1
    event_handler = MyHandler
```

(continues on next page)

(continued from previous page)

```
MyEventRegister.register_event()

event_input = Event(
    name="get:event:example",
    version=1,
    id="9230c47c-3bcf-11ea-b77f-2e728ce88125",
    flow_id="a47830ca-3bcf-11ea-a232-2e728ce88125",
    payload={
        "example": "example"
    },
    identity={
        "userId": "USER_ID",
    },
    metadata={
        "date": "00-00-0000",
    },
)
input_body = event_input.to_json()

response = EventProcessor.process_event(input_body)
```

2.3.3 Python API Reference

Client

Events are sent through the *EventClient*, so far we can just use HTTP Layer to transport the event.

HTTP Layer

For now we just have Event transport over HTTP, we just use *HttpClient*.

Server

Event Register

EventRegister: `EventRegister`

Event Handler

EventHandler: *EventHandler*

Event Processor

EventProcessor: *EventProcessor*

2.3.4 events_protocol

Subpackages

events_protocol.client

Subpackages

events_protocol.client.exception

Submodules

events_protocol.client.exception.request_exception

Module Contents

exception events_protocol.client.exception.request_exception.**BaseRequestException**(*message: str*)

Bases: Exception

exception events_protocol.client.exception.request_exception.**TimeoutException**

Bases: *events_protocol.client.exception.request_exception.BaseRequestException*

exception events_protocol.client.exception.request_exception.**FailedDependencyException**

Bases: *events_protocol.client.exception.request_exception.BaseRequestException*

exception events_protocol.client.exception.request_exception.**BadProtocolException**

Bases: *events_protocol.client.exception.request_exception.BaseRequestException*

Submodules

events_protocol.client.event_client

Module Contents

class events_protocol.client.event_client.**EventClient**(*url: str*)

send_event(*self*, *name: str*, *version: int*, *payload: Optional[Dict] = {}*, *id: str = None*, *flow_id: str = None*, *identity: Dict = {}*, *auth: Dict = {}*, *metadata: Dict = {}*, *timeout: Optional[int] = None*)

send_request_event(*self*, *request_event: RequestEvent*, *timeout: Optional[int] = None*)

parse_event(*self*, *raw_response: str*)

build_request_event(*self*, *name: str*, *version: int*, *payload: Dict*, *id: str = None*, *flow_id: str = None*, *identity: Dict = {}*, *auth: Dict = {}*, *metadata: Dict = {}*)

events_protocol.client.http

Module Contents

class events_protocol.client.http.**HttpClient**

post(*self, url: str, headers: Dict[str, str], payload: str, timeout: int*)

Package Contents

class events_protocol.client.**EventClient**(*url: str*)

send_event(*self*, *name: str*, *version: int*, *payload: Optional[Dict] = {}*, *id: str = None*, *flow_id: str = None*, *identity: Dict = {}*, *auth: Dict = {}*, *metadata: Dict = {}*, *timeout: Optional[int] = None*)

send_request_event(*self*, *request_event: RequestEvent*, *timeout: Optional[int] = None*)

parse_event(*self*, *raw_response: str*)

build_request_event(*self*, *name: str*, *version: int*, *payload: Dict*, *id: str = None*, *flow_id: str = None*, *identity: Dict = {}*, *auth: Dict = {}*, *metadata: Dict = {}*)

events_protocol.core

Subpackages

events_protocol.core.logging

Subpackages

events_protocol.core.logging.mixins

Submodules

events_protocol.core.logging.mixins.loggable

Module Contents

class events_protocol.core.logging.mixins.loggable.**LoggableMixin**

logger

Package Contents

class events_protocol.core.logging.mixins.**LoggableMixin**

logger

Submodules

events_protocol.core.logging.logger

Module Contents

class events_protocol.core.logging.logger.**Logger**(*log_name: str*, *log_format: str = '$BOLD%(asctime)s$RESET %(name)-12s %(levelname)-18s %(message)s'*, *date_format: str = '%Y-%m-%d %H:%M:%S'*, *level: int = logging.INFO*, *_custom_formatter: logging.Formatter = None*)

Bases: logging.LoggerAdapter

This class is responsible to be an interface for other classes and modules use the logging with the Events Protocol format.

Events Protocol.package.utils.Logger This class is responsible to be an interface for other classes and modules use the logging with the Events Protocol format

HANDLERS :List[StreamHandler]

static _create_log(*log_name: str*)
Creates the logger object.

log_name [str] The name of the logger.

logging.Logger return the logging.Logger with the logger name provided.

add_handler(*self*, *handler: logging.Handler*, *set_formatter: bool = True*, *formatter: logging.Formatter = None*)
Method to add a new handler to the logger with the defined formatter and level.

handler [Handler] The handler class to be added in the logger.

set_formatter [bool, optional] If the level and Formatter should be applied, by default True

formatter: logging.Formatter, optional The custom formatter to be set in the handler.

__add_default_handlers(*self*)
Method to add the default handlers.

classmethod get_logger(*cls*, *log_name: str*, *log_format: str = '$BOLD%(asctime)s$RESET %(name)-12s %(levelname)-18s %(message)s'*, *date_format: str = '%Y-%m-%d %H:%M:%S'*, *level: int = logging.INFO*, *_custom_formatter: logging.Formatter = None*)
Get a logger object with Events Protocol Colored Formatter.

log_name [str] Name of the logger instance

log_format [str, optional] Format of the logger message, by default "$BOLD%(asctime)s$RESET %(name)-12s %(levelname)-18s %(message)s"

date_format [str, optional] Date format of the logger message, by default "%Y-%m-%d %H:%M:%S"

level [int, optional]

The logger level, you can use the ones predefined inside the logging module, or provide an int according to the following pattern: CRITICAL/FATAL = 50, ERROR = 40, WARNING = 30, INFO = 20, DEBUG = 10, NOTSET = 0. For more information: https://docs.python.org/3/library/logging.html#levels, by default 20 (INFO).

_custom_formatter [logging.Formatter, optional] Replaces the Events Protocol ColoredFormatter by the formatter provided and will ignore the log_format parameter.

logging.Logger The formatted logger.

class events_protocol.core.logging.logger.**ColoredFormatter**(*msg: str*, *datefmt: str = '%Y-%m-%d %H:%M:%S'*, *use_color: bool = True*)

Bases: logging.Formatter

Events Protocol's Custom Colored Formatter for Logging.

Events Protocol.package.utils.ColoredFormatter This class is responsible for setting the format for Events Protocol Loggings, by the default set timestamp and the module name with bold and will use the following color scheme:

GREEN: DEBUG BLUE: INFO YELLOW: WARNING RED: ERROR MAGENTA: CRITICAL

RESET_SEQ = [0m

COLOR_SEQ = [%dm

BOLD_SEQ = [1m

COLORS

format(*self*, *record: logging.LogRecord*)

Method responsible for formatting the record.

record: logging.LogRecord The LogRecord with the message to be printed out.

str The message formatted.

classmethod formatter_message(*cls*, *message: str*, *use_color: bool = True*)

The method to parse the Events Protocol format string.

message [str] The message to be formatted.

use_color [bool] Flag to signalize if the output should be colored or not.

str The message formatted.

events_protocol.core.logging.logger.**logger_monitor**(*package_name: str = None*, *level: int = logging.INFO*)

A logger decorator to monitor legacy methods and functions.

package_name [str, optional.] The name of the package where the function or method is located. To get automatically use the __name__ variable.

level [int, optional.] The level of logger to be shown, by default: logging.INFO

events_protocol.core.logging.picpay_logger

Module Contents

class events_protocol.core.logging.picpay_logger.**PicpayLogger**(*cls*, *level: int = logging.INFO*, *environment: str = 'PRD'*)

Bases: logging.Logger

Create and format loggers in the PicPay standard format

level: int Logging level. Set using the enum defined in the logging std library

logging.Logger Interface to use the logs of the logging library

logger_name = events_protocol

version :str = UNDEFINED

is_production_environment :bool = True

internal_logger :logging.Logger

classmethod set_version(*cls*, *version: str*)
Set application version that will be logged

version: str Application version

__create_logger_with_environment(*self*, *name: str = 'events_protocol'*, *environment: str = 'PRD'*)
Creates the logger using the desired environment

name: str Application version

environment: str Set the current environment that will be used in the application. If it's PRD or HML it will format the logger in the PicPay application standards. Otherwise it'll use development settings Default PRD

__dev_log(*self*, *level*, *message*, **args*, ***kawrgs*)
Default logger. Used for development environment.

level: int Logging level. Set using the enum defined in the logging std library

message: str Message that will be send to the logger

__prod_log(*self*, *level*, *message*, **args*, ***kwargs*)
Production format logger. Used for production and QA environment.

level: int Logging level. Set using the enum defined in the logging std library

message: str Message that will be send to the logger

_log(*self*, *level*, *message*, **args*, ***kwargs*)
Overrides log method of the logging library

level: int Logging level. Set using the enum defined in the logging std library

message: str Message that will be send to the logger

events_protocol.core.logging.supressor

Module Contents

events_protocol.core.logging.supressor.**supress_log**(*f*)

events_protocol.core.logging.supressor.**disable_logs**(*log_names: typing.List[str] = []*) → None

Package Contents

class events_protocol.core.logging.**Logger**(*log_name: str*, *log_format: str = '$BOLD%(asctime)s$RESET %(name)-12s %(levelname)-18s %(message)s'*, *date_format: str = '%Y-%m-%d %H:%M:%S'*, *level: int = logging.INFO*, *_custom_formatter: logging.Formatter = None*)

Bases: logging.LoggerAdapter

This class is responsible to be an interface for other classes and modules use the logging with the Events Protocol format.

Events Protocol.package.utils.Logger This class is responsible to be an interface for other classes and modules use the logging with the Events Protocol format

HANDLERS :List[StreamHandler]

static _create_log(*log_name: str*)

Creates the logger object.

log_name [str] The name of the logger.

logging.Logger return the logging.Logger with the logger name provided.

add_handler(*self*, *handler: logging.Handler*, *set_formatter: bool = True*, *formatter: logging.Formatter = None*)

Method to add a new handler to the logger with the defined formatter and level.

handler [Handler] The handler class to be added in the logger.

set_formatter [bool, optional] If the level and Formatter should be applied, by default True

formatter: logging.Formatter, optional The custom formatter to be set in the handler.

__add_default_handlers(*self*)

Method to add the default handlers.

classmethod get_logger(*cls*, *log_name: str*, *log_format: str = '$BOLD%(asctime)s$RESET %(name)-12s %(levelname)-18s %(message)s'*, *date_format: str = '%Y-%m-%d %H:%M:%S'*, *level: int = logging.INFO*, *_custom_formatter: logging.Formatter = None*)

Get a logger object with Events Protocol Colored Formatter.

log_name [str] Name of the logger instance

log_format [str, optional] Format of the logger message, by default "$BOLD%(asctime)s$RESET %(name)-12s %(levelname)-18s %(message)s"

date_format [str, optional] Date format of the logger message, by default "%Y-%m-%d %H:%M:%S"

level [int, optional]

The logger level, you can use the ones predefined inside the logging module, or provide an int according to the following pattern: CRITICAL/FATAL = 50, ERROR = 40, WARNING = 30, INFO = 20, DEBUG = 10, NOTSET = 0. For more information: https://docs.python.org/3/library/logging.html#levels, by default 20 (INFO).

_custom_formatter [logging.Formatter, optional] Replaces the Events Protocol ColoredFormatter by the formatter provided and will ignore the log_format parameter.

logging.Logger The formatted logger.

class events_protocol.core.logging.**ColoredFormatter**(*msg: str*, *datefmt: str = '%Y-%m-%d %H:%M:%S'*, *use_color: bool = True*)

Bases: logging.Formatter

Events Protocol's Custom Colored Formatter for Logging.

Events Protocol.package.utils.ColoredFormatter This class is responsible for setting the format for Events Protocol Loggings, by the default set timestamp and the module name with bold and will use the following color scheme:

GREEN: DEBUG BLUE: INFO YELLOW: WARNING RED: ERROR MAGENTA: CRITICAL

RESET_SEQ = [0m

COLOR_SEQ = [%dm

BOLD_SEQ = [1m

COLORS

format(*self*, *record: logging.LogRecord*)
Method responsible for formatting the record.

record: logging.LogRecord The LogRecord with the message to be printed out.

str The message formatted.

classmethod formatter_message(*cls*, *message: str*, *use_color: bool = True*)
The method to parse the Events Protocol format string.

message [str] The message to be formatted.

use_color [bool] Flag to signalize if the output should be colored or not.

str The message formatted.

events_protocol.core.logging.**logger_monitor**(*package_name: str = None*, *level: int = logging.INFO*)
A logger decorator to monitor legacy methods and functions.

package_name [str, optional.] The name of the package where the function or method is located. To get automatically use the __name__ variable.

level [int, optional.] The level of logger to be shown, by default: logging.INFO

class events_protocol.core.logging.**PicpayLogger**(*cls*, *level: int = logging.INFO*, *environment: str = 'PRD'*)

Bases: logging.Logger

Create and format loggers in the PicPay standard format

level: int Logging level. Set using the enum defined in the logging std library

logging.Logger Interface to use the logs of the logging library

logger_name = events_protocol

version :str = UNDEFINED

is_production_environment :bool = True

internal_logger :logging.Logger

classmethod set_version(*cls*, *version: str*)

Set application version that will be logged

version: str Application version

__create_logger_with_environment(*self*, *name: str = 'events_protocol'*, *environment: str = 'PRD'*)

Creates the logger using the desired environment

name: str Application version

environment: str Set the current environment that will be used in the application. If it's PRD or HML it will format the logger in the PicPay application standards. Otherwise it'll use development settings Default PRD

__dev_log(*self*, *level*, *message*, **args*, ***kawrgs*)

Default logger. Used for development environment.

level: int Logging level. Set using the enum defined in the logging std library

message: str Message that will be send to the logger

__prod_log(*self*, *level*, *message*, **args*, ***kwargs*)

Production format logger. Used for production and QA environment.

level: int Logging level. Set using the enum defined in the logging std library

message: str Message that will be send to the logger

_log(*self*, *level*, *message*, **args*, ***kwargs*)

Overrides log method of the logging library

level: int Logging level. Set using the enum defined in the logging std library

message: str Message that will be send to the logger

events_protocol.core.model

Submodules

events_protocol.core.model.base

Module Contents

events_protocol.core.model.base.**_to_camel**(*string: str*) → str

class events_protocol.core.model.base.**Field**

name :str

error_type :str

message :str

to_dict(*self*)

exception events_protocol.core.model.base.**ValidationError**
Bases: Exception

fields :typing.List[Field]

to_dict(*self*)

to_json(*self*)

class events_protocol.core.model.base.**BaseModel**(*__pydantic_self__*, ***data*)
Bases: pydantic.BaseModel

classmethod from_object(*cls*, *obj: BaseModel*)

to_dict(*self*, **args*, ***kwargs*)

to_json(*self*, **args*, ***kwargs*)

class events_protocol.core.model.base.**CamelPydanticMixin**(*by_alias=True*, ***data: typing.Any*)
Bases: *events_protocol.core.model.base.BaseModel*

class Config

alias_generator

classmethod from_json(*cls*, *data: str*)

events_protocol.core.model.event

Module Contents

events_protocol.core.model.event.**PayloadType**

class events_protocol.core.model.event.**Event**
Bases: *events_protocol.core.model.base.CamelPydanticMixin*

name :str

version :int

payload :PayloadType

id :Optional[str]

flow_id :Optional[str]

identity :Optional[Dict[str, Any]]

auth :Optional[Dict[str, Any]]

metadata :Optional[Dict[str, Any]]

payload_as(*self*, *clazz: CamelPydanticMixin*)

identity_as(*self*, *clazz: Generic*)

auth_as(*self*, *clazz: Generic*)

property user_id(*self*)

property user_type(*self*)

property user(*self*)

property origin(*self*)

class events_protocol.core.model.event.**ResponseEvent**

Bases: *events_protocol.core.model.event.Event*

static from_event(*event: Event*, *event_type: EventType = EventSuccessType.SUCCESS*)

property is_success(*self*)

property is_redirect(*self*)

property is_error(*self*)

property _event_name(*self*)

property event_type(*self*)

property error_type(*self*)

class events_protocol.core.model.event.**RequestEvent**

Bases: *events_protocol.core.model.event.Event*

class events_protocol.core.model.event.**EventMessage**

Bases: *events_protocol.core.model.base.CamelPydanticMixin*

code :str

parameters :Dict[str, Optional[Any]]

events_protocol.core.model.event_type

Module Contents

class events_protocol.core.model.event_type.**EventType**

Bases: enum.Enum

classmethod is_in(*cls*, *item: typing.Any*)

__str__(*self*)

class events_protocol.core.model.event_type.**EventErrorType**
Bases: *events_protocol.core.model.event_type.EventType*

GENERIC = error

BAD_PROTOCOL = badProtocol

BAD_REQUEST = badRequest

UNAUTHORIZED = unauthorized

NOT_FOUND = notFound

FORBIDDEN = forbidden

USER_DENIED = userDenied

RESOURCE_DENIED = resourceDenied

EXPIRED = expired

UNKNOWN = unknown

classmethod get_type(*cls*, *error: str*)

class events_protocol.core.model.event_type.**EventSuccessType**
Bases: *events_protocol.core.model.event_type.EventType*

SUCCESS = response

REDIRECT = redirect

classmethod get_type(*cls*, *success: str*)

events_protocol.core.model.payload

Module Contents

class events_protocol.core.model.payload.**RedirectPayload**

url :str

query_parameters :Dict[str, Any]

events_protocol.core.model.user

Module Contents

class events_protocol.core.model.user.**User**

user_id :Optional[int]

user_type :Optional[str]

__hash__(*self*)

__eq__(*self*, *other*)

events_protocol.core.utils

Submodules

events_protocol.core.utils.config

Module Contents

events_protocol.core.utils.config.**config**(*config_name: str*, *default: typing.Any = None*, *cast: typing.Type = None*, *env_file=os.environ.get('APPLICATION_ENVIRONMENT') or 'dev'*)

events_protocol.core.utils.encoder

Module Contents

class events_protocol.core.utils.encoder.**JSONEncoder**
Bases: json.JSONEncoder

default(*self*, *obj*)

Package Contents

events_protocol.core.utils.**config**(*config_name: str*, *default: typing.Any = None*, *cast: typing.Type = None*, *env_file=os.environ.get('APPLICATION_ENVIRONMENT') or 'dev'*)

events_protocol.core.views

Submodules

events_protocol.core.views.aiohttp

Module Contents

async events_protocol.core.views.aiohttp.**init_app**(*routes: typing.List[typing.Tuple[View, str]]*, *middlewares: typing.Iterable[AIOHTTPMiddleware] = []*) → Application

events_protocol.core.views.aiohttp.**_NOT_ALLOWED_AIOHTTP**

class events_protocol.core.views.aiohttp.**AIOHTTPView**(**args*, ***kwargs*)
Bases: *events_protocol.core.views.base.BaseView*, aiohttp.web.View

request :Request

body :str

__iter__(*self*)

async get_body(*self*)

async get(*self*, **args*, ***kwargs*)

async _get(*self*, **args*, ***kwargs*)

async put(*self*, **args*, ***kwargs*)

async _put(*self*, **args*, ***kwargs*)

async post(*self*, **args*, ***kwargs*)

async _post(*self*, **args*, ***kwargs*)

async delete(*self*, **args*, ***kwargs*)

async _delete(*self*, **args*, ***kwargs*)

get_query_args(*self*)

async write_response(*self*, *http_status: HTTPStatus*, *response_body: dict*, *headers: dict = {}*)

class events_protocol.core.views.aiohttp.**AIOHTTPHealthCheckView**

Bases: *events_protocol.core.views.base.BaseHealth*, *events_protocol.core.views.aiohttp.AIOHTTPView*

events_protocol.core.views.base

Module Contents

class events_protocol.core.views.base.**BaseView**

request

_base_header

async send_response(*self*, *message: str = None*, *data: dict = None*, *description: str = None*, *http_status: int = None*, *code: int = 1000*, *log_level=None*)

abstract get_query_args(*self*)

abstract async write_response(*self*, *http_status: int*, *description: str*, *response_body: dict*, *log_level: str = None*, *headers: dict = {}*)

async _treat_general_exception(*self*, *exception: Exception*)

class events_protocol.core.views.base.**BaseHealth**

Bases: *events_protocol.core.views.base.BaseView*

_checkers :typing.List[typing.Dict[str, typing.Union[typing.Awaitable[typing.Callable]

classmethod add_checker(*cls*, *checker_name: str*, *checker_func: typing.Union[typing.Awaitable[typing.Callable], typing.Callable]*)

async _get(*self*)

events_protocol.core.views.event

Module Contents

class events_protocol.core.views.event.**DataDogAsyncEventProcessor**
Bases: *events_protocol.server.parser.event_processor.AsyncEventProcessor*

async classmethod process_event(*cls*, *raw_event: str*, *request*)

class events_protocol.core.views.event.**EventView**
Bases: *events_protocol.core.views.aiohttp.AIOHTTPView*

async _post(*self*)

Package Contents

class events_protocol.core.views.**BaseHealth**
Bases: *events_protocol.core.views.base.BaseView*

_checkers :typing.List[typing.Dict[str, typing.Union[typing.Awaitable[typing.Callable]

classmethod add_checker(*cls, checker_name: str, checker_func: typing.Union[typing.Awaitable[typing.Callable], typing.Callable]*)

async _get(*self*)

class events_protocol.core.views.**BaseView**

request

_base_header

async send_response(*self*, *message: str = None*, *data: dict = None*, *description: str = None*, *http_status: int = None*, *code: int = 1000*, *log_level=None*)

abstract get_query_args(*self*)

abstract async write_response(*self*, *http_status: int*, *description: str*, *response_body: dict*, *log_level: str = None*, *headers: dict = {}*)

async _treat_general_exception(*self*, *exception: Exception*)

class events_protocol.core.views.**JSONEncoder**
Bases: json.JSONEncoder

default(*self*, *obj*)

async events_protocol.core.views.**init_app**(*routes: typing.List[typing.Tuple[View, str]]*, *middlewares: typing.Iterable[AIOHTTPMiddleware] = []*) → Application

events_protocol.core.views.**_NOT_ALLOWED_AIOHTTP**

class events_protocol.core.views.**AIOHTTPView**(**args*, ***kwargs*)
Bases: *events_protocol.core.views.base.BaseView*, aiohttp.web.View

request :Request

body :str

__iter__(*self*)

async get_body(*self*)

async get(*self*, **args*, ***kwargs*)

async _get(*self*, **args*, ***kwargs*)

async put(*self*, **args*, ***kwargs*)

async _put(*self*, **args*, ***kwargs*)

async post(*self*, **args*, ***kwargs*)

async _post(*self*, **args*, ***kwargs*)

async delete(*self*, **args*, ***kwargs*)

async _delete(*self*, **args*, ***kwargs*)

get_query_args(*self*)

async write_response(*self*, *http_status: HTTPStatus*, *response_body: dict*, *headers: dict = {}*)

class events_protocol.core.views.**AIOHTTPHealthCheckView**
Bases: *events_protocol.core.views.base.BaseHealth*, *events_protocol.core.views.aiohttp.AIOHTTPView*

class events_protocol.core.views.**AIOHTTPView**(**args*, ***kwargs*)
Bases: *events_protocol.core.views.base.BaseView*, aiohttp.web.View

request :Request

body :str

__iter__(*self*)

async get_body(*self*)

async get(*self*, **args*, ***kwargs*)

async _get(*self*, **args*, ***kwargs*)

async put(*self*, **args*, ***kwargs*)

async _put(*self*, **args*, ***kwargs*)

async post(*self*, **args*, ***kwargs*)

async _post(*self*, **args*, ***kwargs*)

async delete(*self*, **args*, ***kwargs*)

async _delete(*self*, **args*, ***kwargs*)

get_query_args(*self*)

async write_response(*self*, *http_status: HTTPStatus*, *response_body: dict*, *headers: dict = {}*)

class events_protocol.core.views.**DataDogAsyncEventProcessor**
Bases: *events_protocol.server.parser.event_processor.AsyncEventProcessor*

async classmethod process_event(*cls*, *raw_event: str*, *request*)

class events_protocol.core.views.**EventView**
Bases: *events_protocol.core.views.aiohttp.AIOHTTPView*

async _post(*self*)

class events_protocol.core.views.**EventContextHolder**

static get()

static set(*event_context: EventContext*)

static clean()

classmethod with_context(*cls*, *context_id: IdType*, *context_flow_id: IdType*, *event_name: str*, *event_version: int*, *user_id: str = None*, *user_type: str = None*)

async classmethod with_async_context(*cls*, *context_id: IdType*, *context_flow_id: IdType*, *event_name: str*, *event_version: int*, *user_id: str = None*, *user_type: str = None*)

exception events_protocol.core.views.**EventException**(*parameters: Dict[str, Optional[Any]]*, *expected: bool = False*)

Bases: RuntimeError

_CODE :str =

_TYPE :EventErrorType

property code(*self*)

property event_error_type(*self*)

exception events_protocol.core.views.**EventParsingException**

Bases: *events_protocol.core.exception.EventException*

_CODE = INVALID_COMMUNICATION_PROTOCOL

_TYPE

class events_protocol.core.views.**LoggableMixin**

logger

exception events_protocol.core.views.**ValidationError**

Bases: Exception

fields :typing.List[Field]

to_dict(*self*)

to_json(*self*)

class events_protocol.core.views.**Event**

Bases: *events_protocol.core.model.base.CamelPydanticMixin*

name :str

version :int

payload :PayloadType

id :Optional[str]

flow_id :Optional[str]

identity :Optional[Dict[str, Any]]

auth :Optional[Dict[str, Any]]

metadata :Optional[Dict[str, Any]]

payload_as(*self*, *clazz: CamelPydanticMixin*)

identity_as(*self*, *clazz: Generic*)

auth_as(*self*, *clazz: Generic*)

property user_id(*self*)

property user_type(*self*)

property user(*self*)

property origin(*self*)

class events_protocol.core.views.**ResponseEvent**
Bases: *events_protocol.core.model.event.Event*

static from_event(*event: Event*, *event_type: EventType = EventSuccessType.SUCCESS*)

property is_success(*self*)

property is_redirect(*self*)

property is_error(*self*)

property _event_name(*self*)

property event_type(*self*)

property error_type(*self*)

class events_protocol.core.views.**EventHandler**
Bases: abc.ABC

event_name :str

event_version :typing.Union[None, int]

_SCHEMA :CamelPydanticMixin

__post_init__(*self*)

abstract handle(*cls*, *event: RequestEvent*)

classmethod parse_event(*cls*, *event: Event*)

class events_protocol.core.views.**AsyncEventHandler**
Bases: *events_protocol.server.handler.event_handler.EventHandler*, abc.ABC

_SCHEMA :CamelPydanticMixin

abstract async handle(*cls*, *event: RequestEvent*)

class events_protocol.core.views.**EventDiscovery**
Bases: *events_protocol.core.logging.mixins.loggable.LoggableMixin*

_events :typing.Dict[typing.Tuple[str, int], typing.Union[EventHandler, AsyncEventHand

_EVENT_NAME_STD :str = [a-z_]+[a-z]:[a-z_]+[a-z](:[a-z]+[a-z])*

classmethod add(*cls*, *event_name: str*, *event_handler: EventHandler*, *version: int = 1*)

classmethod get(*cls*, *event_name: str*, *event_version: int = 1*)

class events_protocol.core.views.**EventBuilder**
Bases: *events_protocol.core.logging.mixins.loggable.LoggableMixin*

classmethod error_for(*cls*, *exception: EventException*, *event: typing.Optional[Event] = Event(name='', version=1, id=str(uuid4()))*, *id_flow=str(uuid4())*, *loggable=True*)

classmethod response_for(*cls*, *event: Event*, *payload: PayloadType*, *event_type: EventSuccessType = EventSuccessType.SUCCESS*, *loggable=True*)

class events_protocol.core.views.**EventProcessor**
Bases: *events_protocol.core.logging.mixins.loggable.LoggableMixin*

event_discovery

event_validator

classmethod process_event(*cls*, *raw_event: str*)

classmethod parse_event(*cls*, *str_event: str*)

class events_protocol.core.views.**AsyncEventProcessor**
Bases: *events_protocol.server.parser.event_processor.EventProcessor*

async classmethod process_event(*cls*, *raw_event: str*)

Submodules

events_protocol.core.builder

Module Contents

class events_protocol.core.builder.**EventBuilder**
Bases: *events_protocol.core.logging.mixins.loggable.LoggableMixin*

classmethod error_for(*cls*, *exception: EventException*, *event: typing.Optional[Event] = Event(name='', version=1, id=str(uuid4()))*, *id_flow=str(uuid4())*, *loggable=True*)

classmethod response_for(*cls*, *event: Event*, *payload: PayloadType*, *event_type: EventSuccessType = EventSuccessType.SUCCESS*, *loggable=True*)

events_protocol.core.context

Module Contents

events_protocol.core.context.**IdType**

class events_protocol.core.context.**EventContext**
Bases: *events_protocol.core.model.base.BaseModel*

id :typing.Optional[IdType]

flow_id :typing.Optional[IdType]

event_name :typing.Optional[str]

event_version :typing.Optional[int]

user_id :typing.Optional[str]

user_type :typing.Optional[str]

events_protocol.core.context.**_context :ContextVar[EventContext]**

class events_protocol.core.context.**EventContextHolder**

static get()

static set(*event_context: EventContext*)

static clean()

classmethod with_context(*cls*, *context_id: IdType*, *context_flow_id: IdType*, *event_name: str*, *event_version: int*, *user_id: str = None*, *user_type: str = None*)

async classmethod with_async_context(*cls*, *context_id: IdType*, *context_flow_id: IdType*, *event_name: str*, *event_version: int*, *user_id: str = None*, *user_type: str = None*)

events_protocol.core.exception

Module Contents

exception events_protocol.core.exception.**EventException**(*parameters: Dict[str, Optional[Any]]*, *expected: bool = False*)

Bases: RuntimeError

_CODE :str =

_TYPE :EventErrorType

property code(*self*)

property event_error_type(*self*)

exception events_protocol.core.exception.**MessagebleEventException**(*message: str*)

Bases: *events_protocol.core.exception.EventException*

exception events_protocol.core.exception.**EventNotFoundException**

Bases: *events_protocol.core.exception.MessagebleEventException*

_CODE = EVENT_NOT_FOUND

_TYPE

exception events_protocol.core.exception.**MissingEventInformationException**

Bases: *events_protocol.core.exception.EventException*

_CODE = MISSING_FIELDS

_TYPE

exception events_protocol.core.exception.**EventParsingException**

Bases: *events_protocol.core.exception.EventException*

_CODE = INVALID_COMMUNICATION_PROTOCOL

_TYPE

exception events_protocol.core.exception.**EventFailedDependencyException**

Bases: *events_protocol.core.exception.MessagebleEventException*

_CODE = FAILED_DEPENDENCY

exception events_protocol.core.exception.**EventTimeoutException**

Bases: *events_protocol.core.exception.MessagebleEventException*

_CODE = EVENT_TIMEOUT

events_protocol.core.urls

Module Contents

events_protocol.core.urls.**URL_PATTERNS = [None, None]**

events_protocol.server

Subpackages

events_protocol.server.handler

Submodules

events_protocol.server.handler.event_handler

Module Contents

class events_protocol.server.handler.event_handler.**EventHandler**
Bases: abc.ABC

event_name :str

event_version :typing.Union[None, int]

_SCHEMA :CamelPydanticMixin

__post_init__(*self*)

abstract handle(*cls*, *event: RequestEvent*)

classmethod parse_event(*cls*, *event: Event*)

class events_protocol.server.handler.event_handler.**AsyncEventHandler**
Bases: *events_protocol.server.handler.event_handler.EventHandler*, abc.ABC

_SCHEMA :CamelPydanticMixin

abstract async handle(*cls*, *event: RequestEvent*)

events_protocol.server.handler.event_handler_discovery

Module Contents

class events_protocol.server.handler.event_handler_discovery.**EventDiscovery**
Bases: *events_protocol.core.logging.mixins.loggable.LoggableMixin*

_events :typing.Dict[typing.Tuple[str, int], typing.Union[EventHandler, AsyncEventHand

_EVENT_NAME_STD :str = [a-z_]+[a-z]:[a-z_]+[a-z](:[a-z]+[a-z])*

classmethod add(*cls*, *event_name: str*, *event_handler: EventHandler*, *version: int = 1*)

classmethod get(*cls*, *event_name: str*, *event_version: int = 1*)

events_protocol.server.handler.safe_event_handler

Module Contents

class events_protocol.server.handler.safe_event_handler.**SafeEventHandler**
Bases: *events_protocol.server.handler.event_handler.AsyncEventHandler*

Defines a safe event handler

event_name [str]

Event endpoint. It should contain only letters and respect the following format:

- {{name}}:{{name}}

event_version [int] Event version

Change the __safe_event_handler to contain the calls to the desired service The run method will be override to create a new handler

SafeHandler

event_name :str

event_version :typing.Union[None, int]

async classmethod handle(*cls*, *event: Event*)
Method that will safely handle the current event in the events_protocol library

event: Event Event sent by the user

ResponseEvent Response event containing the result of the service

async classmethod run(*cls*, *event: Event*)

async classmethod __safe_event_handler(*cls*, *event: Event*)
Implements the integrity verification of the payload and runs the service

event: Event Event sent by the user

ResponseEvent Response event containing the result of the service

Package Contents

class events_protocol.server.handler.**SafeEventHandler**
Bases: *events_protocol.server.handler.event_handler.AsyncEventHandler*

Defines a safe event handler

event_name [str]

Event endpoint. It should contain only letters and respect the following format:

- {{name}}:{{name}}

event_version [int] Event version

Change the __safe_event_handler to contain the calls to the desired service The run method will be override to create a new handler

SafeHandler

event_name :str

event_version :typing.Union[None, int]

async classmethod handle(*cls*, *event: Event*)
Method that will safely handle the current event in the events_protocol library

event: Event Event sent by the user

ResponseEvent Response event containing the result of the service

async classmethod run(*cls*, *event: Event*)

async classmethod __safe_event_handler(*cls*, *event: Event*)
Implements the integrity verification of the payload and runs the service

event: Event Event sent by the user

ResponseEvent Response event containing the result of the service

class events_protocol.server.handler.**EventHandler**
Bases: abc.ABC

event_name :str

event_version :typing.Union[None, int]

_SCHEMA :CamelPydanticMixin

__post_init__(*self*)

abstract handle(*cls*, *event: RequestEvent*)

classmethod parse_event(*cls*, *event: Event*)

events_protocol.server.parser

Submodules

events_protocol.server.parser.event_processor

Module Contents

class events_protocol.server.parser.event_processor.**EventProcessor**
Bases: *events_protocol.core.logging.mixins.loggable.LoggableMixin*

event_discovery

event_validator

classmethod process_event(*cls*, *raw_event: str*)

classmethod parse_event(*cls*, *str_event: str*)

class events_protocol.server.parser.event_processor.**AsyncEventProcessor**
Bases: *events_protocol.server.parser.event_processor.EventProcessor*

async classmethod process_event(*cls*, *raw_event: str*)

Package Contents

events_protocol.**__version__ = 0.3.2**

- search

PYTHON MODULE INDEX

e

INDEX

Symbols

A

B

F

G

H

I

J

L

M

N

O

T

U

V

W