pw.io.http

class pw.io.http.**PathwayWebserver**(host, port, with_schema_endpoint=True) [source]

The basic configuration class for `pw.io.http.rest_connector`.

It contains essential information about the host and the port on which the webserver should run and accept queries.

- **Parameters**
  - **host** – TCP/IP host or a sequence of hosts for the created endpoint.
  - **port** – Port for the created endpoint.
  - **with_schema_endpoint** – If set to True, the server will also provide `/_schema` endpoint containing Open API 3.0.3 schema for the handlers generated with `pw.io.http.rest_connector` calls.

property **openapi_description**

Returns Open API description for the added set of endpoints in yaml format.

property **openapi_description_json**: *dict*

Returns Open API description for the added set of endpoints in JSON format.

class pw.io.http.**RetryPolicy**(first_delay_ms, backoff_factor, jitter_ms) [source]

Class representing policy of delays or backoffs for the retries.

pw.io.http.read(url, *, schema=None, method='GET', payload=None, headers=None, response_mapper=None, format='json', delimiter=None, n_retries=0, retry_policy=<pathway.io.http._common.RetryPolicy object>, connect_timeout_ms=None, request_timeout_ms=None, allow_redirects=True, retry_codes=(429, 500, 502, 503, 504), autocommit_duration_ms=10000, debug_data=None, value_columns=None, primary_key=None, types=None, default_values=None)

Reads a table from an HTTP stream. source

- **Parameters**
  - **url** (`str`) – the full URL of streaming endpoint to fetch data from.
  - **schema** (`Optional` [`type` [`Schema`]]) – Schema of the resulting table.
  - **method** (`str`) – request method for streaming. It should be one of **HTTP request methods**.
  - **payload** (`Optional` [`Any`]) – data to be send in the body of the request.
  - **headers** (`Optional` [`dict` [`str` , `str`]]) – request headers in the form of dict. Wildcards are allowed both, in keys and in values.
  - **response_mapper** (`Optional` [`Callable` [[`str` | `bytes`], `bytes`]]) – in case a response needs to be processed, this method can be provided. It will be applied to each slice of a stream.
  - **format** (`str`) – format of the data, “json” or “raw”. In case of a “raw” format, table with single “data” column will be produced. For “json” format, bytes encoded json is expected.
  - **delimiter** (`UnionType` [`str` , `bytes` , `None`]) – delimiter used to split stream into messages.
  - **n_retries** (`int`) – how many times to retry the failed request.

- **retry_policy** (`RetryPolicy`) – policy of delays or backoffs for the retries.
- **connect_timeout_ms** (`Optional` [`int`]) – connection timeout, specified in milliseconds. In case it's None, no restrictions on connection duration will be applied.
- **request_timeout_ms** (`Optional` [`int`]) – request timeout, specified in milliseconds. In case it's None, no restrictions on request duration will be applied.
- **allow_redirects** (`bool`) – whether to allow redirects.
- **retry_codes** (`Optional` [`tuple`]) – HTTP status codes that trigger retries.
- **content_type** – content type of the data to send. In case the chosen format is JSON, it will be defaulted to "application/json".
- **autocommit_duration_ms** (`int`) – the maximum time between two commits. Every autocommit_duration_ms milliseconds, the updates received by the connector are committed and pushed into Pathway's computation graph.
- **debug_data** – static data replacing original one when debug mode is active.
- **value_columns** (`Optional` [`list` [`str`]]) – columns to extract for a table. [will be deprecated soon]
- **primary_key** (`Optional` [`list` [`str`]]) – in case the table should have a primary key generated according to a subset of its columns, the set of columns should be specified in this field. Otherwise, the primary key will be generated as uuid4. [will be deprecated soon]
- **types** (`Optional` [`dict` [`str` , `PathwayType`]]) – dictionary containing the mapping between the columns and the data types (`pw.Type`) of the values of those columns. This parameter is optional, and if not provided the default type is `pw.Type.ANY` . [will be deprecated soon]
- **default_values** (`Optional` [`dict` [`str` , `Any`]]) – dictionary containing default values for columns replacing blank entries. The default value of the column must be specified explicitly, otherwise there will be no default value. [will be deprecated soon]

Examples:

Raw format:

```
import os
import pathway as pw
table = pw.io.http.read(
  "https://localhost:8000/stream",
  method="GET",
  headers={"Authorization": f"Bearer {os.environ['BEARER_TOKEN']}"},
  format="raw",
)
```

JSON with response mapper:

Input can be adjusted using a mapping function that will be applied to each slice of a stream. The mapping function should return bytes.

```
def mapper(msg: bytes) -> bytes:
  result = json.loads(msg.decode())
  return json.dumps({"key": result["id"], "text": result["data"]}).encode()
class InputSchema(pw.Schema):
 key: int
 text: str
t = pw.io.http.read(
  "https://localhost:8000/stream",
  method="GET",
  headers={"Authorization": f"Bearer {os.environ['BEARER_TOKEN']}"},
  schema=InputSchema,
  response_mapper=mapper
)
```

pw.io.http.**rest_connector**(host=None, port=None, *, webserver=None, route='/', schema=None, autocommit_duration_ms=1500, keep_queries=None, delete_completed_queries=None)

Runs a lightweight HTTP server and inputs a collection from the HTTP endpoint, configured by the parameters of this method. **source**

On the output, the method provides a table and a callable, which needs to accept the result table of the computation, which entries will be tracked and put into respective request's responses.

- **Parameters**
  - **webserver** (`Optional` [`PathwayWebserver`]) – configuration object containing host and port information. You only need to create only one instance of this class per single host-port pair;
  - **route** (`str`) – route which will be listened to by the web server;
  - **schema** (`Optional` [`type` [`Schema`]]) – schema of the resulting table;
  - **autocommit_duration_ms** – the maximum time between two commits. Every autocommit_duration_ms milliseconds, the updates received by the connector are committed and pushed into Pathway's computation graph;
  - **keep_queries** (`Optional` [`bool`]) – whether to keep queries after processing; defaults to False. [deprecated]
  - **delete_completed_queries** (`Optional` [`bool`]) – whether to send a deletion entry after the query is processed. Allows to remove it from the system if it is stored by operators such as `join` or `groupby` ;
- **Returns**
  table – the table read; response_writer: a callable, where the result table should be provided. The result table must contain columns query_id corresponding to the primary key of an object from the input table and result, corresponding to the endpoint's return value.

Example:

Let's consider the following example: there is a collection of words that are received through HTTP REST endpoint /uppercase located at 127.0.0.1, port 9999. The Pathway program processes this table by converting these words to the upper case. This conversion result must be provided to the user on the output.

Then, you can proceed with the following REST connector configuration code.

First, the schema and the webserver object need to be created:

```
import pathway as pw
class WordsSchema(pw.Schema):
    word: str


webserver = pw.io.http.PathwayWebserver(host="127.0.0.1", port=9999)
```

Then, the endpoint that inputs this collection can be configured:

```
words, response_writer = pw.io.http.rest_connector(
    webserver=webserver,
    route="/uppercase",
    schema=WordsSchema,
)
```

Finally, you can define the logic that takes the input table words, calculates the result in the form of a table, and provides it for the endpoint's output:

```
uppercase_words = words.select(
    query_id=words.id,
    result=pw.apply(lambda x: x.upper(), pw.this.word)
)
response_writer(uppercase_words)
```

Please note that you don't need to create another web server object if you need to have more than one endpoint running on the same host and port. For example, if you need to create another endpoint that converts words to lower case, in the same way, you need to reuse the existing webserver object. That is, the configuration would start with:

```
words_for_lowercase, response_writer_for_lowercase = pw.io.http.rest_connector(
    webserver=webserver,
    route="/lowercase",
    schema=WordsSchema,
)
```

pw.io.http.**write**(table, url, *, method='POST', format='json', request_payload_template=None, n_retries=0, retry_policy=

pathway
for developers

Table of Contents >

request_timeout_ms=None, content_type=None, headers=None, allow_redirects=True, retry_codes=(429, 500, 502, 503, 504))

Sends the stream of updates from the table to the specified HTTP API. **source**

- **Parameters**
  - **table** (`Table`) – table to be tracked.
  - **method** (`str`) – request method for streaming. It should be one of **HTTP request methods**.
  - **url** (`str`) – the full URL of the endpoint to push data into. Can contain wildcards.
  - **format** (`str`) – the payload format, one of {“json”, “custom”}. If “json” is specified, the plain JSON will be formed and sent. Otherwise, the contents of the field request_payload_template will be used.
  - **request_payload_template** (`Optional` [`str`]) – the template to format and send in case “custom” was specified in the format field. Can include wildcards.
  - **n_retries** (`int`) – how many times to retry the failed request.
  - **retry_policy** (`RetryPolicy`) – policy of delays or backoffs for the retries.

- **connect_timeout_ms** (`Optional` [`int`]) – connection timeout, specified in milliseconds. In case it's None, no restrictions on connection duration will be applied.
- **request_timeout_ms** (`Optional` [`int`]) – request timeout, specified in milliseconds. In case it's None, no restrictions on request duration will be applied.
- **allow_redirects** (`bool`) – Whether to allow redirects.
- **retry_codes** (`Optional` [`tuple`]) – HTTP status codes that trigger retries.
- **content_type** (`Optional` [`str`]) – content type of the data to send. In case the chosen format is JSON, it will be defaulted to "application/json".
- **headers** (`Optional` [`dict` [`str` , `str`]]) – request headers in the form of dict. Wildcards are allowed both, in keys and in values.

Wildcards:

Wildcards are the proposed way to customize the HTTP requests composed. The engine will replace all entries of `{table.<column_name>}` with a value from the column `<column_name>` in the row sent. This wildcard resolving will happen in url, request payload template and headers.

Examples:

For the sake of demonstation, let's try diffirent ways to send the stream of changes on a table `pets` , containing data about pets and their owners. The table contains just two columns: the pet and the owner's name.

```
import pathway as pw
pets = pw.debug.table_from_markdown("owner pet \n Alice dog \n Bob cat \n Alice
```

Consider that there is a need to send the stream of changes on such table to the external API endpoint (let's pick some exemplary URL for the sake of demonstation).

To keep things simple, we can suppose that this API accepts flat JSON objects, which are sent in POST requests. Then, the communication can be done with a simple code snippet:

```
pw.io.http.write(pets, "http://www.example.com/api/event")
```

Now let’s do something more custom. Suppose that the API endpoint requires us to communicate via PUT method and to pass the values as CGI-parameters. In this case, wildcards are the way to go:

```
pw.io.http.write(
    pets,
    "http://www.example.com/api/event?owner={table.owner}&pet={table.pet}",
    method="PUT"
)
```

A custom payload can also be formed from the outside. What if the endpoint requires the data in tskv format in request body?

First of all, let’s form a template for the message body:

```
message_template_tokens = [
    "owner={table.owner}",
    "pet={table.pet}",
    "time={table.time}",
    "diff={table.diff}",
]
message_template = "\t".join(message_template_tokens)
```

Now, we can use this template and the custom format, this way:

```
pw.io.http.write(
    pets,
    "http://www.example.com/api/event",
    method="POST",
    format="custom",
    request_payload_template=message_template
)
```

Pathway Io
← pw.io.fs

Pathway Io
pw.io.jsonlines →

Chat with us on Discord

Pathway
96bis Boulevard Raspail
Agoranov
75006 Paris, France

contact@pathway.com