Application Heartbeats

Release 0.2.0

Connor Imes

Jan 21, 2024

CONTENTS:

A Python application-level heartbeats interface.

CHAPTER

ONE

INSTALLATION

The package is available on PyPI:

```
pip install apphb
```

and on Conda Forge:

```
conda install apphb
```

CHAPTER

TWO

GETTING STARTED

The core component is the `Heartbeat` class. The user defines a window period (`window_size`) that specifies a sliding window length over which performance is computed. Users may optionally specify other fields to compute sums and rates for.

For example:

```
total_iters = 10
window_size = 2
hbt = Heartbeat(window_size)
for tag in range(total_iters):
    start_time = time.monotonic()
    application_kernel()
    end_time = time.monotonic()
    hbt.heartbeat(tag, (end_time - start_time,))
    print(str(tag) + ': Instant performance: ' + str(hbt.get_instant_rate()))
    print(str(tag) + ': Window performance: ' + str(hbt.get_window_rate()))
print('Global performance: ' + str(hbt.get_global_rate()))
```

See the `examples` directory in the project source for more detailed use cases, including specifying custom fields.

CHAPTER

THREE

API REFERENCE

3.1 apphb

3.1.1 apphb package

An application-level heartbeats interface.

class apphb.**Heartbeat**(*window_size: int*, *time_shape: int = 1*, *fields_shape: Tuple[int, ...] | None = None*)

Bases: `object`

An application heartbeat interface for recording time and user-specified fields.

Heartbeats store user-provided field values and compute global, window, and instant counts and rates. A circular window buffer stores recent heartbeats in memory. Users are responsible for saving (e.g., logging) data older than a window period, if desired.

Time is the only required field, which computes rates as heart rates. All other user-specified fields compute rates w.r.t. the time field. Field units (including time units) are not specified by the API.

Notes

It may be desirable to capture and store metrics using high-precision units. For example, consider using a monotonic clock with high granularity like `time.monotonic_ns()`.

Because the API doesn't enforce particular units, users might need to normalize when retrieving heartbeat data. For example, if time is provided in nanoseconds, divide values and counts by 1 billion to compute seconds, and multiply heart rates by 1 billion to compute heartbeats per second. Similarly, custom fields might need to be normalized w.r.t. their units, and in the case of rates, w.r.t. the ratio between their units and the time units.

Parameters

- **window_size** (*int*) – The heartbeat window period, where `window_size > 0`.
- **time_shape** (*int, optional*) – The shape to be used for the *heartbeat() time* parameter. The only acceptable values are `1` and `2`. `1` implies using elapsed time, e.g., `time=(elapsed_time,)`. `2` implies using start and end times, e.g., `time=(start_time, end_time)`.
- **fields_shape** (*Tuple[int, ...], optional*) – The shape that will be used if supplying additional fields with each heartbeat. The only acceptable values in the tuple are `1` and `2`. For example, if the *heartbeat() fields* param is going to be: `fields=[(total_value_1, ), (start_value_2, end_value_2)]`, then the shape would be: `fields_shape=(1, 2)`.

get_global_count(*off: int = 0, fld: int | None = None*) → int | float

Get a heartbeat field global count.

Parameters

- **off** (*int, optional*) – A negative offset relative to the last heartbeat to get older data. The value must be in range: `-window_size <= off <= 0`.
- **fld** (*int, optional*) – The *HeartbeatRecord* field. If *None*, the *time* field is used, otherwise `field_records[fld]` is used.

Returns
The requested global count.

Return type
HeartbeatFieldCount

get_global_rate(*off: int = 0, fld: int | None = None*) → float

Get a heartbeat field global rate.

Parameters

- **off** (*int, optional*) – A negative offset relative to the last heartbeat to get older data. The value must be in range: `-window_size <= off <= 0`.
- **fld** (*int, optional*) – The *HeartbeatRecord* field. If *None*, the *time* field is used, otherwise `field_records[fld]` is used.

Returns
The requested global rate.

Return type
HeartbeatFieldCount

get_instant_count(*off: int = 0, fld: int | None = None*) → int | float

Get a heartbeat field instant count.

Parameters

- **off** (*int, optional*) – A negative offset relative to the last heartbeat to get older data. The value must be in range: `-window_size <= off <= 0`.
- **fld** (*int, optional*) – The *HeartbeatRecord* field. If *None*, the *time* field is used, otherwise `field_records[fld]` is used.

Returns
The requested instant count.

Return type
HeartbeatFieldCount

get_instant_rate(*off: int = 0, fld: int | None = None*) → float

Get a heartbeat field instant rate.

Parameters

- **off** (*int, optional*) – A negative offset relative to the last heartbeat to get older data. The value must be in range: `-window_size <= off <= 0`.
- **fld** (*int, optional*) – The *HeartbeatRecord* field. If *None*, the *time* field is used, otherwise `field_records[fld]` is used.

Returns
The requested instant rate.

Return type
HeartbeatFieldCount

`get_record`(*off: int = 0*) → *HeartbeatRecord*

Get a heartbeat record (the last one, by default).

Parameters
off (*int, optional*) – A negative offset relative to the last heartbeat to get older data. The value must be in range: `-window_size <= off <= 0`.

Returns
The desired record.

Return type
HeartbeatRecord

Notes

Modifying the record is strongly discouraged. Changes can affect future heartbeats which compute new values based on prior heartbeats, e.g., for global and window values.

`get_value`(*off: int = 0, fld: int | None = None*) → Tuple[int | float] | Tuple[int | float, int | float]

Get a heartbeat field value.

Parameters

- **off** (*int, optional*) – A negative offset relative to the last heartbeat to get older data. The value must be in range: `-window_size <= off <= 0`.
- **fld** (*int, optional*) – The *HeartbeatRecord* field. If *None*, the *time* field is used, otherwise `field_records[fld]` is used.

Returns
The requested value.

Return type
HeartbeatFieldValue

`get_window_count`(*off: int = 0, fld: int | None = None*) → int | float

Get a heartbeat field window count.

Parameters

- **off** (*int, optional*) – A negative offset relative to the last heartbeat to get older data. The value must be in range: `-window_size <= off <= 0`.
- **fld** (*int, optional*) – The *HeartbeatRecord* field. If *None*, the *time* field is used, otherwise `field_records[fld]` is used.

Returns
The requested window count.

Return type
HeartbeatFieldCount

`get_window_rate`(*off: int = 0, fld: int | None = None*) → float

Get a heartbeat field window rate.

Parameters

- **off** (*int, optional*) – A negative offset relative to the last heartbeat to get older data. The value must be in range: `-window_size <= off <= 0`.

- **fld** (*int, optional*) – The *HeartbeatRecord* field. If *None*, the *time* field is used, otherwise `field_records[fld]` is used.

Returns
The requested window rate.

Return type
HeartbeatFieldCount

heartbeat(*tag: int | str, time: Tuple[int | float] | Tuple[int | float, int | float], fields: Tuple[Tuple[int | float] | Tuple[int | float, int | float], ...] | None = None*)

Issue a heartbeat.

Parameters

- **tag** (*HeartbeatIdentifier*) – A user-specified identifier - most likely a unique *int* value.
- **time** (*HeartbeatFieldValue*) – The elapsed/total or the start and end times for the record, depending on *time_shape* specified during initialization. However specified, the elapsed time must be positive, i.e., the following must hold: `len(time) == 1 and time[0] > 0` or `len(time) == 2 and (time[1] - time[0]) > 0`.
- **fields** (*Tuple[HeartbeatFieldValue, ...], optional*) – The elapsed/total or start and end values for each field, depending on *fields_shape* specified during initialization.

property count: int

The heartbeat count.

Type
int

property fields_shape: Tuple[int, ...]

The fields shape.

Type
Tuple[int, ...]

property time_shape: int

The time shape.

Type
int

property window_size: int

The window size.

Type
int

apphb.**HeartbeatFieldRate**

Type for heartbeat field record *glbl_rate*, *wndw_rate*, and *inst_rate* fields.

class apphb.**HeartbeatFieldRecord**(*val: Tuple[int | float] | Tuple[int | float, int | float] = (0,), glbl: int | float = 0, wndw: int | float = 0, inst: int | float = 0, glbl_rate: float = 0, wndw_rate: float = 0, inst_rate: float = 0*)

Bases: `object`

Contains user-specified values and computed global, window, and instant counts and rates.

copy(*norm: int | float | None = None*, *rate_norm: float | None = None*) → *HeartbeatFieldRecord*

Create a copy, optionally by normalizing values, counts, and rates.

Parameters

- **norm** (*HeartbeatFieldCount, optional*) – The normalization factor for *val* tuple values, *glbl*, *wndw*, and *inst*.
- **rate_norm** (*HeartbeatFieldRate, optional*) – The normalization factor for *glbl_rate*, *wndw_rate*, and *inst_rate*.

Returns

A new normalized instance.

Return type

HeartbeatFieldRecord

glbl: int | float = 0

The record's global sum.

glbl_rate: float = 0

The record's global rate.

inst: int | float = 0

The record's instant value.

inst_rate: float = 0

The record's instant rate.

val: Tuple[int | float] | Tuple[int | float, int | float] = (0,)

The record's reported value.

wndw: int | float = 0

The record's window sum.

wndw_rate: float = 0

The record's window rate.

class apphb.**HeartbeatRecord**(*ident: int | str = 0*, *tag: int | str = 0*, *time: ~apphb.HeartbeatFieldRecord = <factory>*, *field_records: ~typing.List[~apphb.HeartbeatFieldRecord] = <factory>*)

Bases: object

Contains identifiers and *HeartbeatFieldRecord* instances for a heartbeat's fields.

field_records: List[*HeartbeatFieldRecord*]

The record's other fields.

ident: int | str = 0

The record's (preferably) unique identifier.

tag: int | str = 0

The record's (preferably) unique alternate identifier.

time: *HeartbeatFieldRecord*

The record's time field.

The *time* field is a special case. Other fields depend on it for computing their rates. In contrast, *time* rates are computed as heartbeat rates.

apphb.**HeartbeatFieldCount**

Type for heartbeat field record *glbl*, *wndw*, and *inst* fields.

alias of Union[int, float]

apphb.**HeartbeatFieldRecordData**

Union of types in a heartbeat field record.

alias of Union[Tuple[Union[int, float]], Tuple[Union[int, float], Union[int, float]], int, float]

apphb.**HeartbeatFieldValue**

Type for heartbeat field record *val* field.

alias of Union[Tuple[Union[int, float]], Tuple[Union[int, float], Union[int, float]]]

apphb.**HeartbeatIdentifier**

Type for heartbeat record *ident* field.

alias of Union[int, str]

apphb.**HeartbeatRecordData**

Union of types in a heartbeat record.

alias of Union[Tuple[Union[int, float]], Tuple[Union[int, float], Union[int, float]], int, float, str]

Submodules

apphb.logging module

Heartbeat logging utilities.

apphb.logging.**get_log_header**(*hbt:* Heartbeat, *time_name: str = 'Time'*, *heartrate_name: str = 'Heart Rate'*, *field_names: Sequence[str] | None = None*, *field_rate_names: Sequence[str] | None = None*) → List[str]

Get a heartbeat header for logging.

The first entries are always 'Heartbeat' and 'Tag'. 'Heartbeat' is the internal identifier. 'Tag' is the user-specified identifier. Then for each field, including *time*:

> The field shape determines whether one or two entries will be used for the value, where two columns implies Start and End values. Then entries for Global, Window, and Instant counts, then Global, Window, and Instant rates.

The first field name is 'Time' with corresponding rate name 'Heart Rate'. User-specified fields and their corresponding rate names follow.

Parameters

- **hbt** (Heartbeat) – The heartbeat instance.
- **time_name** (*str, optional*) – Custom name for the *time* field, e.g., to specify units.
- **heartrate_name** (*str, optional*) – Custom name for the heartrate (rates for the *time* field), e.g., to specify units.
- **field_names** (*Sequence[str], optional*) – Names for user-specified fields.
- **field_rate_names** (*Sequence[str], optional*) – Rate names for user-specified fields. If not specified, the corresponding field name is used with 'Rate' appended to it. If `len(field_rate_names) < len(field_names)`, 'Rate' is appended to remaining names.

Returns
The list of names for all identifiers and fields in a heartbeat record.

Return type
List[str]

apphb.logging.**get_log_record**(*hbr: HeartbeatRecord*, *time_norm: int | float | None = None*, *heartrate_norm: float | None = None*, *field_norms: Sequence[int | float] | None = None*, *field_rate_norms: Sequence[float] | None = None*) → List[Tuple[int | float] | Tuple[int | float, int | float] | int | float | str]

Get a heartbeat record for logging.

Parameters

- **hbr** (*HeartbeatRecord*) – The heartbeat record to be logged.
- **time_norm** (*HeartbeatFieldCount, optional*) – The normalization factor for the *time* field's *val*, *glbl*, *wndw*, and *inst* values.
- **heartrate_norm** (*HeartbeatFieldRate, optional*) – The normalization factor for the *time* field's *glbl_rate*, *wndw_rate*, and *inst_rate* values.
- **field_norms** (*Sequence[HeartbeatFieldCount], optional*) – The normalization factor for user-specified fields' *val*, *glbl*, *wndw*, and *inst* values. The entire parameter or individual elements may be *None*.
- **field_rate_norms** (*Sequence[HeartbeatFieldRate], optional*) – The normalization factor for user-specified fields' *glbl_rate*, *wndw_rate*, and *inst_rate* values. The entire parameter or individual elements may be *None*.

Returns
The heartbeat record data.

Return type
List[HeartbeatRecordData]

apphb.logging.**get_log_records**(*hbt: Heartbeat*, *count: int | None = None*, *time_norm: int | float | None = None*, *heartrate_norm: float | None = None*, *field_norms: Sequence[int | float] | None = None*, *field_rate_norms: Sequence[float] | None = None*) → List[List[Tuple[int | float] | Tuple[int | float, int | float] | int | float | str]]

Get heartbeat records for logging.

Parameters

- **hbt** (*Heartbeat*) – The heartbeat to be logged.
- **count** (*int, optional*) – The number of historical records to get, where `0 <= count <= window_size`. If *None*, returns the previous window history.
- **time_norm** (*HeartbeatFieldCount, optional*) – The normalization factor for the *time* field's *val*, *glbl*, *wndw*, and *inst* values.
- **heartrate_norm** (*HeartbeatFieldRate, optional*) – The normalization factor for the *time* field's *glbl_rate*, *wndw_rate*, and *inst_rate* values.
- **field_norms** (*Sequence[HeartbeatFieldCount], optional*) – The normalization factor for user-specified fields' *val*, *glbl*, *wndw*, and *inst* values. The entire parameter or individual elements may be *None*.
- **field_rate_norms** (*Sequence[HeartbeatFieldRate], optional*) – The normalization factor for user-specified fields' *glbl_rate*, *wndw_rate*, and *inst_rate* values. The entire parameter or individual elements may be *None*.

Returns
The heartbeat records data.

Return type
List[List[HeartbeatRecordData]]

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX