Tracing User Guide

Release 1.0

Bruce Mitchener, Jr.

Sep 04, 2020

CONTENTS

This is a tracing library based on the Dapper work by Google. It is also similar to Twitter's Zipkin and HTrace from Cloudera which were also inspired by the Dapper work.

Tracing is different from simply recording metrics about a service or logging text strings to a log file. Tracing is intended to capture information about the units of work that are being performed, how they relate to one another, and how long the pieces of the units of work take to execute.

Each unit of work at the top level is captured by a trace which consists of a tree of spans (**). Each unit of work is tracked by a span in the tree.

Each span can have key/value data associated with it via either *trace-add-data* or *span-add-data*. This data typically represents data associated with the computation or action. Examples might include the user name making a request, the SQL being executed, the table being queried, or the IP address associated with a request.

Each span can also have timestamped annotations provided via either *trace-annotate* or *span-annotate*. This associates a text description of an event with a timestamp and the span. This might be used to indicate progress through a task, unusual events, or anything interesting.

Other important details are discussed below, such as *Writers* and *Sampling*.

CHAPTER

ONE

INSTRUMENTING YOUR CODE

Instrumenting your code with tracing should be pretty easy.

At application startup, call *trace-set-host* with a string identifier that is unique among the machines running the software.

To start tracing, either call *trace-push* and *trace-pop* or use *with-tracing*:

```
with-tracing ("http-request")
  trace-add-data("client-ip-address", ...);
  trace-add-data("requested-path", ...);

  ... do work ...
end;
```

1.1 Setting up Sampling

Sampling of traces is left up to the application. A few basic primitives are supplied in *always-sample*, *never-sample*, *if-tracing-sample*, and related functions. See *Sampling* for more information.

CHAPTER

TWO

THE TRACING-CORE MODULE

2.1 Tracing

The tracing functions in this section represent the high level interface to the tracing library and are what would typically be used, rather than the span-specific functions.

There may be times though when using the lower level, span-specific functions is appropriate, such as when you have multiple units of work executing asynchronously. The asynchronous tasks may find it easier to track their own spans separately.

`trace-push` `Function`

Signature trace-push (description #key sampler) => (span?)

Parameters

- **`description`** – An instance of `<string>`.
- **`sampler`** (*`#key`*) – An instance of `<function>`.

Values

- **`span?`** – An instance of `false-or(<span>)`.

Discussion

Create a new *`<span>`* and make it the current tracing span. If there is already a span, the new span will use the existing span as the parent.

`sampler` defaults to *`if-tracing-sample`*.

See also

- *trace-pop*

trace-add-data Function

Signature trace-add-data (key data) => ()

Parameters

- **key** – An instance of <string>.
- **data** – An instance of <string>.

Discussion Adds key / value data to the current trace span (if any), using *span-add-data*.

See also

- *span-add-data*

trace-annotate Function

Signature trace-annotate (description) => ()

Parameters

- **description** – An instance of <string>.

Discussion Adds an annotation to the current trace span (if any), using *span-annotate*.

See also

- *span-annotate*

trace-pop Function

Signature trace-pop (span?) => ()

Parameters

- **span?** – An instance of false-or().

Discussion Stops the current span and pops it from the stack, returning the previous span to the current slot.

See also

- *trace-push*

trace-set-host Function

Sets the global host identifier that will be set on all spans created by this process.

Signature trace-set-host (host) => ()

Parameters

- **host** – An instance of <string>.

Discussion

Sets the global host identifier that will be set on all spans created by this process.

This may just be a hostname, but if multiple processes are running on the same host, then it should include additional identifying data. Ideally, this identifier will be unique per process within a cluster in a distributed system.

Note: This function should be called early in the application startup, before any tracing is performed.

See also

- *span-host*

with-tracing Macro

Macro Call

```
with-tracing ("Span description")
  trace-add-data("Table", "users");
  ...
end with-tracing;

with-tracing ("Span description", sampler: never-sample)
  ...
end with-tracing;
```

Discussion The `with-tracing` macro simplifies the process of calling *trace-push* and *trace-pop*. It can also take any keywords that *trace-push* takes and will pass them along.

2.2 Spans

** Class**

Superclasses <object>

Init-Keywords

- **description** –
- **parent-id** –
- **trace-id** –

Discussion A span tracks a period of time associated with a computation or action, along with annotations and key / value data. Spans exist within a tree of spans all of which share the same `trace-id`.

span-add-data Generic function

Signature span-add-data (span key data) => ()

Parameters

- **span** – An instance of **.
- **key** – An instance of `<string>`.
- **data** – An instance of `<string>`.

Discussion Key / value pairs may be stored on a span to provide better context. This might include the query being executed, address or host information or whatever is relevant to the application being traced.

See also

- *span-data*

span-annotate Generic function

Signature span-annotate (span description) => ()

Parameters

- **span** – An instance of **.
- **description** – An instance of <string>.

Discussion Annotations are to record an occurrence of an event during a span. They have a specific timestamp associated with them that is automatically set to the time when the annotation is created.

See also

- *span-annotations*
- *<span-annotation>*
- *annotation-description*
- *annotation-timestamp*

span-annotations Generic function

Returns the collection of *<span-annotation>* associated with this span.

Signature span-annotations (span) => (annotations)

Parameters

- **span** – An instance of **.

Values

- **annotations** – An instance of <vector>.

See also

- *span-annotate*
- *<span-annotation>*
- *annotation-description*
- *annotation-timestamp*

span-data Generic function

Returns the property list of data associated with this span.

Signature span-data (span) => (data)

Parameters

- **span** – An instance of **.

Values

- **data** – An instance of <vector>.

See also

- *span-add-data*

span-description Generic function

Returns the description of the span.

Signature span-description (span) => (description)

Parameters

- **span** – An instance of **.

Values

- **description** – An instance of <string>.

span-duration Generic function

Signature span-duration (span) => (time?)

Parameters

- **span** – An instance of **.

Values

- **time?** – An instance of false-or(<duration>).

Discussion If the span has not yet been stopped, this returns #f. Once the span has been stopped, the duration that the span was running will be returned.

See also

- *span-start-time*
- *<duration>*
- *duration-microseconds*
- *duration-seconds*

span-host Generic function

Returns the host identifier for the process which created this span.

Signature span-host (span) => (host)

Parameters

- **span** – An instance of **.

Values

- **host** – An instance of <string>.

span-id Generic function

Returns the unique ID associated with this span.

Signature span-id (span) => (id)

Parameters

- **span** – An instance of **.

Values

- **id** – An instance of <object>.

span-parent-id Generic function

Signature span-parent-id (span) => (id)

Parameters

- **span** – An instance of **.

Values

- **id** – An instance of <object>.

span-process-id Generic function

Signature span-process-id (span) => (process-id)

Parameters

- **span** – An instance of **.

Values

- **process-id** – An instance of <integer>.

span-start-time Generic function

Returns the time that the span was created.

Signature span-start-time (span) => (timestamp)

Parameters

- **span** – An instance of **.

Values

- **timestamp** – An instance of *<timestamp>*.

Like other time values in Dylan, this is the time since January 1, 1900.

See also

- *span-duration*
- *<timestamp>*
- *timestamp-days*
- *timestamp-microseconds*
- *timestamp-seconds*

span-stop Generic function

Stops a span and sends it to the current registered *<span-writer>* instances.

Signature span-stop (span) => ()

Parameters

- **span** – An instance of **.

See also

- *span-stopped?*
- *store-span*

span-stopped? Generic function

Has the span been stopped yet?

Signature span-stopped? (span) => (stopped?)

Parameters

- **span** – An instance of **.

Values

- **stopped?** – An instance of <boolean>.

See also

- *span-stop*

span-thread-id Generic function

Signature span-thread-id (span) => (thread-id)

Parameters

- **span** – An instance of **.

Values

- **thread-id** – An instance of <integer>.

span-trace-id Generic function

Return the trace-id for a span.

Signature span-trace-id (span) => (id)

Parameters

- **span** – An instance of **.

Values

- **id** – An instance of <object>.

Discussion Returns the trace-id for a span. This ID is the same for all spans within a single trace.

2.3 Annotations

Annotations let you attach events that happened at a point in time (noted by a timestamp) to a span.

<span-annotation> Class

Superclasses <object>

Init-Keywords

- **description** –
- **timestamp** –

annotation-description Generic function

Return the description of an annotation.

Signature annotation-description (annotation) => (description)

Parameters

- **annotation** – An instance of *<span-annotation>*.

Values

- **description** – An instance of <string>.

annotation-timestamp Generic function

Return the timestamp at which the annotation was created and attached.

Signature annotation-timestamp (annotation) => (timestamp)

Parameters

- **annotation** – An instance of *<span-annotation>*.

Values

- **timestamp** – An instance of *<timestamp>*.

2.4 Sampling

Samplers allow for collecting a subset of the data, making the usage of this tracing framework in a heavily loaded production scenario more realistic.

Samplers are simply functions that return a boolean value indicating whether or not an actual trace should be generated and recorded. They should be called at the appropriate point within the application being traced. Some applications may wish to limit which traces or parts of traces are collected in ways that are not readily representable within this framework.

always-sample Function

Always returns true, so that the trace is sampled.

Signature always-sample () => #t

Values

- **record-sample?** – Always `#t`.

if-tracing-sample Function

Returns true if tracing is enabled, otherwise `#f`.

Signature if-tracing-sample () => (record-sample?)

Values

- **record-sample?** – An instance of `<boolean>`.

See also

- *disable-tracing*
- *enable-tracing*

never-sample Function

Always returns false, so that the trace isn't sampled.

Signature never-sample () => #f

Values

- **record-sample?** – Always `#f`.

disable-tracing Function

Signature disable-tracing () => ()

Discussion This function only modifies the return value of *if-tracing-sample* and does not globally disable tracing.

See also

- *enable-tracing*

enable-tracing Function

Signature enable-tracing () => ()

Discussion This function only modifies the return value of *if-tracing-sample* and does not globally enable tracing.

See also

- *disable-tracing*

2.5 Writers

Spans are stored by using instances of *<span-writer>* which have been registered using *register-span-writer*. Spans are stored when they are stopped (*trace-pop*, *span-stop*). Spans are also stored when they are finalized without having been stopped previously. This finalization is only present to prevent data from being lost and should not be a default mode of operation.

<span-writer> Class

Superclasses <object>

See also

- *register-span-writer*
- *registered-span-writers*
- *unregister-span-writer*

register-span-writer Function

Signature register-span-writer (span-writer) => ()

Parameters

- **span-writer** – An instance of *<span-writer>*.

See also

- *<span-writer>*
- *registered-span-writers*
- *unregister-span-writer*

registered-span-writers Function

Signature registered-span-writers () => (span-writers)

Values

- **span-writers** – An instance of `<span-writer-vector>`.

See also

- *<span-writer>*
- *register-span-writer*
- *unregister-span-writer*

store-span Function

Signature store-span (span) => ()

Parameters

- **span** – An instance of **.

See also

- *registered-span-writers*

unregister-span-writer Function

Signature unregister-span-writer (span-writer) => ()

Parameters

- **span-writer** – An instance of *<span-writer>*.

See also

- *<span-writer>*
- *register-span-writer*
- *registered-span-writers*

2.6 Writer Implementation

To add a new storage class, subclass *<span-writer>* and implement the *span-writer-add-span* method. Then, call *register-span-writer* with an instance of your span writer and all subsequent spans completed will be written to it.

span-writer-add-span Generic function

Signature span-writer-add-span (span span-writer) => ()

Parameters

- **span** – An instance of **.
- **span-writer** – An instance of *<span-writer>*.

Discussion This method is specialized for each subclass of *<span-writer>*. It is called whenever a span needs to be processed by a span writer.

2.7 Time Utilities

<duration> Class

Measure of time elapsed.

Superclasses <object>

Init-Keywords

- **microseconds** –
- **seconds** –

See also

- *duration-microseconds*
- *duration-seconds*

duration-microseconds Generic function

Signature duration-microseconds (duration) => (microseconds)

Parameters

- **duration** – An instance of *<duration>*.

Values

- **microseconds** – An instance of <integer>.

See also

- *duration-seconds*

duration-seconds Generic function

Signature duration-seconds (duration) => (seconds)

Parameters

- **duration** – An instance of *<duration>*.

Values

- **seconds** – An instance of <integer>.

See also

- *duration-microseconds*

<timestamp> Class

A point in time.

Superclasses <object>

Init-Keywords

- **microseconds** –
- **seconds** –

See also

- *timestamp-days*
- *timestamp-microseconds*
- *timestamp-seconds*

timestamp-days Generic function

Signature timestamp-days (timestamp) => (days)

Parameters

- **timestamp** – An instance of *<timestamp>*.

Values

- **days** – An instance of <integer>.

See also

- *timestamp-microseconds*
- *timestamp-seconds*

timestamp-microseconds Generic function

Signature timestamp-microseconds (timestamp) => (microseconds)

Parameters

- **timestamp** – An instance of *<timestamp>*.

Values

- **microseconds** – An instance of <integer>.

See also

- *timestamp-days*
- *timestamp-seconds*

timestamp-seconds Generic function

Signature timestamp-seconds (timestamp) => (seconds)

Parameters

- **timestamp** – An instance of *<timestamp>*.

Values

- **seconds** – An instance of <integer>.

See also

- *timestamp-days*
- *timestamp-microseconds*

2.8 Tags

These constants are available to help standardize tracing across applications.

$tag/peer/host-name Constant

$tag/peer/ip Constant

$tag/peer/port Constant

$tag/http/uri Constant

$tag/http/response/size Constant

$tag/http/status Constant

2.9 Miscellaneous

get-unique-id Function

Signature get-unique-id () => (id)

Values

- **id** – An instance of <unique-id>.

API INDEX

INDEX