OpenTelemetry C++

Release 0.6.0

OpenTelemetry authors

May 11, 2021

OPENTELEMETRY C++ API

1 Overview 1

2 Getting started 3

3 Reference documentation 5

4 Getting help 37

Index 39

CHAPTER

ONE

OVERVIEW

The OpenTelemetry C++ API enables developers to instrument their applications and libraries in order to make them ready to create and emit telemetry data. The OpenTelemetry C++ API exclusively focuses on instrumentation and does not address concerns like exporting, sampling, and aggregating telemetry data. Those concerns are addressed by the OpenTelemetry C++ SDK. This architecture enables developers to instrument applications and libraries with the OpenTelemetry C++ API while being completely agnostic of how telemetry data is exported and processed.

1.1 Library design

The OpenTelemetry C++ API is provided as a header-only library and supports all recent versions of the C++ standard, down to C++11.

A single application might dynamically or statically link to different libraries that were compiled with different compilers, while several of the linked libraries are instrumented with OpenTelemetry. OpenTelemetry C++ supports those scenarios by providing a stable ABI. This is achieved by a careful API design, and most notably by providing ABI stable versions of classes from the standard library. All those classes are provided in the `opentelemetry::nostd` namespace.

CHAPTER

TWO

GETTING STARTED

2.1 Tracing

When instrumenting libraries and applications, the most simple approach requires three steps.

2.1.1 Obtain a tracer

```
auto provider = opentelemetry::trace::Provider::GetTracerProvider();
auto tracer = provider->GetTracer("foo_library", "1.0.0");
```

The `TracerProvider` acquired in the first step is a singleton object that is usually provided by the OpenTelemetry C++ SDK. It is used to provide specific implementations for API interfaces. In case no SDK is used, the API provides a default no-op implementation of a `TracerProvider`.

The `Tracer` acquired in the second step is needed to create and start `Span`s.

2.1.2 Start a span

```
auto span = tracer->StartSpan("HandleRequest");
```

This creates a span, sets its name to `"HandleRequest"`, and sets its start time to the current time. Refer to the API documentation for other operations that are available to enrich spans with additional data.

2.1.3 Mark a span as active

```
auto scope = tracer->WithActiveSpan(span);
```

This marks a span as active and returns a `Scope` object bound to the lifetime of the span. When the `Scope` object is destroyed, the related span is ended.

The concept of an active span is important, as any span that is created without explicitly specifying a parent is parented to the currently active span.

CHAPTER

THREE

REFERENCE DOCUMENTATION

3.1 Class Hierarchy

3.2 File Hierarchy

3.3 Full API

3.3.1 Namespaces

Namespace opentelemetry

Namespaces

- *Namespace opentelemetry::common*
- *Namespace opentelemetry::trace*

Namespace opentelemetry::common

Classes

- *Class KeyValueIterable*
- *Class SteadyTimestamp*
- *Class SystemTimestamp*

Typedefs

- *Typedef opentelemetry::common::AttributeValue*

Namespace opentelemetry::trace

Namespaces

- *Namespace opentelemetry::trace::propagation*

Classes

- *Struct EndSpanOptions*
- *Struct StartSpanOptions*
- *Class DefaultSpan*
- *Class DefaultTracer*
- *Class NoopSpan*
- *Class NoopTracer*
- *Class NoopTracerProvider*
- *Class NullSpanContext*
- *Class Provider*
- *Class Scope*
- *Class Span*
- *Class SpanContext*
- *Class SpanContextKeyValueIterable*
- *Class SpanId*
- *Class TraceFlags*
- *Class TraceId*
- *Class Tracer*
- *Class TracerProvider*
- *Class TraceState*

Enums

- *Enum CanonicalCode*
- *Enum SpanKind*
- *Enum StatusCode*

Functions

- *Template Function opentelemetry::trace::to_span_ptr*

Variables

- *Variable opentelemetry::trace::kSpanKey*

Namespace opentelemetry::trace::propagation

Namespaces

- *Namespace opentelemetry::trace::propagation::detail*

Classes

- *Class B3Propagator*
- *Class B3PropagatorExtractor*
- *Class B3PropagatorMultiHeader*
- *Class HttpTraceContext*
- *Class JaegerPropagator*

Functions

- *Function opentelemetry::trace::propagation::GetSpan*
- *Function opentelemetry::trace::propagation::SetSpan*

Variables

- *Variable opentelemetry::trace::propagation::kB3CombinedHeader*
- *Variable opentelemetry::trace::propagation::kB3SampledHeader*
- *Variable opentelemetry::trace::propagation::kB3SpanIdHeader*
- *Variable opentelemetry::trace::propagation::kB3TraceIdHeader*
- *Variable opentelemetry::trace::propagation::kSpanIdHexStrLength*
- *Variable opentelemetry::trace::propagation::kSpanIdSize*
- *Variable opentelemetry::trace::propagation::kTraceFlagsSize*
- *Variable opentelemetry::trace::propagation::kTraceHeader*
- *Variable opentelemetry::trace::propagation::kTraceIdHexStrLength*
- *Variable opentelemetry::trace::propagation::kTraceIdSize*
- *Variable opentelemetry::trace::propagation::kTraceParent*
- *Variable opentelemetry::trace::propagation::kTraceParentSize*

- *Variable opentelemetry::trace::propagation::kTraceState*
- *Variable opentelemetry::trace::propagation::kVersionSize*

Namespace opentelemetry::trace::propagation::detail

Functions

- *Function opentelemetry::trace::propagation::detail::HexToBinary*
- *Function opentelemetry::trace::propagation::detail::HexToInt*
- *Function opentelemetry::trace::propagation::detail::IsValidHex*
- *Function opentelemetry::trace::propagation::detail::SplitString*

Variables

- *Variable opentelemetry::trace::propagation::detail::kHexDigits*

3.3.2 Classes and Structs

Struct EndSpanOptions

- Defined in file_include_opentelemetry_trace_span.h

Struct Documentation

struct opentelemetry::trace::**EndSpanOptions**
StartEndOptions provides options to set properties of a *Span* when it is ended.

Public Members

common::*SteadyTimestamp* **end_steady_time**

Struct StartSpanOptions

- Defined in file_include_opentelemetry_trace_span.h

Struct Documentation

struct opentelemetry::trace::**StartSpanOptions**
StartSpanOptions provides options to set properties of a *Span* at the time of its creation

Public Members

common::*SystemTimestamp* **start_system_time**

common::*SteadyTimestamp* **start_steady_time**

SpanContext **parent** = *SpanContext*::*GetInvalid*()

SpanKind **kind** = *SpanKind*::*kInternal*

Class KeyValueIterable

- Defined in file_include_opentelemetry_common_key_value_iterable.h

Class Documentation

class opentelemetry::common::**KeyValueIterable**

Supports internal iteration over a collection of key-value pairs.

Public Functions

virtual ~KeyValueIterable() = default

virtual bool **ForEachKeyValue**(nostd::function_ref<bool) nostd::string_view, common::*AttributeValue* > *callback* **const noexcept** = 0Iterate over key-value pairs

Parameters **callback** – a callback to invoke for each key-value. If the callback returns false, the iteration is aborted.

Returns true if every key-value pair was iterated over

virtual size_t **size**() **const noexcept** = 0

Returns the number of key-value pairs

Class SteadyTimestamp

- Defined in file_include_opentelemetry_common_timestamp.h

Class Documentation

class opentelemetry::common::**SteadyTimestamp**

A timepoint relative to the monotonic clock epoch.

This is used for calculating the duration of an operation.

Public Functions

inline SteadyTimestamp() noexcept
Initializes a monotonic timestamp pointing to the start of the epoch.

template<class **Rep**, class **Period**>
inline explicit SteadyTimestamp(const std::chrono::duration<*Rep*, *Period*> *&time_since_epoch*) **noexcept**
Initializes a monotonic timestamp from a duration.

Parameters **time_since_epoch** – Time elapsed since the beginning of the epoch.

inline SteadyTimestamp(const std::chrono::steady_clock::time_point *&time_point*) **noexcept**
Initializes a monotonic timestamp based on a point in time.

Parameters **time_point** – A point in time.

inline operator std::chrono::steady_clock::time_point() const noexcept
Returns a time point for the time stamp.

Returns A time point corresponding to the time stamp.

inline std::chrono::nanoseconds **time_since_epoch() const noexcept**
Returns the nanoseconds since the beginning of the epoch.

Returns Elapsed nanoseconds since the beginning of the epoch for this timestamp.

inline bool **operator==(const** *SteadyTimestamp* *&other*) **const noexcept**
Compare two steady time stamps.

Returns true if the two time stamps are equal.

inline bool **operator!=(const** *SteadyTimestamp* *&other*) **const noexcept**
Compare two steady time stamps for inequality.

Returns true if the two time stamps are not equal.

Class SystemTimestamp

- Defined in file_include_opentelemetry_common_timestamp.h

Class Documentation

class opentelemetry::common::**SystemTimestamp**
A timepoint relative to the system clock epoch.

This is used for marking the beginning and end of an operation.

Public Functions

inline SystemTimestamp() noexcept
Initializes a system timestamp pointing to the start of the epoch.

template<class **Rep**, class **Period**>
inline explicit SystemTimestamp(const std::chrono::duration<*Rep*, *Period*> *&time_since_epoch*) **noexcept**
Initializes a system timestamp from a duration.

Parameters **time_since_epoch** – Time elapsed since the beginning of the epoch.

inline SystemTimestamp(const std::chrono::system_clock::time_point *&time_point*) **noexcept**
Initializes a system timestamp based on a point in time.

Parameters **time_point** – A point in time.

inline operator std::chrono::system_clock::time_point() const noexcept
Returns a time point for the time stamp.

Returns A time point corresponding to the time stamp.

inline std::chrono::nanoseconds **time_since_epoch() const noexcept**
Returns the nanoseconds since the beginning of the epoch.

Returns Elapsed nanoseconds since the beginning of the epoch for this timestamp.

inline bool **operator==(const** *SystemTimestamp* *&other*) **const noexcept**
Compare two steady time stamps.

Returns true if the two time stamps are equal.

inline bool **operator!=(const** *SystemTimestamp* *&other*) **const noexcept**
Compare two steady time stamps for inequality.

Returns true if the two time stamps are not equal.

Class DefaultSpan

- Defined in file_include_opentelemetry_trace_default_span.h

Inheritance Relationships

Base Type

- `public opentelemetry::trace::Span` (*Class Span*)

Class Documentation

class opentelemetry::trace::**DefaultSpan** : **public** opentelemetry::trace::*Span*

Public Functions

inline trace::*SpanContext* **GetContext**() **const noexcept**

inline bool **IsRecording**() **const noexcept**

inline void **SetAttribute**(nostd::string_view, **const** common::*AttributeValue*&) **noexcept**

inline void **AddEvent**(nostd::string_view) **noexcept**

inline void **AddEvent**(nostd::string_view, common::*SystemTimestamp*) **noexcept**

inline void **AddEvent**(nostd::string_view, common::*SystemTimestamp*, **const** common::*KeyValueIterable*&) **noexcept**

inline void **AddEvent**(nostd::string_view *name*, **const** common::*KeyValueIterable* &*attributes*) **noexcept**

inline void **SetStatus**(*StatusCode*, nostd::string_view) **noexcept**

inline void **UpdateName**(nostd::string_view) **noexcept**

inline void **End**(**const** *EndSpanOptions*& = {}) **noexcept**

inline nostd::string_view **ToString**()

inline **DefaultSpan**(*SpanContext span_context*)

inline **DefaultSpan**(*DefaultSpan* &&*spn*)

inline **DefaultSpan**(**const** *DefaultSpan* &*spn*)

Public Static Functions

static inline *DefaultSpan* **GetInvalid**()

Class DefaultTracer

- Defined in file_include_opentelemetry_trace_default_tracer.h

Inheritance Relationships

Base Type

- public opentelemetry::trace::Tracer (*Class Tracer*)

Class Documentation

class opentelemetry::trace::**DefaultTracer** : **public** opentelemetry::trace::*Tracer*

Public Functions

~DefaultTracer() = default

inline nostd::unique_ptr< Span > StartSpan (nostd::string_view name, const common::Key
Starts a span.

Optionally sets attributes at *Span* creation from the given key/value pairs.

Attributes will be processed in order, previous attributes with the same key will be overwritten.

inline void ForceFlushWithMicroseconds (uint64_t timeout) override noexcept

inline void CloseWithMicroseconds (uint64_t timeout) override noexcept

Class NoopSpan

- Defined in file_include_opentelemetry_trace_noop.h

Inheritance Relationships

Base Type

- public opentelemetry::trace::Span (*Class Span*)

Class Documentation

class opentelemetry::trace::**NoopSpan** : **public** opentelemetry::trace::*Span*
No-op implementation of *Span*. This class should not be used directly.

Public Functions

inline explicit NoopSpan(**const** std::shared_ptr<*Tracer*> &*tracer*) **noexcept**

inline virtual void **SetAttribute**(nostd::string_view, **const** common::*AttributeValue*&) **noexcept override**

inline virtual void **AddEvent**(nostd::string_view) **noexcept override**

inline virtual void **AddEvent**(nostd::string_view, common::*SystemTimestamp*) **noexcept override**

inline virtual void **AddEvent**(nostd::string_view, common::*SystemTimestamp*, **const** common::*KeyValueIterable*&) **noexcept override**

inline virtual void **SetStatus**(*StatusCode*, nostd::string_view) **noexcept override**

inline virtual void **UpdateName**(nostd::string_view) **noexcept override**

inline virtual void **End**(**const** *EndSpanOptions*&) **noexcept override**
Mark the end of the *Span*. Only the timing of the first End call for a given *Span* will be recorded, and implementations are free to ignore all further calls.

Parameters **options** – can be used to manually define span properties like the end timestamp

inline virtual bool **IsRecording**() **const noexcept override**

inline virtual *SpanContext* **GetContext**() **const noexcept override**

Class NoopTracer

- Defined in file_include_opentelemetry_trace_noop.h

Inheritance Relationships

Base Types

- public opentelemetry::trace::Tracer (*Class Tracer*)
- public std::enable_shared_from_this< NoopTracer >

Class Documentation

class opentelemetry::trace::**NoopTracer** : **public** opentelemetry::trace::*Tracer*, **public** std::enable_shared_from_th
No-op implementation of *Tracer*.

Public Functions

inline virtual nostd::shared_ptr<*Span*> **StartSpan**(nostd::string_view, **const** common::*KeyValueIterable*&, **const** *SpanContextKeyValueIterable*&, **const** *StartSpanOptions*&) **noexcept override**

Starts a span.

Optionally sets attributes at *Span* creation from the given key/value pairs.

Attributes will be processed in order, previous attributes with the same key will be overwritten.

inline virtual void **ForceFlushWithMicroseconds**(uint64_t) **noexcept override**

inline virtual void **CloseWithMicroseconds**(uint64_t) **noexcept override**

Class NoopTracerProvider

- Defined in file_include_opentelemetry_trace_noop.h

Inheritance Relationships

Base Type

- `public opentelemetry::trace::TracerProvider` (*Class TracerProvider*)

Class Documentation

class `opentelemetry::trace::`**NoopTracerProvider** : **public** opentelemetry::trace::*TracerProvider*
No-op implementation of a *TracerProvider*.

Public Functions

inline **NoopTracerProvider**()

inline **virtual** nostd::shared_ptr<opentelemetry::trace::*Tracer*> **GetTracer**(nostd::string_view *library_name*, nostd::string_view *library_version*) **override**

Gets or creates a named tracer instance.

Optionally a version can be passed to create a named and versioned tracer instance.

Class NullSpanContext

- Defined in file_include_opentelemetry_trace_span_context_kv_iterable.h

Inheritance Relationships

Base Type

- `public opentelemetry::trace::SpanContextKeyValueIterable` (*Class SpanContextKeyValueIterable*)

Class Documentation

class `opentelemetry::trace::`**NullSpanContext** : **public** opentelemetry::trace::*SpanContextKeyValueIterable*
Null *Span* context that does not carry any information.

Public Functions

inline virtual bool **ForEachKeyValue**(nostd::function_ref<bool) *SpanContext*, **const** opentelemetry::common::*KeyValueIterable*& > **const noexcept override**Iterate over SpanContext/key-value pairs

Parameters **callback** – a callback to invoke for each key-value for each SpanContext. If the callback returns false, the iteration is aborted.

Returns true if every SpanContext/key-value pair was iterated over

inline virtual size_t **size**() **const noexcept override**

Returns the number of key-value pairs

Class B3Propagator

- Defined in file_include_opentelemetry_trace_propagation_b3_propagator.h

Inheritance Relationships

Base Type

- `public opentelemetry::trace::propagation::B3PropagatorExtractor`

Class Documentation

class `opentelemetry::trace::propagation::`**B3Propagator** : **public** opentelemetry::trace::propagation::*B3Propa*

Public Functions

inline void **Inject**(opentelemetry::context::propagation::TextMapCarrier &*carrier*, **const** *context*::Context &*context*) **noexcept override**

Class B3PropagatorExtractor

- Defined in file_include_opentelemetry_trace_propagation_b3_propagator.h

Inheritance Relationships

Base Type

- `public TextMapPropagator`

Derived Types

- `public opentelemetry::trace::propagation::B3Propagator` (*Class B3Propagator*)
- `public opentelemetry::trace::propagation::B3PropagatorMultiHeader` (*Class B3PropagatorMultiHeader*)

Class Documentation

class `opentelemetry::trace::propagation::B3PropagatorExtractor` : **public** TextMapPropagator

Subclassed by *opentelemetry::trace::propagation::B3Propagator*, *opentelemetry::trace::propagation::B3PropagatorMultiHeader*

Public Functions

inline *context*::Context **Extract** (**const** opentelemetry::context::propagation::TextMapCarrier &*carrier*, *context*::Context &*context*) **noexcept override**

Public Static Functions

static inline *TraceId* **TraceIdFromHex** (nostd::string_view *trace_id*)

static inline *SpanId* **SpanIdFromHex** (nostd::string_view *span_id*)

static inline *TraceFlags* **TraceFlagsFromHex** (nostd::string_view *trace_flags*)

Class B3PropagatorMultiHeader

- Defined in file_include_opentelemetry_trace_propagation_b3_propagator.h

Inheritance Relationships

Base Type

- `public opentelemetry::trace::propagation::B3PropagatorExtractor`

Class Documentation

class `opentelemetry::trace::propagation::B3PropagatorMultiHeader` : **public** opentelemetry::trace::propa

Public Functions

inline void **Inject**(opentelemetry::context::propagation::TextMapCarrier &*carrier*, **const** *context*::Context &*context*) **noexcept override**

Class HttpTraceContext

- Defined in file_include_opentelemetry_trace_propagation_http_trace_context.h

Inheritance Relationships

Base Type

- `public TextMapPropagator`

Class Documentation

class `opentelemetry::trace::propagation::HttpTraceContext` : **public** TextMapPropagator

Public Functions

inline void **Inject**(opentelemetry::context::propagation::TextMapCarrier &*carrier*, **const** *context*::Context &*context*) **noexcept override**

inline *context*::Context **Extract**(**const** opentelemetry::context::propagation::TextMapCarrier &*carrier*, *context*::Context &*context*) **noexcept override**

Public Static Functions

static inline *TraceId* **TraceIdFromHex**(nostd::string_view *trace_id*)

static inline *SpanId* **SpanIdFromHex**(nostd::string_view *span_id*)

static inline *TraceFlags* **TraceFlagsFromHex**(nostd::string_view *trace_flags*)

Class JaegerPropagator

- Defined in file_include_opentelemetry_trace_propagation_jaeger.h

Inheritance Relationships

Base Type

- `public TextMapPropagator`

Class Documentation

class opentelemetry::trace::propagation::**JaegerPropagator** : **public** TextMapPropagator

Public Functions

inline void **Inject**(*context*::propagation::TextMapCarrier &*carrier*, **const** *context*::Context &*context*) **noexcept override**

inline *context*::Context **Extract**(**const** *context*::propagation::TextMapCarrier &*carrier*, *context*::Context &*context*) **noexcept override**

Class Provider

- Defined in file_include_opentelemetry_trace_provider.h

Class Documentation

class opentelemetry::trace::**Provider**

Stores the singleton global *TracerProvider*.

Public Static Functions

static inline nostd::shared_ptr<*TracerProvider*> **GetTracerProvider**() **noexcept**

Returns the singleton *TracerProvider*.

By default, a no-op *TracerProvider* is returned. This will never return a nullptr *TracerProvider*.

static inline void **SetTracerProvider**(nostd::shared_ptr<*TracerProvider*> *tp*) **noexcept**

Changes the singleton *TracerProvider*.

Class Scope

- Defined in file_include_opentelemetry_trace_scope.h

Class Documentation

class opentelemetry::trace::**Scope**

Controls how long a span is active.

On creation of the *Scope* object, the given span is set to the currently active span. On destruction, the given span is ended and the previously active span will be the currently active span again.

Public Functions

inline Scope (**const** nostd::shared_ptr<*Span*> &*span*) **noexcept**
Initialize a new scope.

Parameters **span** – the given span will be set as the currently active span.

Class Span

- Defined in file_include_opentelemetry_trace_span.h

Inheritance Relationships

Derived Types

- `public opentelemetry::trace::DefaultSpan` (*Class DefaultSpan*)
- `public opentelemetry::trace::NoopSpan` (*Class NoopSpan*)

Class Documentation

class `opentelemetry::trace::`**Span**
A *Span* represents a single operation within a Trace.

Subclassed by *opentelemetry::trace::DefaultSpan*, *opentelemetry::trace::NoopSpan*

Public Functions

Span () = default

virtual ~Span () = default

Span (**const** *Span*&) = delete

Span (*Span*&&) = delete

Span &**operator=** (**const** *Span*&) = delete

Span &**operator=** (*Span*&&) = delete

virtual void **SetAttribute** (nostd::string_view *key*, **const** common::*AttributeValue* &*value*) **noexcept** = 0

virtual void **AddEvent** (nostd::string_view *name*) **noexcept** = 0

virtual void **AddEvent** (nostd::string_view *name*, common::*SystemTimestamp* *timestamp*) **noexcept** = 0

virtual void **AddEvent** (nostd::string_view *name*, common::*SystemTimestamp* *timestamp*, **const** common::*KeyValueIterable* &*attributes*) **noexcept** = 0

inline virtual void **AddEvent** (nostd::string_view *name*, **const** common::*KeyValueIterable* &*attributes*) **noexcept**

template<class **T**, nostd::enable_if_t<common::detail::is_key_value_iterable<*T*>::value>* = nullptr>
inline void **AddEvent** (nostd::string_view *name*, common::*SystemTimestamp* *timestamp*, **const** *T* &*attributes*) **noexcept**

template<class **T**, nostd::enable_if_t<common::detail::is_key_value_iterable<*T*>::value>* = nullptr>

inline void **AddEvent**(nostd::string_view *name*, **const** *T* &*attributes*) **noexcept**

inline void **AddEvent**(nostd::string_view *name*, common::*SystemTimestamp* *timestamp*, std::initializer_list<std::pair<nostd::string_view, common::*AttributeValue*>> *attributes*) **noexcept**

inline void **AddEvent**(nostd::string_view *name*, std::initializer_list<std::pair<nostd::string_view, common::*AttributeValue*>> *attributes*) **noexcept**

virtual void **SetStatus**(*StatusCode* *code*, nostd::string_view *description* = "") **noexcept** = 0

virtual void **UpdateName**(nostd::string_view *name*) **noexcept** = 0

virtual void **End**(**const** *EndSpanOptions* &*options* = {}) **noexcept** = 0

Mark the end of the *Span*. Only the timing of the first End call for a given *Span* will be recorded, and implementations are free to ignore all further calls.

Parameters **options** – can be used to manually define span properties like the end timestamp

virtual trace::*SpanContext* **GetContext**() **const** **noexcept** = 0

virtual bool **IsRecording**() **const** **noexcept** = 0

Class SpanContext

- Defined in file_include_opentelemetry_trace_span_context.h

Class Documentation

class opentelemetry::trace::**SpanContext**

Public Functions

inline **SpanContext**(bool *sampled_flag*, bool *is_remote*)

inline **SpanContext**(*TraceId* *trace_id*, *SpanId* *span_id*, *TraceFlags* *trace_flags*, bool *is_remote*, nostd::shared_ptr<*TraceState*> *trace_state* = *TraceState*::*GetDefault*()) **noexcept**

SpanContext(**const** *SpanContext* &*ctx*) = default

inline bool **IsValid**() **const** **noexcept**

inline **const** trace_api::TraceFlags &**trace_flags**() **const** **noexcept**

inline **const** trace_api::TraceId &**trace_id**() **const** **noexcept**

inline **const** trace_api::SpanId &**span_id**() **const** **noexcept**

inline **const** nostd::shared_ptr<trace_api::TraceState> **trace_state**() **const** **noexcept**

inline bool **operator==**(**const** *SpanContext* &*that*) **const** **noexcept**

SpanContext &**operator=**(**const** *SpanContext* &*ctx*) = default

inline bool **IsRemote**() **const** **noexcept**

inline bool **IsSampled**() **const** **noexcept**

Public Static Functions

static inline *SpanContext* **GetInvalid**()

Class SpanContextKeyValueIterable

- Defined in file_include_opentelemetry_trace_span_context_kv_iterable.h

Inheritance Relationships

Derived Type

- public opentelemetry::trace::NullSpanContext (*Class NullSpanContext*)

Class Documentation

class opentelemetry::trace::**SpanContextKeyValueIterable**

Supports internal iteration over a collection of SpanContext/key-value pairs.

Subclassed by *opentelemetry::trace::NullSpanContext*

Public Functions

virtual ~SpanContextKeyValueIterable() = default

virtual bool **ForEachKeyValue**(nostd::function_ref<bool(*SpanContext*, **const** opentelemetry::common::*KeyValueIterable*&)> *callback*) **const noexcept** = 0

Iterate over SpanContext/key-value pairs

Parameters **callback** – a callback to invoke for each key-value for each SpanContext. If the callback returns false, the iteration is aborted.

Returns true if every SpanContext/key-value pair was iterated over

virtual size_t **size**() **const noexcept** = 0

Returns the number of key-value pairs

Class SpanId

- Defined in file_include_opentelemetry_trace_span_id.h

Class Documentation

class opentelemetry::trace::**SpanId**

Public Functions

inline SpanId() **noexcept**

inline explicit SpanId(nostd::span<**const** uint8_t, *kSize*> *id*) **noexcept**

inline void **ToLowerBase16**(nostd::span<char, 2 * *kSize*> *buffer*) **const noexcept**

inline nostd::span<**const** uint8_t, *kSize*> **Id**() **const noexcept**

inline bool **operator==**(**const** *SpanId* &*that*) **const noexcept**

inline bool **operator!=**(**const** *SpanId* &*that*) **const noexcept**

inline bool **IsValid**() **const noexcept**

inline void **CopyBytesTo**(nostd::span<uint8_t, *kSize*> *dest*) **const noexcept**

Public Static Attributes

static constexpr int **kSize** = 8

Class TraceFlags

- Defined in file_include_opentelemetry_trace_trace_flags.h

Class Documentation

class opentelemetry::trace::**TraceFlags**

Public Functions

inline TraceFlags() **noexcept**

inline explicit TraceFlags(uint8_t *flags*) **noexcept**

inline bool **IsSampled**() **const noexcept**

inline void **ToLowerBase16**(nostd::span<char, 2> *buffer*) **const noexcept**

inline uint8_t **flags**() **const noexcept**

inline bool **operator==**(**const** *TraceFlags* &*that*) **const noexcept**

inline bool **operator!=**(**const** *TraceFlags* &*that*) **const noexcept**

inline void **CopyBytesTo**(nostd::span<uint8_t, 1> *dest*) **const noexcept**

Public Static Attributes

static constexpr uint8_t **kIsSampled** = 1

Class TraceId

- Defined in file_include_opentelemetry_trace_trace_id.h

Class Documentation

class opentelemetry::trace::**TraceId**

Public Functions

inline TraceId() **noexcept**

inline explicit TraceId(nostd::span<**const** uint8_t, *kSize*> *id*) **noexcept**

inline void **ToLowerBase16**(nostd::span<char, 2 * *kSize*> *buffer*) **const noexcept**

inline nostd::span<**const** uint8_t, *kSize*> **Id**() **const noexcept**

inline bool **operator==**(**const** *TraceId* &*that*) **const noexcept**

inline bool **operator!=**(**const** *TraceId* &*that*) **const noexcept**

inline bool **IsValid**() **const noexcept**

inline void **CopyBytesTo**(nostd::span<uint8_t, *kSize*> *dest*) **const noexcept**

Public Static Attributes

static constexpr int **kSize** = 16

Class Tracer

- Defined in file_include_opentelemetry_trace_tracer.h

Inheritance Relationships

Derived Types

- public opentelemetry::trace::DefaultTracer (*Class DefaultTracer*)
- public opentelemetry::trace::NoopTracer (*Class NoopTracer*)

Class Documentation

class opentelemetry::trace::**Tracer**

Handles span creation and in-process context propagation.

This class provides methods for manipulating the context, creating spans, and controlling spans' lifecycles.

Subclassed by *opentelemetry::trace::DefaultTracer*, *opentelemetry::trace::NoopTracer*

Public Functions

virtual ~Tracer() = default

virtual nostd::shared_ptr<*Span*> **StartSpan**(nostd::string_view *name*, **const** common::*KeyValueIterable* &*attributes*, **const** *SpanContextKeyValueIterable* &*links*, **const** *StartSpanOptions* &*options* = {}) **noexcept** = 0

Starts a span.

Optionally sets attributes at *Span* creation from the given key/value pairs.

Attributes will be processed in order, previous attributes with the same key will be overwritten.

inline nostd::shared_ptr<*Span*> **StartSpan**(nostd::string_view *name*, **const** *StartSpanOptions* &*options* = {}) **noexcept**

template<class **T**, nostd::enable_if_t<common::detail::is_key_value_iterable<*T*>::value>* = nullptr>
inline nostd::shared_ptr<*Span*> **StartSpan**(nostd::string_view *name*, **const** *T* &*attributes*, **const** *StartSpanOptions* &*options* = {}) **noexcept**

inline nostd::shared_ptr<*Span*> **StartSpan**(nostd::string_view *name*, **const** common::*KeyValueIterable* &*attributes*, **const** *StartSpanOptions* &*options* = {}) **noexcept**

template<class **T**, class **U**, nostd::enable_if_t<common::detail::is_key_value_iterable<*T*>::value>* = nullptr, nostd::enable_if_t<c
inline nostd::shared_ptr<*Span*> **StartSpan**(nostd::string_view *name*, **const** *T* &*attributes*, **const** *U* &*links*, **const** *StartSpanOptions* &*options* = {}) **noexcept**

inline nostd::shared_ptr<*Span*> **StartSpan**(nostd::string_view *name*, std::initializer_list<std::pair<nostd::string_view, common::*AttributeValue*>> *attributes*, **const** *StartSpanOptions* &*options* = {}) **noexcept**

template<class **T**, nostd::enable_if_t<common::detail::is_key_value_iterable<*T*>::value>* = nullptr>
inline nostd::shared_ptr<*Span*> **StartSpan**(nostd::string_view *name*, **const** *T* &*attributes*, std::initializer_list<std::pair<*SpanContext*, std::initializer_list<std::pair<nostd::string_view, common::*AttributeValue*>>>> *links*, **const** *StartSpanOptions* &*options* = {}) **noexcept**

template<class **T**, nostd::enable_if_t<common::detail::is_key_value_iterable<*T*>::value>* = nullptr>
inline nostd::shared_ptr<*Span*> **StartSpan**(nostd::string_view *name*, std::initializer_list<std::pair<nostd::string_view, common::*AttributeValue*>> *attributes*, **const** *T* &*links*, **const** *StartSpanOptions* &*options* = {}) **noexcept**

inline nostd::shared_ptr<*Span*> **StartSpan**(nostd::string_view *name*, std::initializer_list<std::pair<nostd::string_view, common::*AttributeValue*>> *attributes*, std::initializer_list<std::pair<*SpanContext*, std::initializer_list<std::pair<nostd::string_view, common::*AttributeValue*>>>> *links*, **const** *StartSpanOptions* &*options* = {}) **noexcept**

template<class **Rep**, class **Period**>
inline void **ForceFlush**(std::chrono::duration<*Rep*, *Period*> *timeout*) **noexcept**
Force any buffered spans to flush.

Parameters **timeout** – to complete the flush

virtual void **ForceFlushWithMicroseconds**(uint64_t *timeout*) **noexcept** = 0

template<class **Rep**, class **Period**>
inline void **Close**(std::chrono::duration<*Rep*, *Period*> *timeout*) **noexcept**
ForceFlush any buffered spans and stop reporting spans.

Parameters **timeout** – to complete the flush

virtual void **CloseWithMicroseconds**(uint64_t *timeout*) **noexcept** = 0

Public Static Functions

static inline nostd::unique_ptr<*Scope*> **WithActiveSpan**(nostd::shared_ptr<*Span*> &*span*) **noexcept**
Set the active span. The span will remain active until the returned *Scope* object is destroyed.

Parameters **span** – the span that should be set as the new active span.

Returns a *Scope* that controls how long the span will be active.

static inline nostd::shared_ptr<*Span*> **GetCurrentSpan**() **noexcept**
Get the currently active span.

Returns the currently active span, or an invalid default span if no span is active.

Class TracerProvider

- Defined in file_include_opentelemetry_trace_tracer_provider.h

Inheritance Relationships

Derived Type

- `public opentelemetry::trace::NoopTracerProvider` (*Class NoopTracerProvider*)

Class Documentation

class opentelemetry::trace::**TracerProvider**

Creates new *Tracer* instances.

Subclassed by *opentelemetry::trace::NoopTracerProvider*

Public Functions

virtual ~TracerProvider() = default

virtual nostd::shared_ptr<*Tracer*> **GetTracer**(nostd::string_view *library_name*, nostd::string_view *library_version* = "") = 0

Gets or creates a named tracer instance.

Optionally a version can be passed to create a named and versioned tracer instance.

Class TraceState

- Defined in file_include_opentelemetry_trace_trace_state.h

Class Documentation

class opentelemetry::trace::**TraceState**

TraceState carries tracing-system specific context in a list of key-value pairs. *TraceState* allows different vendors to propagate additional information and inter-operate with their legacy id formats.

For more information, see the W3C Trace Context specification: https://www.w3.org/TR/trace-context

Public Functions

inline std::string **ToHeader**()

Creates a w3c tracestate header from *TraceState* object

inline bool **Get**(nostd::string_view *key*, std::string &*value*) **const noexcept**

Returns `value` associated with `key` passed as argument Returns empty string if key is invalid or not found

inline nostd::shared_ptr<*TraceState*> **Set**(**const** nostd::string_view &*key*, **const** nostd::string_view &*value*)

Returns shared_ptr of `new` *TraceState* object with following mutations applied to the existing instance: Update Key value: The updated value must be moved to beginning of List Add : The new key-value pair SHOULD be added to beginning of List

If the provided key-value pair is invalid, or results in transtate that violates the tracecontext specification, empty *TraceState* instance will be returned.

If the existing object has maximum list members, it's copy is returned.

inline nostd::shared_ptr<*TraceState*> **Delete**(**const** nostd::string_view &*key*)

Returns shared_ptr to a `new` *TraceState* object after removing the attribute with given key (if present)

Returns empty *TraceState* object if key is invalid

Returns copy of original *TraceState* object if key is not present (??)

inline bool **Empty**() **const noexcept**

inline bool **GetAllEntries**(nostd::function_ref<bool) nostd::string_view, nostd::string_view > *callback* **const noexcept**

Public Static Functions

static inline nostd::shared_ptr<*TraceState*> **GetDefault**()

static inline nostd::shared_ptr<*TraceState*> **FromHeader**(nostd::string_view *header*)
Returns shared_ptr to a newly created *TraceState* parsed from the header provided.

Parameters **header** – Encoding of the tracestate header defined by the W3C Trace Context specification https://www.w3.org/TR/trace-context/

Returns *TraceState* A new *TraceState* instance or DEFAULT

static inline bool **IsValidKey**(nostd::string_view *key*)
Returns whether key is a valid key. See https://www.w3.org/TR/trace-context/#key Identifiers MUST begin with a lowercase letter or a digit, and can only contain lowercase letters (a-z), digits (0-9), underscores (_), dashes (-), asterisks (*), and forward slashes (/). For multi-tenant vendor scenarios, an at sign (@) can be used to prefix the vendor name.

static inline bool **IsValidValue**(nostd::string_view *value*)
Returns whether value is a valid value. See https://www.w3.org/TR/trace-context/#value The value is an opaque string containing up to 256 printable ASCII (RFC0020) characters ((i.e., the range 0x20 to 0x7E) except comma , and equal =)

Public Static Attributes

static constexpr int **kKeyMaxSize** = 256

static constexpr int **kValueMaxSize** = 256

static constexpr int **kMaxKeyValuePairs** = 32

static constexpr auto **kKeyValueSeparator** = '='

static constexpr auto **kMembersSeparator** = ','

3.3.3 Enums

Enum CanonicalCode

- Defined in file_include_opentelemetry_trace_canonical_code.h

Enum Documentation

enum opentelemetry::trace::**CanonicalCode**
Values:

enumerator OK
The operation completed successfully.

enumerator CANCELLED
The operation was cancelled (typically by the caller).

enumerator UNKNOWN
Unknown error. An example of where this error may be returned is if a Status value received from another address space belongs to an error-space that is not known in this address space. Also errors raised by APIs that do not return enough error information may be converted to this error.

enumerator INVALID_ARGUMENT
Client specified an invalid argument. Note that this differs from FAILED_PRECONDITION. INVALID_ARGUMENT indicates arguments that are problematic regardless of the state of the system (e.g., a malformed file name).

enumerator DEADLINE_EXCEEDED
Deadline expired before operation could complete. For operations that change the state of the system, this error may be returned even if the operation has completed successfully. For example, a successful response from a server could have been delayed long enough for the deadline to expire.

enumerator NOT_FOUND
Some requested entity (e.g., file or directory) was not found.

enumerator ALREADY_EXISTS
Some entity that we attempted to create (e.g., file or directory) already exists.

enumerator PERMISSION_DENIED
The caller does not have permission to execute the specified operation. PERMISSION_DENIED must not be used for rejections caused by exhausting some resource (use RESOURCE_EXHAUSTED instead for those errors). PERMISSION_DENIED must not be used if the caller cannot be identified (use UNAUTHENTICATED instead for those errors).

enumerator RESOURCE_EXHAUSTED
Some resource has been exhausted, perhaps a per-user quota, or perhaps the entire file system is out of space.

enumerator FAILED_PRECONDITION
Operation was rejected because the system is not in a state required for the operation's execution. For example, directory to be deleted may be non-empty, an rmdir operation is applied to a non-directory, etc.

A litmus test that may help a service implementor in deciding between FAILED_PRECONDITION, ABORTED, and UNAVAILABLE: (a) Use UNAVAILABLE if the client can retry just the failing call. (b) Use ABORTED if the client should retry at a higher-level (e.g., restarting a read-modify-write sequence). (c) Use FAILED_PRECONDITION if the client should not retry until the system state has been explicitly fixed. E.g., if an "rmdir" fails because the directory is non-empty, FAILED_PRECONDITION should be returned since the client should not retry unless they have first fixed up the directory by deleting files from it.

enumerator ABORTED
The operation was aborted, typically due to a concurrency issue like sequencer check failures, transaction aborts, etc.

See litmus test above for deciding between FAILED_PRECONDITION, ABORTED, and UNAVAILABLE.

enumerator OUT_OF_RANGE
Operation was attempted past the valid range. E.g., seeking or reading past end of file.

Unlike INVALID_ARGUMENT, this error indicates a problem that may be fixed if the system state changes. For example, a 32-bit file system will generate INVALID_ARGUMENT if asked to read at an offset that is not in the range [0,2^32-1], but it will generate OUT_OF_RANGE if asked to read from an offset past the current file size.

There is a fair bit of overlap between FAILED_PRECONDITION and OUT_OF_RANGE. We recommend using OUT_OF_RANGE (the more specific error) when it applies so that callers who are iterating through

a space can easily look for an OUT_OF_RANGE error to detect when they are done.

enumerator UNIMPLEMENTED
Operation is not implemented or not supported/enabled in this service.

enumerator INTERNAL
Internal errors. Means some invariants expected by underlying system has been broken. If you see one of these errors, something is very broken.

enumerator UNAVAILABLE
The service is currently unavailable. This is a most likely a transient condition and may be corrected by retrying with a backoff.

See litmus test above for deciding between FAILED_PRECONDITION, ABORTED, and UNAVAILABLE.

enumerator DATA_LOSS
Unrecoverable data loss or corruption.

enumerator UNAUTHENTICATED
The request does not have valid authentication credentials for the operation.

Enum SpanKind

- Defined in file_include_opentelemetry_trace_span.h

Enum Documentation

enum opentelemetry::trace::**SpanKind**
Values:

enumerator kInternal

enumerator kServer

enumerator kClient

enumerator kProducer

enumerator kConsumer

Enum StatusCode

- Defined in file_include_opentelemetry_trace_span.h

Enum Documentation

enum opentelemetry::trace::**StatusCode**
Values:

enumerator kUnset

enumerator kOk

enumerator kError

3.3.4 Functions

Function opentelemetry::trace::propagation::detail::HexToBinary

- Defined in file_include_opentelemetry_trace_propagation_detail_hex.h

Function Documentation

inline bool opentelemetry::trace::propagation::detail::**HexToBinary**(nostd::string_view *hex*, uint8_t **buffer*, size_t *buffer_size*)

Converts a hexadecimal to binary format if the hex string will fit the buffer. Smaller hex strings are left padded with zeroes.

Function opentelemetry::trace::propagation::detail::HexToInt

- Defined in file_include_opentelemetry_trace_propagation_detail_hex.h

Function Documentation

inline int8_t opentelemetry::trace::propagation::detail::**HexToInt**(char *c*)

Function opentelemetry::trace::propagation::detail::IsValidHex

- Defined in file_include_opentelemetry_trace_propagation_detail_hex.h

Function Documentation

inline bool opentelemetry::trace::propagation::detail::**IsValidHex**(nostd::string_view *s*)

Function opentelemetry::trace::propagation::detail::SplitString

- Defined in file_include_opentelemetry_trace_propagation_detail_string.h

Function Documentation

inline size_t opentelemetry::trace::propagation::detail::**SplitString**(nostd::string_view *s*, char *separator*, nostd::string_view **results*, size_t *count*)

Splits a string by separator, up to given buffer count words. Returns the amount of words the input was split into.

Function opentelemetry::trace::propagation::GetSpan

- Defined in file_include_opentelemetry_trace_propagation_detail_context.h

Function Documentation

inline nostd::shared_ptr<trace::*Span*> `opentelemetry::trace::propagation::`**GetSpan**(**const** *context*::Context &*context*)

Function opentelemetry::trace::propagation::SetSpan

- Defined in file_include_opentelemetry_trace_propagation_detail_context.h

Function Documentation

inline *context*::Context `opentelemetry::trace::propagation::`**SetSpan**(*context*::Context &*context*, nostd::shared_ptr<trace::*Span*> *span*)

Template Function opentelemetry::trace::to_span_ptr

- Defined in file_include_opentelemetry_trace_span.h

Function Documentation

template<class **SpanType**, class **TracerType**>
nostd::shared_ptr<trace::*Span*> `opentelemetry::trace::`**to_span_ptr**(*TracerType* **const** *objPtr*, nostd::string_view *name*, trace::*StartSpanOptions* &*options*)

3.3.5 Variables

Variable opentelemetry::trace::kSpanKey

- Defined in file_include_opentelemetry_trace_span.h

Variable Documentation

constexpr char opentelemetry::trace::**kSpanKey**[] = "active_span"

Variable opentelemetry::trace::propagation::detail::kHexDigits

- Defined in file_include_opentelemetry_trace_propagation_detail_hex.h

Variable Documentation

constexpr int8_t opentelemetry::trace::propagation::detail::**kHexDigits**[256] = {-1, -1, -1, -1, -1, -1, -1, -1,

Variable opentelemetry::trace::propagation::kB3CombinedHeader

- Defined in file_include_opentelemetry_trace_propagation_b3_propagator.h

Variable Documentation

static const nostd::string_view opentelemetry::trace::propagation::**kB3CombinedHeader** = "b3"

Variable opentelemetry::trace::propagation::kB3SampledHeader

- Defined in file_include_opentelemetry_trace_propagation_b3_propagator.h

Variable Documentation

static const nostd::string_view opentelemetry::trace::propagation::**kB3SampledHeader** = "X-B3-Sampled"

Variable opentelemetry::trace::propagation::kB3SpanIdHeader

- Defined in file_include_opentelemetry_trace_propagation_b3_propagator.h

Variable Documentation

static const nostd::string_view opentelemetry::trace::propagation::**kB3SpanIdHeader** = "X-B3-SpanId"

Variable opentelemetry::trace::propagation::kB3TraceIdHeader

- Defined in file_include_opentelemetry_trace_propagation_b3_propagator.h

Variable Documentation

static const nostd::string_view opentelemetry::trace::propagation::**kB3TraceIdHeader** = "X-B3-TraceId"

Variable opentelemetry::trace::propagation::kSpanIdHexStrLength

- Defined in file_include_opentelemetry_trace_propagation_b3_propagator.h

Variable Documentation

static const int opentelemetry::trace::propagation::**kSpanIdHexStrLength** = 16

Variable opentelemetry::trace::propagation::kSpanIdSize

- Defined in file_include_opentelemetry_trace_propagation_http_trace_context.h

Variable Documentation

static const size_t opentelemetry::trace::propagation::**kSpanIdSize** = 16

Variable opentelemetry::trace::propagation::kTraceFlagsSize

- Defined in file_include_opentelemetry_trace_propagation_http_trace_context.h

Variable Documentation

static const size_t opentelemetry::trace::propagation::**kTraceFlagsSize** = 2

Variable opentelemetry::trace::propagation::kTraceHeader

- Defined in file_include_opentelemetry_trace_propagation_jaeger.h

Variable Documentation

static const nostd::string_view opentelemetry::trace::propagation::**kTraceHeader** = "uber-trace-id"

Variable opentelemetry::trace::propagation::kTraceIdHexStrLength

- Defined in file_include_opentelemetry_trace_propagation_b3_propagator.h

Variable Documentation

static const int opentelemetry::trace::propagation::**kTraceIdHexStrLength** = 32

Variable opentelemetry::trace::propagation::kTraceIdSize

- Defined in file_include_opentelemetry_trace_propagation_http_trace_context.h

Variable Documentation

static const size_t opentelemetry::trace::propagation::**kTraceIdSize** = 32

Variable opentelemetry::trace::propagation::kTraceParent

- Defined in file_include_opentelemetry_trace_propagation_http_trace_context.h

Variable Documentation

static const nostd::string_view opentelemetry::trace::propagation::**kTraceParent** = "traceparent"

Variable opentelemetry::trace::propagation::kTraceParentSize

- Defined in file_include_opentelemetry_trace_propagation_http_trace_context.h

Variable Documentation

static const size_t opentelemetry::trace::propagation::**kTraceParentSize** = 55

Variable opentelemetry::trace::propagation::kTraceState

- Defined in file_include_opentelemetry_trace_propagation_http_trace_context.h

Variable Documentation

static const nostd::string_view opentelemetry::trace::propagation::**kTraceState** = "tracestate"

Variable opentelemetry::trace::propagation::kVersionSize

- Defined in file_include_opentelemetry_trace_propagation_http_trace_context.h

Variable Documentation

static const size_t `opentelemetry::trace::propagation::`**kVersionSize** = 2

3.3.6 Defines

Define HAVE_WORKING_REGEX

- Defined in file_include_opentelemetry_trace_trace_state.h

Define Documentation

HAVE_WORKING_REGEX

3.3.7 Typedefs

Typedef opentelemetry::common::AttributeValue

- Defined in file_include_opentelemetry_common_attribute_value.h

Typedef Documentation

using `opentelemetry::common::`**AttributeValue** = nostd::variant<bool, int32_t, int64_t, uint32_t, double, nostd::string_

OpenTelemetry signals can be enriched by adding attributes. The `AttributeValue` type is defined as a variant of all attribute value types the OpenTelemetry C++ API supports.

The following attribute value types are supported by the OpenTelemetry specification:

- Primitive types: string, boolean, double precision floating point (IEEE 754-1985) or signed 64 bit integer.
- Homogenous arrays of primitive type values.

> **Warning:**
>
> The OpenTelemetry C++ API currently supports several attribute value types that are not covered by the OpenTelemetry specification:
>
> - `uint64_t`
> - `nostd::span<const uint64_t>`
> - `nostd::span<uint8_t>`
>
> Those types are reserved for future use and currently should not be used. There are no guarantees around how those values are handled by exporters.

CHAPTER

FOUR

GETTING HELP

- Refer to opentelemetry.io for general information about OpenTelemetry.
- Refer to the OpenTelemetry C++ GitHub repository for further information and resources related to OpenTelemetry C++.
- For questions related to OpenTelemetry C++ that are not covered by the existing documentation, please ask away in GitHub discussions.
- Feel free to join the CNCF OpenTelemetry C++ Slack channel. If you are new, you can create a CNCF Slack account here.
- For bugs and feature requests, write a GitHub issue.

INDEX

H

HAVE_WORKING_REGEX (*C macro*), 36

O

opentelemetry::common::AttributeValue (*C++ type*), 36
opentelemetry::common::KeyValueIterable (*C++ class*), 9
opentelemetry::common::KeyValueIterable::~KeyValueIterable (*C++ function*), 9
opentelemetry::common::KeyValueIterable::ForEachKeyValue (*C++ function*), 9
opentelemetry::common::KeyValueIterable::size (*C++ function*), 9
opentelemetry::common::SteadyTimestamp (*C++ class*), 9
opentelemetry::common::SteadyTimestamp::operator std::chrono::steady_clock::time_point (*C++ function*), 10
opentelemetry::common::SteadyTimestamp::operator!= (*C++ function*), 10
opentelemetry::common::SteadyTimestamp::operator== (*C++ function*), 10
opentelemetry::common::SteadyTimestamp::SteadyTimestamp (*C++ function*), 10
opentelemetry::common::SteadyTimestamp::time_since_epoch (*C++ function*), 10
opentelemetry::common::SystemTimestamp (*C++ class*), 10
opentelemetry::common::SystemTimestamp::operator std::chrono::system_clock::time_point (*C++ function*), 11
opentelemetry::common::SystemTimestamp::operator!= (*C++ function*), 11
opentelemetry::common::SystemTimestamp::operator== (*C++ function*), 11
opentelemetry::common::SystemTimestamp::SystemTimestamp (*C++ function*), 11
opentelemetry::common::SystemTimestamp::time_since_epoch (*C++ function*), 11
opentelemetry::trace::CanonicalCode (*C++ enum*), 28
opentelemetry::trace::CanonicalCode::ABORTED (*C++ enumerator*), 29
opentelemetry::trace::CanonicalCode::ALREADY_EXISTS (*C++ enumerator*), 29
opentelemetry::trace::CanonicalCode::CANCELLED (*C++ enumerator*), 28
opentelemetry::trace::CanonicalCode::DATA_LOSS (*C++ enumerator*), 30
opentelemetry::trace::CanonicalCode::DEADLINE_EXCEE (*C++ enumerator*), 29
opentelemetry::trace::CanonicalCode::FAILED_PRECOND (*C++ enumerator*), 29
opentelemetry::trace::CanonicalCode::INTERNAL (*C++ enumerator*), 30
opentelemetry::trace::CanonicalCode::INVALID_ARGUME (*C++ enumerator*), 29
opentelemetry::trace::CanonicalCode::NOT_FOUND (*C++ enumerator*), 29
opentelemetry::trace::CanonicalCode::OK (*C++ enumerator*), 28
opentelemetry::trace::CanonicalCode::OUT_OF_RANGE (*C++ enumerator*), 29
opentelemetry::trace::CanonicalCode::PERMISSION_DEN (*C++ enumerator*), 29
opentelemetry::trace::CanonicalCode::RESOURCE_EXHAU (*C++ enumerator*), 29
opentelemetry::trace::CanonicalCode::UNAUTHENTICATE (*C++ enumerator*), 30
opentelemetry::trace::CanonicalCode::UNAVAILABLE (*C++ enumerator*), 30
opentelemetry::trace::CanonicalCode::UNIMPLEMENTED (*C++ enumerator*), 30
opentelemetry::trace::CanonicalCode::UNKNOWN (*C++ enumerator*), 28
opentelemetry::trace::DefaultSpan (*C++ class*), 12
opentelemetry::trace::DefaultSpan::AddEvent (*C++ function*), 12
opentelemetry::trace::DefaultSpan::DefaultSpan (*C++ function*), 12
opentelemetry::trace::DefaultSpan::End (*C++ function*), 12

opentelemetry::trace::DefaultSpan::GetContext (C++ function), 12
opentelemetry::trace::DefaultSpan::GetInvalid (C++ function), 12
opentelemetry::trace::DefaultSpan::IsRecording (C++ function), 12
opentelemetry::trace::DefaultSpan::SetAttribute (C++ function), 12
opentelemetry::trace::DefaultSpan::SetStatus (C++ function), 12
opentelemetry::trace::DefaultSpan::ToString (C++ function), 12
opentelemetry::trace::DefaultSpan::UpdateName (C++ function), 12
opentelemetry::trace::DefaultTracer (C++ class), 13
opentelemetry::trace::DefaultTracer::~DefaultTracer (C++ function), 13
opentelemetry::trace::EndSpanOptions (C++ struct), 8
opentelemetry::trace::EndSpanOptions::end_steady_time (C++ member), 8
opentelemetry::trace::kSpanKey (C++ member), 33
opentelemetry::trace::NoopSpan (C++ class), 13
opentelemetry::trace::NoopSpan::AddEvent (C++ function), 13
opentelemetry::trace::NoopSpan::End (C++ function), 13
opentelemetry::trace::NoopSpan::GetContext (C++ function), 14
opentelemetry::trace::NoopSpan::IsRecording (C++ function), 14
opentelemetry::trace::NoopSpan::NoopSpan (C++ function), 13
opentelemetry::trace::NoopSpan::SetAttribute (C++ function), 13
opentelemetry::trace::NoopSpan::SetStatus (C++ function), 13
opentelemetry::trace::NoopSpan::UpdateName (C++ function), 13
opentelemetry::trace::NoopTracer (C++ class), 14
opentelemetry::trace::NoopTracer::CloseWithMicroseconds (C++ function), 14
opentelemetry::trace::NoopTracer::ForceFlushWithMicroseconds (C++ function), 14
opentelemetry::trace::NoopTracer::StartSpan (C++ function), 14
opentelemetry::trace::NoopTracerProvider (C++ class), 15
opentelemetry::trace::NoopTracerProvider::GetTracer (C++ function), 15
opentelemetry::trace::NoopTracerProvider::NoopTracerProvider (C++ function), 15
opentelemetry::trace::NullSpanContext (C++ class), 15
opentelemetry::trace::NullSpanContext::ForEachKeyValue (C++ function), 16
opentelemetry::trace::NullSpanContext::size (C++ function), 16
opentelemetry::trace::propagation::B3Propagator (C++ class), 16
opentelemetry::trace::propagation::B3Propagator::Inject (C++ function), 16
opentelemetry::trace::propagation::B3PropagatorExtractor (C++ class), 17
opentelemetry::trace::propagation::B3PropagatorExtractor (C++ function), 17
opentelemetry::trace::propagation::B3PropagatorExtractor (C++ function), 17
opentelemetry::trace::propagation::B3PropagatorExtractor (C++ function), 17
opentelemetry::trace::propagation::B3PropagatorExtractor (C++ function), 17
opentelemetry::trace::propagation::B3PropagatorMultiHeaderExtractor (C++ class), 17
opentelemetry::trace::propagation::B3PropagatorMultiHeaderExtractor (C++ function), 18
opentelemetry::trace::propagation::detail::HexToBinary (C++ function), 31
opentelemetry::trace::propagation::detail::HexToInt (C++ function), 31
opentelemetry::trace::propagation::detail::IsValidHex (C++ function), 31
opentelemetry::trace::propagation::detail::kHexDigits (C++ member), 33
opentelemetry::trace::propagation::detail::SplitString (C++ function), 31
opentelemetry::trace::propagation::GetSpan (C++ function), 32
opentelemetry::trace::propagation::HttpTraceContext (C++ class), 18
opentelemetry::trace::propagation::HttpTraceContext (C++ function), 18
opentelemetry::trace::propagation::HttpTraceContext (C++ function), 18
opentelemetry::trace::propagation::HttpTraceContext (C++ function), 18
opentelemetry::trace::propagation::HttpTraceContext (C++ function), 18
opentelemetry::trace::propagation::JaegerPropagator (C++ class), 19
opentelemetry::trace::propagation::JaegerPropagator (C++ function), 19

opentelemetry::trace::propagation::JaegerPropagator::Inject (C++ function), 19
opentelemetry::trace::propagation::kB3CombinedHeader (C++ member), 33
opentelemetry::trace::propagation::kB3SampledHeader (C++ member), 33
opentelemetry::trace::propagation::kB3SpanIdHeader (C++ member), 33
opentelemetry::trace::propagation::kB3TraceIdHeader (C++ member), 34
opentelemetry::trace::propagation::kSpanIdHexStrLength (C++ member), 34
opentelemetry::trace::propagation::kSpanIdSize (C++ member), 34
opentelemetry::trace::propagation::kTraceFlagsHexStrLength (C++ member), 34
opentelemetry::trace::propagation::kTraceFlagsSize (C++ member), 34
opentelemetry::trace::propagation::kTraceIdHexStrLength (C++ member), 35
opentelemetry::trace::propagation::kTraceIdSize (C++ member), 35
opentelemetry::trace::propagation::kTraceParentHeader (C++ member), 35
opentelemetry::trace::propagation::kTraceParentSize (C++ member), 35
opentelemetry::trace::propagation::kTraceStateHeader (C++ member), 35
opentelemetry::trace::propagation::kVersionSize (C++ member), 36
opentelemetry::trace::propagation::SetSpan (C++ function), 32
opentelemetry::trace::Provider (C++ class), 19
opentelemetry::trace::Provider::GetTracerProvider (C++ function), 19
opentelemetry::trace::Provider::SetTracerProvider (C++ function), 19
opentelemetry::trace::Scope (C++ class), 19
opentelemetry::trace::Scope::Scope (C++ function), 20
opentelemetry::trace::Span (C++ class), 20
opentelemetry::trace::Span::~Span (C++ function), 20
opentelemetry::trace::Span::AddEvent (C++ function), 20, 21
opentelemetry::trace::Span::End (C++ function), 21
opentelemetry::trace::Span::GetContext (C++ function), 21
opentelemetry::trace::Span::IsRecording (C++ function), 21
opentelemetry::trace::Span::operator= (C++ function), 20
opentelemetry::trace::Span::SetAttribute (C++ function), 20
opentelemetry::trace::Span::SetStatus (C++ function), 21
opentelemetry::trace::Span::Span (C++ function), 20
opentelemetry::trace::Span::UpdateName (C++ function), 21
opentelemetry::trace::SpanContext (C++ class), 21
opentelemetry::trace::SpanContext::GetInvalid (C++ function), 22
opentelemetry::trace::SpanContext::IsRemote (C++ function), 21
opentelemetry::trace::SpanContext::IsSampled (C++ function), 21
opentelemetry::trace::SpanContext::IsValid (C++ function), 21
opentelemetry::trace::SpanContext::operator= (C++ function), 21
opentelemetry::trace::SpanContext::operator== (C++ function), 21
opentelemetry::trace::SpanContext::span_id (C++ function), 21
opentelemetry::trace::SpanContext::SpanContext (C++ function), 21
opentelemetry::trace::SpanContext::trace_flags (C++ function), 21
opentelemetry::trace::SpanContext::trace_id (C++ function), 21
opentelemetry::trace::SpanContext::trace_state (C++ function), 21
opentelemetry::trace::SpanContextKeyValueIterable (C++ class), 22
opentelemetry::trace::SpanContextKeyValueIterable:: (C++ function), 22
opentelemetry::trace::SpanContextKeyValueIterable:: (C++ function), 22
opentelemetry::trace::SpanContextKeyValueIterable:: (C++ function), 22
opentelemetry::trace::SpanId (C++ class), 22
opentelemetry::trace::SpanId::CopyBytesTo (C++ function), 23
opentelemetry::trace::SpanId::Id (C++ function), 23
opentelemetry::trace::SpanId::IsValid (C++ function), 23
opentelemetry::trace::SpanId::kSize (C++ member), 23
opentelemetry::trace::SpanId::operator!= (C++ function), 23
opentelemetry::trace::SpanId::operator== (C++ function), 23

opentelemetry::trace::SpanId::SpanId (C++ function), 23
opentelemetry::trace::SpanId::ToLowerBase16 (C++ function), 23
opentelemetry::trace::SpanKind (C++ enum), 30
opentelemetry::trace::SpanKind::kClient (C++ enumerator), 30
opentelemetry::trace::SpanKind::kConsumer (C++ enumerator), 30
opentelemetry::trace::SpanKind::kInternal (C++ enumerator), 30
opentelemetry::trace::SpanKind::kProducer (C++ enumerator), 30
opentelemetry::trace::SpanKind::kServer (C++ enumerator), 30
opentelemetry::trace::StartSpanOptions (C++ struct), 8
opentelemetry::trace::StartSpanOptions::kind (C++ member), 9
opentelemetry::trace::StartSpanOptions::parent (C++ member), 9
opentelemetry::trace::StartSpanOptions::start_system_time (C++ member), 9
opentelemetry::trace::StartSpanOptions::start_steady_time (C++ member), 9
opentelemetry::trace::StatusCode (C++ enum), 30
opentelemetry::trace::StatusCode::kError (C++ enumerator), 30
opentelemetry::trace::StatusCode::kOk (C++ enumerator), 30
opentelemetry::trace::StatusCode::kUnset (C++ enumerator), 30
opentelemetry::trace::to_span_ptr (C++ function), 32
opentelemetry::trace::TraceFlags (C++ class), 23
opentelemetry::trace::TraceFlags::CopyBytesTo (C++ function), 23
opentelemetry::trace::TraceFlags::flags (C++ function), 23
opentelemetry::trace::TraceFlags::IsSampled (C++ function), 23
opentelemetry::trace::TraceFlags::kIsSampled (C++ member), 24
opentelemetry::trace::TraceFlags::operator!= (C++ function), 23
opentelemetry::trace::TraceFlags::operator== (C++ function), 23
opentelemetry::trace::TraceFlags::ToLowerBase16 (C++ function), 23
opentelemetry::trace::TraceFlags::TraceFlags (C++ function), 23
opentelemetry::trace::TraceId (C++ class), 24
opentelemetry::trace::TraceId::CopyBytesTo (C++ function), 24
opentelemetry::trace::TraceId::Id (C++ function), 24
opentelemetry::trace::TraceId::IsValid (C++ function), 24
opentelemetry::trace::TraceId::kSize (C++ member), 24
opentelemetry::trace::TraceId::operator!= (C++ function), 24
opentelemetry::trace::TraceId::operator== (C++ function), 24
opentelemetry::trace::TraceId::ToLowerBase16 (C++ function), 24
opentelemetry::trace::TraceId::TraceId (C++ function), 24
opentelemetry::trace::Tracer (C++ class), 25
opentelemetry::trace::Tracer::~Tracer (C++ function), 25
opentelemetry::trace::Tracer::Close (C++ function), 26
opentelemetry::trace::Tracer::CloseWithMicroseconds (C++ function), 26
opentelemetry::trace::Tracer::ForceFlush (C++ function), 26
opentelemetry::trace::Tracer::ForceFlushWithMicrose (C++ function), 26
opentelemetry::trace::Tracer::GetCurrentSpan (C++ function), 26
opentelemetry::trace::Tracer::StartSpan (C++ function), 25
opentelemetry::trace::Tracer::WithActiveSpan (C++ function), 26
opentelemetry::trace::TracerProvider (C++ class), 27
opentelemetry::trace::TracerProvider::~TracerProvi (C++ function), 27
opentelemetry::trace::TracerProvider::GetTracer (C++ function), 27
opentelemetry::trace::TraceState (C++ class), 27
opentelemetry::trace::TraceState::Delete (C++ function), 27
opentelemetry::trace::TraceState::Empty (C++ function), 27
opentelemetry::trace::TraceState::FromHeader (C++ function), 28
opentelemetry::trace::TraceState::Get (C++ function), 27
opentelemetry::trace::TraceState::GetAllEntries (C++ function), 27

opentelemetry::trace::TraceState::GetDefault
(*C++ function*), 28
opentelemetry::trace::TraceState::IsValidKey
(*C++ function*), 28
opentelemetry::trace::TraceState::IsValidValue
(*C++ function*), 28
opentelemetry::trace::TraceState::kKeyMaxSize
(*C++ member*), 28
opentelemetry::trace::TraceState::kKeyValueSeparator
(*C++ member*), 28
opentelemetry::trace::TraceState::kMaxKeyValuePairs
(*C++ member*), 28
opentelemetry::trace::TraceState::kMembersSeparator
(*C++ member*), 28
opentelemetry::trace::TraceState::kValueMaxSize
(*C++ member*), 28
opentelemetry::trace::TraceState::Set
(*C++ function*), 27
opentelemetry::trace::TraceState::ToHeader
(*C++ function*), 27