

---

# OpenTelemetry C++

*Release 0.7.0*

OpenTelemetry authors

May 27, 2021



# OPENTELEMETRY C++ API

|          |                                |           |
|----------|--------------------------------|-----------|
| <b>1</b> | <b>Overview</b>                | <b>1</b>  |
| <b>2</b> | <b>Getting started</b>         | <b>3</b>  |
| <b>3</b> | <b>Reference documentation</b> | <b>5</b>  |
| <b>4</b> | <b>Getting help</b>            | <b>47</b> |
|          | <b>Index</b>                   | <b>49</b> |



## 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.



## 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 `Spans`.

#### 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. A span without a parent is called root span.

### 2.1.4 Create nested Spans

```
auto outer_span = tracer->StartSpan("Outer operation");
auto outer_scope = tracer->WithActiveSpan(outer_span);
{
    auto inner_span = tracer->StartSpan("Inner operation");
    auto inner_scope = tracer->WithActiveSpan(inner_span);
    // ... perform inner operation
    inner_span->End();
}
// ... perform outer operation
outer_span->End();
```

Spans can be nested, and have a parent-child relationship with other spans. When a given span is active, the newly created span inherits the active span's trace ID, and other context attributes.

### 2.1.5 Context Propagation

```
// set global propagator
opentelemetry::context::propagation::GlobalTextMapPropagator::SetGlobalPropagator(
    nostd::shared_ptr<opentelemetry::context::propagation::TextMapPropagator>(
        new opentelemetry::trace::propagation::HttpTraceContext()));

// get global propagator
HttpTextMapCarrier<opentelemetry::ext::http::client::Headers> carrier;
auto propagator =
    opentelemetry::context::propagation::GlobalTextMapPropagator::GetGlobalPropagator();

//inject context to headers
auto current_ctx = opentelemetry::context::RuntimeContext::GetCurrent();
propagator->Inject(carrier, current_ctx);

//Extract headers to context
auto current_ctx = opentelemetry::context::RuntimeContext::GetCurrent();
auto new_context = propagator->Extract(carrier, current_ctx);
auto remote_span = opentelemetry::trace::propagation::GetSpan(new_context);
```

Context contains the meta-data of the currently active Span including Span Id, Trace Id, and flags. Context Propagation is an important mechanism in distributed tracing to transfer this Context across service boundary often through HTTP headers. OpenTelemetry provides a text-based approach to propagate context to remote services using the W3C Trace Context HTTP headers.



## REFERENCE DOCUMENTATION

### 3.1 Class Hierarchy

### 3.2 File Hierarchy

### 3.3 Full API

#### 3.3.1 Namespaces

##### Namespace opentelemetry

##### Namespaces

- *Namespace opentelemetry::baggage*
- *Namespace opentelemetry::common*
- *Namespace opentelemetry::context*
- *Namespace opentelemetry::trace*

##### Namespace opentelemetry::baggage

##### Namespaces

- *Namespace opentelemetry::baggage::propagation*

##### Classes

- *Class Baggage*

## Namespace `opentelemetry::baggage::propagation`

### Classes

- *Class `BaggagePropagator`*

### Functions

- *Function `opentelemetry::baggage::propagation::GetBaggage`*
- *Function `opentelemetry::baggage::propagation::SetBaggage`*

### Variables

- *Variable `opentelemetry::baggage::propagation::kBaggageHeader`*

## Namespace `opentelemetry::common`

### Classes

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

### Typedefs

- *Typedef `opentelemetry::common::AttributeValue`*

## Namespace `opentelemetry::context`

### Namespaces

- *Namespace `opentelemetry::context::propagation`*

### Classes

- *Class `Context`*
- *Class `RuntimeContext`*
- *Class `RuntimeContextStorage`*
- *Class `ThreadLocalContextStorage`*
- *Class `Token`*

## Functions

- *Function `opentelemetry::context::GetDefaultStorage`*

## Typedefs

- *Typedef `opentelemetry::context::ContextValue`*

## Namespace `opentelemetry::context::propagation`

### Classes

- *Class `CompositePropagator`*
- *Class `GlobalTextMapPropagator`*
- *Class `NoOpPropagator`*
- *Class `TextMapCarrier`*
- *Class `TextMapPropagator`*

## 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 Baggage

- Defined in file\_include\_opentelemetry\_baggage\_baggage.h

## Class Documentation

**class** opentelemetry::baggage::Baggage

### Public Functions

**inline** Baggage ()

**inline** Baggage (size\_t size)

template<class T>

**inline** Baggage (const T &keys\_and\_values)

**inline** bool GetValue (nostd::string\_view key, std::string &value) **const**

**inline** nostd::shared\_ptr<Baggage> Set (const nostd::string\_view &key, const nostd::string\_view &value)

**inline** bool GetAllEntries (nostd::function\_ref<bool> nostd::string\_view, nostd::string\_view > callback) **const noexcept**

```
inline nostd::shared_ptr<Baggage> Delete (nostd::string_view key)
inline std::string ToHeader () const
```

### Public Static Functions

```
static inline nostd::shared_ptr<Baggage> GetDefault ()
static inline nostd::shared_ptr<Baggage> FromHeader (nostd::string_view header)
```

### Public Static Attributes

```
static constexpr size_t kMaxKeyValuePairs = 180
static constexpr size_t kMaxKeyValueSize = 4096
static constexpr size_t kMaxSize = 8192
static constexpr char kKeyValueSeparator = '='
static constexpr char kMembersSeparator = ','
static constexpr char kMetadataSeparator = ';'
```

## Class BaggagePropagator

- Defined in file\_include\_opentelemetry\_baggage\_propagation\_baggage\_propagator.h

## Inheritance Relationships

### Base Type

- public opentelemetry::context::propagation::TextMapPropagator

## Class Documentation

```
class opentelemetry::baggage::propagation::BaggagePropagator : public opentelemetry::context::propagation::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 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, com-  
mon::AttributeValue  
> *callback* **const noexcept** = 0 Iterate 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  
                        noexcept &time_point)
```

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 Context

- Defined in file\_include\_opentelemetry\_context\_context.h

## Class Documentation

**class** opentelemetry::context::Context

### Public Functions

**Context** () = default

template<class **T**>

**inline Context** (const *T* &*keys\_and\_values*)

**inline Context** (nstd::string\_view *key*, *ContextValue* *value*)

template<class **T**>

**inline Context SetValue** (*T* &*values*) **noexcept**

**inline Context SetValue** (nstd::string\_view *key*, *ContextValue* *value*) **noexcept**

**inline context::ContextValue GetValue** (const nstd::string\_view *key*) **const noexcept**

**inline bool HasKey** (const nstd::string\_view *key*) **const noexcept**

**inline bool operator==** (const *Context* &*other*) **const noexcept**

## Class CompositePropagator

- Defined in file\_include\_opentelemetry\_context\_propagation\_composite\_propagator.h

## Inheritance Relationships

### Base Type

- public opentelemetry::context::propagation::TextMapPropagator

## Class Documentation

```
class opentelemetry::context::propagation::CompositePropagator : public opentelemetry::context::propagation::Propagator
```

### Public Functions

```
inline CompositePropagator (std::vector<std::unique_ptr<TextMapPropagator>> propagators)
```

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

Run each of the configured propagators with the given context and carrier. Propagators are run in the order they are configured, so if multiple propagators write the same carrier key, the propagator later in the list will “win”.

#### Parameters

- **carrier** – Carrier into which context will be injected
- **context** – Context to inject

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

Run each of the configured propagators with the given context and carrier. Propagators are run in the order they are configured, so if multiple propagators write the same context key, the propagator later in the list will “win”.

#### Parameters

- **carrier** – Carrier from which to extract context
- **context** – Context to add values to

## Class GlobalTextMapPropagator

- Defined in file\_include\_opentelemetry\_context\_propagation\_global\_propagator.h

## Class Documentation

```
class opentelemetry::context::propagation::GlobalTextMapPropagator
```

### Public Static Functions

```
static inline nostd::shared_ptr<TextMapPropagator> GetGlobalPropagator () noexcept
```

```
static inline void SetGlobalPropagator (nostd::shared_ptr<TextMapPropagator> prop)  
                                noexcept
```

## Class NoOpPropagator

- Defined in file\_include\_opentelemetry\_context\_propagation\_noop\_propagator.h

## Inheritance Relationships

### Base Type

- `public opentelemetry::context::propagation::TextMapPropagator`

## Class Documentation

**class** `opentelemetry::context::propagation::NoOpPropagator` : **public** `opentelemetry::context::propagation::TextMapPropagator`  
No-op implementation TextMapPropagator

### Public Functions

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

Noop extract function does nothing and returns the input context

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

Noop inject function does nothing

## Class TextMapCarrier

- Defined in file\_include\_opentelemetry\_context\_propagation\_text\_map\_propagator.h

## Class Documentation

**class** `opentelemetry::context::propagation::TextMapCarrier`

### Public Functions

**virtual** `nstd::string_view` **Get** (`nstd::string_view` *key*) **const noexcept = 0**

**virtual** **void** **Set** (`nstd::string_view` *key*, `nstd::string_view` *value*) **noexcept = 0**

## Class TextMapPropagator

- Defined in file\_include\_opentelemetry\_context\_propagation\_text\_map\_propagator.h

## Inheritance Relationships

### Derived Types

- `public opentelemetry::baggage::propagation::BaggagePropagator` (*Class [BaggagePropagator](#)*)
- `public opentelemetry::context::propagation::CompositePropagator` (*Class [CompositePropagator](#)*)
- `public opentelemetry::context::propagation::NoOpPropagator` (*Class [NoOpPropagator](#)*)
- `public opentelemetry::trace::propagation::B3PropagatorExtractor` (*Class [B3PropagatorExtractor](#)*)
- `public opentelemetry::trace::propagation::HttpTraceContext` (*Class [HttpTraceContext](#)*)
- `public opentelemetry::trace::propagation::JaegerPropagator` (*Class [JaegerPropagator](#)*)

### Class Documentation

**class** `opentelemetry::context::propagation::TextMapPropagator`  
 Subclassed by `opentelemetry::baggage::propagation::BaggagePropagator`, `opentelemetry::context::propagation::CompositePropagator`, `opentelemetry::context::propagation::NoOpPropagator`, `opentelemetry::trace::propagation::B3PropagatorExtractor`, `opentelemetry::trace::propagation::HttpTraceContext`, `opentelemetry::trace::propagation::JaegerPropagator`

#### Public Functions

```
virtual context::Context Extract(const TextMapCarrier &carrier, context::Context &context)
    noexcept = 0
virtual void Inject(TextMapCarrier &carrier, const context::Context &context) noexcept =
    0
```

### Class RuntimeContext

- Defined in file `include/opentelemetry_context_runtime_context.h`

### Class Documentation

**class** `opentelemetry::context::RuntimeContext`

## Public Static Functions

```
static inline Context GetCurrent () noexcept
static inline nostd::unique_ptr<Token> Attach (const Context &context) noexcept
static inline bool Detach (Token &token) noexcept
static inline Context SetValue (nostd::string_view key, const ContextValue &value, Context
                               *context = nullptr) noexcept
static inline ContextValue GetValue (nostd::string_view key, Context *context = nullptr)
                               noexcept
static inline void SetRuntimeContextStorage (nostd::shared_ptr<RuntimeContextStorage>
                                             storage) noexcept
```

Provide a custom runtime context storage.

This provides a possibility to override the default thread-local runtime context storage. This has to be set before any spans are created by the application, otherwise the behavior is undefined.

**Parameters** *storage* – a custom runtime context storage

## Class RuntimeContextStorage

- Defined in file\_include\_opentelemetry\_context\_runtime\_context.h

## Inheritance Relationships

### Derived Type

- public opentelemetry::context::ThreadLocalContextStorage (*Class ThreadLocalContextStorage*)

## Class Documentation

**class** opentelemetry::context::RuntimeContextStorage

*RuntimeContextStorage* is used by RuntimeContext to store Context frames.

Custom context management strategies can be implemented by deriving from this class and passing an initialized *RuntimeContextStorage* object to RuntimeContext::SetRuntimeContextStorage.

Subclassed by *opentelemetry::context::ThreadLocalContextStorage*

## Public Functions

**virtual** *Context* GetCurrent () noexcept = 0

Return the current context.

**Returns** the current context

**virtual** nostd::unique\_ptr<*Token*> Attach (const *Context* &context) noexcept = 0

Set the current context.

**Parameters** *the* – new current context

**Returns** a token for the new current context. This never returns a nullptr.

```
virtual bool Detach (Token &token) noexcept = 0
```

Detach the context related to the given token.

**Parameters** **token** – a token related to a context

**Returns** true if the context could be detached

```
inline virtual ~RuntimeContextStorage ()
```

### Protected Functions

```
inline nstd::unique_ptr<Token> CreateToken (const Context &context) noexcept
```

## Class ThreadLocalContextStorage

- Defined in file\_include\_opentelemetry\_context\_runtime\_context.h

## Inheritance Relationships

### Base Type

- public opentelemetry::context::RuntimeContextStorage (*Class RuntimeContextStorage*)

## Class Documentation

```
class opentelemetry::context::ThreadLocalContextStorage : public opentelemetry::context::RuntimeContextStorage
```

### Public Functions

```
ThreadLocalContextStorage () noexcept = default
```

```
inline Context GetCurrent () noexcept override
```

```
inline bool Detach (Token &token) noexcept override
```

```
inline nstd::unique_ptr<Token> Attach (const Context &context) noexcept override
```

## Class Token

- Defined in file\_include\_opentelemetry\_context\_runtime\_context.h

## Class Documentation

```
class opentelemetry::context::Token
```

### Public Functions

```
inline bool operator==(const Context &other) const noexcept
inline ~Token()
```

### 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 (nstd::string_view, const common::AttributeValue&) noexcept
inline void AddEvent (nstd::string_view) noexcept
inline void AddEvent (nstd::string_view, common::SystemTimestamp) noexcept
inline void AddEvent (nstd::string_view, common::SystemTimestamp, const common::KeyValueIterable&) noexcept
inline void AddEvent (nstd::string_view name, const common::KeyValueIterable &attributes)
    noexcept
inline void SetStatus (StatusCode, nstd::string_view) noexcept
inline void UpdateName (nstd::string_view) noexcept
inline void End (const EndSpanOptions& = {}) noexcept
inline nstd::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 com-
mon::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_this<NoopTracer>`  
No-op implementation of *Tracer*.

## Public Functions

```
inline virtual nostd::shared_ptr<Span> StartSpan (nostd::string_view, const com-
mon::KeyValueIterable&, const Span-
ContextKeyValueIterable&, 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::B3Propagator

### 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 opentelemetry::context::propagation::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 opentelemetry::context::propagation::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** (nstd::string\_view trace\_id)

**static inline** SpanId **SpanIdFromHex** (nstd::string\_view span\_id)

**static inline** TraceFlags **TraceFlagsFromHex** (nstd::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::propagation::B3PropagatorExtractor
```

### 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 opentelemetry::context::propagation::TextMapPropagator`

## Class Documentation

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

### Class Documentation

```
class opentelemetry::trace::propagation::JaegerPropagator : public opentelemetry::context::propagation::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 nstd::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** (nstd::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 &at-
    tributes) 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 (nstd::span<const uint8_t, kSize> id) noexcept
inline void ToLowerBase16 (nstd::span<char, 2 * kSize> buffer) const noexcept
inline nstd::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 (nstd::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 (nstd::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 (nstd::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 *SpanContext*KeyValueIterable &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<

**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, com-
                                         mon::AttributeValue>>>> links, const StartSpanOp-
                                         tions &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 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::baggage::propagation::GetBaggage`

- Defined in `file_include_opentelemetry_baggage_propagation_baggage_propagator.h`

#### Function Documentation

```
inline nostd::shared_ptr<baggage::Baggage> opentelemetry::baggage::propagation::GetBaggage (const
                                                                    con-
                                                                    text::Context
                                                                    &con-
                                                                    text)
```

#### Function `opentelemetry::baggage::propagation::SetBaggage`

- Defined in `file_include_opentelemetry_baggage_propagation_baggage_propagator.h`

#### Function Documentation

```
inline context::Context opentelemetry::baggage::propagation::SetBaggage (context::Context
                                                                    &context,
                                                                    nostd::shared_ptr<baggage::Baggage>
                                                                    baggage)
```

#### Function `opentelemetry::context::GetDefaultStorage`

- Defined in `file_include_opentelemetry_context_runtime_context.h`

#### Function Documentation

**Warning:** doxygenfunction: Unable to resolve function “`opentelemetry::context::GetDefaultStorage`” with arguments () in doxygen xml output for project “OpenTelemetry C++ API” from directory: `../api/docs/doxyoutput/xml`. Potential matches:

```
- RuntimeContextStorage *GetDefaultStorage() noexcept
```

#### 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 *objPtr,
                                                                    nostd::string_view
                                                                    name,
                                                                    const
                                                                    trace::StartSpanOptions
                                                                    &options)
```

## 3.3.5 Variables

### Variable `opentelemetry::baggage::propagation::kBaggageHeader`

- Defined in `file_include_opentelemetry_baggage_propagation_baggage_propagator.h`

## Variable Documentation

**static const** `nostd::string_view opentelemetry::baggage::propagation::kBaggageHeader` = "baggage"

## 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.

### Typedef `opentelemetry::context::ContextValue`

- Defined in file `include/opentelemetry/context/context_value.h`

### Typedef Documentation

`using opentelemetry::context::ContextValue = nostd::variant<nostd::monostate, bool, int64_t, uint64_t, double, nostd::string_view>;`

## GETTING HELP

- Refer to [opentelemetry.io](https://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*), 45

### O

opentelemetry::baggage::Baggage (*C++ class*), 10

opentelemetry::baggage::Baggage::Baggage (*C++ function*), 10

opentelemetry::baggage::Baggage::Delete (*C++ function*), 10

opentelemetry::baggage::Baggage::FromHeader (*C++ function*), 11

opentelemetry::baggage::Baggage::GetAllEntries (*C++ function*), 10

opentelemetry::baggage::Baggage::GetDefault (*C++ function*), 11

opentelemetry::baggage::Baggage::GetValue (*C++ function*), 10

opentelemetry::baggage::Baggage::kKeyValueSeparator (*C++ member*), 11

opentelemetry::baggage::Baggage::kMaxKeyValuePairs (*C++ member*), 11

opentelemetry::baggage::Baggage::kMaxKeyValueSize (*C++ member*), 11

opentelemetry::baggage::Baggage::kMaxSize (*C++ member*), 11

opentelemetry::baggage::Baggage::kMembersSeparator (*C++ member*), 11

opentelemetry::baggage::Baggage::kMetadataSeparator (*C++ member*), 11

opentelemetry::baggage::Baggage::Set (*C++ function*), 10

opentelemetry::baggage::Baggage::ToHeader (*C++ function*), 11

opentelemetry::baggage::propagation::BaggagePropagator (*C++ class*), 11

opentelemetry::baggage::propagation::BaggagePropagator::Extract (*C++ function*), 11

opentelemetry::baggage::propagation::BaggagePropagator::Inject (*C++ function*), 11

opentelemetry::baggage::propagation::GetBaggage (*C++ function*), 39

opentelemetry::baggage::propagation::kBaggageHeader (*C++ member*), 42

opentelemetry::baggage::propagation::SetBaggage (*C++ function*), 39

opentelemetry::common::AttributeValue (*C++ type*), 45

opentelemetry::common::KeyValueIterable (*C++ class*), 12

opentelemetry::common::KeyValueIterable::~~KeyValueIterable (*C++ function*), 12

opentelemetry::common::KeyValueIterable::ForEachKey (*C++ function*), 12

opentelemetry::common::KeyValueIterable::size (*C++ function*), 12

opentelemetry::common::SteadyTimestamp (*C++ class*), 12

opentelemetry::common::SteadyTimestamp::operator std::chrono::steady\_clock::time\_point (*C++ function*), 13

opentelemetry::common::SteadyTimestamp::operator!= (*C++ function*), 13

opentelemetry::common::SteadyTimestamp::operator== (*C++ function*), 13

opentelemetry::common::SteadyTimestamp::SteadyTime (*C++ function*), 12

opentelemetry::common::SteadyTimestamp::time\_since\_epoch (*C++ function*), 13

opentelemetry::common::SystemTimestamp (*C++ class*), 13

opentelemetry::common::SystemTimestamp::operator std::chrono::system\_clock::time\_point (*C++ function*), 13

opentelemetry::common::SystemTimestamp::operator!= (*C++ function*), 14

opentelemetry::common::SystemTimestamp::operator== (*C++ function*), 14

opentelemetry::common::SystemTimestamp::SystemTime (*C++ function*), 13

opentelemetry::common::SystemTimestamp::time\_since\_epoch (*C++ function*), 13

opentelemetry::context::Context (*C++ class*), 14

```

opentelemetry::context::Context::Context opentelemetry::context::RuntimeContext::GetValue
    (C++ function), 14 (C++ function), 18
opentelemetry::context::Context::GetValue opentelemetry::context::RuntimeContext::SetRuntime
    (C++ function), 14 (C++ function), 18
opentelemetry::context::Context::HasKey opentelemetry::context::RuntimeContext::SetValue
    (C++ function), 14 (C++ function), 18
opentelemetry::context::Context::operator opentelemetry::context::RuntimeContextStorage
    (C++ function), 14 (C++ class), 18
opentelemetry::context::Context::SetValues opentelemetry::context::RuntimeContextStorage::~~Run
    (C++ function), 14 (C++ function), 19
opentelemetry::context::Context::SetValues opentelemetry::context::RuntimeContextStorage::Att
    (C++ function), 14 (C++ function), 18
opentelemetry::context::ContextValue opentelemetry::context::RuntimeContextStorage::Cre
    (C++ type), 46 (C++ function), 19
opentelemetry::context::propagation::CompositePropagator opentelemetry::context::RuntimeContextStorage::Det
    (C++ class), 15 (C++ function), 18
opentelemetry::context::propagation::CompositePropagator opentelemetry::context::RuntimeContextStorage::Get
    (C++ function), 15 (C++ function), 18
opentelemetry::context::propagation::CompositePropagator opentelemetry::context::RuntimeContextStorage::Thre
    (C++ function), 15 (C++ class), 19
opentelemetry::context::propagation::CompositePropagator opentelemetry::context::RuntimeContextStorage::Thre
    (C++ function), 15 (C++ function), 19
opentelemetry::context::propagation::GlobalTextMapPropagator opentelemetry::context::ThreadLocalContextStorage:
    (C++ class), 15 (C++ function), 19
opentelemetry::context::propagation::GlobalTextMapPropagator opentelemetry::context::ThreadLocalContextStorage:
    (C++ function), 15 (C++ function), 19
opentelemetry::context::propagation::GlobalTextMapPropagator opentelemetry::context::ThreadLocalContextStorage:
    (C++ function), 15 (C++ function), 19
opentelemetry::context::propagation::NoOpPropagator opentelemetry::context::Token (C++ class),
    (C++ class), 16 19
opentelemetry::context::propagation::NoOpPropagator opentelemetry::context::Token::~~Token
    (C++ function), 16 (C++ function), 20
opentelemetry::context::propagation::NoOpPropagator opentelemetry::context::Token::operator==
    (C++ function), 16 (C++ function), 20
opentelemetry::context::propagation::TextMapCarrier opentelemetry::trace::CanonicalCode
    (C++ class), 16 (C++ enum), 36
opentelemetry::context::propagation::TextMapCarrier opentelemetry::trace::CanonicalCode::ABORTED
    (C++ function), 16 (C++ enumerator), 37
opentelemetry::context::propagation::TextMapCarrier opentelemetry::trace::CanonicalCode::ALREADY_EXISTS
    (C++ function), 16 (C++ enumerator), 37
opentelemetry::context::propagation::TextMapPropagator opentelemetry::trace::CanonicalCode::CANCELLED
    (C++ class), 17 (C++ enumerator), 36
opentelemetry::context::propagation::TextMapPropagator opentelemetry::trace::CanonicalCode::DATA_LOSS
    (C++ function), 17 (C++ enumerator), 38
opentelemetry::context::propagation::TextMapPropagator opentelemetry::trace::CanonicalCode::DEADLINE_EXCEL
    (C++ function), 17 (C++ enumerator), 37
opentelemetry::context::RuntimeContext opentelemetry::trace::CanonicalCode::FAILED_PRECONI
    (C++ class), 17 (C++ enumerator), 37
opentelemetry::context::RuntimeContext::Append opentelemetry::trace::CanonicalCode::INTERNAL
    (C++ function), 18 (C++ enumerator), 38
opentelemetry::context::RuntimeContext::Delete opentelemetry::trace::CanonicalCode::INVALID_ARGUMI
    (C++ function), 18 (C++ enumerator), 37
opentelemetry::context::RuntimeContext::Get opentelemetry::trace::CanonicalCode::NOT_FOUND
    (C++ function), 18 (C++ enumerator), 37

```

`opentelemetry::trace::CanonicalCode::OK` `opentelemetry::trace::NoopSpan::GetContext`  
 (`C++ enumerator`), 36 (`C++ function`), 22  
`opentelemetry::trace::CanonicalCode::OUT_OF_RANGE` `opentelemetry::trace::NoopSpan::IsRecording`  
 (`C++ enumerator`), 37 (`C++ function`), 22  
`opentelemetry::trace::CanonicalCode::PERMISSION_DENIED` `opentelemetry::trace::NoopSpan::NoopSpan`  
 (`C++ enumerator`), 37 (`C++ function`), 22  
`opentelemetry::trace::CanonicalCode::RESOURCE_EXHAUSTED` `opentelemetry::trace::NoopSpan::SetAttribute`  
 (`C++ enumerator`), 37 (`C++ function`), 22  
`opentelemetry::trace::CanonicalCode::UNAUTHENTICATED` `opentelemetry::trace::NoopSpan::SetStatus`  
 (`C++ enumerator`), 38 (`C++ function`), 22  
`opentelemetry::trace::CanonicalCode::UNAVAILABLE` `opentelemetry::trace::NoopSpan::UpdateName`  
 (`C++ enumerator`), 38 (`C++ function`), 22  
`opentelemetry::trace::CanonicalCode::UNIMPLEMENTED` `opentelemetry::trace::NoopTracer` (`C++`  
 (`C++ enumerator`), 38 `class`), 22  
`opentelemetry::trace::CanonicalCode::UNKNOWN` `opentelemetry::trace::NoopTracer::CloseWithMicroseconds`  
 (`C++ enumerator`), 36 (`C++ function`), 23  
`opentelemetry::trace::DefaultSpan` (`C++ class`), 20 `opentelemetry::trace::NoopTracer::ForceFlushWithMicroseconds`  
 (`C++ function`), 20 (`C++ function`), 23  
`opentelemetry::trace::DefaultSpan::AddEvent` `opentelemetry::trace::NoopTracer::StartSpan`  
 (`C++ function`), 20 (`C++ function`), 23  
`opentelemetry::trace::DefaultSpan::DefaultSpan` `opentelemetry::trace::NoopTracerProvider`  
 (`C++ function`), 20 (`C++ class`), 23  
`opentelemetry::trace::DefaultSpan::End` `opentelemetry::trace::NoopTracerProvider::GetTracer`  
 (`C++ function`), 20 (`C++ function`), 23  
`opentelemetry::trace::DefaultSpan::GetContext` `opentelemetry::trace::NoopTracerProvider::NoopTracerProvider`  
 (`C++ function`), 20 (`C++ function`), 23  
`opentelemetry::trace::DefaultSpan::GetInvalid` `opentelemetry::trace::NullSpanContext`  
 (`C++ function`), 21 (`C++ class`), 24  
`opentelemetry::trace::DefaultSpan::IsRecording` `opentelemetry::trace::NullSpanContext::ForEachKeyValue`  
 (`C++ function`), 20 (`C++ function`), 24  
`opentelemetry::trace::DefaultSpan::SetAttribute` `opentelemetry::trace::NullSpanContext::size`  
 (`C++ function`), 20 (`C++ function`), 24  
`opentelemetry::trace::DefaultSpan::SetStatus` `opentelemetry::trace::propagation::B3Propagator`  
 (`C++ function`), 20 (`C++ class`), 25  
`opentelemetry::trace::DefaultSpan::ToString` `opentelemetry::trace::propagation::B3Propagator::Inject`  
 (`C++ function`), 20 (`C++ function`), 25  
`opentelemetry::trace::DefaultSpan::UpdateName` `opentelemetry::trace::propagation::B3PropagatorExtractor`  
 (`C++ function`), 20 (`C++ class`), 25  
`opentelemetry::trace::DefaultTracer` `opentelemetry::trace::propagation::B3PropagatorExtractor`  
 (`C++ class`), 21 (`C++ function`), 25  
`opentelemetry::trace::DefaultTracer::~DefaultTracer` `opentelemetry::trace::propagation::B3PropagatorExtractor`  
 (`C++ function`), 21 (`C++ function`), 25  
`opentelemetry::trace::EndSpanOptions` `opentelemetry::trace::propagation::B3PropagatorExtractor`  
 (`C++ struct`), 10 (`C++ function`), 25  
`opentelemetry::trace::EndSpanOptions::end_span_time` `opentelemetry::trace::propagation::B3PropagatorExtractor`  
 (`C++ member`), 10 (`C++ function`), 25  
`opentelemetry::trace::kSpanKey` (`C++ member`), 42 `opentelemetry::trace::propagation::B3PropagatorMultiHeader`  
 (`C++ class`), 26  
`opentelemetry::trace::NoopSpan` (`C++ class`), 22 `opentelemetry::trace::propagation::B3PropagatorMultiHeader`  
 (`C++ function`), 22 (`C++ function`), 26  
`opentelemetry::trace::NoopSpan::AddEvent` `opentelemetry::trace::propagation::detail::HexToBinary`  
 (`C++ function`), 22 (`C++ function`), 40  
`opentelemetry::trace::NoopSpan::End` `opentelemetry::trace::propagation::detail::HexToInteger`  
 (`C++ function`), 22 (`C++ function`), 40

```

opentelemetry::trace::propagation::detail::PbSpanHeaderHex::trace::propagation::SetSpan
  (C++ function), 40
opentelemetry::trace::propagation::detail::PbSpanHeaderHex::trace::Provider (C++
  (C++ member), 42
opentelemetry::trace::propagation::detail::PbSpanHeaderHex::trace::Provider::GetTracerProvider
  (C++ function), 40
opentelemetry::trace::propagation::GetSpan opentelemetry::trace::Provider::SetTracerProvider
  (C++ function), 41
opentelemetry::trace::propagation::HttpTraceContext opentelemetry::trace::Scope (C++ class), 28
  (C++ class), 26
opentelemetry::trace::propagation::HttpTraceContext opentelemetry::trace::Scope::Scope (C++
  (C++ function), 26
opentelemetry::trace::propagation::HttpTraceContext opentelemetry::trace::Span (C++ class), 28
  (C++ function), 26
opentelemetry::trace::propagation::HttpTraceContext opentelemetry::trace::Span::~Span (C++
  (C++ function), 26
opentelemetry::trace::propagation::HttpTraceContext opentelemetry::Span::AddEvent
  (C++ function), 27
opentelemetry::trace::propagation::HttpTraceContext opentelemetry::TraceFlagsFromHex (C++
  (C++ function), 27
opentelemetry::trace::propagation::HttpTraceContext opentelemetry::TraceIdFromSpanHex::GetContext
  (C++ function), 27
opentelemetry::trace::propagation::JaegerProtocol opentelemetry::trace::Span::IsRecording
  (C++ class), 27
opentelemetry::trace::propagation::JaegerProtocol opentelemetry::Trace::Span::operator=
  (C++ function), 27
opentelemetry::trace::propagation::JaegerProtocol opentelemetry::Trace::Span::SetAttribute
  (C++ function), 27
opentelemetry::trace::propagation::kB3Component opentelemetry::trace::Span::SetStatus
  (C++ member), 42
opentelemetry::trace::propagation::kB3Sampled opentelemetry::trace::Span::Span (C++
  (C++ member), 42
opentelemetry::trace::propagation::kB3SpanId opentelemetry::trace::Span::UpdateName
  (C++ member), 43
opentelemetry::trace::propagation::kB3TraceId opentelemetry::trace::SpanContext (C++
  (C++ member), 43
opentelemetry::trace::propagation::kSpanContextLength opentelemetry::trace::SpanContext::GetInvalid
  (C++ member), 43
opentelemetry::trace::propagation::kSpanContextSize opentelemetry::trace::SpanContext::IsRemote
  (C++ member), 43
opentelemetry::trace::propagation::kTraceFlagsSize opentelemetry::trace::SpanContext::IsSampled
  (C++ member), 43
opentelemetry::trace::propagation::kTraceHeader opentelemetry::trace::SpanContext::IsValid
  (C++ member), 44
opentelemetry::trace::propagation::kTraceHeaderLength opentelemetry::trace::SpanContext::operator=
  (C++ member), 44
opentelemetry::trace::propagation::kTraceHeaderSize opentelemetry::trace::SpanContext::operator==
  (C++ member), 44
opentelemetry::trace::propagation::kTraceParent opentelemetry::trace::SpanContext::span_id
  (C++ member), 44
opentelemetry::trace::propagation::kTraceParentSize opentelemetry::trace::SpanContext::SpanContext
  (C++ member), 44
opentelemetry::trace::propagation::kTraceSpan opentelemetry::trace::SpanContext::trace_flags
  (C++ member), 45
opentelemetry::trace::propagation::kVersionSize opentelemetry::trace::SpanContext::trace_id
  (C++ member), 45

```





```

opentelemetry::trace::Tracer::GetCurrentSpan
    (C++ function), 34
opentelemetry::trace::Tracer::StartSpan
    (C++ function), 33
opentelemetry::trace::Tracer::WithActiveSpan
    (C++ function), 34
opentelemetry::trace::TracerProvider
    (C++ class), 35
opentelemetry::trace::TracerProvider::~~TracerProvider
    (C++ function), 35
opentelemetry::trace::TracerProvider::GetTracer
    (C++ function), 35
opentelemetry::trace::TraceState (C++
    class), 35
opentelemetry::trace::TraceState::Delete
    (C++ function), 35
opentelemetry::trace::TraceState::Empty
    (C++ function), 35
opentelemetry::trace::TraceState::FromHeader
    (C++ function), 36
opentelemetry::trace::TraceState::Get
    (C++ function), 35
opentelemetry::trace::TraceState::GetAllEntries
    (C++ function), 35
opentelemetry::trace::TraceState::GetDefault
    (C++ function), 36
opentelemetry::trace::TraceState::IsValidKey
    (C++ function), 36
opentelemetry::trace::TraceState::IsValidValue
    (C++ function), 36
opentelemetry::trace::TraceState::kKeyMaxSize
    (C++ member), 36
opentelemetry::trace::TraceState::kKeyValueSeparator
    (C++ member), 36
opentelemetry::trace::TraceState::kMaxKeyValuePairs
    (C++ member), 36
opentelemetry::trace::TraceState::kMembersSeparator
    (C++ member), 36
opentelemetry::trace::TraceState::kValueMaxSize
    (C++ member), 36
opentelemetry::trace::TraceState::Set
    (C++ function), 35
opentelemetry::trace::TraceState::ToHeader
    (C++ function), 35

```