OpenTelemetry C++

Release 0.3.0

OpenTelemetry authors

Apr 16, 2021

OPENTELEMETRY C++ API

1 Overview 1

2 Getting started 3

3 Reference documentation 5

4 Getting help 67

Index 69

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

Contents

- *Namespaces*

Namespaces

- *Namespace opentelemetry::common*
- *Namespace opentelemetry::core*
- *Namespace opentelemetry::nostd*
- *Namespace opentelemetry::trace*

Namespace opentelemetry::common

Contents

- *Classes*
- *Enums*
- *Typedefs*

Classes

- *Class KeyValueIterable*
- *Class NullKeyValueIterable*

Enums

- *Enum AttributeType*

Typedefs

- *Typedef opentelemetry::common::AttributeValue*

Namespace opentelemetry::core

Contents

- *Classes*

Classes

- *Class SteadyTimestamp*
- *Class SystemTimestamp*

Namespace opentelemetry::nostd

Contents

- *Namespaces*
- *Classes*
- *Functions*
- *Typedefs*
- *Variables*

Namespaces

- *Namespace opentelemetry::nostd::detail*

Classes

- *Struct shared_ptr::PlacementBuffer*
- *Template Class shared_ptr*
- *Class shared_ptr::shared_ptr_wrapper*
- *Template Class span*
- *Template Class span< T, dynamic_extent >*
- *Class string_view*
- *Template Class unique_ptr*

Functions

- *Function opentelemetry::nostd::operator!=(string_view, string_view)*
- *Function opentelemetry::nostd::operator!=(string_view, const std::string&)*
- *Function opentelemetry::nostd::operator!=(const std::string&, string_view)*
- *Function opentelemetry::nostd::operator!=(string_view, const char *)*
- *Function opentelemetry::nostd::operator!=(const char *, string_view)*
- *Template Function opentelemetry::nostd::operator!=(const shared_ptr<T1>&, const shared_ptr<T2>&)*
- *Template Function opentelemetry::nostd::operator!=(const unique_ptr<T1>&, const unique_ptr<T2>&)*
- *Template Function opentelemetry::nostd::operator!=(const shared_ptr<T>&, std::nullptr_t)*
- *Template Function opentelemetry::nostd::operator!=(const unique_ptr<T>&, std::nullptr_t)*
- *Template Function opentelemetry::nostd::operator!=(std::nullptr_t, const shared_ptr<T>&)*
- *Template Function opentelemetry::nostd::operator!=(std::nullptr_t, const unique_ptr<T>&)*
- *Function opentelemetry::nostd::operator<<*
- *Template Function opentelemetry::nostd::operator==(const shared_ptr<T1>&, const shared_ptr<T2>&)*
- *Template Function opentelemetry::nostd::operator==(const unique_ptr<T1>&, const unique_ptr<T2>&)*
- *Template Function opentelemetry::nostd::operator==(const shared_ptr<T>&, std::nullptr_t)*
- *Template Function opentelemetry::nostd::operator==(const unique_ptr<T>&, std::nullptr_t)*
- *Template Function opentelemetry::nostd::operator==(std::nullptr_t, const unique_ptr<T>&)*
- *Template Function opentelemetry::nostd::operator==(std::nullptr_t, const shared_ptr<T>&)*
- *Function opentelemetry::nostd::operator==(string_view, string_view)*
- *Function opentelemetry::nostd::operator==(string_view, const std::string&)*
- *Function opentelemetry::nostd::operator==(const std::string&, string_view)*
- *Function opentelemetry::nostd::operator==(string_view, const char *)*

- *Function opentelemetry::nostd::operator==(const char *, string_view)*

Typedefs

- *Typedef opentelemetry::nostd::Traits*

Variables

- *Variable opentelemetry::nostd::dynamic_extent*

Namespace opentelemetry::nostd::detail

Contents

- *Classes*

Classes

- *Template Struct is_specialized_span_convertible*
- *Template Struct is_specialized_span_convertible< span< T, Extent > >*
- *Template Struct is_specialized_span_convertible< std::array< T, N > >*
- *Template Struct is_specialized_span_convertible< T[N]>*
- *Template Struct unique_ptr_element_type*
- *Template Struct unique_ptr_element_type< T[]>*

Namespace opentelemetry::trace

Contents

- *Namespaces*
- *Classes*
- *Enums*
- *Functions*
- *Variables*

Namespaces

- *Namespace opentelemetry::trace::propagation*

Classes

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

Enums

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

Functions

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

Variables

- *Variable opentelemetry::trace::kSpanKey*

Namespace opentelemetry::trace::propagation

Contents

- *Namespaces*
- *Classes*
- *Variables*

Namespaces

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

Classes

- *Template Class B3Propagator*
- *Template Class B3PropagatorExtractor*
- *Template Class B3PropagatorMultiHeader*
- *Template Class CompositePropagator*
- *Template Class HttpTraceContext*
- *Template Class JaegerPropagator*
- *Template Class TextMapPropagator*

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

Contents

- *Functions*
- *Variables*

Functions

- *Function opentelemetry::trace::propagation::detail::GetCurrentSpan*
- *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*

Namespace std

Contents

- *Classes*

Classes

- *Template Struct hash< OPENTELEMETRY_NAMESPACE::nostd::string_view >*

3.3.2 Classes and Structs

Template Struct is_specialized_span_convertible

- Defined in file_include_opentelemetry_nostd_span.h

Inheritance Relationships

Base Type

- `public false_type`

Struct Documentation

template<class **T**>
struct is_specialized_span_convertible : **public** false_type
Helper class to resolve overloaded constructors

Template Struct is_specialized_span_convertible< span< T, Extent > >

- Defined in file_include_opentelemetry_nostd_span.h

Inheritance Relationships

Base Type

- `public true_type`

Struct Documentation

template<class **T**, size_t **Extent**>
struct is_specialized_span_convertible<*span*<*T*, *Extent*>> : **public** true_type

Template Struct is_specialized_span_convertible< std::array< T, N > >

- Defined in file_include_opentelemetry_nostd_span.h

Inheritance Relationships

Base Type

- `public true_type`

Struct Documentation

template<class **T**, size_t **N**>
struct is_specialized_span_convertible<std::array<*T*, *N*>> : **public** true_type

Template Struct is_specialized_span_convertible< T[N]>

- Defined in file_include_opentelemetry_nostd_span.h

Inheritance Relationships

Base Type

- `public true_type`

Struct Documentation

template<class **T**, size_t **N**>
struct is_specialized_span_convertible<*T*[N]> : **public** true_type

Template Struct unique_ptr_element_type

- Defined in file_include_opentelemetry_nostd_unique_ptr.h

Struct Documentation

template<class **T**>
struct opentelemetry::nostd::detail::**unique_ptr_element_type**

Public Types

using type = *T*

Template Struct unique_ptr_element_type< T[]>

- Defined in file_include_opentelemetry_nostd_unique_ptr.h

Struct Documentation

template<class **T**>
struct opentelemetry::nostd::detail::**unique_ptr_element_type**<*T*[]>

Public Types

using type = *T*

Struct shared_ptr::PlacementBuffer

- Defined in file_include_opentelemetry_nostd_shared_ptr.h

Nested Relationships

This struct is a nested type of *Template Class shared_ptr*.

Struct Documentation

struct opentelemetry::nostd::*shared_ptr*::**PlacementBuffer**

Public Members

char **data**[**kMaxSize**]

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

core::*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

core::*SystemTimestamp* **start_system_time**

core::*SteadyTimestamp* **start_steady_time**

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

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

Template Struct hash< OPENTELEMETRY_NAMESPACE::nostd::string_view >

- Defined in file_include_opentelemetry_nostd_string_view.h

Struct Documentation

template<>
struct std::**hash**<OPENTELEMETRY_NAMESPACE::nostd::string_view>

Public Functions

inline std::size_t **operator()**(**const** OPENTELEMETRY_NAMESPACE::nostd::string_view &*k*) **const**

Class KeyValueIterable

- Defined in file_include_opentelemetry_common_key_value_iterable.h

Inheritance Relationships

Derived Type

- `public opentelemetry::common::NullKeyValueIterable` (*Class NullKeyValueIterable*)

Class Documentation

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

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

Subclassed by *opentelemetry::common::NullKeyValueIterable*

Public Functions

virtual ~KeyValueIterable() = default

virtual bool **ForEachKeyValue**(nostd::function_ref<bool(nostd::*string_view*, common::*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 NullKeyValueIterable

- Defined in file_include_opentelemetry_common_key_value_iterable.h

Inheritance Relationships

Base Type

- `public opentelemetry::common::KeyValueIterable` (*Class KeyValueIterable*)

Class Documentation

class `opentelemetry::common::`**NullKeyValueIterable** : **public** opentelemetry::common::*KeyValueIterable*

Public Functions

inline NullKeyValueIterable()

inline virtual bool **ForEachKeyValue**(nostd::function_ref<bool) nostd::*string_view*, common::*AttributeValue* > **const noexcept**

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

Class SteadyTimestamp

- Defined in file_include_opentelemetry_core_timestamp.h

Class Documentation

class opentelemetry::core::**SteadyTimestamp**

Represents a timepoint relative to the monotonic clock epoch

Public Functions

inline SteadyTimestamp() **noexcept**

template<class **Rep**, class **Period**>
inline explicit SteadyTimestamp(**const** std::chrono::duration<*Rep*, *Period*> &*time_since_epoch*) **noexcept**

inline SteadyTimestamp(**const** std::chrono::steady_clock::time_point &*time_point*) **noexcept**

inline operator std::chrono::steady_clock::time_point() **const noexcept**

inline std::chrono::nanoseconds **time_since_epoch**() **const noexcept**

Returns the amount of time that has passed since the monotonic clock epoch

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

inline bool **operator!=**(**const** *SteadyTimestamp* &*other*) **const noexcept**

Class SystemTimestamp

- Defined in file_include_opentelemetry_core_timestamp.h

Class Documentation

class opentelemetry::core::**SystemTimestamp**

Represents a timepoint relative to the system clock epoch

Public Functions

inline SystemTimestamp() **noexcept**

template<class **Rep**, class **Period**>
inline explicit SystemTimestamp(**const** std::chrono::duration<*Rep*, *Period*> &*time_since_epoch*) **noexcept**

inline SystemTimestamp(**const** std::chrono::system_clock::time_point &*time_point*) **noexcept**

inline operator std::chrono::system_clock::time_point() **const noexcept**

inline std::chrono::nanoseconds **time_since_epoch**() **const noexcept**

Returns the amount of time that has passed since the system clock epoch

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

inline bool **operator!=**(**const** *SystemTimestamp* &*other*) **const noexcept**

Template Class shared_ptr

- Defined in file_include_opentelemetry_nostd_shared_ptr.h

Nested Relationships

Nested Types

- *Struct shared_ptr::PlacementBuffer*
- *Class shared_ptr::shared_ptr_wrapper*

Class Documentation

template<class **T**>
class opentelemetry::nostd::**shared_ptr**
Provide a type-erased version of std::shared_ptr that has ABI stability.

Public Types

using element_type = *T*

using pointer = *element_type**

Public Functions

inline shared_ptr() **noexcept**

inline explicit shared_ptr(*pointer ptr*)

inline shared_ptr(std::shared_ptr<*T*> *ptr*) **noexcept**

inline shared_ptr(*shared_ptr* &&*other*) **noexcept**

template<class **U**, **typename** std::enable_if<std::is_convertible<*U**, *pointer*>::value>::type* = nullptr>
inline shared_ptr(*shared_ptr*<*U*> &&*other*) **noexcept**

inline shared_ptr(**const** *shared_ptr* &*other*) **noexcept**

inline ~shared_ptr()

inline *shared_ptr* &**operator=**(*shared_ptr* &&*other*) **noexcept**

inline *shared_ptr* &**operator=**(std::nullptr_t) **noexcept**

inline *shared_ptr* &**operator=**(**const** *shared_ptr* &*other*) **noexcept**

inline *element_type* &**operator***() **const noexcept**

inline *pointer* **operator->**() **const noexcept**

inline operator bool() **const noexcept**

inline *pointer* **get**() **const noexcept**

inline void **swap**(*shared_ptr*<*T*> &*other*) **noexcept**

Friends

friend class shared_ptr

Class shared_ptr::shared_ptr_wrapper

- Defined in file_include_opentelemetry_nostd_shared_ptr.h

Nested Relationships

This class is a nested type of *Template Class shared_ptr*.

Class Documentation

class opentelemetry::nostd::*shared_ptr*::**shared_ptr_wrapper**

Public Functions

shared_ptr_wrapper() **noexcept** = default

inline **shared_ptr_wrapper**(std::shared_ptr<T> &&*ptr*) **noexcept**

inline **virtual** **~shared_ptr_wrapper**()

inline **virtual** void **CopyTo**(*PlacementBuffer* &*buffer*) **const** **noexcept**

inline **virtual** void **MoveTo**(*PlacementBuffer* &*buffer*) **noexcept**

template<class **U**, **typename** std::enable_if<std::is_convertible<pointer, *U**>::value>::type* = nullptr>
inline void **MoveTo**(**typename** shared_ptr<*U*>::PlacementBuffer &*buffer*) **noexcept**

inline **virtual** pointer **Get**() **const** **noexcept**

inline **virtual** void **Reset**() **noexcept**

Template Class span

- Defined in file_include_opentelemetry_nostd_span.h

Class Documentation

template<class **T**, size_t **Extent**>
class `opentelemetry::nostd::span`

Back port of std::span.

See https://en.cppreference.com/w/cpp/container/span for interface documentation.

Note: This provides a subset of the methods available on std::span.

Note: The std::span API specifies error cases to have undefined behavior, so this implementation chooses to terminate or assert rather than throw exceptions.

Public Functions

template<bool **B** = *Extent* == 0, **typename** std::enable_if<*B*>::type* = nullptr>
inline **span**() **noexcept**

inline **span**(*T* **data*, size_t *count*) **noexcept**

inline **span**(*T* **first*, *T* **last*) **noexcept**

template<size_t **N**, **typename** std::enable_if<*Extent* == *N*>::type* = nullptr>
inline **span**(*T* (&*array*)[N]) **noexcept**

template<size_t **N**, **typename** std::enable_if<*Extent* == *N*>::type* = nullptr>
inline **span**(std::array<*T*, *N*> &*array*) **noexcept**

template<size_t **N**, **typename** std::enable_if<*Extent* == *N*>::type* = nullptr>
inline **span**(**const** std::array<*T*, *N*> &*array*) **noexcept**

template<class **C**, **typename** std::enable_if<!detail::*is_specialized_span_convertible*<*C*>::value && std::is_convertible<**typen**
inline **span**(*C* &*c*) **noexcept**(noexcept(**data**(**c**), **size**(**c**)))

template<class **C**, **typename** std::enable_if<!detail::*is_specialized_span_convertible*<*C*>::value && std::is_convertible<**typen**
inline **span**(**const** *C* &*c*) **noexcept**(noexcept(**data**(**c**), **size**(**c**)))

template<class **U**, size_t **N**, **typename** std::enable_if<*N* == *Extent* && std::is_convertible<*U* (*)[], *T* (*)[]>::value>::type* = nu
inline span(**const** *span*<*U*, *N*> &*other*) **noexcept**

span(**const** *span*&) **noexcept** = default

inline bool **empty**() **const noexcept**

inline *T* ***data**() **const noexcept**

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

inline *T* &**operator[]**(size_t *index*) **const noexcept**

inline *T* ***begin**() **const noexcept**

inline *T* ***end**() **const noexcept**

Public Static Attributes

static constexpr size_t **extent** = *Extent*

Template Class span< T, dynamic_extent >

- Defined in file_include_opentelemetry_nostd_span.h

Class Documentation

template<class **T**>
class `opentelemetry::nostd::span`<*T*, *dynamic_extent*>

Public Functions

inline span() **noexcept**

inline span(*T* **data*, size_t *count*) **noexcept**

inline span(*T* **first*, *T* **last*) **noexcept**

template<size_t **N**>
inline span(*T* (&*array*)[N]) **noexcept**

template<size_t **N**>
inline span(std::array<*T*, *N*> &*array*) **noexcept**

template<size_t **N**>
inline span(**const** std::array<*T*, *N*> &*array*) **noexcept**

template<class **C**, **typename** std::enable_if<!detail::*is_specialized_span_convertible*<*C*>::value && std::is_convertible<**typen**
inline span(*C* &*c*) **noexcept**(noexcept(**data**(**c**), **size**(**c**)))

template<class **C**, **typename** std::enable_if<!detail::*is_specialized_span_convertible*<*C*>::value && std::is_convertible<**typen**
inline span(**const** *C* &*c*) **noexcept**(noexcept(**data**(**c**), **size**(**c**)))

template<class **U**, size_t **N**, **typename** std::enable_if<std::is_convertible<*U* (*)[], *T* (*)[]>::value>::type* = nullptr>
inline span(**const** *span*<*U*, *N*> &*other*) **noexcept**

span(**const** *span*&) **noexcept** = default

inline bool **empty**() **const noexcept**

inline *T* ***data**() **const noexcept**

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

inline *T* &**operator[]**(size_t *index*) **const noexcept**

inline *T* ***begin**() **const noexcept**

inline *T* ***end**() **const noexcept**

Public Static Attributes

static constexpr size_t **extent** = *dynamic_extent*

Class string_view

- Defined in file_include_opentelemetry_nostd_string_view.h

Class Documentation

class opentelemetry::nostd::**string_view**

Back port of std::string_view to work with pre-cpp-17 compilers.

Note: This provides a subset of the methods available on std::string_view but tries to be as compatible as possible with the std::string_view interface.

Public Types

typedef std::size_t **size_type**

Public Functions

inline string_view() **noexcept**

inline string_view(**const** char **str*) **noexcept**

inline string_view(**const** std::basic_string<char> &*str*) **noexcept**

inline string_view(**const** char **str*, *size_type len*) **noexcept**

inline explicit operator std::string() **const**

inline const char ***data**() **const noexcept**

inline bool **empty**() **const noexcept**

inline *size_type* **length**() **const noexcept**

inline *size_type* **size**() **const noexcept**

inline const char ***begin**() **const noexcept**

inline const char ***end**() **const noexcept**

inline const char &**operator[]**(*size_type i*)

inline *string_view* **substr**(*size_type pos*, *size_type n* = *npos*) **const**

inline int **compare**(*string_view v*) **const noexcept**

inline int **compare**(*size_type pos1*, *size_type count1*, *string_view v*) **const**

inline int **compare**(*size_type pos1*, *size_type count1*, *string_view v*, *size_type pos2*, *size_type count2*) **const**

inline int **compare**(**const** char **s*) **const**

inline int **compare**(*size_type pos1*, *size_type count1*, **const** char **s*) **const**

inline int **compare**(*size_type pos1*, *size_type count1*, **const** char **s*, *size_type count2*) **const**

inline *size_type* **find**(char *ch*, *size_type pos* = 0) **const noexcept**

inline bool **operator<**(**const** *string_view v*) **const noexcept**

inline bool **operator>**(**const** *string_view v*) **const noexcept**

Public Static Attributes

static constexpr *size_type* **npos** = static_cast<*size_type*>(-1)

Template Class unique_ptr

- Defined in file_include_opentelemetry_nostd_unique_ptr.h

Class Documentation

template<class **T**>
class `opentelemetry::nostd::unique_ptr`

Provide a simplified port of std::unique_ptr that has ABI stability.

Note: This implementation doesn't allow for a custom deleter.

Public Types

using element_type = **typename** detail::*unique_ptr_element_type*<*T*>::type

using pointer = *element_type**

Public Functions

inline unique_ptr() **noexcept**

inline unique_ptr(std::nullptr_t) **noexcept**

inline explicit unique_ptr(*pointer ptr*) **noexcept**

inline unique_ptr(*unique_ptr* &&*other*) **noexcept**

template<class **U**, **typename** std::enable_if<std::is_convertible<*U**, *pointer*>::value>::type* = nullptr>
inline unique_ptr(*unique_ptr*<*U*> &&*other*) **noexcept**

template<class **U**, **typename** std::enable_if<std::is_convertible<*U**, *pointer*>::value>::type* = nullptr>
inline unique_ptr(std::unique_ptr<*U*> &&*other*) **noexcept**

inline ~unique_ptr()

inline *unique_ptr* &**operator=**(*unique_ptr* &&*other*) **noexcept**

inline *unique_ptr* &**operator=**(std::nullptr_t) **noexcept**

template<class **U**, **typename** std::enable_if<std::is_convertible<*U**, *pointer*>::value>::type* = nullptr>
inline *unique_ptr* &**operator=**(*unique_ptr*<*U*> &&*other*) **noexcept**

template<class **U**, **typename** std::enable_if<std::is_convertible<*U**, *pointer*>::value>::type* = nullptr>
inline *unique_ptr* &**operator=**(std::unique_ptr<*U*> &&*other*) **noexcept**

inline **operator std::unique_ptr<T>**() && **noexcept**

inline **operator bool**() **const** **noexcept**

inline *element_type* &**operator***() **const** **noexcept**

inline *pointer* **operator->**() **const** **noexcept**

inline *pointer* **get**() **const** **noexcept**

inline void **reset**(*pointer ptr* = nullptr) **noexcept**

inline *pointer* **release**() **noexcept**

inline void **swap**(*unique_ptr* &*other*) **noexcept**

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 key*, **const** common::*AttributeValue* &*value*) **noexcept**

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

inline void **AddEvent**(nostd::*string_view name*, core::*SystemTimestamp timestamp*) **noexcept**

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

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

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

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

inline void **End**(**const** *EndSpanOptions* &*options* = {}) **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*, core::*SystemTimestamp*) **noexcept override**

inline virtual void **AddEvent**(nostd::*string_view*, core::*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.

Template Class B3Propagator

- Defined in file_include_opentelemetry_trace_propagation_b3_propagator.h

Inheritance Relationships

Base Type

- `public opentelemetry::trace::propagation::B3PropagatorExtractor< T >`

Class Documentation

template<typename **T**>
class `opentelemetry::trace::propagation::B3Propagator` : **public** opentelemetry::trace::propagation::*B3Propa*

Public Types

using Setter = void (*) (*T* &carrier, nostd::*string_view* trace_type, nostd::*string_view* trace_description)

Public Functions

inline void **Inject** (*Setter setter*, *T* &*carrier*, **const** *context*::Context &*context*) **noexcept override**

Template Class B3PropagatorExtractor

- Defined in file_include_opentelemetry_trace_propagation_b3_propagator.h

Inheritance Relationships

Base Type

- `public opentelemetry::trace::propagation::TextMapPropagator< T >`

Derived Types

- `public opentelemetry::trace::propagation::B3Propagator< T >` (*Template Class B3Propagator*)
- `public opentelemetry::trace::propagation::B3PropagatorMultiHeader< T >` (*Template Class B3PropagatorMultiHeader*)

Class Documentation

template<typename **T**>
class opentelemetry::trace::propagation::**B3PropagatorExtractor** : **public** opentelemetry::trace::propagat

Subclassed by *opentelemetry::trace::propagation::B3Propagator< T >*, *opentelemetry::trace::propagation::B3PropagatorMultiHeader< T >*

Public Types

using Getter = nostd::*string_view* (*) (**const** *T* &carrier, nostd::*string_view* trace_type)

Public Functions

inline *context*::Context **Extract** (*Getter getter*, **const** *T* &*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*)

Template Class B3PropagatorMultiHeader

- Defined in file_include_opentelemetry_trace_propagation_b3_propagator.h

Inheritance Relationships

Base Type

- public opentelemetry::trace::propagation::B3PropagatorExtractor< T >

Class Documentation

template<typename **T**>
class opentelemetry::trace::propagation::**B3PropagatorMultiHeader** : **public** opentelemetry::trace::propa

Public Types

using Setter = void (*) (*T* &carrier, nostd::*string_view* trace_type, nostd::*string_view* trace_description)

Public Functions

inline void **Inject**(*Setter* *setter*, *T* &*carrier*, **const** *context*::Context &*context*) **noexcept override**

Template Class CompositePropagator

- Defined in file_include_opentelemetry_trace_propagation_composite_propagator.h

Inheritance Relationships

Base Type

- `public opentelemetry::trace::propagation::TextMapPropagator< T >`

Class Documentation

template<typename **T**>
class `opentelemetry::trace::propagation::`**CompositePropagator** : **public** opentelemetry::trace::propagation

Public Types

using Getter = nostd::*string_view* (*)(**const** *T* &carrier, nostd::*string_view* trace_type)

using Setter = void (*)(*T* &carrier, nostd::*string_view* trace_type, nostd::*string_view* trace_description)

Public Functions

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

inline void **Inject**(*Setter* *setter*, *T* &*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

- **setter** – Rules that manages how context will be injected to carrier.
- **carrier** – Carrier into which context will be injected
- **context** – Context to inject

inline *context*::Context **Extract**(*Getter* *getter*, **const** *T* &*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

- **setter** – Rules that manages how context will be extracte from carrier.

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

Template Class HttpTraceContext

- Defined in file_include_opentelemetry_trace_propagation_http_trace_context.h

Inheritance Relationships

Base Type

- `public opentelemetry::trace::propagation::TextMapPropagator< T >`

Class Documentation

template<typename **T**>
class `opentelemetry::trace::propagation::`**HttpTraceContext** : **public** opentelemetry::trace::propagation::*Tex*

Public Types

using Getter = nostd::*string_view* (*) (**const** *T* &carrier, nostd::*string_view* trace_type)

using Setter = void (*) (*T* &carrier, nostd::*string_view* trace_type, nostd::*string_view* trace_description)

Public Functions

inline void **Inject** (*Setter setter*, *T* &*carrier*, **const** *context*::Context &*context*) **noexcept override**

inline *context*::Context **Extract** (*Getter getter*, **const** *T* &*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*)

Template Class JaegerPropagator

- Defined in file_include_opentelemetry_trace_propagation_jaeger.h

Inheritance Relationships

Base Type

- `public opentelemetry::trace::propagation::TextMapPropagator< T >`

Class Documentation

template<typename **T**>
class `opentelemetry::trace::propagation::`**JaegerPropagator** : **public** opentelemetry::trace::propagation::*Tex*

Public Types

using Getter = nostd::*string_view* (*) (**const** *T* &carrier, nostd::*string_view* trace_type)

using Setter = void (*) (*T* &carrier, nostd::*string_view* trace_type, nostd::*string_view* trace_description)

Public Functions

inline void **Inject** (*Setter setter*, *T* &*carrier*, **const** *context*::Context &*context*) **noexcept override**

inline *context*::Context **Extract** (*Getter getter*, **const** *T* &*carrier*, *context*::Context &*context*) **noexcept override**

Template Class TextMapPropagator

- Defined in file_include_opentelemetry_trace_propagation_text_map_propagator.h

Inheritance Relationships

Derived Types

- `public opentelemetry::trace::propagation::B3PropagatorExtractor< T >` (*Template Class B3PropagatorExtractor*)
- `public opentelemetry::trace::propagation::CompositePropagator< T >` (*Template Class CompositePropagator*)
- `public opentelemetry::trace::propagation::HttpTraceContext< T >` (*Template Class HttpTraceContext*)
- `public opentelemetry::trace::propagation::JaegerPropagator< T >` (*Template Class JaegerPropagator*)

Class Documentation

template<typename **T**>
class opentelemetry::trace::propagation::**TextMapPropagator**

Subclassed by *opentelemetry::trace::propagation::B3PropagatorExtractor< T >*, *opentelemetry::trace::propagation::CompositePropagator< T >*, *opentelemetry::trace::propagation::HttpTraceContext< T >*, *opentelemetry::trace::propagation::JaegerPropagator< T >*

Public Types

using Getter = nostd::*string_view* (*) (**const** *T* &carrier, nostd::*string_view* trace_type)

using Setter = void (*) (*T* &carrier, nostd::*string_view* trace_type, nostd::*string_view* trace_description)

Public Functions

virtual *context*::Context **Extract** (*Getter get_from_carrier*, **const** *T* &*carrier*, *context*::Context &*context*) **noexcept** = 0

virtual void **Inject** (*Setter set_from_carrier*, *T* &*carrier*, **const** *context*::Context &*context*) **noexcept** = 0

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*, core::*SystemTimestamp timestamp*) **noexcept** = 0

virtual void **AddEvent**(nostd::*string_view name*, core::*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*, core::*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*, core::*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

Class Documentation

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

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

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**

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**

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.

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.

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

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

Nested Relationships

Nested Types

- *Class TraceState::Entry*

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 std::string **Get**(nostd::*string_view* *key*) **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 nostd::*span*<*Entry*> **Entries** () **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** = ','

class Entry

Public Functions

inline Entry ()

inline Entry (**const** *Entry* &*copy*)

inline *Entry* &**operator=** (*Entry* &*other*)

Entry (*Entry* &&*other*) = default

Entry &**operator=** (*Entry* &&*other*) = default

inline Entry (nostd::*string_view* *key*, nostd::*string_view* *value*) **noexcept**

inline nostd::*string_view* **GetKey** () **const**

inline nostd::*string_view* **GetValue** () **const**

inline void **SetValue** (nostd::*string_view* *value*)

Class TraceState::Entry

- Defined in file_include_opentelemetry_trace_trace_state.h

Nested Relationships

This class is a nested type of *Class TraceState*.

Class Documentation

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

Public Functions

inline **Entry**()

inline **Entry**(**const** *Entry* &*copy*)

inline *Entry* &**operator=**(*Entry* &*other*)

Entry(*Entry* &&*other*) = default

Entry &**operator=**(*Entry* &&*other*) = default

inline **Entry**(nostd::*string_view* *key*, nostd::*string_view* *value*) **noexcept**

inline nostd::*string_view* **GetKey**() **const**

inline nostd::*string_view* **GetValue**() **const**

inline void **SetValue**(nostd::*string_view* *value*)

3.3.3 Enums

Enum AttributeType

- Defined in file_include_opentelemetry_common_attribute_value.h

Enum Documentation

enum opentelemetry::common::**AttributeType**

Values:

enumerator **TYPE_BOOL**

enumerator **TYPE_INT**

enumerator **TYPE_INT64**

enumerator **TYPE_UINT**

enumerator **TYPE_UINT64**

enumerator **TYPE_DOUBLE**

enumerator **TYPE_STRING**

enumerator TYPE_SPAN_BOOL

enumerator TYPE_SPAN_INT

enumerator TYPE_SPAN_INT64

enumerator TYPE_SPAN_UINT

enumerator TYPE_SPAN_UINT64

enumerator TYPE_SPAN_DOUBLE

enumerator TYPE_SPAN_STRING

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

Template Function opentelemetry::nostd::operator!=(const shared_ptr<T1>&, const shared_ptr<T2>&)

- Defined in file_include_opentelemetry_nostd_shared_ptr.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "opentelemetry::nostd::operator!=" with arguments (const shared_ptr<T1>&, const shared_ptr<T2>&) in doxygen xml output for project "OpenTelemetry C++ API" from directory: ../../api/docs/doxyoutput/xml. Potential matches:

```
- bool operator!=(const char *lhs, string_view rhs) noexcept
- bool operator!=(const std::string &lhs, string_view rhs) noexcept
- bool operator!=(string_view lhs, const char *rhs) noexcept
- bool operator!=(string_view lhs, const std::string &rhs) noexcept
- bool operator!=(string_view lhs, string_view rhs) noexcept
- template<class T1, class T2> bool operator!=(const shared_ptr<T1> &lhs, const␣
↪shared_ptr<T2> &rhs) noexcept
```

```
- template<class T1, class T2> bool operator!=(const unique_ptr<T1> &lhs, const␣
↪unique_ptr<T2> &rhs) noexcept
- template<class T> bool operator!=(const shared_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator!=(const unique_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator!=(std::nullptr_t, const shared_ptr<T> &rhs)␣
↪noexcept
- template<class T> bool operator!=(std::nullptr_t, const unique_ptr<T> &rhs)␣
↪noexcept
```

Template Function opentelemetry::nostd::operator!=(const shared_ptr<T>&, std::nullptr_t)

- Defined in file_include_opentelemetry_nostd_shared_ptr.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "opentelemetry::nostd::operator!=" with arguments (const shared_ptr<T>&, std::nullptr_t) in doxygen xml output for project "OpenTelemetry C++ API" from directory: ../../api/docs/doxyoutput/xml. Potential matches:

```
- bool operator!=(const char *lhs, string_view rhs) noexcept
- bool operator!=(const std::string &lhs, string_view rhs) noexcept
- bool operator!=(string_view lhs, const char *rhs) noexcept
- bool operator!=(string_view lhs, const std::string &rhs) noexcept
- bool operator!=(string_view lhs, string_view rhs) noexcept
- template<class T1, class T2> bool operator!=(const shared_ptr<T1> &lhs, const␣
↪shared_ptr<T2> &rhs) noexcept
- template<class T1, class T2> bool operator!=(const unique_ptr<T1> &lhs, const␣
↪unique_ptr<T2> &rhs) noexcept
- template<class T> bool operator!=(const shared_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator!=(const unique_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator!=(std::nullptr_t, const shared_ptr<T> &rhs)␣
↪noexcept
- template<class T> bool operator!=(std::nullptr_t, const unique_ptr<T> &rhs)␣
↪noexcept
```

Template Function opentelemetry::nostd::operator!=(std::nullptr_t, const shared_ptr<T>&)

- Defined in file_include_opentelemetry_nostd_shared_ptr.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "opentelemetry::nostd::operator!=" with arguments (std::nullptr_t, const shared_ptr<T>&) in doxygen xml output for project "OpenTelemetry C++ API" from directory: ../../api/docs/doxyoutput/xml. Potential matches:

```
- bool operator!=(const char *lhs, string_view rhs) noexcept
- bool operator!=(const std::string &lhs, string_view rhs) noexcept
- bool operator!=(string_view lhs, const char *rhs) noexcept
- bool operator!=(string_view lhs, const std::string &rhs) noexcept
- bool operator!=(string_view lhs, string_view rhs) noexcept
- template<class T1, class T2> bool operator!=(const shared_ptr<T1> &lhs, const␣
↪shared_ptr<T2> &rhs) noexcept
- template<class T1, class T2> bool operator!=(const unique_ptr<T1> &lhs, const␣
↪unique_ptr<T2> &rhs) noexcept
- template<class T> bool operator!=(const shared_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator!=(const unique_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator!=(std::nullptr_t, const shared_ptr<T> &rhs)␣
↪noexcept
- template<class T> bool operator!=(std::nullptr_t, const unique_ptr<T> &rhs)␣
↪noexcept
```

Function opentelemetry::nostd::operator!=(string_view, string_view)

- Defined in file_include_opentelemetry_nostd_string_view.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "opentelemetry::nostd::operator!=" with arguments (string_view, string_view) in doxygen xml output for project "OpenTelemetry C++ API" from directory: ../../api/docs/doxyoutput/xml. Potential matches:

```
- bool operator!=(const char *lhs, string_view rhs) noexcept
- bool operator!=(const std::string &lhs, string_view rhs) noexcept
- bool operator!=(string_view lhs, const char *rhs) noexcept
- bool operator!=(string_view lhs, const std::string &rhs) noexcept
- bool operator!=(string_view lhs, string_view rhs) noexcept
- template<class T1, class T2> bool operator!=(const shared_ptr<T1> &lhs, const␣
↪shared_ptr<T2> &rhs) noexcept
- template<class T1, class T2> bool operator!=(const unique_ptr<T1> &lhs, const␣
↪unique_ptr<T2> &rhs) noexcept
- template<class T> bool operator!=(const shared_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator!=(const unique_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator!=(std::nullptr_t, const shared_ptr<T> &rhs)␣
↪noexcept
- template<class T> bool operator!=(std::nullptr_t, const unique_ptr<T> &rhs)␣
↪noexcept
```

Function opentelemetry::nostd::operator!=(string_view, const std::string&)

- Defined in file_include_opentelemetry_nostd_string_view.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "opentelemetry::nostd::operator!=" with arguments (string_view, const std::string&) in doxygen xml output for project "OpenTelemetry C++ API" from directory: ../../api/docs/doxyoutput/xml. Potential matches:

```
- bool operator!=(const char *lhs, string_view rhs) noexcept
- bool operator!=(const std::string &lhs, string_view rhs) noexcept
- bool operator!=(string_view lhs, const char *rhs) noexcept
- bool operator!=(string_view lhs, const std::string &rhs) noexcept
- bool operator!=(string_view lhs, string_view rhs) noexcept
- template<class T1, class T2> bool operator!=(const shared_ptr<T1> &lhs, const␣
↪shared_ptr<T2> &rhs) noexcept
- template<class T1, class T2> bool operator!=(const unique_ptr<T1> &lhs, const␣
↪unique_ptr<T2> &rhs) noexcept
- template<class T> bool operator!=(const shared_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator!=(const unique_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator!=(std::nullptr_t, const shared_ptr<T> &rhs)␣
↪noexcept
- template<class T> bool operator!=(std::nullptr_t, const unique_ptr<T> &rhs)␣
↪noexcept
```

Function opentelemetry::nostd::operator!=(const std::string&, string_view)

- Defined in file_include_opentelemetry_nostd_string_view.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "opentelemetry::nostd::operator!=" with arguments (const std::string&, string_view) in doxygen xml output for project "OpenTelemetry C++ API" from directory: ../../api/docs/doxyoutput/xml. Potential matches:

```
- bool operator!=(const char *lhs, string_view rhs) noexcept
- bool operator!=(const std::string &lhs, string_view rhs) noexcept
- bool operator!=(string_view lhs, const char *rhs) noexcept
- bool operator!=(string_view lhs, const std::string &rhs) noexcept
- bool operator!=(string_view lhs, string_view rhs) noexcept
- template<class T1, class T2> bool operator!=(const shared_ptr<T1> &lhs, const␣
↪shared_ptr<T2> &rhs) noexcept
- template<class T1, class T2> bool operator!=(const unique_ptr<T1> &lhs, const␣
↪unique_ptr<T2> &rhs) noexcept
- template<class T> bool operator!=(const shared_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator!=(const unique_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
```

```
- template<class T> bool operator!=(std::nullptr_t, const shared_ptr<T> &rhs)␣
↪noexcept
- template<class T> bool operator!=(std::nullptr_t, const unique_ptr<T> &rhs)␣
↪noexcept
```

Function opentelemetry::nostd::operator!=(string_view, const char *)

- Defined in file_include_opentelemetry_nostd_string_view.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "opentelemetry::nostd::operator!=" with arguments (string_view, const char*) in doxygen xml output for project "OpenTelemetry C++ API" from directory: ../../api/docs/doxyoutput/xml. Potential matches:

```
- bool operator!=(const char *lhs, string_view rhs) noexcept
- bool operator!=(const std::string &lhs, string_view rhs) noexcept
- bool operator!=(string_view lhs, const char *rhs) noexcept
- bool operator!=(string_view lhs, const std::string &rhs) noexcept
- bool operator!=(string_view lhs, string_view rhs) noexcept
- template<class T1, class T2> bool operator!=(const shared_ptr<T1> &lhs, const␣
↪shared_ptr<T2> &rhs) noexcept
- template<class T1, class T2> bool operator!=(const unique_ptr<T1> &lhs, const␣
↪unique_ptr<T2> &rhs) noexcept
- template<class T> bool operator!=(const shared_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator!=(const unique_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator!=(std::nullptr_t, const shared_ptr<T> &rhs)␣
↪noexcept
- template<class T> bool operator!=(std::nullptr_t, const unique_ptr<T> &rhs)␣
↪noexcept
```

Function opentelemetry::nostd::operator!=(const char *, string_view)

- Defined in file_include_opentelemetry_nostd_string_view.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "opentelemetry::nostd::operator!=" with arguments (const char*, string_view) in doxygen xml output for project "OpenTelemetry C++ API" from directory: ../../api/docs/doxyoutput/xml. Potential matches:

```
- bool operator!=(const char *lhs, string_view rhs) noexcept
- bool operator!=(const std::string &lhs, string_view rhs) noexcept
- bool operator!=(string_view lhs, const char *rhs) noexcept
- bool operator!=(string_view lhs, const std::string &rhs) noexcept
- bool operator!=(string_view lhs, string_view rhs) noexcept
```

```
- template<class T1, class T2> bool operator!=(const shared_ptr<T1> &lhs, const␣
↪shared_ptr<T2> &rhs) noexcept
- template<class T1, class T2> bool operator!=(const unique_ptr<T1> &lhs, const␣
↪unique_ptr<T2> &rhs) noexcept
- template<class T> bool operator!=(const shared_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator!=(const unique_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator!=(std::nullptr_t, const shared_ptr<T> &rhs)␣
↪noexcept
- template<class T> bool operator!=(std::nullptr_t, const unique_ptr<T> &rhs)␣
↪noexcept
```

Template Function opentelemetry::nostd::operator!=(const unique_ptr<T1>&, const unique_ptr<T2>&)

- Defined in file_include_opentelemetry_nostd_unique_ptr.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "opentelemetry::nostd::operator!=" with arguments (const unique_ptr<T1>&, const unique_ptr<T2>&) in doxygen xml output for project "OpenTelemetry C++ API" from directory: ../../api/docs/doxyoutput/xml. Potential matches:

```
- bool operator!=(const char *lhs, string_view rhs) noexcept
- bool operator!=(const std::string &lhs, string_view rhs) noexcept
- bool operator!=(string_view lhs, const char *rhs) noexcept
- bool operator!=(string_view lhs, const std::string &rhs) noexcept
- bool operator!=(string_view lhs, string_view rhs) noexcept
- template<class T1, class T2> bool operator!=(const shared_ptr<T1> &lhs, const␣
↪shared_ptr<T2> &rhs) noexcept
- template<class T1, class T2> bool operator!=(const unique_ptr<T1> &lhs, const␣
↪unique_ptr<T2> &rhs) noexcept
- template<class T> bool operator!=(const shared_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator!=(const unique_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator!=(std::nullptr_t, const shared_ptr<T> &rhs)␣
↪noexcept
- template<class T> bool operator!=(std::nullptr_t, const unique_ptr<T> &rhs)␣
↪noexcept
```

Template Function opentelemetry::nostd::operator!=(const unique_ptr<T>&, std::nullptr_t)

- Defined in file_include_opentelemetry_nostd_unique_ptr.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "opentelemetry::nostd::operator!=" with arguments (const unique_ptr<T>&, std::nullptr_t) in doxygen xml output for project "OpenTelemetry C++ API" from directory: ../../api/docs/doxyoutput/xml. Potential matches:

```
- bool operator!=(const char *lhs, string_view rhs) noexcept
- bool operator!=(const std::string &lhs, string_view rhs) noexcept
- bool operator!=(string_view lhs, const char *rhs) noexcept
- bool operator!=(string_view lhs, const std::string &rhs) noexcept
- bool operator!=(string_view lhs, string_view rhs) noexcept
- template<class T1, class T2> bool operator!=(const shared_ptr<T1> &lhs, const␣
↪shared_ptr<T2> &rhs) noexcept
- template<class T1, class T2> bool operator!=(const unique_ptr<T1> &lhs, const␣
↪unique_ptr<T2> &rhs) noexcept
- template<class T> bool operator!=(const shared_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator!=(const unique_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator!=(std::nullptr_t, const shared_ptr<T> &rhs)␣
↪noexcept
- template<class T> bool operator!=(std::nullptr_t, const unique_ptr<T> &rhs)␣
↪noexcept
```

Template Function opentelemetry::nostd::operator!=(std::nullptr_t, const unique_ptr<T>&)

- Defined in file_include_opentelemetry_nostd_unique_ptr.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "opentelemetry::nostd::operator!=" with arguments (std::nullptr_t, const unique_ptr<T>&) in doxygen xml output for project "OpenTelemetry C++ API" from directory: ../../api/docs/doxyoutput/xml. Potential matches:

```
- bool operator!=(const char *lhs, string_view rhs) noexcept
- bool operator!=(const std::string &lhs, string_view rhs) noexcept
- bool operator!=(string_view lhs, const char *rhs) noexcept
- bool operator!=(string_view lhs, const std::string &rhs) noexcept
- bool operator!=(string_view lhs, string_view rhs) noexcept
- template<class T1, class T2> bool operator!=(const shared_ptr<T1> &lhs, const␣
↪shared_ptr<T2> &rhs) noexcept
- template<class T1, class T2> bool operator!=(const unique_ptr<T1> &lhs, const␣
↪unique_ptr<T2> &rhs) noexcept
- template<class T> bool operator!=(const shared_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator!=(const unique_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
```

```
- template<class T> bool operator!=(std::nullptr_t, const shared_ptr<T> &rhs) noexcept
- template<class T> bool operator!=(std::nullptr_t, const unique_ptr<T> &rhs) noexcept
```

Function opentelemetry::nostd::operator<<

Function Documentation

inline std::ostream &opentelemetry::nostd::**operator<<**(std::ostream &*os*, *string_view s*)

Template Function opentelemetry::nostd::operator==(const shared_ptr<T1>&, const shared_ptr<T2>&)

- Defined in file_include_opentelemetry_nostd_shared_ptr.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "opentelemetry::nostd::operator==" with arguments (const shared_ptr<T1>&, const shared_ptr<T2>&) in doxygen xml output for project "OpenTelemetry C++ API" from directory: ../../api/docs/doxyoutput/xml. Potential matches:

```
- bool operator==(const char *lhs, string_view rhs) noexcept
- bool operator==(const std::string &lhs, string_view rhs) noexcept
- bool operator==(string_view lhs, const char *rhs) noexcept
- bool operator==(string_view lhs, const std::string &rhs) noexcept
- bool operator==(string_view lhs, string_view rhs) noexcept
- template<class T1, class T2> bool operator==(const shared_ptr<T1> &lhs, const shared_ptr<T2> &rhs) noexcept
- template<class T1, class T2> bool operator==(const unique_ptr<T1> &lhs, const unique_ptr<T2> &rhs) noexcept
- template<class T> bool operator==(const shared_ptr<T> &lhs, std::nullptr_t) noexcept
- template<class T> bool operator==(const unique_ptr<T> &lhs, std::nullptr_t) noexcept
- template<class T> bool operator==(std::nullptr_t, const shared_ptr<T> &rhs) noexcept
- template<class T> bool operator==(std::nullptr_t, const unique_ptr<T> &rhs) noexcept
```

Template Function opentelemetry::nostd::operator==(const shared_ptr<T>&, std::nullptr_t)

- Defined in file_include_opentelemetry_nostd_shared_ptr.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "opentelemetry::nostd::operator==" with arguments (const shared_ptr<T>&, std::nullptr_t) in doxygen xml output for project "OpenTelemetry C++ API" from directory: ../../api/docs/doxyoutput/xml. Potential matches:

```
- bool operator==(const char *lhs, string_view rhs) noexcept
- bool operator==(const std::string &lhs, string_view rhs) noexcept
- bool operator==(string_view lhs, const char *rhs) noexcept
- bool operator==(string_view lhs, const std::string &rhs) noexcept
- bool operator==(string_view lhs, string_view rhs) noexcept
- template<class T1, class T2> bool operator==(const shared_ptr<T1> &lhs, const␣
↪shared_ptr<T2> &rhs) noexcept
- template<class T1, class T2> bool operator==(const unique_ptr<T1> &lhs, const␣
↪unique_ptr<T2> &rhs) noexcept
- template<class T> bool operator==(const shared_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator==(const unique_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator==(std::nullptr_t, const shared_ptr<T> &rhs)␣
↪noexcept
- template<class T> bool operator==(std::nullptr_t, const unique_ptr<T> &rhs)␣
↪noexcept
```

Template Function opentelemetry::nostd::operator==(std::nullptr_t, const shared_ptr<T>&)

- Defined in file_include_opentelemetry_nostd_shared_ptr.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "opentelemetry::nostd::operator==" with arguments (std::nullptr_t, const shared_ptr<T>&) in doxygen xml output for project "OpenTelemetry C++ API" from directory: ../../api/docs/doxyoutput/xml. Potential matches:

```
- bool operator==(const char *lhs, string_view rhs) noexcept
- bool operator==(const std::string &lhs, string_view rhs) noexcept
- bool operator==(string_view lhs, const char *rhs) noexcept
- bool operator==(string_view lhs, const std::string &rhs) noexcept
- bool operator==(string_view lhs, string_view rhs) noexcept
- template<class T1, class T2> bool operator==(const shared_ptr<T1> &lhs, const␣
↪shared_ptr<T2> &rhs) noexcept
- template<class T1, class T2> bool operator==(const unique_ptr<T1> &lhs, const␣
↪unique_ptr<T2> &rhs) noexcept
- template<class T> bool operator==(const shared_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator==(const unique_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
```

```
- template<class T> bool operator==(std::nullptr_t, const shared_ptr<T> &rhs)␣
↪noexcept
- template<class T> bool operator==(std::nullptr_t, const unique_ptr<T> &rhs)␣
↪noexcept
```

Function opentelemetry::nostd::operator==(string_view, string_view)

- Defined in file_include_opentelemetry_nostd_string_view.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "opentelemetry::nostd::operator==" with arguments (string_view, string_view) in doxygen xml output for project "OpenTelemetry C++ API" from directory: ../../api/docs/doxyoutput/xml. Potential matches:

```
- bool operator==(const char *lhs, string_view rhs) noexcept
- bool operator==(const std::string &lhs, string_view rhs) noexcept
- bool operator==(string_view lhs, const char *rhs) noexcept
- bool operator==(string_view lhs, const std::string &rhs) noexcept
- bool operator==(string_view lhs, string_view rhs) noexcept
- template<class T1, class T2> bool operator==(const shared_ptr<T1> &lhs, const␣
↪shared_ptr<T2> &rhs) noexcept
- template<class T1, class T2> bool operator==(const unique_ptr<T1> &lhs, const␣
↪unique_ptr<T2> &rhs) noexcept
- template<class T> bool operator==(const shared_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator==(const unique_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator==(std::nullptr_t, const shared_ptr<T> &rhs)␣
↪noexcept
- template<class T> bool operator==(std::nullptr_t, const unique_ptr<T> &rhs)␣
↪noexcept
```

Function opentelemetry::nostd::operator==(string_view, const std::string&)

- Defined in file_include_opentelemetry_nostd_string_view.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "opentelemetry::nostd::operator==" with arguments (string_view, const std::string&) in doxygen xml output for project "OpenTelemetry C++ API" from directory: ../../api/docs/doxyoutput/xml. Potential matches:

```
- bool operator==(const char *lhs, string_view rhs) noexcept
- bool operator==(const std::string &lhs, string_view rhs) noexcept
- bool operator==(string_view lhs, const char *rhs) noexcept
- bool operator==(string_view lhs, const std::string &rhs) noexcept
- bool operator==(string_view lhs, string_view rhs) noexcept
```

```
- template<class T1, class T2> bool operator==(const shared_ptr<T1> &lhs, const␣
↪shared_ptr<T2> &rhs) noexcept
- template<class T1, class T2> bool operator==(const unique_ptr<T1> &lhs, const␣
↪unique_ptr<T2> &rhs) noexcept
- template<class T> bool operator==(const shared_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator==(const unique_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator==(std::nullptr_t, const shared_ptr<T> &rhs)␣
↪noexcept
- template<class T> bool operator==(std::nullptr_t, const unique_ptr<T> &rhs)␣
↪noexcept
```

Function opentelemetry::nostd::operator==(const std::string&, string_view)

- Defined in file_include_opentelemetry_nostd_string_view.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "opentelemetry::nostd::operator==" with arguments (const std::string&, string_view) in doxygen xml output for project "OpenTelemetry C++ API" from directory: ../../api/docs/doxyoutput/xml. Potential matches:

```
- bool operator==(const char *lhs, string_view rhs) noexcept
- bool operator==(const std::string &lhs, string_view rhs) noexcept
- bool operator==(string_view lhs, const char *rhs) noexcept
- bool operator==(string_view lhs, const std::string &rhs) noexcept
- bool operator==(string_view lhs, string_view rhs) noexcept
- template<class T1, class T2> bool operator==(const shared_ptr<T1> &lhs, const␣
↪shared_ptr<T2> &rhs) noexcept
- template<class T1, class T2> bool operator==(const unique_ptr<T1> &lhs, const␣
↪unique_ptr<T2> &rhs) noexcept
- template<class T> bool operator==(const shared_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator==(const unique_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator==(std::nullptr_t, const shared_ptr<T> &rhs)␣
↪noexcept
- template<class T> bool operator==(std::nullptr_t, const unique_ptr<T> &rhs)␣
↪noexcept
```

Function opentelemetry::nostd::operator==(string_view, const char *)

- Defined in file_include_opentelemetry_nostd_string_view.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "opentelemetry::nostd::operator==" with arguments (string_view, const char*) in doxygen xml output for project "OpenTelemetry C++ API" from directory: ../../api/docs/doxyoutput/xml. Potential matches:

```
- bool operator==(const char *lhs, string_view rhs) noexcept
- bool operator==(const std::string &lhs, string_view rhs) noexcept
- bool operator==(string_view lhs, const char *rhs) noexcept
- bool operator==(string_view lhs, const std::string &rhs) noexcept
- bool operator==(string_view lhs, string_view rhs) noexcept
- template<class T1, class T2> bool operator==(const shared_ptr<T1> &lhs, const␣
↪shared_ptr<T2> &rhs) noexcept
- template<class T1, class T2> bool operator==(const unique_ptr<T1> &lhs, const␣
↪unique_ptr<T2> &rhs) noexcept
- template<class T> bool operator==(const shared_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator==(const unique_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator==(std::nullptr_t, const shared_ptr<T> &rhs)␣
↪noexcept
- template<class T> bool operator==(std::nullptr_t, const unique_ptr<T> &rhs)␣
↪noexcept
```

Function opentelemetry::nostd::operator==(const char *, string_view)

- Defined in file_include_opentelemetry_nostd_string_view.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "opentelemetry::nostd::operator==" with arguments (const char*, string_view) in doxygen xml output for project "OpenTelemetry C++ API" from directory: ../../api/docs/doxyoutput/xml. Potential matches:

```
- bool operator==(const char *lhs, string_view rhs) noexcept
- bool operator==(const std::string &lhs, string_view rhs) noexcept
- bool operator==(string_view lhs, const char *rhs) noexcept
- bool operator==(string_view lhs, const std::string &rhs) noexcept
- bool operator==(string_view lhs, string_view rhs) noexcept
- template<class T1, class T2> bool operator==(const shared_ptr<T1> &lhs, const␣
↪shared_ptr<T2> &rhs) noexcept
- template<class T1, class T2> bool operator==(const unique_ptr<T1> &lhs, const␣
↪unique_ptr<T2> &rhs) noexcept
- template<class T> bool operator==(const shared_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator==(const unique_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
```

```
- template<class T> bool operator==(std::nullptr_t, const shared_ptr<T> &rhs)␣
↪noexcept
- template<class T> bool operator==(std::nullptr_t, const unique_ptr<T> &rhs)␣
↪noexcept
```

Template Function opentelemetry::nostd::operator==(const unique_ptr<T1>&, const unique_ptr<T2>&)

- Defined in file_include_opentelemetry_nostd_unique_ptr.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "opentelemetry::nostd::operator==" with arguments (const unique_ptr<T1>&, const unique_ptr<T2>&) in doxygen xml output for project "OpenTelemetry C++ API" from directory: ../../api/docs/doxyoutput/xml. Potential matches:

```
- bool operator==(const char *lhs, string_view rhs) noexcept
- bool operator==(const std::string &lhs, string_view rhs) noexcept
- bool operator==(string_view lhs, const char *rhs) noexcept
- bool operator==(string_view lhs, const std::string &rhs) noexcept
- bool operator==(string_view lhs, string_view rhs) noexcept
- template<class T1, class T2> bool operator==(const shared_ptr<T1> &lhs, const␣
↪shared_ptr<T2> &rhs) noexcept
- template<class T1, class T2> bool operator==(const unique_ptr<T1> &lhs, const␣
↪unique_ptr<T2> &rhs) noexcept
- template<class T> bool operator==(const shared_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator==(const unique_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator==(std::nullptr_t, const shared_ptr<T> &rhs)␣
↪noexcept
- template<class T> bool operator==(std::nullptr_t, const unique_ptr<T> &rhs)␣
↪noexcept
```

Template Function opentelemetry::nostd::operator==(const unique_ptr<T>&, std::nullptr_t)

- Defined in file_include_opentelemetry_nostd_unique_ptr.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "opentelemetry::nostd::operator==" with arguments (const unique_ptr<T>&, std::nullptr_t) in doxygen xml output for project "OpenTelemetry C++ API" from directory: ../../api/docs/doxyoutput/xml. Potential matches:

```
- bool operator==(const char *lhs, string_view rhs) noexcept
- bool operator==(const std::string &lhs, string_view rhs) noexcept
- bool operator==(string_view lhs, const char *rhs) noexcept
- bool operator==(string_view lhs, const std::string &rhs) noexcept
```

```
- bool operator==(string_view lhs, string_view rhs) noexcept
- template<class T1, class T2> bool operator==(const shared_ptr<T1> &lhs, const␣
↪shared_ptr<T2> &rhs) noexcept
- template<class T1, class T2> bool operator==(const unique_ptr<T1> &lhs, const␣
↪unique_ptr<T2> &rhs) noexcept
- template<class T> bool operator==(const shared_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator==(const unique_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator==(std::nullptr_t, const shared_ptr<T> &rhs)␣
↪noexcept
- template<class T> bool operator==(std::nullptr_t, const unique_ptr<T> &rhs)␣
↪noexcept
```

Template Function opentelemetry::nostd::operator==(std::nullptr_t, const unique_ptr<T>&)

- Defined in file_include_opentelemetry_nostd_unique_ptr.h

Function Documentation

Warning: doxygenfunction: Unable to resolve function "opentelemetry::nostd::operator==" with arguments (std::nullptr_t, const unique_ptr<T>&) in doxygen xml output for project "OpenTelemetry C++ API" from directory: ../../api/docs/doxyoutput/xml. Potential matches:

```
- bool operator==(const char *lhs, string_view rhs) noexcept
- bool operator==(const std::string &lhs, string_view rhs) noexcept
- bool operator==(string_view lhs, const char *rhs) noexcept
- bool operator==(string_view lhs, const std::string &rhs) noexcept
- bool operator==(string_view lhs, string_view rhs) noexcept
- template<class T1, class T2> bool operator==(const shared_ptr<T1> &lhs, const␣
↪shared_ptr<T2> &rhs) noexcept
- template<class T1, class T2> bool operator==(const unique_ptr<T1> &lhs, const␣
↪unique_ptr<T2> &rhs) noexcept
- template<class T> bool operator==(const shared_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator==(const unique_ptr<T> &lhs, std::nullptr_t)␣
↪noexcept
- template<class T> bool operator==(std::nullptr_t, const shared_ptr<T> &rhs)␣
↪noexcept
- template<class T> bool operator==(std::nullptr_t, const unique_ptr<T> &rhs)␣
↪noexcept
```

Function opentelemetry::trace::propagation::detail::GetCurrentSpan

- Defined in file_include_opentelemetry_trace_propagation_detail_context.h

Function Documentation

trace::*SpanContext* opentelemetry::trace::propagation::detail::**GetCurrentSpan**(**const** *context*::Context &*context*)

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

- Defined in file_include_opentelemetry_trace_propagation_detail_hex.h

Function Documentation

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

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

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.

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::nostd::dynamic_extent

- Defined in file_include_opentelemetry_nostd_span.h

Variable Documentation

constexpr size_t opentelemetry::nostd::**dynamic_extent** = static_cast<size_t>(-1)

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, uint64_t, double, nos

Typedef opentelemetry::nostd::Traits

- Defined in file_include_opentelemetry_nostd_string_view.h

Typedef Documentation

using opentelemetry::nostd::**Traits** = std::char_traits<char>

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*), 65

O

opentelemetry::common::AttributeType (*C++ enum*), 43
opentelemetry::common::AttributeType::TYPE_BOOL (*C++ enumerator*), 43
opentelemetry::common::AttributeType::TYPE_DOUBLE (*C++ enumerator*), 43
opentelemetry::common::AttributeType::TYPE_INT (*C++ enumerator*), 43
opentelemetry::common::AttributeType::TYPE_INT64 (*C++ enumerator*), 43
opentelemetry::common::AttributeType::TYPE_SPAN_BOOL (*C++ enumerator*), 43
opentelemetry::common::AttributeType::TYPE_SPAN_DOUBLE (*C++ enumerator*), 44
opentelemetry::common::AttributeType::TYPE_SPAN_INT (*C++ enumerator*), 44
opentelemetry::common::AttributeType::TYPE_SPAN_INT64 (*C++ enumerator*), 44
opentelemetry::common::AttributeType::TYPE_SPAN_STRING (*C++ enumerator*), 44
opentelemetry::common::AttributeType::TYPE_SPAN_UINT (*C++ enumerator*), 44
opentelemetry::common::AttributeType::TYPE_SPAN_UINT64 (*C++ enumerator*), 44
opentelemetry::common::AttributeType::TYPE_STRING (*C++ enumerator*), 43
opentelemetry::common::AttributeType::TYPE_UINT (*C++ enumerator*), 43
opentelemetry::common::AttributeType::TYPE_UINT64 (*C++ enumerator*), 43
opentelemetry::common::AttributeValue (*C++ type*), 65
opentelemetry::common::KeyValueIterable (*C++ class*), 16
opentelemetry::common::KeyValueIterable::~KeyValueIterable (*C++ function*), 16
opentelemetry::common::KeyValueIterable::ForEachKeyValue (*C++ function*), 16
opentelemetry::common::KeyValueIterable::size (*C++ function*), 16
opentelemetry::common::NullKeyValueIterable (*C++ class*), 16
opentelemetry::common::NullKeyValueIterable::ForEac (*C++ function*), 17
opentelemetry::common::NullKeyValueIterable::NullKe (*C++ function*), 17
opentelemetry::common::NullKeyValueIterable::size (*C++ function*), 17
opentelemetry::core::SteadyTimestamp (*C++ class*), 17
opentelemetry::core::SteadyTimestamp::operator std::chrono::steady_clock::time_point (*C++ function*), 17
opentelemetry::core::SteadyTimestamp::operator!= (*C++ function*), 17
opentelemetry::core::SteadyTimestamp::operator== (*C++ function*), 17
opentelemetry::core::SteadyTimestamp::SteadyTimesta (*C++ function*), 17
opentelemetry::core::SteadyTimestamp::time_since_ep (*C++ function*), 17
opentelemetry::core::SystemTimestamp (*C++ class*), 17
opentelemetry::core::SystemTimestamp::operator std::chrono::system_clock::time_point (*C++ function*), 18
opentelemetry::core::SystemTimestamp::operator!= (*C++ function*), 18
opentelemetry::core::SystemTimestamp::operator== (*C++ function*), 18
opentelemetry::core::SystemTimestamp::SystemTimesta (*C++ function*), 18
opentelemetry::core::SystemTimestamp::time_since_ep (*C++ function*), 18
opentelemetry::nostd::detail::is_specialized_span_c (*C++ struct*), 12
opentelemetry::nostd::detail::is_specialized_span_c Extent>> (*C++ struct*), 12
opentelemetry::nostd::detail::is_specialized_span_c N>> (*C++ struct*), 13

opentelemetry::nostd::detail::is_specialization_of_span_convertible< (C++ struct), 13
opentelemetry::nostd::detail::unique_ptr_element_type (C++ struct), 13
opentelemetry::nostd::detail::unique_ptr_element_type::type (C++ type), 14
opentelemetry::nostd::detail::unique_ptr_element_type<T[]> (C++ struct), 14
opentelemetry::nostd::detail::unique_ptr_element_type<T[]>::type (C++ type), 14
opentelemetry::nostd::dynamic_extent (C++ member), 61
opentelemetry::nostd::operator<< (C++ function), 53
opentelemetry::nostd::shared_ptr (C++ class), 18
opentelemetry::nostd::shared_ptr::~shared_ptr (C++ function), 19
opentelemetry::nostd::shared_ptr::element_type (C++ type), 18
opentelemetry::nostd::shared_ptr::get (C++ function), 19
opentelemetry::nostd::shared_ptr::operator bool (C++ function), 19
opentelemetry::nostd::shared_ptr::operator* (C++ function), 19
opentelemetry::nostd::shared_ptr::operator= (C++ function), 19
opentelemetry::nostd::shared_ptr::operator-> (C++ function), 19
opentelemetry::nostd::shared_ptr::PlacementBuffer (C++ struct), 14
opentelemetry::nostd::shared_ptr::PlacementBuffer::data_ (C++ member), 14
opentelemetry::nostd::shared_ptr::pointer (C++ type), 18
opentelemetry::nostd::shared_ptr::shared_ptr (C++ function), 19
opentelemetry::nostd::shared_ptr::shared_ptr_wrapper (C++ class), 19
opentelemetry::nostd::shared_ptr::shared_ptr_wrapper::~shared_ptr_wrapper (C++ function), 20
opentelemetry::nostd::shared_ptr::shared_ptr_wrapper::CopyTo (C++ function), 20
opentelemetry::nostd::shared_ptr::shared_ptr_wrapper::Get (C++ function), 20
opentelemetry::nostd::shared_ptr::shared_ptr_wrapper::IsShared (C++ function), 20
opentelemetry::nostd::shared_ptr::shared_ptr_wrapper::IsSharedWith (C++ function), 20
opentelemetry::nostd::shared_ptr::shared_ptr_wrapper::shared_ptr_wrapper (C++ function), 20
opentelemetry::nostd::shared_ptr::swap (C++ function), 19
opentelemetry::nostd::span (C++ class), 20
opentelemetry::nostd::span::begin (C++ function), 21
opentelemetry::nostd::span::data (C++ function), 21
opentelemetry::nostd::span::empty (C++ function), 21
opentelemetry::nostd::span::end (C++ function), 21
opentelemetry::nostd::span::extent (C++ member), 21
opentelemetry::nostd::span::operator[] (C++ function), 21
opentelemetry::nostd::span::size (C++ function), 21
opentelemetry::nostd::span::span (C++ function), 20, 21
opentelemetry::nostd::span<T, dynamic_extent> (C++ class), 21
opentelemetry::nostd::span<T, dynamic_extent>::begin (C++ function), 22
opentelemetry::nostd::span<T, dynamic_extent>::data (C++ function), 21
opentelemetry::nostd::span<T, dynamic_extent>::empty (C++ function), 21
opentelemetry::nostd::span<T, dynamic_extent>::end (C++ function), 22
opentelemetry::nostd::span<T, dynamic_extent>::extent (C++ member), 22
opentelemetry::nostd::span<T, dynamic_extent>::operator[] (C++ function), 22
opentelemetry::nostd::span<T, dynamic_extent>::size (C++ function), 22
opentelemetry::nostd::span<T, dynamic_extent>::span (C++ function), 21
opentelemetry::nostd::string_view (C++ class), 22
opentelemetry::nostd::string_view::begin (C++ function), 22
opentelemetry::nostd::string_view::compare (C++ function), 22, 23
opentelemetry::nostd::string_view::data (C++ function), 22
opentelemetry::nostd::string_view::empty (C++ function), 22
opentelemetry::nostd::string_view::end

(*C++ function*), 22
opentelemetry::nostd::string_view::find (*C++ function*), 23
opentelemetry::nostd::string_view::length (*C++ function*), 22
opentelemetry::nostd::string_view::npos (*C++ member*), 23
opentelemetry::nostd::string_view::operator std::string (*C++ function*), 22
opentelemetry::nostd::string_view::operator> (*C++ function*), 23
opentelemetry::nostd::string_view::operator< (*C++ function*), 23
opentelemetry::nostd::string_view::operator[] (*C++ function*), 22
opentelemetry::nostd::string_view::size (*C++ function*), 22
opentelemetry::nostd::string_view::size_type (*C++ type*), 22
opentelemetry::nostd::string_view::string_view (*C++ function*), 22
opentelemetry::nostd::string_view::substr (*C++ function*), 22
opentelemetry::nostd::Traits (*C++ type*), 65
opentelemetry::nostd::unique_ptr (*C++ class*), 23
opentelemetry::nostd::unique_ptr::~unique_ptr (*C++ function*), 23
opentelemetry::nostd::unique_ptr::element_type (*C++ type*), 23
opentelemetry::nostd::unique_ptr::get (*C++ function*), 24
opentelemetry::nostd::unique_ptr::operator bool (*C++ function*), 24
opentelemetry::nostd::unique_ptr::operator std::unique_ptr<T> (*C++ function*), 24
opentelemetry::nostd::unique_ptr::operator* (*C++ function*), 24
opentelemetry::nostd::unique_ptr::operator= (*C++ function*), 23, 24
opentelemetry::nostd::unique_ptr::operator-> (*C++ function*), 24
opentelemetry::nostd::unique_ptr::pointer (*C++ type*), 23
opentelemetry::nostd::unique_ptr::release (*C++ function*), 24
opentelemetry::nostd::unique_ptr::reset (*C++ function*), 24
opentelemetry::nostd::unique_ptr::swap (*C++ function*), 24
opentelemetry::nostd::unique_ptr::unique_ptr (*C++ function*), 23
opentelemetry::trace::CanonicalCode (*C++ enum*), 44
opentelemetry::trace::CanonicalCode::ABORTED (*C++ enumerator*), 45
opentelemetry::trace::CanonicalCode::ALREADY_EXISTS (*C++ enumerator*), 44
opentelemetry::trace::CanonicalCode::CANCELLED (*C++ enumerator*), 44
opentelemetry::trace::CanonicalCode::DATA_LOSS (*C++ enumerator*), 45
opentelemetry::trace::CanonicalCode::DEADLINE_EXCEE (*C++ enumerator*), 44
opentelemetry::trace::CanonicalCode::FAILED_PRECOND (*C++ enumerator*), 44
opentelemetry::trace::CanonicalCode::INTERNAL (*C++ enumerator*), 45
opentelemetry::trace::CanonicalCode::INVALID_ARGUME (*C++ enumerator*), 44
opentelemetry::trace::CanonicalCode::NOT_FOUND (*C++ enumerator*), 44
opentelemetry::trace::CanonicalCode::OK (*C++ enumerator*), 44
opentelemetry::trace::CanonicalCode::OUT_OF_RANGE (*C++ enumerator*), 45
opentelemetry::trace::CanonicalCode::PERMISSION_DEN (*C++ enumerator*), 44
opentelemetry::trace::CanonicalCode::RESOURCE_EXHAU (*C++ enumerator*), 44
opentelemetry::trace::CanonicalCode::UNAUTHENTICATE (*C++ enumerator*), 45
opentelemetry::trace::CanonicalCode::UNAVAILABLE (*C++ enumerator*), 45
opentelemetry::trace::CanonicalCode::UNIMPLEMENTED (*C++ enumerator*), 45
opentelemetry::trace::CanonicalCode::UNKNOWN (*C++ enumerator*), 44
opentelemetry::trace::DefaultSpan (*C++ class*), 24
opentelemetry::trace::DefaultSpan::AddEvent (*C++ function*), 24
opentelemetry::trace::DefaultSpan::DefaultSpan (*C++ function*), 25
opentelemetry::trace::DefaultSpan::End (*C++ function*), 24
opentelemetry::trace::DefaultSpan::GetContext (*C++ function*), 24
opentelemetry::trace::DefaultSpan::GetInvalid (*C++ function*), 25
opentelemetry::trace::DefaultSpan::IsRecording (*C++ function*), 24
opentelemetry::trace::DefaultSpan::SetAttribute (*C++ function*), 24
opentelemetry::trace::DefaultSpan::SetStatus (*C++ function*), 24
opentelemetry::trace::DefaultSpan::ToString (*C++ function*), 25

opentelemetry::trace::DefaultSpan::UpdateName (C++ function), 24
opentelemetry::trace::DefaultTracer (C++ class), 25
opentelemetry::trace::DefaultTracer::~DefaultTracer (C++ function), 25
opentelemetry::trace::EndSpanOptions (C++ struct), 14
opentelemetry::trace::EndSpanOptions::end_steady_time (C++ member), 15
opentelemetry::trace::kSpanKey (C++ member), 62
opentelemetry::trace::NoopSpan (C++ class), 26
opentelemetry::trace::NoopSpan::AddEvent (C++ function), 26
opentelemetry::trace::NoopSpan::End (C++ function), 26
opentelemetry::trace::NoopSpan::GetContext (C++ function), 26
opentelemetry::trace::NoopSpan::IsRecording (C++ function), 26
opentelemetry::trace::NoopSpan::NoopSpan (C++ function), 26
opentelemetry::trace::NoopSpan::SetAttribute (C++ function), 26
opentelemetry::trace::NoopSpan::SetStatus (C++ function), 26
opentelemetry::trace::NoopSpan::UpdateName (C++ function), 26
opentelemetry::trace::NoopTracer (C++ class), 27
opentelemetry::trace::NoopTracer::CloseWithMicroseconds (C++ function), 27
opentelemetry::trace::NoopTracer::ForceFlushWithMicroseconds (C++ function), 27
opentelemetry::trace::NoopTracer::StartSpan (C++ function), 27
opentelemetry::trace::NoopTracerProvider (C++ class), 27
opentelemetry::trace::NoopTracerProvider::GetTracer (C++ function), 27
opentelemetry::trace::NoopTracerProvider::NoopTracerProvider (C++ function), 27
opentelemetry::trace::propagation::B3Propagator (C++ class), 28
opentelemetry::trace::propagation::B3Propagator::Inject (C++ function), 28
opentelemetry::trace::propagation::B3Propagator::kB3Header (C++ type), 28
opentelemetry::trace::propagation::B3PropagatorExtractor (C++ class), 29
opentelemetry::trace::propagation::B3PropagatorExtractor::Extract (C++ function), 29
opentelemetry::trace::propagation::B3PropagatorExtractor::kB3Header (C++ type), 29
opentelemetry::trace::propagation::B3PropagatorExtractor::kFields (C++ function), 29
opentelemetry::trace::propagation::B3PropagatorExtractor::kValidBitForDebug (C++ function), 29
opentelemetry::trace::propagation::B3PropagatorExtractor::kValidBitForSampled (C++ function), 29
opentelemetry::trace::propagation::B3PropagatorMultiHeader (C++ class), 29
opentelemetry::trace::propagation::B3PropagatorMultiHeader::Inject (C++ function), 30
opentelemetry::trace::propagation::B3PropagatorMultiHeader::kB3Header (C++ type), 29
opentelemetry::trace::propagation::CompositePropagator (C++ class), 30
opentelemetry::trace::propagation::CompositePropagator::Inject (C++ function), 30
opentelemetry::trace::propagation::CompositePropagator::kFields (C++ function), 30
opentelemetry::trace::propagation::CompositePropagator::kSupportedPropagators (C++ type), 30
opentelemetry::trace::propagation::CompositePropagator::SupportedPropagators (C++ function), 30
opentelemetry::trace::propagation::CompositePropagator::SupportedPropagators (C++ type), 30
opentelemetry::trace::propagation::detail::GetCurrentTimeStamp (C++ function), 60
opentelemetry::trace::propagation::detail::HexToBinary (C++ function), 60
opentelemetry::trace::propagation::detail::HexToInt (C++ function), 60
opentelemetry::trace::propagation::detail::IsValidHex (C++ function), 60
opentelemetry::trace::propagation::detail::kHexDigitToBinary (C++ member), 62
opentelemetry::trace::propagation::detail::SplitString (C++ function), 61
opentelemetry::trace::propagation::HttpTraceContext (C++ class), 31
opentelemetry::trace::propagation::HttpTraceContext::GetTracer (C++ function), 31
opentelemetry::trace::propagation::HttpTraceContext::kTracerProvider (C++ type), 31
opentelemetry::trace::propagation::HttpTraceContext::HttpTraceContext (C++ function), 31
opentelemetry::trace::propagation::HttpTraceContext::Inject (C++ type), 31
opentelemetry::trace::propagation::HttpTraceContext::kTraceParent (C++ function), 31
opentelemetry::trace::propagation::HttpTraceContext::kTraceState (C++ function), 31
opentelemetry::trace::propagation::HttpTraceContext::Extract (C++ function), 31

opentelemetry::trace::propagation::JaegerPropagator (C++ class), 32
opentelemetry::trace::propagation::JaegerPropagator::Extract (C++ function), 32
opentelemetry::trace::propagation::JaegerPropagator::kTraceContextKey (C++ type), 32
opentelemetry::trace::propagation::JaegerPropagator::Inject (C++ function), 32
opentelemetry::trace::propagation::JaegerPropagator::kTraceHeader (C++ type), 32
opentelemetry::trace::propagation::kB3CombinedHeader (C++ member), 62
opentelemetry::trace::propagation::kB3SampledHeader (C++ member), 62
opentelemetry::trace::propagation::kB3SpanIdHeader (C++ member), 62
opentelemetry::trace::propagation::kB3TraceIdHeader (C++ member), 63
opentelemetry::trace::propagation::kSpanIdHexSize (C++ member), 63
opentelemetry::trace::propagation::kTraceFlagsSize (C++ member), 63
opentelemetry::trace::propagation::kTraceIdHexSize (C++ member), 63
opentelemetry::trace::propagation::kTraceParentSize (C++ member), 64
opentelemetry::trace::propagation::kTraceStateSize (C++ member), 64
opentelemetry::trace::propagation::kVersionSize (C++ member), 64
opentelemetry::trace::propagation::TextMapPropagator (C++ class), 33
opentelemetry::trace::propagation::TextMapPropagator::Extract (C++ function), 33
opentelemetry::trace::propagation::TextMapPropagator::kTraceFlagsHexSize (C++ type), 33
opentelemetry::trace::propagation::TextMapPropagator::Inject (C++ function), 33
opentelemetry::trace::propagation::TextMapPropagator::kTraceIdHexSize (C++ type), 33
opentelemetry::trace::Provider (C++ class), 33
opentelemetry::trace::Provider::GetTracer (C++ function), 33
opentelemetry::trace::Provider::SetTracer (C++ function), 33
opentelemetry::trace::Scope (C++ class), 34
opentelemetry::trace::Scope::Scope (C++ function), 34
opentelemetry::trace::Span (C++ class), 34
opentelemetry::trace::Span::~Span (C++ function), 34
opentelemetry::trace::Span::AddEvent (C++ function), 34, 35
opentelemetry::trace::Span::End (C++ function), 35
opentelemetry::trace::Span::GetContext (C++ function), 35
opentelemetry::trace::Span::IsRecording (C++ function), 35
opentelemetry::trace::Span::operator= (C++ function), 34
opentelemetry::trace::Span::SetAttribute (C++ function), 34
opentelemetry::trace::Span::SetStatus (C++ function), 35
opentelemetry::trace::Span::Span (C++ function), 34
opentelemetry::trace::Span::UpdateName (C++ function), 35
opentelemetry::trace::SpanContext (C++ class), 35
opentelemetry::trace::SpanContext::GetInvalid (C++ function), 36
opentelemetry::trace::SpanContext::IsRemote (C++ function), 36
opentelemetry::trace::SpanContext::IsSampled (C++ function), 36
opentelemetry::trace::SpanContext::IsValid (C++ function), 35
opentelemetry::trace::SpanContext::operator= (C++ function), 36
opentelemetry::trace::SpanContext::operator== (C++ function), 36
opentelemetry::trace::SpanContext::span_id (C++ function), 35
opentelemetry::trace::SpanContext::SpanContext (C++ function), 35
opentelemetry::trace::SpanContext::trace_flags (C++ function), 35
opentelemetry::trace::SpanContext::trace_id (C++ function), 35
opentelemetry::trace::SpanContext::trace_state (C++ function), 36
opentelemetry::trace::SpanContextKeyValueIterable (C++ class), 36
opentelemetry::trace::SpanContextKeyValueIterable:: (C++ function), 36
opentelemetry::trace::SpanContextKeyValueIterable:: (C++ function), 36

opentelemetry::trace::SpanContextKeyValueIterable (C++ class), 36
opentelemetry::trace::SpanId (C++ class), 37
opentelemetry::trace::SpanId::CopyBytesTo (C++ function), 37
opentelemetry::trace::SpanId::Id (C++ function), 37
opentelemetry::trace::SpanId::IsValid (C++ function), 37
opentelemetry::trace::SpanId::kSize (C++ member), 37
opentelemetry::trace::SpanId::operator!= (C++ function), 37
opentelemetry::trace::SpanId::operator== (C++ function), 37
opentelemetry::trace::SpanId::SpanId (C++ function), 37
opentelemetry::trace::SpanId::ToLowerBase16 (C++ function), 37
opentelemetry::trace::SpanKind (C++ enum), 46
opentelemetry::trace::SpanKind::kClient (C++ enumerator), 46
opentelemetry::trace::SpanKind::kConsumer (C++ enumerator), 46
opentelemetry::trace::SpanKind::kInternal (C++ enumerator), 46
opentelemetry::trace::SpanKind::kProducer (C++ enumerator), 46
opentelemetry::trace::SpanKind::kServer (C++ enumerator), 46
opentelemetry::trace::StartSpanOptions (C++ struct), 15
opentelemetry::trace::StartSpanOptions::kind (C++ member), 15
opentelemetry::trace::StartSpanOptions::parent (C++ member), 15
opentelemetry::trace::StartSpanOptions::span_kind (C++ member), 15
opentelemetry::trace::StartSpanOptions::start_system_time (C++ member), 15
opentelemetry::trace::StatusCode (C++ enum), 46
opentelemetry::trace::StatusCode::kError (C++ enumerator), 46
opentelemetry::trace::StatusCode::kOk (C++ enumerator), 46
opentelemetry::trace::StatusCode::kUnset (C++ enumerator), 46
opentelemetry::trace::to_span_ptr (C++ function), 61
opentelemetry::trace::TraceFlags (C++ class), 37
opentelemetry::trace::TraceFlags::CopyBytesTo (C++ function), 37
opentelemetry::trace::TraceFlags::flags (C++ function), 37
opentelemetry::trace::TraceFlags::IsSampled (C++ function), 37
opentelemetry::trace::TraceFlags::kIsSampled (C++ member), 38
opentelemetry::trace::TraceFlags::operator!= (C++ function), 37
opentelemetry::trace::TraceFlags::operator== (C++ function), 37
opentelemetry::trace::TraceFlags::ToLowerBase16 (C++ function), 37
opentelemetry::trace::TraceFlags::TraceFlags (C++ function), 37
opentelemetry::trace::TraceId (C++ class), 38
opentelemetry::trace::TraceId::CopyBytesTo (C++ function), 38
opentelemetry::trace::TraceId::Id (C++ function), 38
opentelemetry::trace::TraceId::IsValid (C++ function), 38
opentelemetry::trace::TraceId::kSize (C++ member), 38
opentelemetry::trace::TraceId::operator!= (C++ function), 38
opentelemetry::trace::TraceId::operator== (C++ function), 38
opentelemetry::trace::TraceId::ToLowerBase16 (C++ function), 38
opentelemetry::trace::TraceId::TraceId (C++ function), 38
opentelemetry::trace::Tracer (C++ class), 39
opentelemetry::trace::Tracer::~Tracer (C++ function), 39
opentelemetry::trace::Tracer::Close (C++ function), 40
opentelemetry::trace::Tracer::CloseWithMicroseconds (C++ function), 40
opentelemetry::trace::Tracer::ForceFlush (C++ function), 40
opentelemetry::trace::Tracer::ForceFlushWithMicrose (C++ function), 40
opentelemetry::trace::Tracer::GetCurrentSpan (C++ function), 40
opentelemetry::trace::Tracer::StartSpan (C++ function), 39
opentelemetry::trace::Tracer::WithActiveSpan (C++ function), 40
opentelemetry::trace::TracerProvider (C++ class), 40

opentelemetry::trace::TracerProvider::~TracerProvider
(C++ function), 41
opentelemetry::trace::TracerProvider::GetTracer
(C++ function), 41
opentelemetry::trace::TraceState (C++
class), 41
opentelemetry::trace::TraceState::Delete
(C++ function), 41
opentelemetry::trace::TraceState::Empty
(C++ function), 41
opentelemetry::trace::TraceState::Entries
(C++ function), 41
opentelemetry::trace::TraceState::Entry
(C++ class), 42, 43
opentelemetry::trace::TraceState::Entry::Entry
(C++ function), 42, 43
opentelemetry::trace::TraceState::Entry::GetKey
(C++ function), 42, 43
opentelemetry::trace::TraceState::Entry::GetValue
(C++ function), 42, 43
opentelemetry::trace::TraceState::Entry::operator=
(C++ function), 42, 43
opentelemetry::trace::TraceState::Entry::SetValue
(C++ function), 42, 43
opentelemetry::trace::TraceState::FromHeader
(C++ function), 42
opentelemetry::trace::TraceState::Get
(C++ function), 41
opentelemetry::trace::TraceState::GetDefault
(C++ function), 42
opentelemetry::trace::TraceState::IsValidKey
(C++ function), 42
opentelemetry::trace::TraceState::IsValidValue
(C++ function), 42
opentelemetry::trace::TraceState::kKeyMaxSize
(C++ member), 42
opentelemetry::trace::TraceState::kKeyValueSeparator
(C++ member), 42
opentelemetry::trace::TraceState::kMaxKeyValuePairs
(C++ member), 42
opentelemetry::trace::TraceState::kMembersSeparator
(C++ member), 42
opentelemetry::trace::TraceState::kValueMaxSize
(C++ member), 42
opentelemetry::trace::TraceState::Set
(C++ function), 41
opentelemetry::trace::TraceState::ToHeader
(C++ function), 41

S

std::hash<OPENTELEMETRY_NAMESPACE::nostd::string_view>
(C++ struct), 15
std::hash<OPENTELEMETRY_NAMESPACE::nostd::string_view>::operator()
(C++ function), 15