

---

# OpenTelemetry C++

*Release 0.5.0*

OpenTelemetry authors

Apr 26, 2021



# OPENTELEMETRY C++ API

<b>1</b>	<b>Overview</b>	<b>1</b>
<b>2</b>	<b>Getting started</b>	<b>3</b>
<b>3</b>	<b>Reference documentation</b>	<b>5</b>
<b>4</b>	<b>Getting help</b>	<b>69</b>
	<b>Index</b>	<b>71</b>



## **OVERVIEW**

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

### **1.1 Library design**

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

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



## GETTING STARTED

### 2.1 Tracing

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

#### 2.1.1 Obtain a tracer

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

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

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

#### 2.1.2 Start a span

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

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

#### 2.1.3 Mark a span as active

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

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

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





## REFERENCE DOCUMENTATION

### 3.1 Class Hierarchy

### 3.2 File Hierarchy

### 3.3 Full API

#### 3.3.1 Namespaces

##### Namespace opentelemetry

##### Namespaces

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

##### Namespace opentelemetry::common

##### Classes

- *Struct KeyValueStringTokenizerOptions*
- *Class KeyValueIterable*
- *Class KeyValueProperties*
- *Class KeyValueProperties::Entry*
- *Class KeyValueStringTokenizer*
- *Class NullKeyValueIterable*
- *Class SteadyTimestamp*
- *Class StringUtil*
- *Class SystemTimestamp*

## Typedefs

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

## Namespace `opentelemetry::nostd`

### 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<<`*
- *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 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&)`*

## Typedefs

- *Typedef `opentelemetry::nostd::Traits`*

## Variables

- *Variable `opentelemetry::nostd::dynamic_extent`*

## Namespace `opentelemetry::nostd::detail`

### 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`

### Namespaces

- *Namespace `opentelemetry::trace::propagation`*

### Classes

- *Struct `EndSpanOptions`*
- *Struct `StartSpanOptions`*
- *Class `DefaultSpan`*
- *Class `DefaultTracer`*
- *Class `NoopSpan`*
- *Class `NoopTracer`*
- *Class `NoopTracerProvider`*

- *Class `NullSpanContext`*
- *Class `Provider`*
- *Class `Scope`*
- *Class `Span`*
- *Class `SpanContext`*
- *Class `SpanContextKeyValueIterable`*
- *Class `SpanId`*
- *Class `TraceFlags`*
- *Class `TraceId`*
- *Class `Tracer`*
- *Class `TracerProvider`*
- *Class `TraceState`*

## **Enums**

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

## **Functions**

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

## **Variables**

- *Variable `opentelemetry::trace::kSpanKey`*

## **Namespace `opentelemetry::trace::propagation`**

### **Namespaces**

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

## **Classes**

- *Class `B3Propagator`*
- *Class `B3PropagatorExtractor`*
- *Class `B3PropagatorMultiHeader`*
- *Class `CompositePropagator`*
- *Class `GlobalTextMapPropagator`*

- *Class `HttpTraceContext`*
- *Class `JaegerPropagator`*
- *Class `NoOpPropagator`*
- *Class `TextMapCarrier`*
- *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`

### 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

## Classes

- Template Struct `hash< OPENTELEMETRY_NAMESPACE::nstd::string_view >`

## 3.3.2 Classes and Structs

### Struct `KeyValueStringTokenizerOptions`

- Defined in file `include_opentelemetry_common_kv_properties.h`

#### Struct Documentation

**struct** `opentelemetry::common::KeyValueStringTokenizerOptions`

#### Public Members

```
char member_separator = ','  
char key_value_separator = '='  
bool ignore_empty_members = true
```

### Template Struct `is_specialized_span_convertible`

- Defined in file `include_opentelemetry_nstd_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

common::SteadyTimestamp end\_steady\_time

## Struct StartSpanOptions

- Defined in file\_include\_opentelemetry\_trace\_span.h

## Struct Documentation

**struct** opentelemetry::trace::StartSpanOptions

*StartSpanOptions* provides options to set properties of a *Span* at the time of its creation

### Public Members

common::SystemTimestamp start\_system\_time

common::SteadyTimestamp start\_steady\_time

SpanContext parent = SpanContext::GetInvalid()

SpanKind kind = SpanKind::kInternal

## Template Struct `hash< OPENTELEMETRY_NAMESPACE::nstd::string_view >`

- Defined in `file_include_opentelemetry_nstd_string_view.h`

### Struct Documentation

```
template<>
struct std::hash<OPENTELEMETRY_NAMESPACE::nstd::string_view>
```

#### Public Functions

```
inline std::size_t operator() (const OPENTELEMETRY_NAMESPACE::nstd::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 (nstd::function_ref<bool>) nstd::string_view, com-
                                mon::AttributeValue
    > callback const noexcept = 0 Iterate over key-value pairs
```

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

**Returns** true if every key-value pair was iterated over

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

**Returns** the number of key-value pairs

## Class KeyValueProperties

- Defined in file\_include\_opentelemetry\_common\_kv\_properties.h

## Nested Relationships

## Nested Types

- *Class KeyValueProperties::Entry*

## Class Documentation

```
class opentelemetry::common::KeyValueProperties
```

### Public Functions

```
inline KeyValueProperties (size_t size)
inline KeyValueProperties ()
template<class T, class = typename std::enable_if<detail::is_key_value_iterable<T>::value>::type>
inline KeyValueProperties (const T &keys_and_values)
inline void AddEntry (nostd::string_view key, nostd::string_view value)
inline bool GetAllEntries (nostd::function_ref<bool> nostd::string_view, nostd::string_view
    > callback) const
inline bool GetValue (nostd::string_view key, std::string &value) const
inline size_t Size () const noexcept
```

### Public Members

```
size_t num_entries_
size_t max_num_entries_
nostd::unique_ptr<Entry[]> entries_
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)
inline nostd::string_view GetKey () const
```

```
inline nostd::string_view GetValue () const
inline void SetValue (nostd::string_view value)
```

### Class KeyValueProperties::Entry

- Defined in file\_include\_opentelemetry\_common\_kv\_properties.h

### Nested Relationships

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

### Class Documentation

```
class opentelemetry::common::KeyValueProperties::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)
inline nostd::string_view GetKey () const
inline nostd::string_view GetValue () const
inline void SetValue (nostd::string_view value)
```

### Class KeyValueStringTokenizer

- Defined in file\_include\_opentelemetry\_common\_kv\_properties.h

### Class Documentation

```
class opentelemetry::common::KeyValueStringTokenizer
```

### Public Functions

```

inline KeyValueStringTokenizer (nostd::string_view str, const KeyValueStringTokenizerOptions &opts = KeyValueStringTokenizerOptions())
inline bool next (bool &valid_kv, nostd::string_view &key, nostd::string_view &value)
inline size_t NumTokens () const noexcept
inline void reset () noexcept

```

### Public Static Functions

```

static inline nostd::string_view GetDefaultKeyOrValue ()

```

### 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,      com-
    > const noexcept                               mon::AttributeValue
inline virtual size_t size () const noexcept

```

### Class SteadyTimestamp

- Defined in file\_include\_opentelemetry\_common\_timestamp.h

## Class Documentation

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

A timepoint relative to the monotonic clock epoch.

This is used for calculating the duration of an operation.

### Public Functions

**inline** `SteadyTimestamp()` **noexcept**

Initializes a monotonic timestamp pointing to the start of the epoch.

template<class **Rep**, class **Period**>

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

Initializes a monotonic timestamp from a duration.

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

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

Initializes a monotonic timestamp based on a point in time.

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

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

Returns a time point for the time stamp.

**Returns** A time point corresponding to the time stamp.

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

Returns the nanoseconds since the beginning of the epoch.

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

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

Compare two steady time stamps.

**Returns** true if the two time stamps are equal.

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

Compare two steady time stamps for inequality.

**Returns** true if the two time stamps are not equal.

## Class StringUtil

- Defined in file `_include_opentelemetry_common_string_util.h`

## Class Documentation

**class** opentelemetry::common::StringUtil

### Public Static Functions

**static inline** nostd::string\_view Trim (nostd::string\_view str, size\_t left, size\_t right)

**static inline** nostd::string\_view Trim (nostd::string\_view str)

## Class SystemTimestamp

- Defined in file\_include\_opentelemetry\_common\_timestamp.h

## Class Documentation

**class** opentelemetry::common::SystemTimestamp

A timepoint relative to the system clock epoch.

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

### Public Functions

**inline SystemTimestamp () noexcept**

Initializes a system timestamp pointing to the start of the epoch.

template<class Rep, class Period>

**inline explicit SystemTimestamp (const std::chrono::duration<Rep, Period> &time\_since\_epoch) noexcept**

Initializes a system timestamp from a duration.

**Parameters** time\_since\_epoch – Time elapsed since the beginning of the epoch.

**inline SystemTimestamp (const std::chrono::system\_clock::time\_point &time\_point) noexcept**

Initializes a system timestamp based on a point in time.

**Parameters** time\_point – A point in time.

**inline operator std::chrono::system\_clock::time\_point () const noexcept**

Returns a time point for the time stamp.

**Returns** A time point corresponding to the time stamp.

**inline std::chrono::nanoseconds time\_since\_epoch () const noexcept**

Returns the nanoseconds since the beginning of the epoch.

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

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

Compare two steady time stamps.

**Returns** true if the two time stamps are equal.

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

Compare two steady time stamps for inequality.

**Returns** true if the two time stamps are not equal.

## 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::nstd::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::nstd::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<typename C::value_type>::value>::type* = nullptr>
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<typename C::value_type>::value>::type* = nullptr>
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* = 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 = 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<typename C::value_type, T>::value>::type* = nullptr>
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<typename C::value_type, T>::value>::type* = nullptr>
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, const common::AttributeValue&) noexcept
inline void AddEvent (nostd::string_view) noexcept
inline void AddEvent (nostd::string_view, common::SystemTimestamp) noexcept
inline void AddEvent (nostd::string_view, common::SystemTimestamp, const common::KeyValueIterable&) noexcept
inline void AddEvent (nostd::string_view name, const common::KeyValueIterable &attributes) noexcept
inline void SetStatus (StatusCode, nostd::string_view) noexcept
inline void UpdateName (nostd::string_view) noexcept
inline void End (const EndSpanOptions& = {}) noexcept
inline nostd::string_view ToString ()
inline DefaultSpan (SpanContext span_context)
inline DefaultSpan (DefaultSpan &&spn)
inline DefaultSpan (const DefaultSpan &spn)
```

### Public Static Functions

```
static inline DefaultSpan GetInvalid ()
```

## Class DefaultTracer

- Defined in file\_include\_opentelemetry\_trace\_default\_tracer.h

## Inheritance Relationships

### Base Type

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

## Class Documentation

```
class opentelemetry::trace::DefaultTracer : public opentelemetry::trace::Tracer
```

### Public Functions

```
~DefaultTracer () = default
```

```
inline nostd::unique_ptr< Span > StartSpan (nostd::string_view name, const common::Key
```

Starts a span.

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

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

```
inline void ForceFlushWithMicroseconds (uint64_t timeout) override noexcept
```

```
inline void CloseWithMicroseconds (uint64_t timeout) override noexcept
```

## Class NoopSpan

- Defined in file\_include\_opentelemetry\_trace\_noop.h

## Inheritance Relationships

### Base Type

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

## Class Documentation

```
class opentelemetry::trace::NoopSpan : public opentelemetry::trace::Span
```

No-op implementation of *Span*. This class should not be used directly.

## Public Functions

```
inline explicit NoopSpan (const std::shared_ptr<Tracer> &tracer) noexcept
inline virtual void SetAttribute (nostd::string_view, const common::AttributeValue&)
                                noexcept override
inline virtual void AddEvent (nostd::string_view) noexcept override
inline virtual void AddEvent (nostd::string_view, common::SystemTimestamp) noexcept
                                override
inline virtual void AddEvent (nostd::string_view, common::SystemTimestamp, const com-
                                mon::KeyValueIterable&) noexcept override
inline virtual void SetStatus (StatusCode, nostd::string_view) noexcept override
inline virtual void UpdateName (nostd::string_view) noexcept override
inline virtual void End (const EndSpanOptions&) noexcept override
    Mark the end of the Span. Only the timing of the first End call for a given Span will be recorded, and
    implementations are free to ignore all further calls.

    Parameters options – can be used to manually define span properties like the end timestamp
inline virtual bool IsRecording () const noexcept override
inline virtual SpanContext GetContext () const noexcept override
```

## Class NoopTracer

- Defined in file\_include\_opentelemetry\_trace\_noop.h

## Inheritance Relationships

### Base Types

- public opentelemetry::trace::Tracer (*Class Tracer*)
- public std::enable\_shared\_from\_this< NoopTracer >

## Class Documentation

**class** opentelemetry::trace::NoopTracer : public opentelemetry::trace::Tracer, public std::enable\_shared\_from\_this<NoopTracer>  
No-op implementation of *Tracer*.

## Public Functions

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

Starts a span.

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

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



```
inline virtual void ForceFlushWithMicroseconds (uint64_t) noexcept override
inline virtual void CloseWithMicroseconds (uint64_t) noexcept override
```

### Class NoopTracerProvider

- Defined in file\_include\_opentelemetry\_trace\_noop.h

### Inheritance Relationships

#### Base Type

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

### Class Documentation

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

#### Public Functions

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

Gets or creates a named tracer instance.

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

### Class NullSpanContext

- Defined in file\_include\_opentelemetry\_trace\_span\_context\_kv\_iterable.h

### Inheritance Relationships

#### Base Type

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

## Class Documentation

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

### Public Functions

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

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

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

**inline virtual size\_t** `size` () **const noexcept override**

**Returns** the number of key-value pairs

## Class B3Propagator

- Defined in `file_include_opentelemetry_trace_propagation_b3_propagator.h`

## Inheritance Relationships

### Base Type

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

## Class Documentation

**class** `opentelemetry::trace::propagation::B3Propagator` : **public** `opentelemetry::trace::propagation::B3PropagatorExtractor`

### Public Functions

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

## Class B3PropagatorExtractor

- Defined in `file_include_opentelemetry_trace_propagation_b3_propagator.h`

## Inheritance Relationships

### Base Type

- `public opentelemetry::trace::propagation::TextMapPropagator`

### Derived Types

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

## Class Documentation

**class** `opentelemetry::trace::propagation::B3PropagatorExtractor` : **public** `opentelemetry::trace::propagation::TextMapPropagator`  
 Subclassed by `opentelemetry::trace::propagation::B3Propagator`, `opentelemetry::trace::propagation::B3PropagatorMultiHeader`

### Public Functions

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

### Public Static Functions

**static inline** `TraceId` **TraceIdFromHex** (`nostd::string_view` `trace_id`)  
**static inline** `SpanId` **SpanIdFromHex** (`nostd::string_view` `span_id`)  
**static inline** `TraceFlags` **TraceFlagsFromHex** (`nostd::string_view` `trace_flags`)

## Class B3PropagatorMultiHeader

- Defined in file `include/opentelemetry_trace_propagation_b3_propagator.h`

## Inheritance Relationships

### Base Type

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

## Class Documentation

**class** opentelemetry::trace::propagation::B3PropagatorMultiHeader : public opentelemetry::trace::propagation::TextMapPropagator

### Public Functions

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

## Class CompositePropagator

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

## Inheritance Relationships

### Base Type

- public opentelemetry::trace::propagation::TextMapPropagator

## Class Documentation

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

### Public Functions

**inline CompositePropagator** (std::vector<std::unique\_ptr<*TextMapPropagator*>> propagators)

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

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

#### Parameters

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

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

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

#### Parameters

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

## Class GlobalTextMapPropagator

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

## Class Documentation

```
class opentelemetry::trace::propagation::GlobalTextMapPropagator
```

### Public Static Functions

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

## Class HttpTraceContext

- Defined in file\_include\_opentelemetry\_trace\_propagation\_http\_trace\_context.h

## Inheritance Relationships

### Base Type

- public opentelemetry::trace::propagation::TextMapPropagator

## Class Documentation

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

### Public Functions

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

### Public Static Functions

```
static inline TraceId TraceIdFromHex (nostd::string_view trace_id)
static inline SpanId SpanIdFromHex (nostd::string_view span_id)
static inline TraceFlags TraceFlagsFromHex (nostd::string_view trace_flags)
```

## Class JaegerPropagator

- Defined in file\_include\_opentelemetry\_trace\_propagation\_jaeger.h

## Inheritance Relationships

### Base Type

- public opentelemetry::trace::propagation::TextMapPropagator

## Class Documentation

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

### Public Functions

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

## Class NoOpPropagator

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

## Inheritance Relationships

### Base Type

- public opentelemetry::trace::propagation::TextMapPropagator

## Class Documentation

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

### Public Functions

```
inline context::Context Extract (const TextMapCarrier&, context::Context &context)  
                noexcept override  
    Noop extract function does nothing and returns the input context  
inline void Inject (TextMapCarrier&, const context::Context &context) noexcept override  
    Noop inject function does nothing
```

## Class TextMapCarrier

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

## Class Documentation

```
class opentelemetry::trace::propagation::TextMapCarrier
```

### Public Functions

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

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

## Class TextMapPropagator

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

## Inheritance Relationships

## Derived Types

- public opentelemetry::trace::propagation::B3PropagatorExtractor (*Class B3PropagatorExtractor*)
- public opentelemetry::trace::propagation::CompositePropagator (*Class CompositePropagator*)
- public opentelemetry::trace::propagation::HttpTraceContext (*Class HttpTraceContext*)
- public opentelemetry::trace::propagation::JaegerPropagator (*Class JaegerPropagator*)
- public opentelemetry::trace::propagation::NoOpPropagator (*Class NoOpPropagator*)

## Class Documentation

```
class opentelemetry::trace::propagation::TextMapPropagator
```

```
Subclassed by opentelemetry::trace::propagation::B3PropagatorExtractor, opentelemetry::trace::propagation::CompositePropagator, opentelemetry::trace::propagation::HttpTraceContext, opentelemetry::trace::propagation::JaegerPropagator, opentelemetry::trace::propagation::NoOpPropagator
```

### Public Functions

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

### Class 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 nstd::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 (nstd::shared_ptr<TracerProvider> tp) noexcept  
    Changes the singleton TracerProvider.
```

### Class Scope

- Defined in file\_include\_opentelemetry\_trace\_scope.h

### Class Documentation

**class** `opentelemetry::trace::Scope`  
Controls how long a span is active.

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

### Public Functions

```
inline Scope (const nstd::shared_ptr<Span> &span) noexcept  
    Initialize a new scope.  
  
    Parameters span – the given span will be set as the currently active span.
```



## Class Span

- Defined in file\_include\_opentelemetry\_trace\_span.h

## Inheritance Relationships

## Derived Types

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

## Class Documentation

**class** opentelemetry::trace::Span

A *Span* represents a single operation within a Trace.

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

### Public Functions

**Span** () = default

**virtual ~Span** () = default

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

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

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

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

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

**virtual void AddEvent** (nostd::string\_view name) noexcept = 0

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

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

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

template<class T, nostd::enable\_if\_t<common::detail::is\_key\_value\_iterable<T>::value>\* = nullptr>

**inline void AddEvent** (nostd::string\_view name, common::SystemTimestamp timestamp, const T &attributes) noexcept

template<class T, nostd::enable\_if\_t<common::detail::is\_key\_value\_iterable<T>::value>\* = nullptr>

**inline void AddEvent** (nostd::string\_view name, const T &attributes) noexcept

**inline void AddEvent** (nostd::string\_view name, common::SystemTimestamp timestamp, std::initializer\_list<std::pair<nostd::string\_view, common::AttributeValue>> attributes) noexcept

**inline void AddEvent** (nostd::string\_view name, std::initializer\_list<std::pair<nostd::string\_view, common::AttributeValue>> attributes) noexcept

```
virtual void SetStatus (StatusCode code, nostd::string_view description = "") noexcept = 0
virtual void UpdateName (nostd::string_view name) noexcept = 0
virtual void End (const EndSpanOptions &options = {}) noexcept = 0
    Mark the end of the Span. Only the timing of the first End call for a given Span will be recorded, and
    implementations are free to ignore all further calls.

    Parameters options – can be used to manually define span properties like the end timestamp
virtual trace::SpanContext GetContext () const noexcept = 0
virtual bool IsRecording () const noexcept = 0
```

## Class SpanContext

- Defined in file `_include_opentelemetry_trace_span_context.h`

## Class Documentation

```
class opentelemetry::trace::SpanContext
```

### Public Functions

```
inline SpanContext (bool sampled_flag, bool is_remote)
inline SpanContext (TraceId trace_id, SpanId span_id, TraceFlags trace_flags, bool is_remote,
    nostd::shared_ptr<TraceState> trace_state = TraceState::GetDefault())
    noexcept
SpanContext (const SpanContext &ctx) = default
inline bool IsValid () const noexcept
inline const trace_api::TraceFlags &trace_flags () const noexcept
inline const trace_api::TraceId &trace_id () const noexcept
inline const trace_api::SpanId &span_id () const noexcept
inline const nostd::shared_ptr<trace_api::TraceState> trace_state () const noexcept
inline bool operator== (const SpanContext &that) const noexcept
SpanContext &operator= (const SpanContext &ctx) = default
inline bool IsRemote () const noexcept
inline bool IsSampled () const noexcept
```

## Public Static Functions

```
static inline SpanContext GetInvalid()
```

## Class *SpanContextKeyValueIterable*

- Defined in file\_include\_opentelemetry\_trace\_span\_context\_kv\_iterable.h

## Inheritance Relationships

### Derived Type

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

## Class Documentation

```
class opentelemetry::trace::SpanContextKeyValueIterable
```

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

Subclassed by *opentelemetry::trace::NullSpanContext*

### Public Functions

```
virtual ~SpanContextKeyValueIterable() = default
```

```
virtual bool ForEachKeyValue (nostd::function_ref<bool> SpanContext, const opentelemetry::common::KeyValueIterable&  
> callback) const noexcept = 0 Iterate over SpanContext/key-value pairs
```

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

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

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

**Returns** the number of key-value pairs

## Class *SpanId*

- Defined in file\_include\_opentelemetry\_trace\_span\_id.h

## Class Documentation

```
class opentelemetry::trace::SpanId
```

### Public Functions

```
inline SpanId() noexcept
inline explicit SpanId(nostd::span<const uint8_t, kSize> id) noexcept
inline void ToLowerBase16(nostd::span<char, 2 * kSize> buffer) const noexcept
inline nostd::span<const uint8_t, kSize> Id() const noexcept
inline bool operator==(const SpanId &that) const noexcept
inline bool operator!=(const SpanId &that) const noexcept
inline bool IsValid() const noexcept
inline void CopyBytesTo(nostd::span<uint8_t, kSize> dest) const noexcept
```

### Public Static Attributes

```
static constexpr int kSize = 8
```

### Class TraceFlags

- Defined in file\_include\_opentelemetry\_trace\_trace\_flags.h

### Class Documentation

```
class opentelemetry::trace::TraceFlags
```

### Public Functions

```
inline TraceFlags() noexcept
inline explicit TraceFlags(uint8_t flags) noexcept
inline bool IsSampled() const noexcept
inline void ToLowerBase16(nostd::span<char, 2> buffer) const noexcept
inline uint8_t flags() const noexcept
inline bool operator==(const TraceFlags &that) const noexcept
inline bool operator!=(const TraceFlags &that) const noexcept
inline void CopyBytesTo(nostd::span<uint8_t, 1> dest) const noexcept
```

### Public Static Attributes

```
static constexpr uint8_t kIsSampled = 1
```

### Class TraceId

- Defined in file\_include\_opentelemetry\_trace\_trace\_id.h

### Class Documentation

```
class opentelemetry::trace::TraceId
```

#### Public Functions

```
inline TraceId() noexcept
inline explicit TraceId(nostd::span<const uint8_t, kSize> id) noexcept
inline void ToLowerBase16(nostd::span<char, 2 * kSize> buffer) const noexcept
inline nostd::span<const uint8_t, kSize> Id() const noexcept
inline bool operator==(const TraceId &that) const noexcept
inline bool operator!=(const TraceId &that) const noexcept
inline bool IsValid() const noexcept
inline void CopyBytesTo(nostd::span<uint8_t, kSize> dest) const noexcept
```

#### Public Static Attributes

```
static constexpr int kSize = 16
```

### Class Tracer

- Defined in file\_include\_opentelemetry\_trace\_tracer.h

### Inheritance Relationships

#### Derived Types

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

## Class Documentation

### **class** `opentelemetry::trace::Tracer`

Handles span creation and in-process context propagation.

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

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

### Public Functions

**virtual** `~Tracer()` = default

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

Starts a span.

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

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

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

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

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

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

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

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

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

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

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

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

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

```
inline nostd::shared_ptr<Span> StartSpan (nostd::string_view name,
std::initializer_list<std::pair<nostd::string_view,
common::AttributeValue>> attributes,
std::initializer_list<std::pair<SpanContext,
std::initializer_list<std::pair<nostd::string_view, com-
mon::AttributeValue>>>> links, const StartSpanOp-
tions &options = { }) noexcept
```

```
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

## Class Documentation

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

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

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

## Public Functions

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

Creates a w3c tracestate header from *TraceState* object

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

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

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

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

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

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

**inline** nostd::*shared\_ptr*<TraceState> **Delete** (**const** nostd::*string\_view* &*key*)

Returns *shared\_ptr* to a new *TraceState* object after removing the attribute with given *key* ( if present )

**Returns** empty *TraceState* object if *key* is invalid

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

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

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



## Public Static Functions

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

**static inline** `nostd::shared_ptr<TraceState> FromHeader (nostd::string_view header)`

Returns `shared_ptr` to a newly created *TraceState* parsed from the header provided.

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

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

**static inline** `bool IsValidKey (nostd::string_view key)`

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

**static inline** `bool IsValidValue (nostd::string_view value)`

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

## Public Static Attributes

**static constexpr** `int kKeyMaxSize = 256`

**static constexpr** `int kValueMaxSize = 256`

**static constexpr** `int kMaxKeyValuePairs = 32`

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

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

## 3.3.3 Enums

### Enum CanonicalCode

- Defined in file `_include_opentelemetry_trace_canonical_code.h`

### Enum Documentation

**enum** `opentelemetry::trace::CanonicalCode`

*Values:*

**enumerator** `OK`

The operation completed successfully.

**enumerator** `CANCELLED`

The operation was cancelled (typically by the caller).

**enumerator** `UNKNOWN`

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

**enumerator INVALID\_ARGUMENT**

Client specified an invalid argument. Note that this differs from FAILED\_PRECONDITION. INVALID\_ARGUMENT indicates arguments that are problematic regardless of the state of the system (e.g., a malformed file name).

**enumerator DEADLINE\_EXCEEDED**

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

**enumerator NOT\_FOUND**

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

**enumerator ALREADY\_EXISTS**

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

**enumerator PERMISSION\_DENIED**

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

**enumerator RESOURCE\_EXHAUSTED**

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

**enumerator FAILED\_PRECONDITION**

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

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

**enumerator ABORTED**

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

See litmus test above for deciding between FAILED\_PRECONDITION, ABORTED, and UNAVAILABLE.

**enumerator OUT\_OF\_RANGE**

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

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

There is a fair bit of overlap between FAILED\_PRECONDITION and OUT\_OF\_RANGE. We recommend using OUT\_OF\_RANGE (the more specific error) when it applies so that callers who are iterating through a space can easily look for an OUT\_OF\_RANGE error to detect when they are done.

**enumerator UNIMPLEMENTED**

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

**enumerator INTERNAL**

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

**enumerator UNAVAILABLE**

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

See litmus test above for deciding between FAILED\_PRECONDITION, ABORTED, and UNAVAILABLE.

**enumerator DATA\_LOSS**

Unrecoverable data loss or corruption.

**enumerator UNAUTHENTICATED**

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

## Enum SpanKind

- Defined in file\_include\_opentelemetry\_trace\_span.h

## Enum Documentation

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

*Values:*

**enumerator** kInternal

**enumerator** kServer

**enumerator** kClient

**enumerator** kProducer

**enumerator** kConsumer

## Enum StatusCode

- Defined in file\_include\_opentelemetry\_trace\_span.h

## Enum Documentation

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

*Values:*

**enumerator** kUnset

**enumerator** kOk

**enumerator** kError

### 3.3.4 Functions

**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

```

inline trace::SpanContext opentelemetry::trace::propagation::detail::GetCurrentSpan(const
con-
text::Context
&con-
text)

```

### Function `opentelemetry::trace::propagation::detail::HexToBinary`

- Defined in `file_include_opentelemetry_trace_propagation_detail_hex.h`

#### Function Documentation

```

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

```

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

### Function `opentelemetry::trace::propagation::detail::HexToInt`

- Defined in `file_include_opentelemetry_trace_propagation_detail_hex.h`

## Function Documentation

**inline** int8\_t opentelemetry::trace::propagation::detail::HexToInt (char c)

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

- Defined in file\_include\_opentelemetry\_trace\_propagation\_detail\_hex.h

## Function Documentation

**inline** bool opentelemetry::trace::propagation::detail::IsValidHex (nostd::string\_view s)

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

- Defined in file\_include\_opentelemetry\_trace\_propagation\_detail\_string.h

## Function Documentation

**inline** size\_t opentelemetry::trace::propagation::detail::SplitString (nostd::string\_view s, char separator, nostd::string\_view \*results, size\_t count)

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

### 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::nstd::dynamic_extent`

- Defined in `file_include_opentelemetry_nstd_span.h`

#### Variable Documentation

**constexpr** `size_t opentelemetry::nstd::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** `nstd::string_view opentelemetry::trace::propagation::kB3CombinedHeader` = `"b3"`

#### Variable `opentelemetry::trace::propagation::kB3SampledHeader`

- Defined in `file_include_opentelemetry_trace_propagation_b3_propagator.h`

## Variable Documentation

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

## Variable `opentelemetry::trace::propagation::kB3SpanIdHeader`

- Defined in file `include_opentelemetry_trace_propagation_b3_propagator.h`

## Variable Documentation

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

## Variable `opentelemetry::trace::propagation::kB3TraceIdHeader`

- Defined in file `include_opentelemetry_trace_propagation_b3_propagator.h`

## Variable Documentation

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

## Variable `opentelemetry::trace::propagation::kSpanIdHexStrLength`

- Defined in file `include_opentelemetry_trace_propagation_b3_propagator.h`

## Variable Documentation

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

## Variable `opentelemetry::trace::propagation::kSpanIdSize`

- Defined in file `include_opentelemetry_trace_propagation_http_trace_context.h`

## Variable Documentation

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

## Variable `opentelemetry::trace::propagation::kTraceFlagsSize`

- Defined in file `include_opentelemetry_trace_propagation_http_trace_context.h`

### Variable Documentation

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

### Variable `opentelemetry::trace::propagation::kTraceHeader`

- Defined in `file_include_opentelemetry_trace_propagation_jaeger.h`

### Variable Documentation

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

### Variable `opentelemetry::trace::propagation::kTraceIdHexStrLength`

- Defined in `file_include_opentelemetry_trace_propagation_b3_propagator.h`

### Variable Documentation

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

### Variable `opentelemetry::trace::propagation::kTraceIdSize`

- Defined in `file_include_opentelemetry_trace_propagation_http_trace_context.h`

### Variable Documentation

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

### Variable `opentelemetry::trace::propagation::kTraceParent`

- Defined in `file_include_opentelemetry_trace_propagation_http_trace_context.h`

### Variable Documentation

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

### Variable `opentelemetry::trace::propagation::kTraceParentSize`

- Defined in `file_include_opentelemetry_trace_propagation_http_trace_context.h`

### Variable Documentation

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

### Variable `opentelemetry::trace::propagation::kTraceState`

- Defined in `file_include_opentelemetry_trace_propagation_http_trace_context.h`

### Variable Documentation

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

### Variable `opentelemetry::trace::propagation::kVersionSize`

- Defined in `file_include_opentelemetry_trace_propagation_http_trace_context.h`

### Variable Documentation

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

## 3.3.6 Defines

### Define `HAVE_WORKING_REGEX`

- Defined in `file_include_opentelemetry_trace_trace_state.h`

### Define Documentation

`HAVE_WORKING_REGEX`

## 3.3.7 Typedefs

### Typedef `opentelemetry::common::AttributeValue`

- Defined in `file_include_opentelemetry_common_attribute_value.h`

### Typedef Documentation

**using** `opentelemetry::common::AttributeValue = nostd::variant<bool, int32_t, int64_t, uint32_t, double, nostd::string>`  
OpenTelemetry signals can be enriched by adding attributes. The `AttributeValue` type is defined as a variant of all attribute value types the OpenTelemetry C++ API supports.

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

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

**Warning:**

The OpenTelemetry C++ API currently supports several attribute value types that are not covered by the OpenTelemetry specification:

- `uint64_t`
- `nostd::span<const uint64_t>`
- `nostd::span<uint8_t>`

Those types are reserved for future use and currently should not be used. There are no guarantees around how those values are handled by exporters.

**Typedef `opentelemetry::nostd::Traits`**

- Defined in `file_include_opentelemetry_nostd_string_view.h`

**Typedef Documentation**

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





## GETTING HELP

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



## INDEX

### H

HAVE\_WORKING\_REGEX (*C macro*), 66

### O

opentelemetry::common::AttributeValue  
(*C++ type*), 66

opentelemetry::common::KeyValueIterable  
(*C++ class*), 14

opentelemetry::common::KeyValueIterable::ForEachKeyValue  
(*C++ function*), 14

opentelemetry::common::KeyValueIterable::ForEachKeyValue  
(*C++ function*), 14

opentelemetry::common::KeyValueIterable::Size  
(*C++ function*), 14

opentelemetry::common::KeyValueProperties  
(*C++ class*), 15

opentelemetry::common::KeyValueProperties::AddEntry  
(*C++ function*), 15

opentelemetry::common::KeyValueProperties::Entries  
(*C++ member*), 15

opentelemetry::common::KeyValueProperties::Entry  
(*C++ class*), 15, 16

opentelemetry::common::KeyValueProperties::Entry::Entry  
(*C++ function*), 15, 16

opentelemetry::common::KeyValueProperties::Entry::GetKey  
(*C++ function*), 15, 16

opentelemetry::common::KeyValueProperties::Entry::GetValue  
(*C++ function*), 15, 16

opentelemetry::common::KeyValueProperties::Entry::operator=  
(*C++ function*), 15, 16

opentelemetry::common::KeyValueProperties::Entry::SetValue  
(*C++ function*), 16

opentelemetry::common::KeyValueProperties::GetAllEntries  
(*C++ function*), 15

opentelemetry::common::KeyValueProperties::GetValue  
(*C++ function*), 15

opentelemetry::common::KeyValueProperties::KeyValueProperties  
(*C++ function*), 15

opentelemetry::common::KeyValueProperties::max\_num\_entries  
(*C++ member*), 15

opentelemetry::common::KeyValueProperties::num\_entries  
(*C++ member*), 15

opentelemetry::common::KeyValueProperties::Size  
(*C++ function*), 15

opentelemetry::common::KeyValueStringTokenizer  
(*C++ class*), 16

opentelemetry::common::KeyValueStringTokenizer::Get  
(*C++ function*), 17

opentelemetry::common::KeyValueStringTokenizer::Key  
(*C++ function*), 17

opentelemetry::common::KeyValueStringTokenizer::next  
(*C++ function*), 17

opentelemetry::common::KeyValueStringTokenizer::Num  
(*C++ function*), 17

opentelemetry::common::KeyValueStringTokenizer::res  
(*C++ function*), 17

opentelemetry::common::KeyValueStringTokenizerOpti  
(*C++ struct*), 10

opentelemetry::common::KeyValueStringTokenizerOpti  
(*C++ member*), 10

opentelemetry::common::KeyValueStringTokenizerOpti  
(*C++ member*), 10

opentelemetry::common::KeyValueStringTokenizerOpti  
(*C++ member*), 10

opentelemetry::common::NullKeyValueIterable  
(*C++ class*), 17

opentelemetry::common::NullKeyValueIterable::ForEa  
(*C++ function*), 17

opentelemetry::common::NullKeyValueIterable::NullKe  
(*C++ function*), 17

opentelemetry::common::NullKeyValueIterable::size  
(*C++ function*), 17

opentelemetry::common::SteadyTimestamp  
(*C++ class*), 18

opentelemetry::common::SteadyTimestamp::operator  
std::chrono::steady\_clock::time\_point  
(*C++ function*), 18

opentelemetry::common::SteadyTimestamp::operator!=  
(*C++ function*), 18

opentelemetry::common::SteadyTimestamp::operator==  
(*C++ function*), 18

opentelemetry::common::SteadyTimestamp::SteadyTime  
(*C++ function*), 18

opentelemetry::common::SteadyTimestamp::time\_since

[illegible]

tion), 23  
 opentelemetry::nstd::span<T, dynamic\_extent>::extent (C++ member), 23  
 opentelemetry::nstd::span<T, dynamic\_extent>::operator[] (C++ function), 23  
 opentelemetry::nstd::span<T, dynamic\_extent>::size (C++ function), 23  
 opentelemetry::nstd::span<T, dynamic\_extent>::span (C++ function), 23  
 opentelemetry::nstd::string\_view (C++ class), 23  
 opentelemetry::nstd::string\_view::begin (C++ function), 24  
 opentelemetry::nstd::string\_view::compare (C++ function), 24  
 opentelemetry::nstd::string\_view::data (C++ function), 24  
 opentelemetry::nstd::string\_view::empty (C++ function), 24  
 opentelemetry::nstd::string\_view::end (C++ function), 24  
 opentelemetry::nstd::string\_view::find (C++ function), 24  
 opentelemetry::nstd::string\_view::length (C++ function), 24  
 opentelemetry::nstd::string\_view::npos (C++ member), 24  
 opentelemetry::nstd::string\_view::operator std::string (C++ function), 24  
 opentelemetry::nstd::string\_view::operator> (C++ function), 24  
 opentelemetry::nstd::string\_view::operator< (C++ function), 24  
 opentelemetry::nstd::string\_view::operator[] (C++ function), 24  
 opentelemetry::nstd::string\_view::size (C++ function), 24  
 opentelemetry::nstd::string\_view::size\_type (C++ type), 24  
 opentelemetry::nstd::string\_view::string\_view (C++ function), 24  
 opentelemetry::nstd::string\_view::substr (C++ function), 24  
 opentelemetry::nstd::Traits (C++ type), 67  
 opentelemetry::nstd::unique\_ptr (C++ class), 25  
 opentelemetry::nstd::unique\_ptr::~~unique\_ptr (C++ function), 25  
 opentelemetry::nstd::unique\_ptr::element\_type (C++ type), 25  
 opentelemetry::nstd::unique\_ptr::get (C++ function), 25  
 opentelemetry::nstd::unique\_ptr::operator bool (C++ function), 25  
 opentelemetry::nstd::unique\_ptr::operator std::unique\_ptr<T> (C++ function), 25  
 opentelemetry::nstd::unique\_ptr::operator\* (C++ function), 25  
 opentelemetry::nstd::unique\_ptr::operator= (C++ function), 25  
 opentelemetry::nstd::unique\_ptr::operator-> (C++ function), 25  
 opentelemetry::nstd::unique\_ptr::pointer (C++ type), 25  
 opentelemetry::nstd::unique\_ptr::release (C++ function), 25  
 opentelemetry::nstd::unique\_ptr::reset (C++ function), 25  
 opentelemetry::nstd::unique\_ptr::swap (C++ function), 25  
 opentelemetry::nstd::unique\_ptr::unique\_ptr (C++ function), 25  
 opentelemetry::trace::CanonicalCode (C++ enum), 45  
 opentelemetry::trace::CanonicalCode::ABORTED (C++ enumerator), 46  
 opentelemetry::trace::CanonicalCode::ALREADY\_EXISTS (C++ enumerator), 46  
 opentelemetry::trace::CanonicalCode::CANCELLED (C++ enumerator), 45  
 opentelemetry::trace::CanonicalCode::DATA\_LOSS (C++ enumerator), 47  
 opentelemetry::trace::CanonicalCode::DEADLINE\_EXCEEDED (C++ enumerator), 46  
 opentelemetry::trace::CanonicalCode::FAILED\_PRECONDITION (C++ enumerator), 46  
 opentelemetry::trace::CanonicalCode::INTERNAL (C++ enumerator), 46  
 opentelemetry::trace::CanonicalCode::INVALID\_ARGUMENT (C++ enumerator), 45  
 opentelemetry::trace::CanonicalCode::NOT\_FOUND (C++ enumerator), 46  
 opentelemetry::trace::CanonicalCode::OK (C++ enumerator), 46  
 opentelemetry::trace::CanonicalCode::OUT\_OF\_RANGE (C++ enumerator), 46  
 opentelemetry::trace::CanonicalCode::PERMISSION\_DENIED (C++ enumerator), 46  
 opentelemetry::trace::CanonicalCode::RESOURCE\_EXHAUSTED (C++ enumerator), 46  
 opentelemetry::trace::CanonicalCode::UNAUTHENTICATED (C++ enumerator), 47  
 opentelemetry::trace::CanonicalCode::UNAVAILABLE (C++ enumerator), 47

```

opentelemetry::trace::CanonicalCode::UNIMPLEMENTED opentelemetry::trace::NoopTracer (C++
(C++ enumerator), 46 class), 28
opentelemetry::trace::CanonicalCode::UNKNOWN opentelemetry::trace::NoopTracer::CloseWithMicrose
(C++ enumerator), 45 (C++ function), 29
opentelemetry::trace::DefaultSpan (C++ opentelemetry::trace::NoopTracer::ForceFlushWithMi
class), 26 (C++ function), 28
opentelemetry::trace::DefaultSpan::AddEvent opentelemetry::trace::NoopTracer::StartSpan
(C++ function), 26 (C++ function), 28
opentelemetry::trace::DefaultSpan::DefaultSpan opentelemetry::trace::NoopTracerProvider
(C++ function), 26 (C++ class), 29
opentelemetry::trace::DefaultSpan::End opentelemetry::trace::NoopTracerProvider::GetTracer
(C++ function), 26 (C++ function), 29
opentelemetry::trace::DefaultSpan::GetContext opentelemetry::trace::NoopTracerProvider::NoopTrace
(C++ function), 26 (C++ function), 29
opentelemetry::trace::DefaultSpan::GetInvalid opentelemetry::trace::NullSpanContext
(C++ function), 26 (C++ class), 30
opentelemetry::trace::DefaultSpan::IsRecording opentelemetry::trace::NullSpanContext::ForEachKeyVa
(C++ function), 26 (C++ function), 30
opentelemetry::trace::DefaultSpan::SetAttribute opentelemetry::trace::NullSpanContext::size
(C++ function), 26 (C++ function), 30
opentelemetry::trace::DefaultSpan::SetStatus opentelemetry::trace::propagation::B3Propagator
(C++ function), 26 (C++ class), 30
opentelemetry::trace::DefaultSpan::ToString opentelemetry::trace::propagation::B3Propagator::In
(C++ function), 26 (C++ function), 30
opentelemetry::trace::DefaultSpan::UpdateName opentelemetry::trace::propagation::B3PropagatorExt
(C++ function), 26 (C++ class), 31
opentelemetry::trace::DefaultTracer opentelemetry::trace::propagation::B3PropagatorExt
(C++ class), 27 (C++ function), 31
opentelemetry::trace::DefaultTracer::~DefaultTracer opentelemetry::trace::propagation::B3PropagatorExt
(C++ function), 27 (C++ function), 31
opentelemetry::trace::EndSpanOptions opentelemetry::trace::propagation::B3PropagatorExt
(C++ struct), 13 (C++ function), 31
opentelemetry::trace::EndSpanOptions::end opentelemetry::trace::propagation::B3PropagatorExt
(C++ member), 13 (C++ function), 31
opentelemetry::trace::kSpanKey (C++ mem- opentelemetry::trace::propagation::B3PropagatorMult
ber), 63 (C++ class), 32
opentelemetry::trace::NoopSpan (C++ opentelemetry::trace::propagation::B3PropagatorMult
class), 27 (C++ function), 32
opentelemetry::trace::NoopSpan::AddEvent opentelemetry::trace::propagation::CompositePropag
(C++ function), 28 (C++ class), 32
opentelemetry::trace::NoopSpan::End opentelemetry::trace::propagation::CompositePropag
(C++ function), 28 (C++ function), 32
opentelemetry::trace::NoopSpan::GetContext opentelemetry::trace::propagation::CompositePropag
(C++ function), 28 (C++ function), 32
opentelemetry::trace::NoopSpan::IsRecording opentelemetry::trace::propagation::CompositePropag
(C++ function), 28 (C++ function), 32
opentelemetry::trace::NoopSpan::NoopSpan opentelemetry::trace::propagation::detail::GetCurre
(C++ function), 28 (C++ function), 61
opentelemetry::trace::NoopSpan::SetAttribute opentelemetry::trace::propagation::detail::HexToBin
(C++ function), 28 (C++ function), 61
opentelemetry::trace::NoopSpan::SetStatus opentelemetry::trace::propagation::detail::HexToInt
(C++ function), 28 (C++ function), 62
opentelemetry::trace::NoopSpan::UpdateName opentelemetry::trace::propagation::detail::IsValid
(C++ function), 28 (C++ function), 62

```





opentelemetry::trace::SpanContext::IsRemote	opentelemetry::trace::SpanKind::kProducer
(C++ function), 38	(C++ enumerator), 47
opentelemetry::trace::SpanContext::IsSampled	opentelemetry::trace::SpanKind::kServer
(C++ function), 38	(C++ enumerator), 47
opentelemetry::trace::SpanContext::IsValid	opentelemetry::trace::StartSpanOptions
(C++ function), 38	(C++ struct), 13
opentelemetry::trace::SpanContext::operation_name	opentelemetry::trace::StartSpanOptions::kind
(C++ function), 38	(C++ member), 13
opentelemetry::trace::SpanContext::operation_name	opentelemetry::trace::StartSpanOptions::parent
(C++ function), 38	(C++ member), 13
opentelemetry::trace::SpanContext::span_id	opentelemetry::trace::StartSpanOptions::start_timestamp
(C++ function), 38	(C++ member), 13
opentelemetry::trace::SpanContext::SpanContext	opentelemetry::trace::StartSpanOptions::start_timestamp
(C++ function), 38	(C++ member), 13
opentelemetry::trace::SpanContext::trace_options	opentelemetry::trace::StatusCode
(C++ function), 38	(C++ enum), 47
opentelemetry::trace::SpanContext::trace_options	opentelemetry::trace::StatusCode::kError
(C++ function), 38	(C++ enumerator), 47
opentelemetry::trace::SpanContext::trace_options	opentelemetry::trace::StatusCode::kOk
(C++ function), 38	(C++ enumerator), 47
opentelemetry::trace::SpanContextKeyValuePairs	opentelemetry::trace::StatusCode::kUnset
(C++ class), 39	(C++ enumerator), 47
opentelemetry::trace::SpanContextKeyValuePairs	opentelemetry::trace::SpanContextKeyValuePairs::operator[]
(C++ function), 39	(C++ function), 62
opentelemetry::trace::SpanContextKeyValuePairs	opentelemetry::trace::TraceFlags
(C++ function), 39	(C++ class), 40
opentelemetry::trace::SpanContextKeyValuePairs::operator[]	opentelemetry::trace::TraceFlags::CopyBytesTo
(C++ function), 39	(C++ function), 40
opentelemetry::trace::SpanId (C++ class),	opentelemetry::trace::TraceFlags::flags
39	(C++ function), 40
opentelemetry::trace::SpanId::CopyBytesTo	opentelemetry::trace::TraceFlags::IsSampled
(C++ function), 40	(C++ function), 40
opentelemetry::trace::SpanId::Id (C++	opentelemetry::trace::TraceFlags::kIsSampled
function), 40	(C++ member), 41
opentelemetry::trace::SpanId::IsValid	opentelemetry::trace::TraceFlags::operator!=
(C++ function), 40	(C++ function), 40
opentelemetry::trace::SpanId::kSize	opentelemetry::trace::TraceFlags::operator==
(C++ member), 40	(C++ function), 40
opentelemetry::trace::SpanId::operator!=	opentelemetry::trace::TraceFlags::ToLowerBase16
(C++ function), 40	(C++ function), 40
opentelemetry::trace::SpanId::operator==	opentelemetry::trace::TraceFlags::TraceFlags
(C++ function), 40	(C++ function), 40
opentelemetry::trace::SpanId::SpanId	opentelemetry::trace::TraceId (C++ class),
(C++ function), 40	41
opentelemetry::trace::SpanId::ToLowerBase16	opentelemetry::trace::TraceId::CopyBytesTo
(C++ function), 40	(C++ function), 41
opentelemetry::trace::SpanKind	opentelemetry::trace::TraceId::Id (C++
(C++ enum), 47	function), 41
opentelemetry::trace::SpanKind::kClient	opentelemetry::trace::TraceId::IsValid
(C++ enumerator), 47	(C++ function), 41
opentelemetry::trace::SpanKind::kConsumer	opentelemetry::trace::TraceId::kSize
(C++ enumerator), 47	(C++ member), 41
opentelemetry::trace::SpanKind::kInternal	opentelemetry::trace::TraceId::operator!=
(C++ enumerator), 47	(C++ function), 41



opentelemetry::trace::TraceId::operator= (C++ function), 41  
 opentelemetry::trace::TraceId::ToLowerBase16 (C++ function), 41  
 opentelemetry::trace::TraceId::TraceId (C++ function), 41  
 opentelemetry::trace::Tracer (C++ class), 42  
 opentelemetry::trace::Tracer::~Tracer (C++ function), 42  
 opentelemetry::trace::Tracer::Close (C++ function), 43  
 opentelemetry::trace::Tracer::CloseWithMicroseconds (C++ function), 43  
 opentelemetry::trace::Tracer::ForceFlush (C++ function), 43  
 opentelemetry::trace::Tracer::ForceFlushWithMicroseconds (C++ function), 43  
 opentelemetry::trace::Tracer::GetCurrentSpan (C++ function), 43  
 opentelemetry::trace::Tracer::StartSpan (C++ function), 42  
 opentelemetry::trace::Tracer::WithActiveSpan (C++ function), 43  
 opentelemetry::trace::TracerProvider (C++ class), 43  
 opentelemetry::trace::TracerProvider::~TracerProvider (C++ function), 44  
 opentelemetry::trace::TracerProvider::GetTracer (C++ function), 44  
 opentelemetry::trace::TraceState (C++ class), 44  
 opentelemetry::trace::TraceState::Delete (C++ function), 44  
 opentelemetry::trace::TraceState::Empty (C++ function), 44  
 opentelemetry::trace::TraceState::FromHeader (C++ function), 45  
 opentelemetry::trace::TraceState::Get (C++ function), 44  
 opentelemetry::trace::TraceState::GetAllEntries (C++ function), 44  
 opentelemetry::trace::TraceState::GetDefault (C++ function), 45  
 opentelemetry::trace::TraceState::IsValidKey (C++ function), 45  
 opentelemetry::trace::TraceState::IsValidValue (C++ function), 45  
 opentelemetry::trace::TraceState::kKeyMaxSize (C++ member), 45  
 opentelemetry::trace::TraceState::kKeyValueSeparator (C++ member), 45  
 opentelemetry::trace::TraceState::kMaxKeyValuePairs (C++ member), 45  
 opentelemetry::trace::TraceState::kMembersSeparator (C++ member), 45  
 opentelemetry::trace::TraceState::kValueMaxSize (C++ member), 45  
 opentelemetry::trace::TraceState::Set (C++ function), 44  
 opentelemetry::trace::TraceState::ToHeader (C++ function), 44  
 S  
 std::hash<OPENTELEMETRY\_NAMESPACE::nostd::string\_view> (C++ struct), 14  
 std::hash<OPENTELEMETRY\_NAMESPACE::nostd::string\_view> (C++ function), 14