

---

# OpenTelemetry C++

*Release 1.1.1*

OpenTelemetry authors

Dec 21, 2021



# OPENTELEMETRY C++ API

<b>1</b>	<b>OpenTelemetry C++ API</b>	<b>1</b>
<b>2</b>	<b>OpenTelemetry C++ SDK</b>	<b>5</b>
<b>3</b>	<b>Reference documentation</b>	<b>11</b>
<b>4</b>	<b>Getting help</b>	<b>95</b>
	<b>Index</b>	<b>97</b>



## **OPENTELEMETRY C++ API**

### **1.1 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.

#### **1.1.1 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.

### **1.2 Getting started**

#### **1.2.1 Tracing**

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

##### **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`.

### 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.

### Mark a span as active

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

This marks a span as active and returns a Scope object. The scope object controls how long a span is active. The span remains active for the lifetime of the scope object.

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

### Create nested Spans

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

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

### Context Propagation

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

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

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

//Extract headers to context
```

(continues on next page)

(continued from previous page)

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

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





## OPENTELEMETRY C++ SDK

### 2.1 Getting started

OpenTelemetry C++ SDK provides the reference implementation of OpenTelemetry C++ API, and also provides implementation for Processor, Sampler, and core Exporters as per the specification.

### 2.2 Exporter

An exporter is responsible for sending the telemetry data to a particular backend. OpenTelemetry offers six tracing exporters out of the box:

- In-Memory Exporter: keeps the data in memory, useful for debugging.
- Jaeger Exporter: prepares and sends the collected telemetry data to a Jaeger backend via UDP and HTTP.
- Zipkin Exporter: prepares and sends the collected telemetry data to a Zipkin backend via the Zipkin APIs.
- Logging Exporter: saves the telemetry data into log streams.
- OpenTelemetry(otlp) Exporter: sends the data to the OpenTelemetry Collector using protobuf/gRPC or protobuf/HTTP.
- ETW Exporter: sends the telemetry data to Event Tracing for Windows (ETW).

```
//namespace alias used in sample code here.
namespace sdktrace = opentelemetry::sdk::trace;

// logging exporter
auto ostream_exporter =
    std::unique_ptr<sdktrace::SpanExporter>(new
    ↪opentelemetry::exporter::trace::OStreamSpanExporter);

// memory exporter
auto memory_exporter =
    std::unique_ptr<sdktrace::SpanExporter>(new
    ↪opentelemetry::exporter::memory::InMemorySpanExporter);

// zipkin exporter
opentelemetry::exporter::zipkin::ZipkinExporterOptions opts;
opts.endpoint = "http://localhost:9411/api/v2/spans" ; // or export OTEL_EXPORTER_ZIPKIN_
    ↪ENDPOINT="..."
opts.service_name = "default_service" ;
```

(continues on next page)

(continued from previous page)

```

auto zipkin_exporter =
    std::unique_ptr<sdktrace::SpanExporter>(new
↳ opentelemetry::exporter::zipkin::ZipkinExporter(opts));

// Jaeger UDP exporter
opentelemetry::exporter::jaeger::JaegerExporterOptions opts;
opts.endpoint = "localhost";
opts.server_port = 6831;
auto jaeger_udp_exporter =
    std::unique_ptr<sdktrace::SpanExporter>(new
↳ opentelemetry::exporter::jaeger::JaegerExporter(opts));

// Jaeger HTTP exporter
opentelemetry::exporter::jaeger::JaegerExporterOptions opts;
opts.transport_format = opentelemetry::exporter::jaeger::TransportFormat::kThriftHttp;
opts.endpoint = "localhost";
opts.server_port = 6831;
opts.headers = {}; // optional headers
auto jaeger_udp_exporter =
    std::unique_ptr<sdktrace::SpanExporter>(new
↳ opentelemetry::exporter::jaeger::JaegerExporter(opts));

// otel grpc exporter
opentelemetry::exporter::otlp::OtlpGrpcExporterOptions opts;
opts.endpoint = "localhost:4317";
opts.use_ssl_credentials = true;
opts.ssl_credentials_cacert_as_string = "ssl-certificate";
auto otlp_grpc_exporter =
    std::unique_ptr<sdktrace::SpanExporter>(new
↳ opentelemetry::exporter::otlp::OtlpGrpcExporter(opts));

// otel http exporter
opentelemetry::exporter::otlp::OtlpHttpExporterOptions opts;
opts.url = "http://localhost:4318/v1/traces";
auto otlp_http_exporter =
    std::unique_ptr<sdktrace::SpanExporter>(new
↳ opentelemetry::exporter::otlp::OtlpHttpExporter(opts));

```

## 2.3 Span Processor

Span Processor is initialised with an Exporter. Different Span Processors are offered by OpenTelemetry C++ SDK:

- SimpleSpanProcessor: immediately forwards ended spans to the exporter.
- BatchSpanProcessor: batches the ended spans and send them to exporter in bulk.
- MultiSpanProcessor: Allows multiple span processors to be active and configured at the same time.

```

// simple processor
auto simple_processor = std::unique_ptr<sdktrace::SpanProcessor>(

```

(continues on next page)

(continued from previous page)

```

    new sdktrace::SimpleSpanProcessor(std::move(ostream_exporter)));

// batch processor
sdktrace::BatchSpanProcessorOptions options{};
auto batch_processor = std::unique_ptr<sdktrace::SpanProcessor>(
    new sdktrace::BatchSpanProcessor(std::move(memory_exporter), options));

// multi-processor
std::vector<std::unique_ptr<SpanProcessor>>
    processors{std::move(simple_processor), std::move(batch_processor)};
auto multi_processor = std::unique_ptr<sdktrace::SpanProcessor>(
    new sdktrace::MultiSpanProcessor(std::move(processors)));

```

## 2.4 Resource

A Resource is an immutable representation of the entity producing telemetry as key-value pair. The OpenTelemetry C++ SDK allow for creation of Resources and for associating them with telemetry.

```

auto resource_attributes = opentelemetry::sdk::resource::ResourceAttributes
{
    {"service.name": "shoppingcart"},
    {"service.instance.id": "instance-12"}
};
auto resource = opentelemetry::sdk::resource::Resource::Create(resource_attributes);
auto received_attributes = resource.GetAttributes();
// received_attributes contains
//   - service.name = shoppingcart
//   - service.instance.id = instance-12
//   - telemetry.sdk.name = opentelemetry
//   - telemetry.sdk.language = cpp
//   - telemetry.sdk.version = <current sdk version>

```

It is possible to define the custom resource detectors by inhering from *opentelemetry::sdk::Resource::ResourceDetector* class.

## 2.5 Sampler

Sampling is mechanism to control/reducing the number of samples of traces collected and sent to the backend. OpenTelemetry C++ SDK offers four samplers out of the box:

- AlwaysOnSampler which samples every trace regardless of upstream sampling decisions.
- AlwaysOffSampler which doesn't sample any trace, regardless of upstream sampling decisions.
- ParentBased which uses the parent span to make sampling decisions, if present.
- TraceIdRatioBased which samples a configurable percentage of traces.

```

//AlwaysOnSampler
opentelemetry::sdk::trace::AlwaysOnSampler always_on_sampler;

```

(continues on next page)

(continued from previous page)

```
//AlwaysOffSampler
opentelemetry::sdk::trace::AlwaysOffSampler always_off_sampler;

//ParentBasedSampler
opentelemetry::sdk::trace::ParentBasedSampler sampler_off(std::make_shared
↳<AlwaysOffSampler>());

//TraceIdRatioBasedSampler - Sample 50% generated spans
double ratio = 0.5;
opentelemetry::sdk::trace::TraceIdRatioBasedSampler s(ratio);
```

## 2.6 TracerContext

SDK configuration are shared between *TracerProvider* and all it's *Tracer* instances through *TracerContext*.

```
auto tracer_context = std::make_shared<sdktrace::TracerContext>
(std::move(multi_processor), resource, std::move(always_on_sampler));
```

## 2.7 TracerProvider

*TracerProvider* instance holds the SDK configurations ( Span Processors, Samplers, Resource). There is single global *TracerProvider* instance for an application, and it is created at the start of application. There are two different mechanisms to create *TraceProvider* instance

- Using constructor which takes already created *TracerContext* shared object as parameter.
- Using constructor which takes SDK configurations as parameter.

```
// Created using `TracerContext` instance
auto tracer_provider = sdktrace::TracerProvider(tracer_context);

// Create using SDK configurations as parameter
auto tracer_provider =
    sdktrace::TracerProvider(std::move(simple_processor), resource, std::move(always_on_
↳sampler));

// set the global tracer TraceProvider
opentelemetry::trace::Provider::SetTracerProvider(provider);
```

## 2.8 Logging and Error Handling

OpenTelemetry C++ SDK provides mechanism for application owner to add customer log and error handler. The default log handler is redirected to standard output ( using `std::cout` ).

The logging macro supports logging using C++ stream format, and key-value pair. The log handler is meant to capture errors and warnings arising from SDK, not supposed to be used for the application errors. The different log levels are supported - Error, Warn, Info and Debug. The default log level is Warn ( to dump both Error and Warn) and it can be changed at compile time.

```
OTEL_INTERNAL_LOG_ERROR
    (" Connection failed. Error string " << error_str << " Error Num: " << errorno);
OTEL_INTERNAL_LOG_ERROR
    (" Connection failed." , {"error message: " : error_str,{"error number":
↪errorno}});
OTEL_INTERNAL_LOG_DEBUG
    (" Connection Established Successfully. Headers:", {"url", url},{"content-length
↪", len}, {"content-type", type});
```

The custom log handler can be defined by inheriting from `sdk::common::internal_log::LogHandler` class.

```
class CustomLogHandler : public sdk::common::internal_log::LogHandler
{
    void Handle(Loglevel level,
                const char \*file,
                int line,
                const char \*msg,
                const sdk::common::AttributeMap &attributes)

    {
        // add implementation here
    }
};
sdk::common::internal_log::GlobalLogHandler::SetLogHandler(CustomLogHandler());
```



## REFERENCE DOCUMENTATION

### 3.1 Full API

#### 3.1.1 Namespaces

##### Namespace opentelemetry

###### Contents

- *Namespaces*

##### Namespaces

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

##### Namespace opentelemetry::baggage

###### Contents

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

## Namespaces

- *Namespace `opentelemetry::baggage::propagation`*

## Classes

- *Class `Baggage`*

## Functions

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

## Variables

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

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

### Contents

- *Classes*

## Classes

- *Class `BaggagePropagator`*

## Namespace `opentelemetry::common`

### Contents

- *Classes*
- *Typedefs*



## Classes

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

## Typedefs

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

## Namespace `opentelemetry::context`

### Contents

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

## Namespaces

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

## Classes

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

## Functions

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

## Typedefs

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

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

### Contents

- *Classes*

## Classes

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

## Namespace `opentelemetry::sdk`

### Contents

- *Namespaces*

## Namespaces

- *Namespace `opentelemetry::sdk::instrumentationlibrary`*
- *Namespace `opentelemetry::sdk::resource`*
- *Namespace `opentelemetry::sdk::trace`*

## Namespace `opentelemetry::sdk::instrumentationlibrary`

### Contents

- *Classes*

## Classes

- *Class InstrumentationLibrary*

## Namespace opentelemetry::sdk::resource

### Contents

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

## Classes

- *Class OTELResourceDetector*
- *Class Resource*
- *Class ResourceDetector*

## Functions

- *Function opentelemetry::sdk::resource::attr*

## Typedefs

- *Typedef opentelemetry::sdk::resource::ResourceAttributes*

## Variables

- *Variable opentelemetry::sdk::resource::attribute\_ids*

## Namespace opentelemetry::sdk::trace

### Contents

- *Namespaces*
- *Classes*
- *Enums*

## Namespaces

- *Namespace `opentelemetry::sdk::trace::@43`*

## Classes

- *Struct `BatchSpanProcessorOptions`*
- *Struct `MultiSpanProcessor::ProcessorNode`*
- *Struct `MultiSpanProcessorOptions`*
- *Struct `SamplingResult`*
- *Class `AlwaysOffSampler`*
- *Class `AlwaysOnSampler`*
- *Class `BatchSpanProcessor`*
- *Class `IdGenerator`*
- *Class `MultiRecordable`*
- *Class `MultiSpanProcessor`*
- *Class `ParentBasedSampler`*
- *Class `RandomIdGenerator`*
- *Class `Recordable`*
- *Class `Sampler`*
- *Class `SimpleSpanProcessor`*
- *Class `SpanData`*
- *Class `SpanDataEvent`*
- *Class `SpanDataLink`*
- *Class `SpanExporter`*
- *Class `SpanProcessor`*
- *Class `TraceIdRatioBasedSampler`*
- *Class `Tracer`*
- *Class `TracerContext`*
- *Class `TracerProvider`*

## Enums

- *Enum Decision*

## Namespace opentelemetry::sdk::trace::@43

## Namespace opentelemetry::trace

### Contents

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

## Namespaces

- *Namespace opentelemetry::trace::propagation*

## Classes

- *Struct EndSpanOptions*
- *Struct StartSpanOptions*
- *Class DefaultSpan*
- *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

- *Function `opentelemetry::trace::attr`*
- *Function `opentelemetry::trace::GetSpan`*
- *Function `opentelemetry::trace::SetSpan`*

### Variables

- *Variable `opentelemetry::trace::attribute_id`*
- *Variable `opentelemetry::trace::attribute_ids`*
- *Variable `opentelemetry::trace::attribute_key`*
- *Variable `opentelemetry::trace::kSpanKey`*

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

#### Contents

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

### Namespaces

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

## Classes

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

## 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::kJaegerTraceHeader*
- *Variable opentelemetry::trace::propagation::kSpanIdHexStrLength*
- *Variable opentelemetry::trace::propagation::kSpanIdSize*
- *Variable opentelemetry::trace::propagation::kTraceFlagsSize*
- *Variable opentelemetry::trace::propagation::kTraceIdHexStrLength*
- *Variable opentelemetry::trace::propagation::kTraceIdSize*
- *Variable opentelemetry::trace::propagation::kTraceParent*
- *Variable opentelemetry::trace::propagation::kTraceParentSize*
- *Variable opentelemetry::trace::propagation::kTraceState*
- *Variable opentelemetry::trace::propagation::kVersionSize*

## Namespace opentelemetry::trace::propagation::detail

### Contents

- *Functions*
- *Variables*

## Functions

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

## Variables

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

### 3.1.2 Classes and Structs

#### Struct `BatchSpanProcessorOptions`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_sdk_include_opentelemetry_sdk_trace_batch_span_processor.h`

#### Struct Documentation

struct `opentelemetry::sdk::trace::BatchSpanProcessorOptions`

Struct to hold batch *SpanProcessor* options.

#### Public Members

size\_t **max\_queue\_size** = 2048

The maximum buffer/queue size. After the size is reached, spans are dropped.

std::chrono::milliseconds **schedule\_delay\_millis** = std::chrono::milliseconds(5000)

size\_t **max\_export\_batch\_size** = 512

The maximum batch size of every export. It must be smaller or equal to `max_queue_size`.

#### Struct `MultiSpanProcessor::ProcessorNode`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_sdk_include_opentelemetry_sdk_trace_multi_span_processor.h`



## Nested Relationships

This struct is a nested type of *Class MultiSpanProcessor*.

## Struct Documentation

```
struct opentelemetry::sdk::trace::MultiSpanProcessor::ProcessorNode
```

### Public Functions

```
inline ProcessorNode(std::unique_ptr<SpanProcessor> &&value, ProcessorNode *prev = nullptr,  
                    ProcessorNode *next = nullptr)
```

### Public Members

```
std::unique_ptr<SpanProcessor> value_  
ProcessorNode *next_  
ProcessorNode *prev_
```

## Struct MultiSpanProcessorOptions

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_v1.1.1\_sdk\_include\_opentelemetry\_sdk\_trace\_multi\_span\_processor.h

## Struct Documentation

```
struct MultiSpanProcessorOptions  
    Instantiation options.
```

## Struct SamplingResult

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_v1.1.1\_sdk\_include\_opentelemetry\_sdk\_trace\_sampler.h

## Struct Documentation

```
struct opentelemetry::sdk::trace::SamplingResult  
    The output of ShouldSample. It contains a sampling Decision and a set of Span Attributes.
```

## Public Members

### *Decision* **decision**

std::unique\_ptr<const std::map<std::string, opentelemetry::common::AttributeValue>> **attributes**

nostd::shared\_ptr<opentelemetry::trace::TraceState> **trace\_state**

## Struct EndSpanOptions

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_v1.1.1\_api\_include\_opentelemetry\_trace\_span\_metadata.h

## Struct Documentation

struct opentelemetry::trace::EndSpanOptions

*EndSpanOptions* provides options to set properties of a *Span* when it is ended.

## Public Members

common::SteadyTimestamp **end\_steady\_time**

## Struct StartSpanOptions

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_v1.1.1\_api\_include\_opentelemetry\_trace\_span\_startoptions.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**

nostd::variant<SpanContext, opentelemetry::context::Context> **parent** = *SpanContext::GetInvalid()*

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

## Class Baggage

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_v1.1.1\_api\_include\_opentelemetry\_baggage\_baggage.h

## Class Documentation

class opentelemetry::baggage::Baggage

### Public Functions

inline Baggage()

inline Baggage(size\_t size)

template<class T>

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

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

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

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

inline nostd::shared\_ptr<Baggage> Delete(nostd::string\_view key)

inline std::string ToHeader() const

### Public Static Functions

static inline nostd::shared\_ptr<Baggage> GetDefault()

static inline nostd::shared\_ptr<Baggage> FromHeader(nostd::string\_view header)

## Public Static Attributes

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

## Class BaggagePropagator

- Defined in file `__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_baggage_propagation_baggage_propagator.h`

## Inheritance Relationships

### Base Type

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

## Class Documentation

```
class opentelemetry::baggage::propagation::BaggagePropagator : public
opentelemetry::context::propagation::TextMapPropagator
```

## Public Functions

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

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

```
inline bool Fields(nostd::function_ref<bool(nostd::string_view)> callback) const noexcept override
```

## Class KeyValueIterable

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_common_key_value_iterable.h`

## Class Documentation

class `opentelemetry::common::KeyValueIterable`

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

### Public Functions

virtual `~KeyValueIterable()` = default

virtual bool **ForEachKeyValue**(`nostd::function_ref<bool(nostd::string_view, common::AttributeValue)>` callback) const noexcept = 0

Iterate over key-value pairs

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

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

virtual `size_t size()` const noexcept = 0

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

## Class SteadyTimestamp

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_common_timestamp.h`

## Class Documentation

class `opentelemetry::common::SteadyTimestamp`

A timepoint relative to the monotonic clock epoch.

This is used for calculating the duration of an operation.

### Public Functions

inline `SteadyTimestamp()` noexcept

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

template<class `Rep`, class `Period`>

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

Initializes a monotonic timestamp from a duration.

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

inline **SteadyTimestamp**(const std::chrono::steady\_clock::time\_point &time\_point) noexcept  
Initializes a monotonic timestamp based on a point in time.

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

inline **operator** std::chrono::steady\_clock::time\_point() const noexcept  
Returns a time point for the time stamp.

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

inline std::chrono::nanoseconds **time\_since\_epoch**() const noexcept  
Returns the nanoseconds since the beginning of the epoch.

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

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

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

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

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

## Class SystemTimestamp

- Defined in file \_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_v1.1.1\_api\_include\_opentelemetry\_common\_timestamp.h

## Class Documentation

class opentelemetry::common::SystemTimestamp

A timepoint relative to the system clock epoch.

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

## Public Functions

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

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

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

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

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

inline **operator** std::chrono::system\_clock::time\_point() const noexcept  
Returns a time point for the time stamp.

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

inline std::chrono::nanoseconds **time\_since\_epoch**() const noexcept  
Returns the nanoseconds since the beginning of the epoch.

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

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

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

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

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

## Class Context

- Defined in file \_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_v1.1.1\_api\_include\_opentelemetry\_context\_context.h

## Class Documentation

class opentelemetry::context::Context

### Public Functions

**Context**() = default

template<class T>  
inline **Context**(const T &keys\_and\_values)

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

template<class T>  
inline *Context* **SetValues**(T &values) noexcept

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

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

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

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

## Class CompositePropagator

- Defined in file `_home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_context_propagation_composite_propagator.h`

## Inheritance Relationships

### Base Type

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

## Class Documentation

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

```
inline bool Fields(nostd::function_ref<bool(nostd::string_view)> callback) const noexcept override
```

Invoke callback with fields set to carrier by `inject` method for all the configured propagators. Returns true if all invocation return true



## Class GlobalTextMapPropagator

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_context_propagation_global_propagator.h`

## Class Documentation

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

### Public Static Functions

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

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

## Class NoOpPropagator

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_context_propagation_noop_propagator.h`

## Inheritance Relationships

### Base Type

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

## Class Documentation

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

### Public Functions

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

```
inline void Inject(TextMapCarrier&, const context::Context &context) noexcept override
    Noop inject function does nothing
```

```
inline bool Fields(nostd::function_ref<bool(nostd::string_view)> callback) const noexcept override
```

## Class TextMapCarrier

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_context_propagation_text_map_propagator.h`

## Class Documentation

class opentelemetry::context::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

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

virtual ~TextMapCarrier() = default

## Class TextMapPropagator

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_context_propagation_text_map_propagator.h`

## Inheritance Relationships

### Derived Types

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

## Class Documentation

class opentelemetry::context::propagation::TextMapPropagator

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

### Public Functions

virtual *context::Context* **Extract**(const *TextMapCarrier* &carrier, *context::Context* &context) noexcept = 0

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

virtual bool **Fields**(nostd::function\_ref<bool(nostd::string\_view)> callback) const noexcept = 0

virtual ~**TextMapPropagator**() = default

## Class RuntimeContext

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_context_runtime_context.h`

## Class Documentation

class opentelemetry::context::RuntimeContext

### Public Static Functions

static inline *Context* **GetCurrent**() noexcept

static inline nostd::unique\_ptr<*Token*> **Attach**(const *Context* &context) noexcept

static inline bool **Detach**(*Token* &token) noexcept

static inline *Context* **SetValue**(nostd::string\_view key, const *ContextValue* &value, *Context* \*context = nullptr) noexcept

static inline *ContextValue* **GetValue**(nostd::string\_view key, *Context* \*context = nullptr) noexcept

static inline void **SetRuntimeContextStorage**(nostd::shared\_ptr<*RuntimeContextStorage*> storage) noexcept

Provide a custom runtime context storage.

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

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

## Class RuntimeContextStorage

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_context_runtime_context.h`

## Inheritance Relationships

### Derived Type

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

## Class Documentation

class `opentelemetry::context::RuntimeContextStorage`

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

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

Subclassed by *opentelemetry::context::ThreadLocalContextStorage*

### Public Functions

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

Return the current context.

**Returns** the current context

virtual `nstd::unique_ptr<Token>` **Attach**(const *Context* &context) noexcept = 0

Set the current context.

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

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

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

Detach the context related to the given token.

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

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

inline virtual `~RuntimeContextStorage`()

## Protected Functions

inline nstd::unique\_ptr<*Token*> **CreateToken**(const *Context* &context) noexcept

## Class ThreadLocalContextStorage

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_v1.1.1\_api\_include\_opentelemetry\_context\_runtime\_context.h

## Inheritance Relationships

### Base Type

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

## Class Documentation

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

## Public Functions

**ThreadLocalContextStorage**() noexcept = default

inline *Context* **GetCurrent**() noexcept override

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

inline nstd::unique\_ptr<*Token*> **Attach**(const *Context* &context) noexcept override

## Class Token

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_v1.1.1\_api\_include\_opentelemetry\_context\_runtime\_context.h

## Class Documentation

class opentelemetry::context::Token

### Public Functions

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

inline ~Token()

## Class InstrumentationLibrary

- Defined in file \_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_v1.1.1\_sdk\_include\_opentelemetry\_sdk\_instrumentationlibrary\_instrumentation\_library.h

## Class Documentation

class opentelemetry::sdk::instrumentationlibrary::InstrumentationLibrary

### Public Functions

**InstrumentationLibrary**(const *InstrumentationLibrary* &) = default

inline bool **operator==**(const *InstrumentationLibrary* &other) const  
Compare 2 instrumentation libraries.

**Parameters** **other** – the instrumentation library to compare to.

**Returns** true if the 2 instrumentation libraries are equal, false otherwise.

inline bool **equal**(const nstd::string\_view name, const nstd::string\_view version, const nstd::string\_view schema\_url = "") const

Check whether the instrumentation library has given name and version. This could be used to check version equality and avoid heap allocation.

#### Parameters

- **name** – name of the instrumentation library to compare.
- **version** – version of the instrumentatoin library to compare.
- **schema\_url** – schema url of the telemetry emitted by the library.

**Returns** true if name and version in this instrumentation library are equal with the given name and version.

inline const std::string &**GetName**() const

inline const std::string &**GetVersion**() const

```
inline const std::string &GetSchemaURL() const
```

## Public Static Functions

```
static inline nostd::unique_ptr<InstrumentationLibrary> Create(nostd::string_view name, nostd::string_view  
                                                                version = "", nostd::string_view schema_url  
                                                                = "")
```

Returns a newly created InstrumentationLibrary with the specified library name and version.

### Parameters

- **name** – name of the instrumentation library.
- **version** – version of the instrumentation library.
- **schema\_url** – schema url of the telemetry emitted by the library.

**Returns** the newly created InstrumentationLibrary.

## Class OTELResourceDetector

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_sdk_include_opentelemetry_sdk_resource_resource_detector.h`

## Inheritance Relationships

### Base Type

- public `opentelemetry::sdk::resource::ResourceDetector` (*Class ResourceDetector*)

## Class Documentation

```
class opentelemetry::sdk::resource::OTELResourceDetector : public
```

```
opentelemetry::sdk::resource::ResourceDetector
```

OTelResourceDetector to detect the presence of and create a Resource from the OTEL\_RESOURCE\_ATTRIBUTES environment variable.

### Public Functions

```
virtual Resource Detect() noexcept override
```

## Class Resource

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_sdk_include_opentelemetry_sdk_resource_resource.h`

## Class Documentation

class opentelemetry::sdk::resource::Resource

### Public Functions

**Resource**(const Resource&) = default

const ResourceAttributes &GetAttributes() const noexcept

const std::string &GetSchemaURL() const noexcept

*Resource* Merge(const Resource &other) noexcept

Returns a new, merged Resource by merging the current Resource with the other Resource. In case of a collision, current Resource takes precedence.

**Parameters** *other* – the Resource that will be merged with this.

**Returns** the newly merged Resource.

### Public Static Functions

static Resource Create(const ResourceAttributes &attributes, const std::string &schema\_url = std::string{ })

Returns a newly created Resource with the specified attributes. It adds (merge) SDK attributes and OTEL attributes before returning.

**Parameters** *attributes* – for this resource

**Returns** the newly created Resource.

static Resource &GetEmpty()

Returns an Empty resource.

static Resource &GetDefault()

Returns a Resource that identifies the SDK in use.

### Protected Functions

**Resource**(const ResourceAttributes &attributes = ResourceAttributes(), const std::string &schema\_url = std::string{ }) noexcept

The constructor is protected and only for use internally by the class and inside *ResourceDetector* class. Users should use the Create factory method to obtain a Resource instance.



## Class ResourceDetector

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_sdk_include_opentelemetry_sdk_resource_resource_detector.h`

## Inheritance Relationships

### Derived Type

- `public opentelemetry::sdk::resource::OTELResourceDetector` (*Class OTELResourceDetector*)

## Class Documentation

class `opentelemetry::sdk::resource::ResourceDetector`

Interface for a Resource Detector

Subclassed by *`opentelemetry::sdk::resource::OTELResourceDetector`*

### Public Functions

virtual *`Resource Detect()`* = 0

## Class AlwaysOffSampler

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_sdk_include_opentelemetry_sdk_trace_samplers_always_off.h`

## Inheritance Relationships

### Base Type

- `public opentelemetry::sdk::trace::Sampler` (*Class Sampler*)

## Class Documentation

class `opentelemetry::sdk::trace::AlwaysOffSampler` : public `opentelemetry::sdk::trace::Sampler`

The always off sampler always returns DROP, effectively disabling tracing functionality.

## Public Functions

```
inline virtual SamplingResult ShouldSample(const opentelemetry::trace::SpanContext &parent_context,
                                             opentelemetry::trace::TraceId, nostd::string_view,
                                             opentelemetry::trace::SpanKind, const
                                             opentelemetry::common::KeyValueIterable&, const
                                             opentelemetry::trace::SpanContextKeyValueIterable&)
    noexcept override
```

**Returns** Returns DROP always

```
inline virtual nostd::string_view GetDescription() const noexcept override
```

**Returns** Description MUST be *AlwaysOffSampler*

## Class AlwaysOnSampler

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_v1.1.1\_sdk\_include\_opentelemetry\_sdk\_trace\_samplers\_always\_on.h

## Inheritance Relationships

### Base Type

- public opentelemetry::sdk::trace::Sampler (*Class Sampler*)

## Class Documentation

```
class opentelemetry::sdk::trace::AlwaysOnSampler : public opentelemetry::sdk::trace::Sampler
    The always on sampler is a default sampler which always return Decision::RECORD_AND_SAMPLE
```

## Public Functions

```
inline virtual SamplingResult ShouldSample(const opentelemetry::trace::SpanContext &parent_context,
                                             opentelemetry::trace::TraceId, nostd::string_view,
                                             opentelemetry::trace::SpanKind, const
                                             opentelemetry::common::KeyValueIterable&, const
                                             opentelemetry::trace::SpanContextKeyValueIterable&)
    noexcept override
```

**Returns** Always return Decision RECORD\_AND\_SAMPLE

```
inline virtual nostd::string_view GetDescription() const noexcept override
```

**Returns** Description MUST be *AlwaysOnSampler*

## Class BatchSpanProcessor

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_sdk_include_opentelemetry_sdk_trace_batch_span_processor.h`

## Inheritance Relationships

### Base Type

- `public opentelemetry::sdk::trace::SpanProcessor` (*Class SpanProcessor*)

## Class Documentation

class `opentelemetry::sdk::trace::BatchSpanProcessor` : public `opentelemetry::sdk::trace::SpanProcessor`  
 This is an implementation of the *SpanProcessor* which creates batches of finished spans and passes the export-friendly span data representations to the configured *SpanExporter*.

### Public Functions

**BatchSpanProcessor**(`std::unique_ptr<SpanExporter> &&exporter`, const *BatchSpanProcessorOptions* &options)

Creates a batch span processor by configuring the specified exporter and other parameters as per the official, language-agnostic opentelemetry specs.

#### Parameters

- exporter** – - The backend exporter to pass the ended spans to.
- options** – - The batch *SpanProcessor* options.

virtual `std::unique_ptr<Recordable> MakeRecordable()` noexcept override

Requests a Recordable(Span) from the configured exporter.

**Returns** A recordable generated by the backend exporter

virtual void **OnStart**(*Recordable* &span, const `opentelemetry::trace::SpanContext` &parent\_context)

noexcept override

Called when a span is started.

NOTE: This method is a no-op.

#### Parameters

- span** – - The span that just started
- parent\_context** – - The parent context of the span that just started

virtual void **OnEnd**(`std::unique_ptr<Recordable> &&span`) noexcept override

Called when a span ends.

**Parameters** **span** – - A recordable for a span that just ended

virtual bool **ForceFlush**(`std::chrono::microseconds timeout = std::chrono::microseconds::max()`) noexcept override

Export all ended spans that have not been exported yet.

NOTE: Timeout functionality not supported yet.

virtual bool **Shutdown**(std::chrono::microseconds timeout = std::chrono::microseconds::max()) noexcept  
override

Shuts down the processor and does any cleanup required. Completely drains the buffer/queue of all its ended spans and passes them to the exporter. Any subsequent calls to OnStart, OnEnd, ForceFlush or Shutdown will return immediately without doing anything.

NOTE: Timeout functionality not supported yet.

#### **~BatchSpanProcessor()**

Class destructor which invokes the *Shutdown()* method. The *Shutdown()* method is supposed to be invoked when the Tracer is shutdown (as per other languages), but the C++ Tracer only takes shared ownership of the processor, and thus doesn't call Shutdown (as the processor might be shared with other Tracers).

### **Class IdGenerator**

- Defined in file `__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_sdk_include_opentelemetry_sdk_trace_id_generator.h`

### **Inheritance Relationships**

#### **Derived Type**

- public `opentelemetry::sdk::trace::RandomIdGenerator` (Class *RandomIdGenerator*)

### **Class Documentation**

class `opentelemetry::sdk::trace::IdGenerator`

*IdGenerator* provides an interface for generating Trace Id and Span Id

Subclassed by *opentelemetry::sdk::trace::RandomIdGenerator*

#### **Public Functions**

virtual **~IdGenerator()** = default

virtual `opentelemetry::trace::SpanId` **GenerateSpanId()** noexcept = 0

Returns a SpanId represented by opaque 128-bit trace identifier

virtual `opentelemetry::trace::TraceId` **GenerateTraceId()** noexcept = 0

Returns a TraceId represented by opaque 64-bit trace identifier

## Class MultiRecordable

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_sdk_include_opentelemetry_sdk_trace_multi_recordable.h`

## Inheritance Relationships

### Base Type

- public `opentelemetry::sdk::trace::Recordable` (*Class Recordable*)

## Class Documentation

```
class opentelemetry::sdk::trace::MultiRecordable : public opentelemetry::sdk::trace::Recordable
```

### Public Functions

```
inline void AddRecordable(const SpanProcessor &processor, std::unique_ptr<Recordable> recordable)
    noexcept
```

```
inline const std::unique_ptr<Recordable> &GetRecordable(const SpanProcessor &processor) const
    noexcept
```

```
inline std::unique_ptr<Recordable> ReleaseRecordable(const SpanProcessor &processor) noexcept
```

```
inline void SetIdentity(const opentelemetry::trace::SpanContext &span_context,
    opentelemetry::trace::SpanId parent_span_id) noexcept override
```

```
inline void SetAttribute(nostd::string_view key, const opentelemetry::common::AttributeValue &value)
    noexcept override
```

```
inline void AddEvent(nostd::string_view name, opentelemetry::common::SystemTimestamp timestamp, const
    opentelemetry::common::KeyValueIterable &attributes) noexcept override
```

```
inline void AddLink(const opentelemetry::trace::SpanContext &span_context, const
    opentelemetry::common::KeyValueIterable &attributes) noexcept override
```

```
inline void SetStatus(opentelemetry::trace::StatusCode code, nostd::string_view description) noexcept
    override
```

```
inline void SetName(nostd::string_view name) noexcept override
```

```
inline void SetSpanKind(opentelemetry::trace::SpanKind span_kind) noexcept override
```

```
inline void SetResource(const opentelemetry::sdk::resource::Resource &resource) noexcept override
```

```
inline void SetStartTime(opentelemetry::common::SystemTimestamp start_time) noexcept override
```

```
inline void SetDuration(std::chrono::nanoseconds duration) noexcept override
```

```
inline void SetInstrumentationLibrary(const InstrumentationLibrary &instrumentation_library)  
noexcept override
```

## Class MultiSpanProcessor

- Defined in file `__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_sdk_include_opentelemetry_sdk_trace_multi_span_processor.h`

## Nested Relationships

### Nested Types

- Struct MultiSpanProcessor::ProcessorNode*

## Inheritance Relationships

### Base Type

- public opentelemetry::sdk::trace::SpanProcessor (*Class SpanProcessor*)

## Class Documentation

```
class opentelemetry::sdk::trace::MultiSpanProcessor : public opentelemetry::sdk::trace::SpanProcessor  
Span processor allow hooks for span start and end method invocations.
```

Built-in span processors are responsible for batching and conversion of spans to exportable representation and passing batches to exporters.

### Public Functions

```
inline MultiSpanProcessor(std::vector<std::unique_ptr<SpanProcessor>> &&processors)
```

```
inline void AddProcessor(std::unique_ptr<SpanProcessor> &&processor)
```

```
inline virtual std::unique_ptr<Recordable> MakeRecordable() noexcept override
```

Create a span recordable. This requests a new span recordable from the associated exporter.

Note: This method must be callable from multiple threads.

**Returns** a newly initialized recordable

```
inline virtual void OnStart(Recordable &span, const opentelemetry::trace::SpanContext &parent_context)
    noexcept override
```

OnStart is called when a span is started.

**Parameters**

- **span** – a recordable for a span that was just started
- **parent\_context** – The parent context of the span that just started

```
inline virtual void OnEnd(std::unique_ptr<Recordable> &&span) noexcept override
```

OnEnd is called when a span is ended.

**Parameters** **span** – a recordable for a span that was ended

```
inline virtual bool ForceFlush(std::chrono::microseconds timeout = (std::chrono::microseconds::max)())
    noexcept override
```

Export all ended spans that have not yet been exported.

**Parameters** **timeout** – an optional timeout, the default timeout of 0 means that no timeout is applied.

```
inline virtual bool Shutdown(std::chrono::microseconds timeout = (std::chrono::microseconds::max)())
    noexcept override
```

Shut down the processor and do any cleanup required. Ended spans are exported before shutdown. After the call to Shutdown, subsequent calls to OnStart, OnEnd, ForceFlush or Shutdown will return immediately without doing anything.

**Parameters** **timeout** – an optional timeout, the default timeout of 0 means that no timeout is applied.

```
inline ~MultiSpanProcessor()
```

## Class ParentBasedSampler

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_sdk_include_opentelemetry_sdk_trace_samplers_parent.h`

## Inheritance Relationships

### Base Type

- public opentelemetry::sdk::trace::Sampler (*Class Sampler*)

## Class Documentation

class opentelemetry::sdk::trace::ParentBasedSampler : public opentelemetry::sdk::trace::Sampler  
The ParentBased sampler is a composite sampler. ParentBased(delegateSampler) either respects the parent span's sampling decision or delegates to delegateSampler for root spans.

### Public Functions

explicit ParentBasedSampler(std::shared\_ptr<Sampler> delegate\_sampler) noexcept

virtual SamplingResult ShouldSample(const opentelemetry::trace::SpanContext &parent\_context,  
opentelemetry::trace::TraceId trace\_id, nostd::string\_view name,  
opentelemetry::trace::SpanKind span\_kind, const  
opentelemetry::common::KeyValueIterable &attributes, const  
opentelemetry::trace::SpanContextKeyValueIterable &links) noexcept  
override

The decision either respects the parent span's sampling decision or delegates to delegateSampler for root spans

**Returns** Returns DROP always

virtual nostd::string\_view GetDescription() const noexcept override

**Returns** Description MUST be ParentBased{delegate\_sampler\_.getDescription()}

## Class RandomIdGenerator

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_v1.1.1\_sdk\_include\_opentelemetry\_sdk\_trace\_random\_id\_generator.h

## Inheritance Relationships

### Base Type

- public opentelemetry::sdk::trace::IdGenerator (Class IdGenerator)

## Class Documentation

class opentelemetry::sdk::trace::RandomIdGenerator : public opentelemetry::sdk::trace::IdGenerator



## Public Functions

opentelemetry::trace::*SpanId* **GenerateSpanId**() noexcept override

opentelemetry::trace::*TraceId* **GenerateTraceId**() noexcept override

## Class Recordable

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_sdk_include_opentelemetry_sdk_trace_recordable.h`

## Inheritance Relationships

### Derived Types

- public opentelemetry::sdk::trace::MultiRecordable (*Class MultiRecordable*)
- public opentelemetry::sdk::trace::SpanData (*Class SpanData*)

## Class Documentation

class opentelemetry::sdk::trace::Recordable

Maintains a representation of a span in a format that can be processed by a recorder.

This class is thread-compatible.

Subclassed by *opentelemetry::sdk::trace::MultiRecordable*, *opentelemetry::sdk::trace::SpanData*

## Public Functions

virtual ~Recordable() = default

virtual void SetIdentity(const opentelemetry::trace::SpanContext &span\_context,  
opentelemetry::trace::SpanId parent\_span\_id) noexcept = 0  
Set the span context and parent span id

### Parameters

- **span\_context** – the span context to set
- **parent\_span\_id** – the parent span id to set

virtual void SetAttribute(nostd::string\_view key, const opentelemetry::common::AttributeValue &value)  
noexcept = 0  
Set an attribute of a span.

### Parameters

- **name** – the name of the attribute
- **value** – the attribute value

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

Add an event to a span.

**Parameters**

- **name** – the name of the event
- **timestamp** – the timestamp of the event
- **attributes** – the attributes associated with the event

inline void **AddEvent**(nstd::string\_view name)

Add an event to a span with default timestamp and attributes.

**Parameters** **name** – the name of the event

inline void **AddEvent**(nstd::string\_view name, opentelemetry::common::SystemTimestamp timestamp)

Add an event to a span with default (empty) attributes.

**Parameters**

- **name** – the name of the event
- **timestamp** – the timestamp of the event

virtual void **AddLink**(const opentelemetry::trace::SpanContext &span\_context, const opentelemetry::common::KeyValueIterable &attributes) noexcept = 0

Add a link to a span.

**Parameters**

- **span\_context** – the span context of the linked span
- **attributes** – the attributes associated with the link

inline void **AddLink**(opentelemetry::trace::SpanContext span\_context)

Add a link to a span with default (empty) attributes.

**Parameters** **span\_context** – the span context of the linked span

virtual void **SetStatus**(opentelemetry::trace::StatusCode code, nstd::string\_view description) noexcept = 0

Set the status of the span.

**Parameters**

- **code** – the status code
- **description** – a description of the status

virtual void **SetName**(nstd::string\_view name) noexcept = 0

Set the name of the span.

**Parameters** **name** – the name to set

virtual void **SetSpanKind**(opentelemetry::trace::SpanKind span\_kind) noexcept = 0

Set the spankind of the span.

**Parameters** **span\_kind** – the spankind to set

virtual void **SetResource**(const opentelemetry::sdk::resource::Resource &resource) noexcept = 0

Set Resource of the span

**Parameters** **Resource** – the resource to set

virtual void **SetStartTime**(opentelemetry::common::SystemTimestamp start\_time) noexcept = 0

Set the start time of the span.

**Parameters** `start_time` – the start time to set

virtual void **SetDuration**(std::chrono::nanoseconds duration) noexcept = 0  
Set the duration of the span.

**Parameters** `duration` – the duration to set

virtual void **SetInstrumentationLibrary**(const InstrumentationLibrary &instrumentation\_library)  
noexcept = 0  
Set the instrumentation library of the span.

**Parameters** `instrumentation_library` – the instrumentation library to set

## Class Sampler

- Defined in file `__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_sdk_include_opentelemetry_sdk_trace_sampler.h`

## Inheritance Relationships

## Derived Types

- public `opentelemetry::sdk::trace::AlwaysOffSampler` (*Class AlwaysOffSampler*)
- public `opentelemetry::sdk::trace::AlwaysOnSampler` (*Class AlwaysOnSampler*)
- public `opentelemetry::sdk::trace::ParentBasedSampler` (*Class ParentBasedSampler*)
- public `opentelemetry::sdk::trace::TraceIdRatioBasedSampler` (*Class TraceIdRatioBasedSampler*)

## Class Documentation

class `opentelemetry::sdk::trace::Sampler`

The *Sampler* interface allows users to create custom samplers which will return a *SamplingResult* based on information that is typically available just before the Span was created.

Subclassed by `opentelemetry::sdk::trace::AlwaysOffSampler`, `opentelemetry::sdk::trace::AlwaysOnSampler`, `opentelemetry::sdk::trace::ParentBasedSampler`, `opentelemetry::sdk::trace::TraceIdRatioBasedSampler`

## Public Functions

virtual `~Sampler()` = default

virtual *SamplingResult* **ShouldSample**(const `opentelemetry::trace::SpanContext` &parent\_context,  
opentelemetry::trace::TraceId trace\_id, nstd::string\_view name,  
opentelemetry::trace::SpanKind span\_kind, const  
opentelemetry::common::KeyValueIterable &attributes, const  
opentelemetry::trace::SpanContextKeyValueIterable &links) noexcept  
= 0

Called during Span creation to make a sampling decision.

Since 0.1.0

#### Parameters

- **parent\_context** – a const reference to the SpanContext of a parent Span. An invalid SpanContext if this is a root span.
- **trace\_id** – the TraceId for the new Span. This will be identical to that in the parentContext, unless this is a root span.
- **name** – the name of the new Span.
- **spanKind** – the opentelemetry::trace::SpanKind of the Span.
- **attributes** – list of AttributeValue with their keys.
- **links** – Collection of links that will be associated with the Span to be created.

**Returns** sampling result whether span should be sampled or not.

virtual nostd::string\_view **GetDescription()** const noexcept = 0

Returns the sampler name or short description with the configuration. This may be displayed on debug pages or in the logs.

**Returns** the description of this *Sampler*.

### Class SimpleSpanProcessor

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_v1.1.1\_sdk\_include\_opentelemetry\_sdk\_trace\_simple\_processor.h

### Inheritance Relationships

#### Base Type

- public opentelemetry::sdk::trace::SpanProcessor (*Class SpanProcessor*)

### Class Documentation

class opentelemetry::sdk::trace::SimpleSpanProcessor : public opentelemetry::sdk::trace::SpanProcessor  
The simple span processor passes finished recordables to the configured *SpanExporter*, as soon as they are finished.

OnEnd and ForceFlush are no-ops.

All calls to the configured *SpanExporter* are synchronized using a spin-lock on an atomic\_flag.

## Public Functions

inline explicit **SimpleSpanProcessor**(std::unique\_ptr<*SpanExporter*> &&exporter) noexcept  
Initialize a simple span processor.

**Parameters** **exporter** – the exporter used by the span processor

inline virtual std::unique\_ptr<*Recordable*> **MakeRecordable**() noexcept override

Create a span recordable. This requests a new span recordable from the associated exporter.

Note: This method must be callable from multiple threads.

**Returns** a newly initialized recordable

inline virtual void **OnStart**(*Recordable* &span, const opentelemetry::trace::*SpanContext* &parent\_context)  
noexcept override

OnStart is called when a span is started.

**Parameters**

- **span** – a recordable for a span that was just started
- **parent\_context** – The parent context of the span that just started

inline virtual void **OnEnd**(std::unique\_ptr<*Recordable*> &&span) noexcept override  
OnEnd is called when a span is ended.

**Parameters** **span** – a recordable for a span that was ended

inline virtual bool **ForceFlush**(std::chrono::microseconds timeout = (std::chrono::microseconds::max)())  
noexcept override

Export all ended spans that have not yet been exported.

**Parameters** **timeout** – an optional timeout, the default timeout of 0 means that no timeout is applied.

inline virtual bool **Shutdown**(std::chrono::microseconds timeout = (std::chrono::microseconds::max)())  
noexcept override

Shut down the processor and do any cleanup required. Ended spans are exported before shutdown. After the call to Shutdown, subsequent calls to OnStart, OnEnd, ForceFlush or Shutdown will return immediately without doing anything.

**Parameters** **timeout** – an optional timeout, the default timeout of 0 means that no timeout is applied.

inline **~SimpleSpanProcessor**()

## Class SpanData

- Defined in file `__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_sdk_include_opentelemetry_sdk_trace_span_data.h`

## Inheritance Relationships

### Base Type

- `public opentelemetry::sdk::trace::Recordable` (*Class Recordable*)

### Class Documentation

class `opentelemetry::sdk::trace::SpanData` : public `opentelemetry::sdk::trace::Recordable`  
*SpanData* is a representation of all data collected by a span.

#### Public Functions

inline `SpanData()`

inline `opentelemetry::trace::TraceId GetTraceId()` const noexcept  
Get the trace id for this span

**Returns** the trace id for this span

inline `opentelemetry::trace::SpanId GetSpanId()` const noexcept  
Get the span id for this span

**Returns** the span id for this span

inline const `opentelemetry::trace::SpanContext &GetSpanContext()` const noexcept  
Get the span context for this span

**Returns** the span context for this span

inline `opentelemetry::trace::SpanId GetParentSpanId()` const noexcept  
Get the parent span id for this span

**Returns** the span id for this span's parent

inline `opentelemetry::nostd::string_view GetName()` const noexcept  
Get the name for this span

**Returns** the name for this span

inline `opentelemetry::trace::SpanKind GetSpanKind()` const noexcept  
Get the kind of this span

**Returns** the kind of this span

inline `opentelemetry::trace::StatusCode GetStatus()` const noexcept  
Get the status for this span

**Returns** the status for this span

inline `opentelemetry::nostd::string_view GetDescription()` const noexcept  
Get the status description for this span

**Returns** the description of the the status of this span

inline const `opentelemetry::sdk::resource::Resource &GetResource()` const noexcept  
Get the attributes associated with the resource

**Returns** the attributes associated with the resource configured for TracerProvider

```
inline const opentelemetry::sdk::trace::InstrumentationLibrary &GetInstrumentationLibrary() const noexcept
```

Get the attributes associated with the resource

**Returns** the attributes associated with the resource configured for TracerProvider

```
inline opentelemetry::common::SystemTimestamp GetStartTime() const noexcept
```

Get the start time for this span

**Returns** the start time for this span

```
inline std::chrono::nanoseconds GetDuration() const noexcept
```

Get the duration for this span

**Returns** the duration for this span

```
inline const std::unordered_map<std::string, common::OwnedAttributeValue> &GetAttributes() const noexcept
```

Get the attributes for this span

**Returns** the attributes for this span

```
inline const std::vector<SpanDataEvent> &GetEvents() const noexcept
```

Get the events associated with this span

**Returns** the events associated with this span

```
inline const std::vector<SpanDataLink> &GetLinks() const noexcept
```

Get the links associated with this span

**Returns** the links associated with this span

```
inline virtual void SetIdentity(const opentelemetry::trace::SpanContext &span_context,
                              opentelemetry::trace::SpanId parent_span_id) noexcept override
```

Set the span context and parent span id

**Parameters**

- **span\_context** – the span context to set
- **parent\_span\_id** – the parent span id to set

```
inline virtual void SetAttribute(nostd::string_view key, const opentelemetry::common::AttributeValue
                               &value) noexcept override
```

Set an attribute of a span.

**Parameters**

- **name** – the name of the attribute
- **value** – the attribute value

```
inline virtual void AddEvent(nostd::string_view name, opentelemetry::common::SystemTimestamp timestamp
                             =
                             opentelemetry::common::SystemTimestamp(std::chrono::system_clock::now()),
                             const opentelemetry::common::KeyValueIterable &attributes =
                             opentelemetry::common::KeyValueIterableView<std::map<std::string,
                             int>>({})) noexcept override
```

Add an event to a span.

**Parameters**

- **name** – the name of the event
- **timestamp** – the timestamp of the event

- **attributes** – the attributes associated with the event

inline virtual void **AddLink**(const opentelemetry::trace::SpanContext &span\_context, const opentelemetry::common::KeyValueIterable &attributes) noexcept override

Add a link to a span.

**Parameters**

- **span\_context** – the span context of the linked span
- **attributes** – the attributes associated with the link

inline virtual void **SetStatus**(opentelemetry::trace::StatusCode code, nostd::string\_view description) noexcept override

Set the status of the span.

**Parameters**

- **code** – the status code
- **description** – a description of the status

inline virtual void **SetName**(nostd::string\_view name) noexcept override

Set the name of the span.

**Parameters** **name** – the name to set

inline virtual void **SetSpanKind**(opentelemetry::trace::SpanKind span\_kind) noexcept override

Set the spankind of the span.

**Parameters** **span\_kind** – the spankind to set

inline virtual void **SetResource**(const opentelemetry::sdk::resource::Resource &resource) noexcept override

Set Resource of the span

**Parameters** **Resource** – the resource to set

inline virtual void **SetStartTime**(opentelemetry::common::SystemTimestamp start\_time) noexcept override

Set the start time of the span.

**Parameters** **start\_time** – the start time to set

inline virtual void **SetDuration**(std::chrono::nanoseconds duration) noexcept override

Set the duration of the span.

**Parameters** **duration** – the duration to set

inline virtual void **SetInstrumentationLibrary**(const InstrumentationLibrary &instrumentation\_library) noexcept override

Set the instrumentation library of the span.

**Parameters** **instrumentation\_library** – the instrumentation library to set

## Class SpanDataEvent

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_v1.1.1\_sdk\_include\_opentelemetry\_sdk\_trace\_span\_data.h



## Class Documentation

class opentelemetry::sdk::trace::SpanDataEvent

Class for storing events in *SpanData*.

### Public Functions

inline SpanDataEvent(std::string name, opentelemetry::common::SystemTimestamp timestamp, const opentelemetry::common::KeyValueIterable &attributes)

inline std::string GetName() const noexcept

Get the name for this event

**Returns** the name for this event

inline opentelemetry::common::SystemTimestamp GetTimestamp() const noexcept

Get the timestamp for this event

**Returns** the timestamp for this event

inline const std::unordered\_map<std::string, common::OwnedAttributeValue> &GetAttributes() const  
noexcept

Get the attributes for this event

**Returns** the attributes for this event

## Class SpanDataLink

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_v1.1.1\_sdk\_include\_opentelemetry\_sdk\_trace\_span\_data.h

## Class Documentation

class opentelemetry::sdk::trace::SpanDataLink

Class for storing links in *SpanData*.

### Public Functions

inline SpanDataLink(opentelemetry::trace::SpanContext span\_context, const opentelemetry::common::KeyValueIterable &attributes)

inline const std::unordered\_map<std::string, common::OwnedAttributeValue> &GetAttributes() const  
noexcept

Get the attributes for this link

**Returns** the attributes for this link

inline const opentelemetry::trace::SpanContext &GetSpanContext() const noexcept

Get the span context for this link

**Returns** the span context for this link

## Class SpanExporter

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_v1.1.1\_sdk\_include\_opentelemetry\_sdk\_trace\_exporter.h

## Class Documentation

class opentelemetry::sdk::trace::SpanExporter

*SpanExporter* defines the interface that protocol-specific span exporters must implement.

### Public Functions

virtual ~SpanExporter() = default

virtual std::unique\_ptr<Recordable> MakeRecordable() noexcept = 0

Create a span recordable. This object will be used to record span data and will subsequently be passed to *SpanExporter::Export*. Vendors can implement custom recordables or use the default *SpanData* recordable provided by the SDK.

Note: This method must be callable from multiple threads.

**Returns** a newly initialized *Recordable* object

virtual sdk::common::ExportResult Export(const  
nstd::span<std::unique\_ptr<opentelemetry::sdk::trace::Recordable>>  
&spans) noexcept = 0

Exports a batch of span recordables. This method must not be called concurrently for the same exporter instance.

**Parameters** **spans** – a span of unique pointers to span recordables

virtual bool Shutdown(std::chrono::microseconds timeout = std::chrono::microseconds::max()) noexcept = 0  
Shut down the exporter.

**Parameters** **timeout** – an optional timeout.

**Returns** return the status of the operation.

## Class SpanProcessor

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_v1.1.1\_sdk\_include\_opentelemetry\_sdk\_trace\_processor.h

## Inheritance Relationships

### Derived Types

- `public opentelemetry::sdk::trace::BatchSpanProcessor` (*Class [BatchSpanProcessor](#)*)
- `public opentelemetry::sdk::trace::MultiSpanProcessor` (*Class [MultiSpanProcessor](#)*)
- `public opentelemetry::sdk::trace::SimpleSpanProcessor` (*Class [SimpleSpanProcessor](#)*)

### Class Documentation

class `opentelemetry::sdk::trace::SpanProcessor`

Span processor allow hooks for span start and end method invocations.

Built-in span processors are responsible for batching and conversion of spans to exportable representation and passing batches to exporters.

Subclassed by `opentelemetry::sdk::trace::BatchSpanProcessor`, `opentelemetry::sdk::trace::MultiSpanProcessor`, `opentelemetry::sdk::trace::SimpleSpanProcessor`

### Public Functions

virtual `~SpanProcessor()` = default

virtual `std::unique_ptr<Recordable> MakeRecordable()` noexcept = 0

Create a span recordable. This requests a new span recordable from the associated exporter.

Note: This method must be callable from multiple threads.

**Returns** a newly initialized recordable

virtual void **OnStart**([Recordable](#) &span, const `opentelemetry::trace::SpanContext` &parent\_context) noexcept = 0

OnStart is called when a span is started.

#### Parameters

- **span** – a recordable for a span that was just started
- **parent\_context** – The parent context of the span that just started

virtual void **OnEnd**(`std::unique_ptr<Recordable> &&span`) noexcept = 0

OnEnd is called when a span is ended.

**Parameters** **span** – a recordable for a span that was ended

virtual bool **ForceFlush**(`std::chrono::microseconds timeout = (std::chrono::microseconds::max)()`) noexcept = 0

Export all ended spans that have not yet been exported.

**Parameters** **timeout** – an optional timeout, the default timeout of 0 means that no timeout is applied.

virtual bool **Shutdown**(std::chrono::microseconds timeout = (std::chrono::microseconds::max)()) noexcept = 0  
Shut down the processor and do any cleanup required. Ended spans are exported before shutdown. After the call to Shutdown, subsequent calls to OnStart, OnEnd, ForceFlush or Shutdown will return immediately without doing anything.

**Parameters** **timeout** – an optional timeout, the default timeout of 0 means that no timeout is applied.

## Class TraceIdRatioBasedSampler

- Defined in file `_home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_sdk_include_opentelemetry_sdk_trace_samplers_trace_id_ratio.h`

## Inheritance Relationships

### Base Type

- public opentelemetry::sdk::trace::Sampler (*Class Sampler*)

## Class Documentation

class opentelemetry::sdk::trace::TraceIdRatioBasedSampler : public opentelemetry::sdk::trace::Sampler  
The TraceIdRatioBased sampler computes and returns a decision based on the provided trace\_id and the configured ratio.

### Public Functions

explicit **TraceIdRatioBasedSampler**(double ratio)

**Parameters** **ratio** – a required value,  $1.0 \geq \text{ratio} \geq 0.0$ . If the given trace\_id falls into a given ratio of all possible trace\_id values, ShouldSample will return RECORD\_AND\_SAMPLE.

**Throws** **invalid\_argument** – if ratio is out of bounds [0.0, 1.0]

virtual *SamplingResult* **ShouldSample**(const opentelemetry::trace::SpanContext&,  
opentelemetry::trace::TraceId trace\_id, nostd::string\_view,  
opentelemetry::trace::SpanKind, const  
opentelemetry::common::KeyValueIterable&, const  
opentelemetry::trace::SpanContextKeyValueIterable&) noexcept  
override

**Returns** Returns either RECORD\_AND\_SAMPLE or DROP based on current sampler configuration and provided trace\_id and ratio. trace\_id is used as a pseudorandom value in conjunction with the predefined ratio to determine whether this trace should be sampled

virtual nostd::string\_view **GetDescription**() const noexcept override

**Returns** Description MUST be *TraceIdRatioBasedSampler*{0.000100}

## Class Tracer

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_sdk_include_opentelemetry_sdk_trace_tracer.h`

## Inheritance Relationships

## Base Types

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

## Class Documentation

class `opentelemetry::sdk::trace::Tracer` : public `opentelemetry::trace::Tracer`, public `std::enable_shared_from_this<Tracer>`

## Public Functions

explicit **Tracer**(`std::shared_ptr<sdk::trace::TracerContext>` context, `std::unique_ptr<InstrumentationLibrary>` instrumentation\_library = `InstrumentationLibrary::Create("")`) noexcept  
Construct a new Tracer with the given context pipeline.

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

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

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

inline *SpanProcessor* &**GetProcessor**() noexcept  
Returns the configured span processor.

inline *IdGenerator* &**GetIdGenerator**() const noexcept  
Returns the configured Id generator

inline const `InstrumentationLibrary` &**GetInstrumentationLibrary**() const noexcept  
Returns the associated instrumentation library

inline const `opentelemetry::sdk::resource::Resource` &**GetResource**()  
Returns the currently configured resource

inline *Sampler* &**GetSampler**()

## Class TracerContext

- Defined in file `__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_sdk_include_opentelemetry_sdk_trace_tracer_context.h`

## Class Documentation

class `opentelemetry::sdk::trace::TracerContext`

A class which stores the TracerProvider context.

This class meets the following design criteria:

- A shared reference between TracerProvider and Tracers instantiated.
- A thread-safe class that allows updating/altering processor/exporter pipelines and sampling config.
- The owner/destroyer of Processors/Exporters. These will remain active until this class is destroyed. I.e. Sampling, Exporting, flushing, Custom Iterator etc. are all ok if this object is alive, and they will work together. If this object is destroyed, then no shared references to Processor, Exporter, *Recordable*, Custom Iterator etc. should exist, and all associated pipelines will have been flushed.

## Public Functions

```
explicit TracerContext(std::vector<std::unique_ptr<SpanProcessor>> &&processor,  
    opentelemetry::sdk::resource::Resource resource =  
    opentelemetry::sdk::resource::Resource::Create({}), std::unique_ptr<Sampler>  
    sampler = std::unique_ptr<AlwaysOnSampler>(new AlwaysOnSampler),  
    std::unique_ptr<IdGenerator> id_generator = std::unique_ptr<IdGenerator>(new  
    RandomIdGenerator())) noexcept
```

```
void AddProcessor(std::unique_ptr<SpanProcessor> processor) noexcept
```

Attaches a span processor to list of configured processors to this tracer context. Processor once attached can't be removed.

Note: This method is not thread safe.

**Parameters** `processor` – The new span processor for this tracer. This must not be a nullptr. Ownership is given to the *TracerContext*.

*Sampler* &**GetSampler**() const noexcept  
Obtain the sampler associated with this tracer.

**Returns** The sampler for this tracer.

*SpanProcessor* &**GetProcessor**() const noexcept  
Obtain the configured (composite) processor.

Note: When more than one processor is active, this will return an “aggregate” processor

const opentelemetry::sdk::resource::Resource &**GetResource**() const noexcept  
Obtain the resource associated with this tracer context.

**Returns** The resource for this tracer context.

opentelemetry::sdk::trace::*IdGenerator* &**GetIdGenerator**() const noexcept  
Obtain the Id Generator associated with this tracer context.

**Returns** The ID Generator for this tracer context.

bool **ForceFlush**(std::chrono::microseconds timeout = (std::chrono::microseconds::max)()) noexcept  
Force all active SpanProcessors to flush any buffered spans within the given timeout.

bool **Shutdown**() noexcept  
Shutdown the span processor associated with this tracer provider.

## Class TracerProvider

- Defined in file \_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_v1.1.1\_sdk\_include\_opentelemetry\_sdk\_trace\_tracer\_provider.h

## Inheritance Relationships

### Base Type

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

## Class Documentation

```
class opentelemetry::sdk::trace::TracerProvider : public opentelemetry::trace::TracerProvider
```

## Public Functions

```
explicit TracerProvider(std::unique_ptr<SpanProcessor> processor,
                        opentelemetry::sdk::resource::Resource resource =
                        opentelemetry::sdk::resource::Resource::Create({}), std::unique_ptr<Sampler>
                        sampler = std::unique_ptr<AlwaysOnSampler>(new AlwaysOnSampler),
                        std::unique_ptr<opentelemetry::sdk::trace::IdGenerator> id_generator =
                        std::unique_ptr<opentelemetry::sdk::trace::IdGenerator>(new
                        RandomIdGenerator())) noexcept
```

Initialize a new tracer provider with a specified sampler

### Parameters

- processor** – The span processor for this tracer provider. This must not be a nullptr.
- resource** – The resources for this tracer provider.
- sampler** – The sampler for this tracer provider. This must not be a nullptr.
- id\_generator** – The custom id generator for this tracer provider. This must not be a nullptr

```
explicit TracerProvider(std::vector<std::unique_ptr<SpanProcessor>> &&processors,  
    opentelemetry::sdk::resource::Resource resource =  
    opentelemetry::sdk::resource::Resource::Create({}), std::unique_ptr<Sampler>  
    sampler = std::unique_ptr<AlwaysOnSampler>(new AlwaysOnSampler),  
    std::unique_ptr<opentelemetry::sdk::trace::IdGenerator> id_generator =  
    std::unique_ptr<opentelemetry::sdk::trace::IdGenerator>(new  
    RandomIdGenerator())) noexcept
```

explicit **TracerProvider**(std::shared\_ptr<sdk::trace::*TracerContext*> context) noexcept  
Initialize a new tracer provider with a specified context

**Parameters context** – The shared tracer configuration/pipeline for this provider.

```
opentelemetry::nostd::shared_ptr<opentelemetry::trace::Tracer> GetTracer(nostd::string_view  
    library_name,  
    nostd::string_view  
    library_version = "",  
    nostd::string_view schema_url =  
    "") noexcept override
```

```
void AddProcessor(std::unique_ptr<SpanProcessor> processor) noexcept
```

Attaches a span processor to list of configured processors for this tracer provider.

Note: This process may not receive any in-flight spans, but will get newly created spans. Note: This method is not thread safe, and should ideally be called from main thread.

**Parameters processor** – The new span processor for this tracer provider. This must not be a nullptr.

```
const opentelemetry::sdk::resource::Resource &GetResource() const noexcept  
Obtain the resource associated with this tracer provider.
```

**Returns** The resource for this tracer provider.

```
std::shared_ptr<SpanProcessor> GetProcessor() const noexcept  
Obtain the span processor associated with this tracer provider.
```

**Returns** The span processor for this tracer provider.

```
bool Shutdown() noexcept  
Shutdown the span processor associated with this tracer provider.
```

```
bool ForceFlush(std::chrono::microseconds timeout = (std::chrono::microseconds::max)()) noexcept  
Force flush the span processor associated with this tracer provider.
```



## Class DefaultSpan

- Defined in file `_home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_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  
*DefaultSpan* provides a non-operational *Span* that propagates the tracer context by wrapping it inside the *Span* object.

### Public Functions

inline virtual trace::SpanContext GetContext() const noexcept

inline virtual bool IsRecording() const noexcept

inline virtual void SetAttribute(nostd::string\_view, const common::AttributeValue&) noexcept

inline virtual void AddEvent(nostd::string\_view) noexcept

inline virtual void AddEvent(nostd::string\_view, common::SystemTimestamp) noexcept

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

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

inline virtual void SetStatus(StatusCode, nostd::string\_view) noexcept

inline virtual void UpdateName(nostd::string\_view) noexcept

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

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 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 NoopSpan

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_v1.1.1\_api\_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 explicit NoopSpan(const std::shared_ptr<Tracer> &tracer, nostd::unique_ptr<SpanContext>
    span_context) noexcept
```

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

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

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

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

inline virtual void **SetStatus**(*StatusCode*, nostd::string\_view) noexcept override

inline virtual void **UpdateName**(nostd::string\_view) noexcept override

inline virtual void **End**(const *EndSpanOptions*&) noexcept override

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

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

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

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

## Class NoopTracer

- Defined in file `__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_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 common::KeyValueIterable&, const *SpanContextKeyValueIterable*&, const *StartSpanOptions*&) noexcept override

Starts a span.

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

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

inline virtual void **ForceFlushWithMicroseconds**(uint64\_t) noexcept override

inline virtual void **CloseWithMicroseconds**(uint64\_t) noexcept override

### Class NoopTracerProvider

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_v1.1.1\_api\_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,  
nostd::string\_view schema\_url)  
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\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_v1.1.1\_api\_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\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_v1.1.1\_api\_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**(opentelemetry::context::propagation::TextMapCarrier &carrier, const context::Context &context) noexcept override

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

## Class B3PropagatorExtractor

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_v1.1.1\_api\_include\_opentelemetry\_trace\_propagation\_b3\_propagator.h

## Inheritance Relationships

### Base Type

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

### Derived Types

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

## Class Documentation

```
class opentelemetry::trace::propagation::B3PropagatorExtractor : public
opentelemetry::context::propagation::TextMapPropagator
    Subclassed by opentelemetry::trace::propagation::B3Propagator, openteleme-
    try::trace::propagation::B3PropagatorMultiHeader
```

### Public Functions

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

### Public Static Functions

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

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

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

## Class B3PropagatorMultiHeader

- Defined in file `__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_trace_propagation_b3_propagator.h`

## Inheritance Relationships

### Base Type

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

## Class Documentation

```
class opentelemetry::trace::propagation::B3PropagatorMultiHeader : public
opentelemetry::trace::propagation::B3PropagatorExtractor
```

### Public Functions

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

```
inline bool Fields(nostd::function_ref<bool(nostd::string_view)> callback) const noexcept override
```

## Class HttpTraceContext

- Defined in file `__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_trace_propagation_http_trace_context.h`

## Inheritance Relationships

### Base Type

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

## Class Documentation

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

## Public Functions

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

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

## Public Static Functions

static inline TraceId **TraceIdFromHex**(nstd::string\_view trace\_id)

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

static inline TraceFlags **TraceFlagsFromHex**(nstd::string\_view trace\_flags)

## Class JaegerPropagator

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_v1.1.1\_api\_include\_opentelemetry\_trace\_propagation\_jaeger.h

## Inheritance Relationships

### Base Type

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

## Class Documentation

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

## Public Functions

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

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

inline bool **Fields**(nstd::function\_ref<bool(nstd::string\_view)> callback) const noexcept override



## Class Provider

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_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__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_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__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_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 trace::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 `__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_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 opentelemetry::trace::TraceFlags &trace_flags() const noexcept
```

```
inline const opentelemetry::trace::TraceId &trace_id() const noexcept
```

```
inline const opentelemetry::trace::SpanId &span_id() const noexcept
```

```
inline const nostd::shared_ptr<opentelemetry::trace::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\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_v1.1.1\_api\_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 `__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_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 `__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_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__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_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__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_trace_tracer.h`

## Inheritance Relationships

### Derived Types

- `public opentelemetry::sdk::trace::Tracer` (*Class Tracer*)
- `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::sdk::trace::Tracer`*, *`opentelemetry::trace::NoopTracer`*

#### Public Functions

virtual `~Tracer()` = default

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

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

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

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

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

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



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

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

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

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

**Parameters** `timeout` – to complete the flush

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

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

**Parameters** `timeout` – to complete the flush

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

## Public Static Functions

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

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

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

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

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

## Class TracerProvider

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_v1.1.1\_api\_include\_opentelemetry\_trace\_tracer\_provider.h

## Inheritance Relationships

## Derived Types

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

## Class Documentation

class opentelemetry::trace::TracerProvider

Creates new *Tracer* instances.

Subclassed by *opentelemetry::sdk::trace::TracerProvider*, *opentelemetry::trace::NoopTracerProvider*

### Public Functions

virtual ~TracerProvider() = default

virtual nostd::shared\_ptr<Tracer> GetTracer(nostd::string\_view library\_name, nostd::string\_view library\_version = "", nostd::string\_view schema\_url = "") = 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\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_v1.1.1\_api\_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 = ','
```

## 3.1.3 Enums

### Enum Decision

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_sdk_include_opentelemetry_sdk_trace_sampler.h`

### Enum Documentation

enum `opentelemetry::sdk::trace::Decision`

A sampling Decision for a Span to be created.

*Values:*

enumerator **DROP**

enumerator **RECORD\_ONLY**

enumerator **RECORD\_AND\_SAMPLE**

### Enum CanonicalCode

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_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__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_trace_span_metadata.h`

### Enum Documentation

enum `opentelemetry::trace::SpanKind`

*Values:*

enumerator **kInternal**

enumerator **kServer**

enumerator **kClient**

enumerator **kProducer**

enumerator **kConsumer**

## Enum StatusCode

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_trace_span_metadata.h`

## Enum Documentation

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

enumerator **kUnset**

enumerator **kOk**

enumerator **kError**

## 3.1.4 Functions

### Function opentelemetry::baggage::GetBaggage

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_baggage_baggage_context.h`

## Function Documentation

```
inline nostd::shared_ptr<opentelemetry::baggage::Baggage> opentelemetry::baggage::GetBaggage(const
                                                                    open-
                                                                    teleme-
                                                                    try::context::Context
                                                                    &con-
                                                                    text)
```

### Function opentelemetry::baggage::SetBaggage

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_baggage_baggage_context.h`

## Function Documentation

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

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

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_context_runtime_context.h`

### Function Documentation

static *RuntimeContextStorage* \*`opentelemetry::context::GetDefaultStorage()` noexcept  
Construct and return the default *RuntimeContextStorage*

**Returns** a ThreadLocalContextStorage

## Function `opentelemetry::sdk::resource::attr`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_sdk_include_opentelemetry_sdk_resource_experimental_semantic_conventions.h`

### Function Documentation

inline const char \*`opentelemetry::sdk::resource::attr`(uint32\_t attr)

## Function `opentelemetry::trace::attr`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_trace_experimental_semantic_conventions.h`

### Function Documentation

inline const char \*`opentelemetry::trace::attr`(uint32\_t attr)

## Function `opentelemetry::trace::GetSpan`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_trace_context.h`

### Function Documentation

inline nostd::shared\_ptr<*Span*> `opentelemetry::trace::GetSpan`(const opentelemetry::context::Context &context)



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

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_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__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_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__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_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__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_trace_propagation_detail_string.h`

#### Function Documentation

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

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

### Function `opentelemetry::trace::SetSpan`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_trace_context.h`

### Function Documentation

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

## 3.1.5 Variables

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

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_baggage_baggage_context.h`

### Variable Documentation

static const `std::string` `opentelemetry::baggage::kBaggageHeader` = "baggage"

### Variable `opentelemetry::sdk::resource::attribute_ids`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_sdk_include_opentelemetry_sdk_resource_experimental_semantic_conventions.h`

### Variable Documentation

static const `std::unordered_map<uint32_t, const char*>` `opentelemetry::sdk::resource::attribute_ids`

### Variable `opentelemetry::trace::attribute_id`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_trace_experimental_semantic_conventions.h`

### Variable Documentation

`uint32_t` `opentelemetry::trace::attribute_id`



### Variable `opentelemetry::trace::propagation::kB3CombinedHeader`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_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__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_trace_propagation_b3_propagator.h`

### Variable Documentation

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

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

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_trace_propagation_b3_propagator.h`

### Variable Documentation

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

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

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_trace_propagation_b3_propagator.h`

### Variable Documentation

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

### Variable `opentelemetry::trace::propagation::kJaegerTraceHeader`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_trace_propagation_jaeger.h`

#### Variable Documentation

```
static const std::string_view opentelemetry::trace::propagation::kJaegerTraceHeader =  
"uber-trace-id"
```

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

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_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__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_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__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_trace_propagation_http_trace_context.h`

#### Variable Documentation

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

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

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_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__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_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__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_trace_propagation_http_trace_context.h`

### Variable Documentation

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

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

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_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__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_trace_propagation_http_trace_context.h`

### Variable Documentation

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

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

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_trace_propagation_http_trace_context.h`

### Variable Documentation

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

## 3.1.6 Defines

### Define `HAVE_WORKING_REGEX`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_trace_trace_state.h`

### Define Documentation

`HAVE_WORKING_REGEX`

### Define `OTEL_CPP_TRACE_ATTRIBUTES_MAX`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_trace_experimental_semantic_conventions.h`

### Define Documentation

`OTEL_CPP_TRACE_ATTRIBUTES_MAX`

### Define OTEL\_GET\_RESOURCE\_ATTR

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_sdk_include_opentelemetry_sdk_resource_experimental_semantic_conventions.h`

### Define Documentation

`OTEL_GET_RESOURCE_ATTR(name)`

### Define OTEL\_GET\_TRACE\_ATTR

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_trace_experimental_semantic_conventions.h`

### Define Documentation

`OTEL_GET_TRACE_ATTR(name)`

## 3.1.7 Typedefs

### Typedef opentelemetry::common::AttributeValue

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_common_attribute_value.h`

### Typedef Documentation

using `opentelemetry::common::AttributeValue` = `nostd::variant<bool, int32_t, int64_t, uint32_t, double, const char*, nostd::string_view, nostd::span<const bool>, nostd::span<const int32_t>, nostd::span<const int64_t>, nostd::span<const uint32_t>, nostd::span<const double>, nostd::span<const nostd::string_view>, uint64_t, nostd::span<const uint64_t>, nostd::span<const uint8_t>>`

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

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

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

#### Warning:

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

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



- `nostd::span<uint8_t>`

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

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

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_api_include_opentelemetry_context_context_value.h`

### Typedef Documentation

```
using opentelemetry::context::ContextValue = nostd::variant<nostd::monostate, bool, int64_t, uint64_t,
double, nostd::shared_ptr<trace::Span>, nostd::shared_ptr<trace::SpanContext>,
nostd::shared_ptr<baggage::Baggage>>
```

### Typedef `opentelemetry::sdk::resource::ResourceAttributes`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_v1.1.1_sdk_include_opentelemetry_sdk_resource_resource.h`

### Typedef Documentation

```
using opentelemetry::sdk::resource::ResourceAttributes = opentelemetry::sdk::common::AttributeMap
```



## 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*), 91

### O

opentelemetry::baggage::Baggage (*C++ class*), 23

opentelemetry::baggage::Baggage::Baggage  
(*C++ function*), 23

opentelemetry::baggage::Baggage::Delete  
(*C++ function*), 23

opentelemetry::baggage::Baggage::FromHeader  
(*C++ function*), 23

opentelemetry::baggage::Baggage::GetAllEntries  
(*C++ function*), 23

opentelemetry::baggage::Baggage::GetDefault  
(*C++ function*), 23

opentelemetry::baggage::Baggage::GetValue  
(*C++ function*), 23

opentelemetry::baggage::Baggage::kKeyValueSeparator  
(*C++ member*), 24

opentelemetry::baggage::Baggage::kMaxKeyValuePairs  
(*C++ member*), 24

opentelemetry::baggage::Baggage::kMaxKeyValueSize  
(*C++ member*), 24

opentelemetry::baggage::Baggage::kMaxSize  
(*C++ member*), 24

opentelemetry::baggage::Baggage::kMembersSeparator  
(*C++ member*), 24

opentelemetry::baggage::Baggage::kMetadataSeparator  
(*C++ member*), 24

opentelemetry::baggage::Baggage::Set (*C++  
function*), 23

opentelemetry::baggage::Baggage::ToHeader  
(*C++ function*), 23

opentelemetry::baggage::GetBaggage (*C++ func-  
tion*), 83

opentelemetry::baggage::kBaggageHeader (*C++  
member*), 86

opentelemetry::baggage::propagation::BaggagePropagator  
(*C++ class*), 24

opentelemetry::baggage::propagation::BaggagePropagator::Extract  
(*C++ function*), 24

opentelemetry::baggage::propagation::BaggagePropagator::Fi  
(*C++ function*), 24

opentelemetry::baggage::propagation::BaggagePropagator::Ir  
(*C++ function*), 24

opentelemetry::baggage::SetBaggage (*C++ func-  
tion*), 83

opentelemetry::common::AttributeValue (*C++  
type*), 92

opentelemetry::common::KeyValueIterable  
(*C++ class*), 25

opentelemetry::common::KeyValueIterable::~~KeyValueIterable  
(*C++ function*), 25

opentelemetry::common::KeyValueIterable::ForEachKeyValue  
(*C++ function*), 25

opentelemetry::common::KeyValueIterable::size  
(*C++ function*), 25

opentelemetry::common::SteadyTimestamp (*C++  
class*), 25

opentelemetry::common::SteadyTimestamp::operator  
std::chrono::steady\_clock::time\_point  
(*C++ function*), 26

opentelemetry::common::SteadyTimestamp::operator!=  
(*C++ function*), 26

opentelemetry::common::SteadyTimestamp::operator==  
(*C++ function*), 26

opentelemetry::common::SteadyTimestamp::SteadyTimestamp  
(*C++ function*), 25, 26

opentelemetry::common::SteadyTimestamp::time\_since\_epoch  
(*C++ function*), 26

opentelemetry::common::SystemTimestamp (*C++  
class*), 26

opentelemetry::common::SystemTimestamp::operator  
std::chrono::system\_clock::time\_point  
(*C++ function*), 26

opentelemetry::common::SystemTimestamp::operator!=  
(*C++ function*), 27

opentelemetry::common::SystemTimestamp::operator==  
(*C++ function*), 27

opentelemetry::common::SystemTimestamp::SystemTimestamp  
(*C++ function*), 26

opentelemetry::common::SystemTimestamp::time\_since\_epoch  
(*C++ function*), 27

opentelemetry::context::Context (C++ class), 27	(C++ function), 31
opentelemetry::context::Context::Context (C++ function), 27	opentelemetry::context::propagation::TextMapPropagator::Extract (C++ function), 31
opentelemetry::context::Context::GetValue (C++ function), 27	opentelemetry::context::propagation::TextMapPropagator::Fields (C++ function), 31
opentelemetry::context::Context::HasKey (C++ function), 27	opentelemetry::context::propagation::TextMapPropagator::Inject (C++ function), 31
opentelemetry::context::Context::operator== (C++ function), 27	opentelemetry::context::RuntimeContext (C++ class), 31
opentelemetry::context::Context::SetValue (C++ function), 27	opentelemetry::context::RuntimeContext::Attach (C++ function), 31
opentelemetry::context::Context::SetValues (C++ function), 27	opentelemetry::context::RuntimeContext::Detach (C++ function), 31
opentelemetry::context::ContextValue (C++ type), 93	opentelemetry::context::RuntimeContext::GetCurrent (C++ function), 31
opentelemetry::context::GetDefaultStorage (C++ function), 84	opentelemetry::context::RuntimeContext::GetValue (C++ function), 31
opentelemetry::context::propagation::CompositePropagator (C++ class), 28	opentelemetry::context::RuntimeContext::SetRuntimeContextStorage (C++ function), 31
opentelemetry::context::propagation::CompositePropagator::CompositePropagator (C++ function), 28	opentelemetry::context::RuntimeContext::SetValue (C++ function), 31
opentelemetry::context::propagation::CompositePropagator::Extract (C++ function), 28	opentelemetry::context::RuntimeContextStorage (C++ class), 32
opentelemetry::context::propagation::CompositePropagator::Fields (C++ function), 28	opentelemetry::context::RuntimeContextStorage::~RuntimeContextStorage (C++ function), 32
opentelemetry::context::propagation::CompositePropagator::Inject (C++ function), 28	opentelemetry::context::RuntimeContextStorage::Attach (C++ function), 32
opentelemetry::context::propagation::GlobalTextMapPropagator (C++ class), 29	opentelemetry::context::RuntimeContextStorage::CreateToken (C++ function), 33
opentelemetry::context::propagation::GlobalTextMapPropagator::GetGlobalPropagator (C++ function), 29	opentelemetry::context::RuntimeContextStorage::Detach (C++ function), 32
opentelemetry::context::propagation::GlobalTextMapPropagator::SetGlobalPropagator (C++ function), 29	opentelemetry::context::RuntimeContextStorage::GetCurrent (C++ function), 32
opentelemetry::context::propagation::NoOpPropagator (C++ class), 29	opentelemetry::context::ThreadLocalContextStorage (C++ class), 33
opentelemetry::context::propagation::NoOpPropagator::Extract (C++ function), 29	opentelemetry::context::ThreadLocalContextStorage::Attach (C++ function), 33
opentelemetry::context::propagation::NoOpPropagator::Fields (C++ function), 29	opentelemetry::context::ThreadLocalContextStorage::Detach (C++ function), 33
opentelemetry::context::propagation::NoOpPropagator::Inject (C++ function), 29	opentelemetry::context::ThreadLocalContextStorage::GetCurrent (C++ function), 33
opentelemetry::context::propagation::TextMapCarrier (C++ class), 30	opentelemetry::context::ThreadLocalContextStorage::ThreadLocalContextStorage (C++ function), 33
opentelemetry::context::propagation::TextMapCarrier::Token (C++ class), 34	opentelemetry::context::Token (C++ class), 34
opentelemetry::context::propagation::TextMapCarrier::~Token (C++ function), 34	opentelemetry::context::Token::~Token (C++ function), 34
opentelemetry::context::propagation::TextMapCarrier::Keys (C++ function), 30	opentelemetry::context::Token::operator== (C++ function), 34
opentelemetry::context::propagation::TextMapCarrier::Set (C++ class), 34	opentelemetry::sdk::instrumentationlibrary::InstrumentationLibrary (C++ class), 34
opentelemetry::context::propagation::TextMapCarrier::SetValues (C++ function), 30	opentelemetry::sdk::instrumentationlibrary::InstrumentationLibrary::Attach (C++ function), 35
opentelemetry::context::propagation::TextMapPropagator (C++ class), 31	opentelemetry::sdk::instrumentationlibrary::InstrumentationLibrary::Detach (C++ function), 35
opentelemetry::context::propagation::TextMapPropagator (C++ TextMapPropagator	



```

opentelemetry::sdk::trace::MultiRecordable::SetIdentity opentelemetry::sdk::trace::RandomIdGenerator::GenerateSpan
(C++ function), 41 (C++ function), 45
opentelemetry::sdk::trace::MultiRecordable::SetInstrumentationLibrary opentelemetry::sdk::trace::RandomIdGenerator::GenerateTrace
(C++ function), 42 (C++ function), 45
opentelemetry::sdk::trace::MultiRecordable::SetName opentelemetry::sdk::trace::Recordable (C++
(C++ function), 41 class), 45
opentelemetry::sdk::trace::MultiRecordable::SetResource opentelemetry::sdk::trace::Recordable::~~Recordable
(C++ function), 41 (C++ function), 45
opentelemetry::sdk::trace::MultiRecordable::SetSpanKind opentelemetry::sdk::trace::Recordable::AddEvent
(C++ function), 41 (C++ function), 45, 46
opentelemetry::sdk::trace::MultiRecordable::SetStartTime opentelemetry::sdk::trace::Recordable::AddLink
(C++ function), 42 (C++ function), 46
opentelemetry::sdk::trace::MultiRecordable::SetStatus opentelemetry::sdk::trace::Recordable::SetAttribute
(C++ function), 41 (C++ function), 45
opentelemetry::sdk::trace::MultiSpanProcessor opentelemetry::sdk::trace::Recordable::SetDuration
(C++ class), 42 (C++ function), 47
opentelemetry::sdk::trace::MultiSpanProcessor::opentelemetry::sdk::trace::Recordable::SetIdentity
(C++ function), 43 (C++ function), 45
opentelemetry::sdk::trace::MultiSpanProcessor::AddProcessor opentelemetry::sdk::trace::Recordable::SetInstrumentationLibrary
(C++ function), 42 (C++ function), 47
opentelemetry::sdk::trace::MultiSpanProcessor::ForceFlush opentelemetry::sdk::trace::Recordable::SetName
(C++ function), 43 (C++ function), 46
opentelemetry::sdk::trace::MultiSpanProcessor::MakeRecordable opentelemetry::sdk::trace::Recordable::SetResource
(C++ function), 42 (C++ function), 46
opentelemetry::sdk::trace::MultiSpanProcessor::MultiSpanProcessor opentelemetry::sdk::trace::Recordable::SetSpanKind
(C++ function), 42 (C++ function), 46
opentelemetry::sdk::trace::MultiSpanProcessor::OnEnd opentelemetry::sdk::trace::Recordable::SetStartTime
(C++ function), 43 (C++ function), 46
opentelemetry::sdk::trace::MultiSpanProcessor::OnStart opentelemetry::sdk::trace::Recordable::SetStatus
(C++ function), 43 (C++ function), 46
opentelemetry::sdk::trace::MultiSpanProcessor::ProcessorNode opentelemetry::sdk::trace::Sampler (C++
(C++ struct), 21 class), 47
opentelemetry::sdk::trace::MultiSpanProcessor::ProcessorNode::next opentelemetry::sdk::trace::Sampler::~~Sampler
(C++ member), 21 (C++ function), 47
opentelemetry::sdk::trace::MultiSpanProcessor::ProcessorNode::prev opentelemetry::sdk::trace::Sampler::GetDescription
(C++ member), 21 (C++ function), 48
opentelemetry::sdk::trace::MultiSpanProcessor::ProcessorNode::ProcessorNode opentelemetry::sdk::trace::Sampler::ShouldSample
(C++ function), 21 (C++ function), 47
opentelemetry::sdk::trace::MultiSpanProcessor::ProcessorNode::value opentelemetry::sdk::trace::SamplingResult
(C++ member), 21 (C++ struct), 21
opentelemetry::sdk::trace::MultiSpanProcessor::Sampler opentelemetry::sdk::trace::SamplingResult::attributes
(C++ function), 43 (C++ member), 22
opentelemetry::sdk::trace::MultiSpanProcessor::Options opentelemetry::sdk::trace::SamplingResult::decision
(C++ struct), 21 (C++ member), 22
opentelemetry::sdk::trace::ParentBasedSampler opentelemetry::sdk::trace::SamplingResult::trace_state
(C++ class), 44 (C++ member), 22
opentelemetry::sdk::trace::ParentBasedSampler::GetDescription opentelemetry::sdk::trace::SimpleSpanProcessor
(C++ function), 44 (C++ class), 48
opentelemetry::sdk::trace::ParentBasedSampler::ParentBasedSampler opentelemetry::sdk::trace::SimpleSpanProcessor::~~SimpleSpanProcessor
(C++ function), 44 (C++ function), 49
opentelemetry::sdk::trace::ParentBasedSampler::ShouldSample opentelemetry::sdk::trace::SimpleSpanProcessor::ForceFlush
(C++ function), 44 (C++ function), 49
opentelemetry::sdk::trace::RandomIdGenerator opentelemetry::sdk::trace::SimpleSpanProcessor::MakeRecordable
(C++ class), 44 (C++ function), 49

```



opentelemetry::sdk::trace::SimpleSpanProcessor opentelemetry::sdk::trace::SpanData::SetResource  
 (C++ function), 49 (C++ function), 52  
 opentelemetry::sdk::trace::SimpleSpanProcessor opentelemetry::sdk::trace::SpanData::SetSpanKind  
 (C++ function), 49 (C++ function), 52  
 opentelemetry::sdk::trace::SimpleSpanProcessor opentelemetry::sdk::trace::SpanData::SetStartTime  
 (C++ function), 49 (C++ function), 52  
 opentelemetry::sdk::trace::SimpleSpanProcessor opentelemetry::sdk::trace::SpanData::SetStatus  
 (C++ function), 49 (C++ function), 52  
 opentelemetry::sdk::trace::SpanData (C++ class), 50 opentelemetry::sdk::trace::SpanData  
 (C++ function), 50  
 opentelemetry::sdk::trace::SpanData::AddEvent opentelemetry::sdk::trace::SpanDataEvent  
 (C++ function), 51 (C++ class), 53  
 opentelemetry::sdk::trace::SpanData::AddLink opentelemetry::sdk::trace::SpanDataEvent::GetAttributes  
 (C++ function), 52 (C++ function), 53  
 opentelemetry::sdk::trace::SpanData::GetAttributes opentelemetry::sdk::trace::SpanDataEvent::GetName  
 (C++ function), 51 (C++ function), 53  
 opentelemetry::sdk::trace::SpanData::GetDescription opentelemetry::sdk::trace::SpanDataEvent::GetTimestamp  
 (C++ function), 50 (C++ function), 53  
 opentelemetry::sdk::trace::SpanData::GetDuration opentelemetry::sdk::trace::SpanDataEvent::SpanDataEvent  
 (C++ function), 51 (C++ function), 53  
 opentelemetry::sdk::trace::SpanData::GetEvents opentelemetry::sdk::trace::SpanDataLink  
 (C++ function), 51 (C++ class), 53  
 opentelemetry::sdk::trace::SpanData::GetInstrumentationLibrary opentelemetry::sdk::trace::SpanDataLink::GetAttributes  
 (C++ function), 50 (C++ function), 53  
 opentelemetry::sdk::trace::SpanData::GetLinks opentelemetry::sdk::trace::SpanDataLink::GetSpanContext  
 (C++ function), 51 (C++ function), 53  
 opentelemetry::sdk::trace::SpanData::GetName opentelemetry::sdk::trace::SpanDataLink::SpanDataLink  
 (C++ function), 50 (C++ function), 53  
 opentelemetry::sdk::trace::SpanData::GetParentSpanId opentelemetry::sdk::trace::SpanExporter  
 (C++ function), 50 (C++ class), 54  
 opentelemetry::sdk::trace::SpanData::GetResource opentelemetry::sdk::trace::SpanExporter::~SpanExporter  
 (C++ function), 50 (C++ function), 54  
 opentelemetry::sdk::trace::SpanData::GetSpanContext opentelemetry::sdk::trace::SpanExporter::Export  
 (C++ function), 50 (C++ function), 54  
 opentelemetry::sdk::trace::SpanData::GetSpanId opentelemetry::sdk::trace::SpanExporter::MakeRecordable  
 (C++ function), 50 (C++ function), 54  
 opentelemetry::sdk::trace::SpanData::GetSpanKind opentelemetry::sdk::trace::SpanExporter::Shutdown  
 (C++ function), 50 (C++ function), 54  
 opentelemetry::sdk::trace::SpanData::GetStartTime opentelemetry::sdk::trace::SpanProcessor  
 (C++ function), 51 (C++ class), 55  
 opentelemetry::sdk::trace::SpanData::GetStatus opentelemetry::sdk::trace::SpanProcessor::~SpanProcessor  
 (C++ function), 50 (C++ function), 55  
 opentelemetry::sdk::trace::SpanData::GetTraceId opentelemetry::sdk::trace::SpanProcessor::ForceFlush  
 (C++ function), 50 (C++ function), 55  
 opentelemetry::sdk::trace::SpanData::SetAttributes opentelemetry::sdk::trace::SpanProcessor::MakeRecordable  
 (C++ function), 51 (C++ function), 55  
 opentelemetry::sdk::trace::SpanData::SetDuration opentelemetry::sdk::trace::SpanProcessor::OnEnd  
 (C++ function), 52 (C++ function), 55  
 opentelemetry::sdk::trace::SpanData::SetIdentity opentelemetry::sdk::trace::SpanProcessor::OnStart  
 (C++ function), 51 (C++ function), 55  
 opentelemetry::sdk::trace::SpanData::SetInstrumentationLibrary opentelemetry::sdk::trace::SpanProcessor::Shutdown  
 (C++ function), 52 (C++ function), 56  
 opentelemetry::sdk::trace::SpanData::SetName opentelemetry::sdk::trace::TraceIdRatioBasedSampler  
 (C++ function), 52 (C++ class), 56

opentelemetry::sdk::trace::TraceIdRatioBasedSampler::GetDescription (C++ function), 56	opentelemetry::sdk::trace::TracerProvider::GetTracer (C++ function), 60
opentelemetry::sdk::trace::TraceIdRatioBasedSampler::ShouldSample (C++ function), 56	opentelemetry::sdk::trace::TracerProvider::Shutdown (C++ function), 60
opentelemetry::sdk::trace::TraceIdRatioBasedSampler::TraceIdRatioBasedSampler (C++ function), 56	opentelemetry::sdk::trace::TracerProvider::TracerProvider (C++ function), 59, 60
opentelemetry::sdk::trace::Tracer (C++ class), 57	opentelemetry::trace::attr (C++ function), 84
opentelemetry::sdk::trace::Tracer::CloseWithMicroseconds (C++ function), 57	opentelemetry::trace::attribute_id (C++ mem- ber), 86
opentelemetry::sdk::trace::Tracer::ForceFlushWithMicroseconds (C++ function), 57	opentelemetry::trace::attribute_ids (C++ member), 87
opentelemetry::sdk::trace::Tracer::GetIdGenerator (C++ function), 57	opentelemetry::trace::attribute_key (C++ member), 87
opentelemetry::sdk::trace::Tracer::GetInstrumentationLibrary (C++ function), 57	opentelemetry::trace::CanonicalCode (C++ enumerator), 80
opentelemetry::sdk::trace::Tracer::GetProcessor (C++ function), 57	opentelemetry::trace::CanonicalCode::ABORTED (C++ enumerator), 81
opentelemetry::sdk::trace::Tracer::GetResource (C++ function), 57	opentelemetry::trace::CanonicalCode::ALREADY_EXISTS (C++ enumerator), 81
opentelemetry::sdk::trace::Tracer::GetSampler (C++ function), 57	opentelemetry::trace::CanonicalCode::CANCELLED (C++ enumerator), 80
opentelemetry::sdk::trace::Tracer::StartSpan (C++ function), 57	opentelemetry::trace::CanonicalCode::DATA_LOSS (C++ enumerator), 82
opentelemetry::sdk::trace::Tracer::Tracer (C++ function), 57	opentelemetry::trace::CanonicalCode::DEADLINE_EXCEEDED (C++ enumerator), 81
opentelemetry::sdk::trace::TracerContext (C++ class), 58	opentelemetry::trace::CanonicalCode::FAILED_PRECONDITION (C++ enumerator), 81
opentelemetry::sdk::trace::TracerContext::AddProcessor (C++ function), 58	opentelemetry::trace::CanonicalCode::INTERNAL (C++ enumerator), 82
opentelemetry::sdk::trace::TracerContext::ForceFlush (C++ function), 59	opentelemetry::trace::CanonicalCode::INVALID_ARGUMENT (C++ enumerator), 81
opentelemetry::sdk::trace::TracerContext::GetIdGenerator (C++ function), 58	opentelemetry::trace::CanonicalCode::NOT_FOUND (C++ enumerator), 81
opentelemetry::sdk::trace::TracerContext::GetProcessor (C++ function), 58	opentelemetry::trace::CanonicalCode::OK (C++ enumerator), 80
opentelemetry::sdk::trace::TracerContext::GetResource (C++ function), 58	opentelemetry::trace::CanonicalCode::OUT_OF_RANGE (C++ enumerator), 81
opentelemetry::sdk::trace::TracerContext::GetSampler (C++ function), 58	opentelemetry::trace::CanonicalCode::PERMISSION_DENIED (C++ enumerator), 81
opentelemetry::sdk::trace::TracerContext::Shutdown (C++ function), 59	opentelemetry::trace::CanonicalCode::RESOURCE_EXHAUSTED (C++ enumerator), 81
opentelemetry::sdk::trace::TracerContext::TracerContext (C++ function), 58	opentelemetry::trace::CanonicalCode::UNAUTHENTICATED (C++ enumerator), 82
opentelemetry::sdk::trace::TracerProvider (C++ class), 59	opentelemetry::trace::CanonicalCode::UNAVAILABLE (C++ enumerator), 82
opentelemetry::sdk::trace::TracerProvider::AddProcessor (C++ function), 60	opentelemetry::trace::CanonicalCode::UNIMPLEMENTED (C++ enumerator), 82
opentelemetry::sdk::trace::TracerProvider::ForceFlush (C++ function), 60	opentelemetry::trace::CanonicalCode::UNKNOWN (C++ enumerator), 80
opentelemetry::sdk::trace::TracerProvider::GetProcessor (C++ function), 60	opentelemetry::trace::DefaultSpan (C++ class), 61
opentelemetry::sdk::trace::TracerProvider::GetResource (C++ function), 60	opentelemetry::trace::DefaultSpan::AddEvent (C++ function), 61
	opentelemetry::trace::DefaultSpan::DefaultSpan (C++ function), 60

(C++ function), 61, 62  
 opentelemetry::trace::DefaultSpan::End (C++ function), 61  
 opentelemetry::trace::DefaultSpan::GetContext (C++ function), 61  
 opentelemetry::trace::DefaultSpan::GetInvalid (C++ function), 62  
 opentelemetry::trace::DefaultSpan::IsRecording (C++ function), 61  
 opentelemetry::trace::DefaultSpan::SetAttribute (C++ function), 61  
 opentelemetry::trace::DefaultSpan::SetStatus (C++ function), 61  
 opentelemetry::trace::DefaultSpan::ToString (C++ function), 61  
 opentelemetry::trace::DefaultSpan::UpdateName (C++ function), 61  
 opentelemetry::trace::EndSpanOptions (C++ struct), 22  
 opentelemetry::trace::EndSpanOptions::end\_steadily (C++ member), 22  
 opentelemetry::trace::GetSpan (C++ function), 84  
 opentelemetry::trace::kSpanKey (C++ member), 87  
 opentelemetry::trace::NoopSpan (C++ class), 62  
 opentelemetry::trace::NoopSpan::AddEvent (C++ function), 62  
 opentelemetry::trace::NoopSpan::End (C++ function), 63  
 opentelemetry::trace::NoopSpan::GetContext (C++ function), 63  
 opentelemetry::trace::NoopSpan::IsRecording (C++ function), 63  
 opentelemetry::trace::NoopSpan::NoopSpan (C++ function), 62  
 opentelemetry::trace::NoopSpan::SetAttribute (C++ function), 62  
 opentelemetry::trace::NoopSpan::SetStatus (C++ function), 62  
 opentelemetry::trace::NoopSpan::UpdateName (C++ function), 63  
 opentelemetry::trace::NoopTracer (C++ class), 63  
 opentelemetry::trace::NoopTracer::CloseWithMicroseconds (C++ function), 63  
 opentelemetry::trace::NoopTracer::ForceFlushWithMicroseconds (C++ function), 63  
 opentelemetry::trace::NoopTracer::StartSpan (C++ function), 63  
 opentelemetry::trace::NoopTracerProvider (C++ class), 64  
 opentelemetry::trace::NoopTracerProvider::GetTracer (C++ function), 64  
 opentelemetry::trace::NoopTracerProvider::NoopTracerProvider (C++ function), 64  
 (C++ function), 64  
 opentelemetry::trace::NullSpanContext (C++ class), 65  
 opentelemetry::trace::NullSpanContext::ForEachKeyValue (C++ function), 65  
 opentelemetry::trace::NullSpanContext::size (C++ function), 65  
 opentelemetry::trace::propagation::B3Propagator (C++ class), 65  
 opentelemetry::trace::propagation::B3Propagator::Fields (C++ function), 65  
 opentelemetry::trace::propagation::B3Propagator::Inject (C++ function), 65  
 opentelemetry::trace::propagation::B3PropagatorExtractor (C++ class), 66  
 opentelemetry::trace::propagation::B3PropagatorExtractor::Fields (C++ function), 66  
 opentelemetry::trace::propagation::B3PropagatorExtractor::Inject (C++ function), 66  
 opentelemetry::trace::propagation::B3PropagatorExtractor::InjectAndGet (C++ function), 66  
 opentelemetry::trace::propagation::B3PropagatorMultiHeader (C++ class), 67  
 opentelemetry::trace::propagation::B3PropagatorMultiHeader::Inject (C++ function), 67  
 opentelemetry::trace::propagation::B3PropagatorMultiHeader::InjectAndGet (C++ function), 67  
 opentelemetry::trace::propagation::detail::HexToBinary (C++ function), 85  
 opentelemetry::trace::propagation::detail::HexToInt (C++ function), 85  
 opentelemetry::trace::propagation::detail::IsValidHex (C++ function), 85  
 opentelemetry::trace::propagation::detail::kHexDigits (C++ member), 87  
 opentelemetry::trace::propagation::detail::SplitString (C++ function), 85  
 opentelemetry::trace::propagation::HttpTraceContext (C++ class), 67  
 opentelemetry::trace::propagation::HttpTraceContext::Extract (C++ function), 68  
 opentelemetry::trace::propagation::HttpTraceContext::Inject (C++ function), 68  
 opentelemetry::trace::propagation::HttpTraceContext::SpanId (C++ function), 68  
 opentelemetry::trace::propagation::HttpTraceContext::TraceId (C++ function), 68  
 opentelemetry::trace::propagation::HttpTraceContext::TraceState (C++ function), 68  
 opentelemetry::trace::propagation::JaegerPropagator (C++ class), 68  
 opentelemetry::trace::propagation::JaegerPropagator::Extract (C++ function), 68

```

    (C++ function), 68
opentelemetry::trace::propagation::JaegerPropagatorFields::trace::Span::SetAttribute
    (C++ function), 68
opentelemetry::trace::propagation::JaegerPropagatorInject::trace::Span::SetStatus (C++
    (C++ function), 68
    function), 71
opentelemetry::trace::propagation::kB3CombinedHeader::trace::Span::Span (C++ func-
    (C++ member), 88
    tion), 70
opentelemetry::trace::propagation::kB3SampledHeader::trace::Span::UpdateName (C++
    (C++ member), 88
    function), 71
opentelemetry::trace::propagation::kB3SpanIdHeader::trace::SpanContext (C++ class),
    (C++ member), 88
    71
opentelemetry::trace::propagation::kB3TraceIdHeader::trace::SpanContext::GetInvalid
    (C++ member), 88
    (C++ function), 72
opentelemetry::trace::propagation::kJaegerTraceIdHeader::trace::SpanContext::IsRemote
    (C++ member), 89
    (C++ function), 72
opentelemetry::trace::propagation::kSpanIdHexSpanIdHeader::trace::SpanContext::IsSampled
    (C++ member), 89
    (C++ function), 72
opentelemetry::trace::propagation::kSpanIdSizeopentelemetry::trace::SpanContext::IsValid
    (C++ member), 89
    (C++ function), 71
opentelemetry::trace::propagation::kTraceFlagsSizeopentelemetry::trace::SpanContext::operator=
    (C++ member), 89
    (C++ function), 72
opentelemetry::trace::propagation::kTraceIdHexSpanIdHeader::trace::SpanContext::operator==
    (C++ member), 90
    (C++ function), 72
opentelemetry::trace::propagation::kTraceIdSizeopentelemetry::trace::SpanContext::span_id
    (C++ member), 90
    (C++ function), 72
opentelemetry::trace::propagation::kTraceParentopentelemetry::trace::SpanContext::SpanContext
    (C++ member), 90
    (C++ function), 71
opentelemetry::trace::propagation::kTraceParentSizeopentelemetry::trace::SpanContext::trace_flags
    (C++ member), 90
    (C++ function), 71
opentelemetry::trace::propagation::kTraceStateopentelemetry::trace::SpanContext::trace_id
    (C++ member), 91
    (C++ function), 71
opentelemetry::trace::propagation::kVersionSizeopentelemetry::trace::SpanContext::trace_state
    (C++ member), 91
    (C++ function), 72
opentelemetry::trace::Provider (C++ class), 69
opentelemetry::trace::Provider::GetTracerProvider (C++ class), 72
    (C++ function), 69
opentelemetry::trace::Provider::SetTracerProvider (C++ function), 73
    (C++ function), 69
opentelemetry::trace::Scope (C++ class), 69
opentelemetry::trace::Scope::Scope (C++ func-
    (C++ function), 69
    tion), 69
opentelemetry::trace::SetSpan (C++ function), 86
opentelemetry::trace::Span (C++ class), 70
opentelemetry::trace::Span::~~Span (C++ func-
    (C++ function), 70
    tion), 70
opentelemetry::trace::Span::AddEvent (C++
    (C++ function), 70, 71
    function), 70, 71
opentelemetry::trace::Span::End (C++ function),
    71
opentelemetry::trace::Span::GetContext (C++
    (C++ function), 71
    function), 71
opentelemetry::trace::Span::IsRecording
    (C++ function), 71
opentelemetry::trace::Span::operator= (C++
    (C++ function), 73
    function), 73
opentelemetry::trace::SpanContext::Fields::trace::Span::SetAttribute
    (C++ function), 70
opentelemetry::trace::SpanContext::Inject::trace::Span::SetStatus (C++
    (C++ function), 71
    function), 71
opentelemetry::trace::SpanContext::Span (C++ func-
    (C++ member), 88
    tion), 70
opentelemetry::trace::SpanContext::UpdateName (C++
    (C++ member), 88
    function), 71
opentelemetry::trace::SpanContext (C++ class),
    71
opentelemetry::trace::SpanContext::GetInvalid
    (C++ function), 72
opentelemetry::trace::SpanContext::IsRemote
    (C++ function), 72
opentelemetry::trace::SpanContext::IsSampled
    (C++ function), 72
opentelemetry::trace::SpanContext::IsValid
    (C++ function), 71
opentelemetry::trace::SpanContext::operator=
    (C++ function), 72
opentelemetry::trace::SpanContext::operator==
    (C++ function), 72
opentelemetry::trace::SpanContext::span_id
    (C++ function), 72
opentelemetry::trace::SpanContext::SpanContext
    (C++ function), 71
opentelemetry::trace::SpanContext::trace_flags
    (C++ function), 71
opentelemetry::trace::SpanContext::trace_id
    (C++ function), 71
opentelemetry::trace::SpanContext::trace_state
    (C++ function), 72
opentelemetry::trace::SpanContextKeyValueIterable
    (C++ class), 72
opentelemetry::trace::SpanContextKeyValueIterable::~~SpanCo
opentelemetry::trace::SpanContextKeyValueIterable::ForEach
    (C++ function), 73
    (C++ function), 73
opentelemetry::trace::SpanContextKeyValueIterable::size
    (C++ function), 73
opentelemetry::trace::SpanId (C++ class), 73
opentelemetry::trace::SpanId::CopyBytesTo
    (C++ function), 73
opentelemetry::trace::SpanId::Id (C++ func-
    (C++ function), 73
    tion), 73
opentelemetry::trace::SpanId::IsValid (C++
    (C++ function), 73
    function), 73
opentelemetry::trace::SpanId::kSize (C++
    (C++ member), 74
    member), 74
opentelemetry::trace::SpanId::operator!=
    (C++ function), 73
opentelemetry::trace::SpanId::operator==
    (C++ function), 73

```

<code>opentelemetry::trace::SpanId::SpanId</code> (C++ function), 73	<code>opentelemetry::trace::TraceId::Id</code> (C++ function), 75
<code>opentelemetry::trace::SpanId::ToLowerBase16</code> (C++ function), 73	<code>opentelemetry::trace::TraceId::IsValid</code> (C++ function), 75
<code>opentelemetry::trace::SpanKind</code> (C++ enum), 82	<code>opentelemetry::trace::TraceId::kSize</code> (C++ member), 75
<code>opentelemetry::trace::SpanKind::kClient</code> (C++ enumerator), 82	<code>opentelemetry::trace::TraceId::operator!=</code> (C++ function), 75
<code>opentelemetry::trace::SpanKind::kConsumer</code> (C++ enumerator), 82	<code>opentelemetry::trace::TraceId::operator==</code> (C++ function), 75
<code>opentelemetry::trace::SpanKind::kInternal</code> (C++ enumerator), 82	<code>opentelemetry::trace::TraceId::ToLowerBase16</code> (C++ function), 75
<code>opentelemetry::trace::SpanKind::kProducer</code> (C++ enumerator), 82	<code>opentelemetry::trace::TraceId::TraceId</code> (C++ function), 75
<code>opentelemetry::trace::SpanKind::kServer</code> (C++ enumerator), 82	<code>opentelemetry::trace::Tracer</code> (C++ class), 76
<code>opentelemetry::trace::StartSpanOptions</code> (C++ struct), 22	<code>opentelemetry::trace::Tracer::~Tracer</code> (C++ function), 76
<code>opentelemetry::trace::StartSpanOptions::kind</code> (C++ member), 22	<code>opentelemetry::trace::Tracer::Close</code> (C++ function), 77
<code>opentelemetry::trace::StartSpanOptions::parent</code> (C++ member), 22	<code>opentelemetry::trace::Tracer::CloseWithMicroseconds</code> (C++ function), 77
<code>opentelemetry::trace::StartSpanOptions::start_time</code> (C++ member), 22	<code>opentelemetry::trace::Tracer::ForceFlush</code> (C++ function), 77
<code>opentelemetry::trace::StartSpanOptions::start_time_system</code> (C++ member), 22	<code>opentelemetry::trace::Tracer::ForceFlushWithMicroseconds</code> (C++ function), 77
<code>opentelemetry::trace::StatusCode</code> (C++ enum), 83	<code>opentelemetry::trace::Tracer::GetCurrentSpan</code> (C++ function), 77
<code>opentelemetry::trace::StatusCode::kError</code> (C++ enumerator), 83	<code>opentelemetry::trace::Tracer::StartSpan</code> (C++ function), 76, 77
<code>opentelemetry::trace::StatusCode::kOk</code> (C++ enumerator), 83	<code>opentelemetry::trace::Tracer::WithActiveSpan</code> (C++ function), 77
<code>opentelemetry::trace::StatusCode::kUnset</code> (C++ enumerator), 83	<code>opentelemetry::trace::TracerProvider</code> (C++ class), 78
<code>opentelemetry::trace::TraceFlags</code> (C++ class), 74	<code>opentelemetry::trace::TracerProvider::~TracerProvider</code> (C++ function), 78
<code>opentelemetry::trace::TraceFlags::CopyBytesTo</code> (C++ function), 74	<code>opentelemetry::trace::TracerProvider::GetTracer</code> (C++ function), 78
<code>opentelemetry::trace::TraceFlags::flags</code> (C++ function), 74	<code>opentelemetry::trace::TraceState</code> (C++ class), 78
<code>opentelemetry::trace::TraceFlags::IsSampled</code> (C++ function), 74	<code>opentelemetry::trace::TraceState::Delete</code> (C++ function), 79
<code>opentelemetry::trace::TraceFlags::kIsSampled</code> (C++ member), 74	<code>opentelemetry::trace::TraceState::Empty</code> (C++ function), 79
<code>opentelemetry::trace::TraceFlags::operator!=</code> (C++ function), 74	<code>opentelemetry::trace::TraceState::FromHeader</code> (C++ function), 79
<code>opentelemetry::trace::TraceFlags::operator==</code> (C++ function), 74	<code>opentelemetry::trace::TraceState::Get</code> (C++ function), 79
<code>opentelemetry::trace::TraceFlags::ToLowerBase16</code> (C++ function), 74	<code>opentelemetry::trace::TraceState::GetAllEntries</code> (C++ function), 79
<code>opentelemetry::trace::TraceFlags::TraceFlags</code> (C++ function), 74	<code>opentelemetry::trace::TraceState::GetDefault</code> (C++ function), 79
<code>opentelemetry::trace::TraceId</code> (C++ class), 75	<code>opentelemetry::trace::TraceState::IsValidKey</code> (C++ function), 79
<code>opentelemetry::trace::TraceId::CopyBytesTo</code> (C++ function), 75	<code>opentelemetry::trace::TraceState::IsValidValue</code>

(C++ *function*), [79](#)  
opentelemetry::trace::TraceState::kKeyMaxSize  
(C++ *member*), [80](#)  
opentelemetry::trace::TraceState::kKeyValueSeparator  
(C++ *member*), [80](#)  
opentelemetry::trace::TraceState::kMaxKeyValuePairs  
(C++ *member*), [80](#)  
opentelemetry::trace::TraceState::kMembersSeparator  
(C++ *member*), [80](#)  
opentelemetry::trace::TraceState::kValueMaxSize  
(C++ *member*), [80](#)  
opentelemetry::trace::TraceState::Set (C++  
*function*), [79](#)  
opentelemetry::trace::TraceState::ToHeader  
(C++ *function*), [79](#)  
OTEL\_CPP\_TRACE\_ATTRIBUTES\_MAX (C *macro*), [91](#)  
OTEL\_GET\_RESOURCE\_ATTR (C *macro*), [92](#)  
OTEL\_GET\_TRACE\_ATTR (C *macro*), [92](#)