

---

# OpenTelemetry C++

*Release 1.8.2*

**OpenTelemetry authors**

**Feb 03, 2023**



# 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>Performance Tests - Benchmarks</b>	<b>187</b>
<b>5</b>	<b>Getting help</b>	<b>189</b>
	<b>Index</b>	<b>191</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 = 14268;
opts.headers = {}; // optional headers
auto jaeger_http_exporter =
    std::unique_ptr<sdktrace::SpanExporter>(new
↳opentelemetry::exporter::jaeger::JaegerExporter(opts));

// otlp 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));

// otlp 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
auto always_on_sampler = std::unique_ptr<sdktrace::AlwaysOnSampler>

```

(continues on next page)

(continued from previous page)

```

    (new sdktrace::AlwaysOnSampler);

//AlwaysOffSampler
auto always_off_sampler = std::unique_ptr<sdktrace::AlwaysOffSampler>
    (new sdktrace::AlwaysOffSampler);

//ParentBasedSampler
auto parent_based_sampler = std::unique_ptr<sdktrace::ParentBasedSampler>
    (new sdktrace::ParentBasedSampler);

//TraceIdRatioBasedSampler - Sample 50% generated spans
double ratio = 0.5;
auto always_off_sampler = std::unique_ptr<sdktrace::TraceIdRatioBasedSampler>
    (new sdktrace::TraceIdRatioBasedSampler(ratio));

```

## 2.6 TracerContext

SDK configuration are shared between *TracerProvider* and all its *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 = nostd::shared_ptr<sdktrace::TracerProvider>
    (new sdktrace::TracerProvider(tracer_context));

// Create using SDK configurations as parameter
auto tracer_provider = nostd::shared_ptr<sdktrace::TracerProvider>
    (std::move(simple_processor), resource, std::move(always_on_sampler));

// set the global tracer TracerProvider
opentelemetry::trace::Provider::SetTracerProvider(tracer_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);
opentelemetry::sdk::common::AttributeMap error_attributes = {
    {"url", url}, {"content-length", len}, {"content-type", type}};
OTEL_INTERNAL_LOG_ERROR(" Connection failed." , error_attributes);
opentelemetry::sdk::common::AttributeMap http_attributes = {
    {"url", url}, {"content-length", len}, {"content-type", type}};
OTEL_INTERNAL_LOG_DEBUG(" Connection Established Successfully. Headers:", http_
↳attributes);
```

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

```
class CustomLogHandler : public opentelemetry::sdk::common::internal_log::LogHandler
{
    void Handle(opentelemetry::sdk::common::internal_log::LogLevel level,
               const char \*file,
               int line,
               const char \*msg,
               const opentelemetry::sdk::common::AttributeMap &attributes) noexcept
↳override
    {
        // add implementation here
    }
};
opentelemetry::sdk::common::internal_
↳log::GlobalLogHandler::SetLogHandler(CustomLogHandler());
opentelemetry::sdk::common::internal_
↳log::GlobalLogHandler::SetLogLevel(opentelemetry::sdk::common::internal_
↳log::LogLevel::Debug);
```



## REFERENCE DOCUMENTATION

### 3.1 Page Hierarchy

### 3.2 Full API

#### 3.2.1 Namespaces

##### Namespace opentelemetry

###### Contents

- *Namespaces*

##### Namespaces

- *Namespace opentelemetry::baggage*
- *Namespace opentelemetry::common*
- *Namespace opentelemetry::context*
- *Namespace opentelemetry::metrics*
- *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 DurationUtil*
- *Class KeyValueIterable*
- *Class NoopKeyValueIterable*
- *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::metrics`

### Contents

- *Classes*
- *Typedefs*

## Classes

- *Template Class `Counter`*
- *Template Class `Histogram`*
- *Class `Meter`*
- *Class `MeterProvider`*
- *Template Class `NoopCounter`*
- *Template Class `NoopHistogram`*
- *Class `NoopMeter`*
- *Class `NoopMeterProvider`*

- *Class NoopObservableInstrument*
- *Template Class NoopUpDownCounter*
- *Class ObservableInstrument*
- *Template Class ObserverResultT*
- *Class Provider*
- *Class SynchronousInstrument*
- *Template Class UpDownCounter*

## Typedefs

- *Typedef opentelemetry::metrics::ObservableCallbackPtr*
- *Typedef opentelemetry::metrics::ObserverResult*

## Namespace opentelemetry::sdk

### Contents

- *Namespaces*

## Namespaces

- *Namespace opentelemetry::sdk::instrumentationlibrary*
- *Namespace opentelemetry::sdk::instrumentationscope*
- *Namespace opentelemetry::sdk::metrics*
- *Namespace opentelemetry::sdk::resource*
- *Namespace opentelemetry::sdk::trace*

## Namespace opentelemetry::sdk::instrumentationlibrary

### Contents

- *Typedefs*

## Typedefs

- *Typedef `opentelemetry::sdk::instrumentationlibrary::InstrumentationLibrary`*

## Namespace `opentelemetry::sdk::instrumentationscope`

### Contents

- *Classes*

## Classes

- *Class `InstrumentationScope`*

## Namespace `opentelemetry::sdk::metrics`

### Contents

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

## Classes

- *Struct `InstrumentDescriptor`*
- *Struct `LastReportedMetrics`*
- *Struct `ObservableCallbackRecord`*
- *Struct `PeriodicExportingMetricReaderOptions`*
- *Struct `PointDataAttributes`*
- *Struct `RegisteredView`*
- *Struct `ResourceMetrics`*
- *Struct `ScopeMetrics`*
- *Class `Aggregation`*
- *Class `AggregationConfig`*
- *Class `AlwaysSampleFilter`*
- *Class `AsyncMetricStorage`*
- *Class `AsyncMultiMetricStorage`*

- *Class AsyncWritableMetricStorage*
- *Class AttributeHashGenerator*
- *Class AttributesHashMap*
- *Class AttributesProcessor*
- *Class CollectorHandle*
- *Class DefaultAggregation*
- *Class DefaultAttributesProcessor*
- *Class DoubleCounter*
- *Class DoubleHistogram*
- *Class DoubleHistogramAggregation*
- *Class DoubleLastValueAggregation*
- *Class DoubleSumAggregation*
- *Class DoubleUpDownCounter*
- *Class DropAggregation*
- *Class DropPointData*
- *Class ExactPredicate*
- *Class ExemplarData*
- *Class ExemplarFilter*
- *Class ExemplarReservoir*
- *Class FilteredExemplarReservoir*
- *Class FilteringAttributesProcessor*
- *Class FixedSizeExemplarReservoir*
- *Class HistogramAggregationConfig*
- *Class HistogramExemplarReservoir*
- *Class HistogramExemplarReservoir::HistogramCellSelector*
- *Class HistogramPointData*
- *Class InstrumentMetaDataValidator*
- *Class InstrumentSelector*
- *Class LastValuePointData*
- *Template Class LongCounter*
- *Template Class LongHistogram*
- *Class LongHistogramAggregation*
- *Class LongLastValueAggregation*
- *Class LongSumAggregation*
- *Class LongUpDownCounter*
- *Class MatchEverythingPattern*

- *Class MatchNothingPattern*
- *Class Meter*
- *Class MeterContext*
- *Class MeterProvider*
- *Class MeterSelector*
- *Class MetricCollector*
- *Class MetricData*
- *Class MetricProducer*
- *Class MetricReader*
- *Class MetricStorage*
- *Class NeverSampleFilter*
- *Class NoExemplarReservoir*
- *Class NoopAsyncWritableMetricStorage*
- *Class NoopMetricStorage*
- *Class NoopWritableMetricStorage*
- *Class ObservableInstrument*
- *Class ObservableRegistry*
- *Template Class ObserverResultT*
- *Class PatternPredicate*
- *Class PeriodicExportingMetricReader*
- *Class Predicate*
- *Class PredicateFactory*
- *Class PushMetricExporter*
- *Class ReservoirCell*
- *Class SumPointData*
- *Class Synchronous*
- *Class SyncMetricStorage*
- *Class SyncMultiMetricStorage*
- *Class SyncWritableMetricStorage*
- *Class TemporalMetricStorage*
- *Class View*
- *Class ViewRegistry*
- *Class WithTraceSampleFilter*

## Enums

- *Enum AggregationTemporality*
- *Enum AggregationType*
- *Enum InstrumentClass*
- *Enum InstrumentType*
- *Enum InstrumentValueType*
- *Enum PredicateType*

## Functions

- *Template Function opentelemetry::sdk::metrics::BucketBinarySearch*
- *Template Function opentelemetry::sdk::metrics::HistogramDiff*
- *Template Function opentelemetry::sdk::metrics::HistogramMerge*

## Typedefs

- *Typedef opentelemetry::sdk::metrics::AggregationTemporalitySelector*
- *Typedef opentelemetry::sdk::metrics::MetricAttributes*
- *Typedef opentelemetry::sdk::metrics::PointAttributes*
- *Typedef opentelemetry::sdk::metrics::PointType*
- *Typedef opentelemetry::sdk::metrics::ValueType*

## Variables

- *Variable opentelemetry::sdk::metrics::kExportIntervalMillis*
- *Variable opentelemetry::sdk::metrics::kExportTimeOutMillis*

## Namespace opentelemetry::sdk::resource

### Contents

- *Classes*
- *Typedefs*

## Classes

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

## Typedefs

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

## Namespace opentelemetry::sdk::trace

### Contents

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

## Namespaces

- *Namespace opentelemetry::sdk::trace::@102*

## Classes

- *Struct BatchSpanProcessor::SynchronizationData*
- *Struct BatchSpanProcessorOptions*
- *Struct MultiSpanProcessor::ProcessorNode*
- *Struct MultiSpanProcessorOptions*
- *Struct SamplingResult*
- *Class AlwaysOffSampler*
- *Class AlwaysOffSamplerFactory*
- *Class AlwaysOnSampler*
- *Class AlwaysOnSamplerFactory*
- *Class BatchSpanProcessor*
- *Class BatchSpanProcessorFactory*
- *Class IdGenerator*
- *Class MultiRecordable*
- *Class MultiSpanProcessor*
- *Class ParentBasedSampler*



- *Class ParentBasedSamplerFactory*
- *Class RandomIdGenerator*
- *Class RandomIdGeneratorFactory*
- *Class Recordable*
- *Class Sampler*
- *Class SimpleSpanProcessor*
- *Class SimpleSpanProcessorFactory*
- *Class SpanData*
- *Class SpanDataEvent*
- *Class SpanDataLink*
- *Class SpanExporter*
- *Class SpanProcessor*
- *Class TraceIdRatioBasedSampler*
- *Class TraceIdRatioBasedSamplerFactory*
- *Class Tracer*
- *Class TracerContext*
- *Class TracerContextFactory*
- *Class TracerProvider*
- *Class TracerProviderFactory*

## Enums

- *Enum Decision*

## Namespace `opentelemetry::sdk::trace::@102`

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

## Variables

- 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`

## 3.2.2 Classes and Structs

### Struct `InstrumentDescriptor`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_instruments.h`

### Struct Documentation

struct `InstrumentDescriptor`

#### Public Members

`std::string` `name_`

`std::string` `description_`

`std::string` `unit_`

*InstrumentType* `type_`

*InstrumentValueType* **value\_type\_**

### Struct LastReportedMetrics

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_state_temporal_metric_storage.h`

### Struct Documentation

struct **LastReportedMetrics**

#### Public Members

`std::unique_ptr<AttributesHashMap>` **attributes\_map**

`opentelemetry::common::SystemTimestamp` **collection\_ts**

### Struct ObservableCallbackRecord

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_state_observable_registry.h`

### Struct Documentation

struct **ObservableCallbackRecord**

#### Public Members

`opentelemetry::metrics::ObservableCallbackPtr` **callback**

`void` **\*state**

`opentelemetry::metrics::ObservableInstrument` **\*instrument**

### Struct `PeriodicExportingMetricReaderOptions`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_export_periodic_exporting_metric_reader.h`

#### Struct Documentation

struct `PeriodicExportingMetricReaderOptions`

##### Public Members

`std::chrono::milliseconds export_interval_millis` = `std::chrono::milliseconds(kExportIntervalMillis)`

`std::chrono::milliseconds export_timeout_millis` = `std::chrono::milliseconds(kExportTimeOutMillis)`

### Struct `PointDataAttributes`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_data_metric_data.h`

#### Struct Documentation

struct `PointDataAttributes`

##### Public Members

*PointAttributes* `attributes`

*PointType* `point_data`

### Struct `RegisteredView`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_view_view_registry.h`

## Struct Documentation

struct **RegisteredView**

### Public Functions

```
inline RegisteredView(std::unique_ptr<opentelemetry::sdk::metrics::InstrumentSelector>
    instrument_selector, std::unique_ptr<opentelemetry::sdk::metrics::MeterSelector>
    meter_selector, std::unique_ptr<opentelemetry::sdk::metrics::View> view)
```

### Public Members

```
std::unique_ptr<opentelemetry::sdk::metrics::InstrumentSelector> instrument_selector_
```

```
std::unique_ptr<opentelemetry::sdk::metrics::MeterSelector> meter_selector_
```

```
std::unique_ptr<opentelemetry::sdk::metrics::View> view_
```

## Struct ResourceMetrics

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_export_metric_producer.h`

## Struct Documentation

struct **ResourceMetrics**

### Public Members

```
const opentelemetry::sdk::resource::Resource *resource_
```

```
std::vector<ScopeMetrics> scope_metric_data_
```

## Struct ScopeMetrics

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_export_metric_producer.h`

## Struct Documentation

### struct **ScopeMetrics**

Metric Data to be exported along with resources and Instrumentation scope.

#### Public Members

const opentelemetry::sdk::instrumentationscope::InstrumentationScope \*scope\_

std::vector<MetricData> metric\_data\_

### Struct BatchSpanProcessor::SynchronizationData

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

## Nested Relationships

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

## Struct Documentation

### struct **SynchronizationData**

#### Public Members

std::condition\_variable cv

std::condition\_variable force\_flush\_cv

std::mutex cv\_m

std::mutex force\_flush\_cv\_m

std::mutex shutdown\_m

std::atomic<bool> is\_force\_wakeup\_background\_worker

std::atomic<bool> is\_force\_flush\_pending

std::atomic<bool> is\_force\_flush\_notified



`std::atomic<bool> is_shutdown`

### Struct `BatchSpanProcessorOptions`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_trace_batch_span_processor_options.h`

### Struct Documentation

struct `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_latest_sdk_include_opentelemetry_sdk_trace_multi_span_processor.h`

### Nested Relationships

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

### Struct Documentation

struct `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_latest_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_latest_sdk_include_opentelemetry_sdk_trace_sampler.h`

### Struct Documentation

```
struct SamplingResult
```

The output of `ShouldSample`. It contains a sampling `Decision` and a set of `Span Attributes`.

### Public Functions

```
inline bool IsRecording()
```

```
inline bool IsSampled()
```

## 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_latest_api_include_opentelemetry_trace_span_metadata.h`

## Struct Documentation

struct **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_latest_api_include_opentelemetry_trace_span_startoptions.h`

## Struct Documentation

struct **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\_latest\_api\_include\_opentelemetry\_baggage\_baggage.h

## Class Documentation

class **Baggage**

### Public Functions

inline **Baggage**() noexcept

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

template<class T>

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

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

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

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) noexcept

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

### Public Static Functions

**static inline OPENTELEMETRY\_API\_SINGLETON** nostd::shared\_ptr< Baggage > **GetDefault** ()

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

### 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_latest_api_include_opentelemetry_baggage_propagation_baggage_propagator.h`

### Inheritance Relationships

#### Base Type

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

### Class Documentation

```
class 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 DurationUtil

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_api_include_opentelemetry_common_timestamp.h`

### Class Documentation

```
class DurationUtil
```

## Public Static Functions

```
template<class Rep, class Period>
static inline std::chrono::duration<Rep, Period> AdjustWaitForTimeout(std::chrono::duration<Rep,
                                                                    Period> timeout,
                                                                    std::chrono::duration<Rep,
                                                                    Period> indefinite_value)
noexcept
```

## Class KeyValueIterable

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_api_include_opentelemetry_common_key_value_iterable.h`

## Inheritance Relationships

### Derived Type

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

## Class Documentation

### class `KeyValueIterable`

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

Subclassed by `opentelemetry::common::NoopKeyValueIterable`

### 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 NoopKeyValueIterable

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_api_include_opentelemetry_common_key_value_iterable.h`

## Inheritance Relationships

### Base Type

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

## Class Documentation

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

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

### Public Functions

`~NoopKeyValueIterable()` override = default

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

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

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

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

## Class SteadyTimestamp

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_api_include_opentelemetry_common_timestamp.h`

## Class Documentation

class **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\_latest\_api\_include\_opentelemetry\_common\_timestamp.h

## Class Documentation

class **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_latest_api_include_opentelemetry_context_context.h`

## Class Documentation

class **Context**

### Public Functions

**Context**() = default

template<class **T**>

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

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

template<class **T**>

```
inline Context SetValues(T &values) noexcept  
inline Context SetValue(nstd::string_view key, ContextValue value) noexcept  
inline context::ContextValue GetValue(const nstd::string_view key) const noexcept  
inline bool HasKey(const nstd::string_view key) const noexcept  
inline bool operator==(const Context &other) const noexcept
```

## Class CompositePropagator

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_latest\_api\_include\_opentelemetry\_context\_propagation\_composite\_propagator.h

## Inheritance Relationships

### Base Type

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

## Class Documentation

```
class 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_latest_api_include_opentelemetry_context_propagation_global_propagator.h`

### Class Documentation

class **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_latest_api_include_opentelemetry_context_propagation_noop_propagator.h`

### Inheritance Relationships

#### Base Type

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

### Class Documentation

class **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`&) noexcept override  
 Noop inject function does nothing

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

## Class TextMapCarrier

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_api_include_opentelemetry_context_propagation_text_map_propagator.h`

## Class Documentation

class **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)>`) const noexcept

virtual `~TextMapCarrier`() = default

## Class TextMapPropagator

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_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 **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_latest_api_include_opentelemetry_context_runtime_context.h`

## Class Documentation

class **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

```
static inline nostd::shared_ptr<const RuntimeContextStorage> GetConstRuntimeContextStorage()
    noexcept
```

Provide a pointer to const runtime context storage.

The returned pointer can only be used for extending the lifetime of the runtime context storage.

## Class `RuntimeContextStorage`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_api_include_opentelemetry_context_runtime_context.h`

## Inheritance Relationships

### Derived Type

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

## Class Documentation

class **`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\_latest\_api\_include\_opentelemetry\_context\_runtime\_context.h

## Inheritance Relationships

### Base Type

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

## Class Documentation

```
class 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\_latest\_api\_include\_opentelemetry\_context\_runtime\_context.h

## Class Documentation

```
class Token
```

## Public Functions

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

```
inline ~Token() noexcept
```

## Template Class Counter

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_api_include_opentelemetry_metrics_sync_instruments.h`

## Inheritance Relationships

### Base Type

- public `opentelemetry::metrics::SynchronousInstrument`

### Derived Types

- public `opentelemetry::metrics::NoopCounter< T >` (*Template Class NoopCounter*)
- public `opentelemetry::sdk::metrics::DoubleCounter` (*Class DoubleCounter*)
- public `opentelemetry::sdk::metrics::LongCounter< T >` (*Template Class LongCounter*)

## Class Documentation

```
template<class T>
```

```
class Counter : public opentelemetry::metrics::SynchronousInstrument
```

Subclassed by `opentelemetry::metrics::NoopCounter< T >`, `opentelemetry::sdk::metrics::DoubleCounter`, `opentelemetry::sdk::metrics::LongCounter< T >`

### Public Functions

```
virtual void Add(T value) noexcept = 0
```

Add adds the value to the counter's sum

**Parameters** `value` – The increment amount. MUST be non-negative.

```
virtual void Add(T value, const opentelemetry::context::Context &context) noexcept = 0
```

```
virtual void Add(T value, const common::KeyValueIterable &attributes) noexcept = 0
```

Add adds the value to the counter's sum. The attributes should contain the keys and values to be associated with this value. Counters only accept positive valued updates.

#### Parameters

- `value` – The increment amount. MUST be non-negative.
- `attributes` – the set of attributes, as key-value pairs



```
virtual void Add(T value, const common::KeyValueIterable &attributes, const opentelemetry::context::Context
    &context) noexcept = 0
```

```
template<class U, nostd::enable_if_t<common::detail::is_key_value_iterable<U>::value>* = nullptr>
inline void Add(T value, const U &attributes) noexcept
```

```
template<class U, nostd::enable_if_t<common::detail::is_key_value_iterable<U>::value>* = nullptr>
inline void Add(T value, const U &attributes, const opentelemetry::context::Context &context) noexcept
```

```
inline void Add(T value, std::initializer_list<std::pair<nostd::string_view, common::AttributeValue>>
    attributes) noexcept
```

```
inline void Add(T value, std::initializer_list<std::pair<nostd::string_view, common::AttributeValue>>
    attributes, const opentelemetry::context::Context &context) noexcept
```

## Template Class Histogram

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_api_include_opentelemetry_metrics_sync_instruments.h`

## Inheritance Relationships

### Base Type

- public `opentelemetry::metrics::SynchronousInstrument`

### Derived Types

- public `opentelemetry::metrics::NoopHistogram< T >` (*Template Class NoopHistogram*)
- public `opentelemetry::sdk::metrics::LongHistogram< T >` (*Template Class LongHistogram*)

## Class Documentation

```
template<class T>
```

```
class Histogram : public opentelemetry::metrics::SynchronousInstrument
```

A histogram instrument that records values.

Subclassed by `opentelemetry::metrics::NoopHistogram< T >`, `opentelemetry::sdk::metrics::LongHistogram< T >`

## Public Functions

virtual void **Record**(*T* value, const opentelemetry::context::Context &context) noexcept = 0

Records a value.

**Parameters** **value** – The increment amount. May be positive, negative or zero.

virtual void **Record**(*T* value, const common::KeyValueIterable &attributes, const opentelemetry::context::Context &context) noexcept = 0

Records a value with a set of attributes.

### Parameters

- **value** – The increment amount. May be positive, negative or zero.
- **attributes** – A set of attributes to associate with the count.

```
template<class U, nostd::enable_if_t<common::detail::is_key_value_iterable<U>::value>* = nullptr>
inline void Record(T value, const U &attributes, const opentelemetry::context::Context &context) noexcept
```

```
inline void Record(T value, std::initializer_list<std::pair<nostd::string_view, common::AttributeValue>>
attributes, const opentelemetry::context::Context &context) noexcept
```

## Class Meter

- Defined in file\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_latest\_api\_include\_opentelemetry\_metrics\_meter.h

## Inheritance Relationships

### Derived Types

- public opentelemetry::metrics::NoopMeter (*Class NoopMeter*)
- public opentelemetry::sdk::metrics::Meter (*Class Meter*)

## Class Documentation

### class Meter

Handles instrument creation and provides a facility for batch recording.

This class provides methods to create new metric instruments, record a batch of values to a specified set of instruments, and collect measurements from all instruments.

Subclassed by *opentelemetry::metrics::NoopMeter*, *opentelemetry::sdk::metrics::Meter*

## Public Functions

virtual **~Meter()** = default

virtual nostd::unique\_ptr<*Counter*<uint64\_t>> **CreateUInt64Counter**(nostd::string\_view name, nostd::string\_view description = "", nostd::string\_view unit = "") noexcept = 0

Creates a Counter with the passed characteristics and returns a unique\_ptr to that Counter.

### Parameters

- **name** – the name of the new Counter.
- **description** – a brief description of what the Counter is used for.
- **unit** – the unit of metric values following <https://unitsofmeasure.org/ucum.html>.

**Returns** a shared pointer to the created Counter.

virtual nostd::unique\_ptr<*Counter*<double>> **CreateDoubleCounter**(nostd::string\_view name, nostd::string\_view description = "", nostd::string\_view unit = "") noexcept = 0

virtual nostd::shared\_ptr<*ObservableInstrument*> **CreateInt64ObservableCounter**(nostd::string\_view name, nostd::string\_view description = "", nostd::string\_view unit = "") noexcept = 0

Creates a Asynchronous (Observable) counter with the passed characteristics and returns a shared\_ptr to that Observable Counter

### Parameters

- **name** – the name of the new Observable Counter.
- **description** – a brief description of what the Observable Counter is used for.
- **unit** – the unit of metric values following <https://unitsofmeasure.org/ucum.html>.

virtual nostd::shared\_ptr<*ObservableInstrument*> **CreateDoubleObservableCounter**(nostd::string\_view name, nostd::string\_view description = "", nostd::string\_view unit = "") noexcept = 0

virtual nostd::unique\_ptr<*Histogram*<uint64\_t>> **CreateUInt64Histogram**(nostd::string\_view name, nostd::string\_view description = "", nostd::string\_view unit = "") noexcept = 0

Creates a *Histogram* with the passed characteristics and returns a unique\_ptr to that *Histogram*.

### Parameters

- **name** – the name of the new *Histogram*.

- **description** – a brief description of what the *Histogram* is used for.
- **unit** – the unit of metric values following <https://unitsofmeasure.org/ucum.html>.

**Returns** a shared pointer to the created *Histogram*.

```
virtual nostd::unique_ptr<Histogram<double>> CreateDoubleHistogram(nostd::string_view name,  
                                                                nostd::string_view description =  
                                                                "", nostd::string_view unit = "")  
                                                                noexcept = 0
```

```
virtual nostd::shared_ptr<ObservableInstrument> CreateInt64ObservableGauge(nostd::string_view name,  
                                                                           nostd::string_view  
                                                                           description = "",  
                                                                           nostd::string_view unit =  
                                                                           "") noexcept = 0
```

Creates a Asynchronous (Observable) Gauge with the passed characteristics and returns a shared\_ptr to that Observable Gauge

#### Parameters

- **name** – the name of the new Observable Gauge.
- **description** – a brief description of what the Observable Gauge is used for.
- **unit** – the unit of metric values following <https://unitsofmeasure.org/ucum.html>.

```
virtual nostd::shared_ptr<ObservableInstrument> CreateDoubleObservableGauge(nostd::string_view  
                                                                              name,  
                                                                              nostd::string_view  
                                                                              description = "",  
                                                                              nostd::string_view unit  
                                                                              = "") noexcept = 0
```

```
virtual nostd::unique_ptr<UpDownCounter<int64_t>> CreateInt64UpDownCounter(nostd::string_view  
                                                                              name,  
                                                                              nostd::string_view  
                                                                              description = "",  
                                                                              nostd::string_view unit  
                                                                              = "") noexcept = 0
```

Creates an *UpDownCounter* with the passed characteristics and returns a unique\_ptr to that *UpDownCounter*.

#### Parameters

- **name** – the name of the new *UpDownCounter*.
- **description** – a brief description of what the *UpDownCounter* is used for.
- **unit** – the unit of metric values following <https://unitsofmeasure.org/ucum.html>.

**Returns** a shared pointer to the created *UpDownCounter*.

```
virtual nostd::unique_ptr<UpDownCounter<double>> CreateDoubleUpDownCounter(nostd::string_view  
                                                                              name,  
                                                                              nostd::string_view  
                                                                              description = "",  
                                                                              nostd::string_view  
                                                                              unit = "") noexcept =  
                                                                              0
```

```
virtual nostd::shared_ptr<ObservableInstrument> CreateInt64ObservableUpDownCounter(nostd::string_view
name,
nostd::string_view
description =
"",
nostd::string_view
unit = "")
noexcept = 0
```

Creates a Asynchronous (Observable) *UpDownCounter* with the passed characteristics and returns a *shared\_ptr* to that Observable *UpDownCounter*

#### Parameters

- **name** – the name of the new Observable *UpDownCounter*.
- **description** – a brief description of what the Observable *UpDownCounter* is used for.
- **unit** – the unit of metric values following <https://unitsofmeasure.org/ucum.html>.

```
virtual nostd::shared_ptr<ObservableInstrument> CreateDoubleObservableUpDownCounter(nostd::string_view
name,
nostd::string_view
description =
"",
nostd::string_view
unit = "")
noexcept = 0
```

### Class MeterProvider

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_api_include_opentelemetry_metrics_meter_provider.h`

### Inheritance Relationships

#### Derived Types

- `public opentelemetry::metrics::NoopMeterProvider` (*Class NoopMeterProvider*)
- `public opentelemetry::sdk::metrics::MeterProvider` (*Class MeterProvider*)

### Class Documentation

#### class **MeterProvider**

Creates new *Meter* instances.

Subclassed by *opentelemetry::metrics::NoopMeterProvider*, *opentelemetry::sdk::metrics::MeterProvider*

### Public Functions

virtual **~MeterProvider**() = default

virtual nostd::shared\_ptr<*Meter*> **GetMeter**(nostd::string\_view library\_name, nostd::string\_view library\_version = "", nostd::string\_view schema\_url = "")  
noexcept = 0

Gets or creates a named *Meter* instance.

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

### Template Class NoopCounter

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_latest\_api\_include\_opentelemetry\_metrics\_noop.h

### Inheritance Relationships

#### Base Type

- public opentelemetry::metrics::Counter< T >

### Class Documentation

template<class T>

class **NoopCounter** : public opentelemetry::metrics::Counter<T>

#### Public Functions

inline **NoopCounter**(nostd::string\_view, nostd::string\_view, nostd::string\_view) noexcept

inline void **Add**(T) noexcept override

inline void **Add**(T, const opentelemetry::context::Context&) noexcept override

inline void **Add**(T, const common::KeyValueIterable&) noexcept override

inline void **Add**(T, const common::KeyValueIterable&, const opentelemetry::context::Context&) noexcept  
override

### Template Class NoopHistogram

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_latest\_api\_include\_opentelemetry\_metrics\_noop.h

## Inheritance Relationships

### Base Type

- public opentelemetry::metrics::Histogram< T > (*Template Class Histogram*)

### Class Documentation

template<class T>

class **NoopHistogram** : public opentelemetry::metrics::Histogram<T>

#### Public Functions

inline **NoopHistogram**(nostd::string\_view, nostd::string\_view, nostd::string\_view) noexcept

inline void **Record**(T, const opentelemetry::context::Context&) noexcept override

inline void **Record**(T, const common::KeyValueIterable&, const opentelemetry::context::Context&) noexcept override

### Class NoopMeter

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_latest\_api\_include\_opentelemetry\_metrics\_noop.h

## Inheritance Relationships

### Base Type

- public opentelemetry::metrics::Meter (*Class Meter*)

### Class Documentation

class **NoopMeter** : public opentelemetry::metrics::Meter

No-op implementation of *Meter*.

#### Public Functions

inline virtual nostd::unique\_ptr<Counter<uint64\_t>> **CreateUInt64Counter**(nostd::string\_view name, nostd::string\_view description = "", nostd::string\_view unit = "") noexcept override

Creates a Counter with the passed characteristics and returns a unique\_ptr to that Counter.

#### Parameters

- **name** – the name of the new Counter.

- **description** – a brief description of what the Counter is used for.
- **unit** – the unit of metric values following <https://unitsofmeasure.org/ucum.html>.

**Returns** a shared pointer to the created Counter.

```
inline virtual nostd::unique_ptr<Counter<double>> CreateDoubleCounter(nostd::string_view name,  
                                                                    nostd::string_view description =  
                                                                    "", nostd::string_view unit = "")  
                                                                    noexcept override
```

```
inline virtual nostd::shared_ptr<ObservableInstrument> CreateInt64ObservableCounter(nostd::string_view  
                                                                                    name,  
                                                                                    nostd::string_view  
                                                                                    description = "",  
                                                                                    nostd::string_view  
                                                                                    unit = "")  
                                                                                    noexcept  
                                                                                    override
```

Creates a Asynchronous (Observable) counter with the passed characteristics and returns a shared\_ptr to that Observable Counter

#### Parameters

- **name** – the name of the new Observable Counter.
- **description** – a brief description of what the Observable Counter is used for.
- **unit** – the unit of metric values following <https://unitsofmeasure.org/ucum.html>.

```
inline virtual nostd::shared_ptr<ObservableInstrument> CreateDoubleObservableCounter(nostd::string_view  
                                                                                    name,  
                                                                                    nostd::string_view  
                                                                                    description =  
                                                                                    "",  
                                                                                    nostd::string_view  
                                                                                    unit = "")  
                                                                                    noexcept  
                                                                                    override
```

```
inline virtual nostd::unique_ptr<Histogram<uint64_t>> CreateUInt64Histogram(nostd::string_view name,  
                                                                              nostd::string_view  
                                                                              description = "",  
                                                                              nostd::string_view  
                                                                              unit =  
                                                                              "") noexcept override
```

Creates a *Histogram* with the passed characteristics and returns a unique\_ptr to that *Histogram*.

#### Parameters

- **name** – the name of the new *Histogram*.
- **description** – a brief description of what the *Histogram* is used for.
- **unit** – the unit of metric values following <https://unitsofmeasure.org/ucum.html>.

**Returns** a shared pointer to the created *Histogram*.

```
inline virtual nostd::unique_ptr<Histogram<double>> CreateDoubleHistogram(nostd::string_view name,  
                                                                            nostd::string_view  
                                                                            description = "",  
                                                                            nostd::string_view  
                                                                            unit =  
                                                                            "") noexcept override
```



```
inline virtual nostd::shared_ptr<ObservableInstrument> CreateInt64ObservableGauge(nostd::string_view
                                                                              name,
                                                                              nostd::string_view
                                                                              description = "",
                                                                              nostd::string_view
                                                                              unit = "")
                                                                              noexcept override
```

Creates a Asynchronous (Observable) Gauge with the passed characteristics and returns a shared\_ptr to that Observable Gauge

#### Parameters

- **name** – the name of the new Observable Gauge.
- **description** – a brief description of what the Observable Gauge is used for.
- **unit** – the unit of metric values following <https://unitsofmeasure.org/ucum.html>.

```
inline virtual nostd::shared_ptr<ObservableInstrument> CreateDoubleObservableGauge(nostd::string_view
                                                                              name,
                                                                              nostd::string_view
                                                                              description = "",
                                                                              nostd::string_view
                                                                              unit = "")
                                                                              noexcept
                                                                              override
```

```
inline virtual nostd::unique_ptr<UpDownCounter<int64_t>> CreateInt64UpDownCounter(nostd::string_view
                                                                              name,
                                                                              nostd::string_view
                                                                              description = "",
                                                                              nostd::string_view
                                                                              unit = "")
                                                                              noexcept
                                                                              override
```

Creates an *UpDownCounter* with the passed characteristics and returns a unique\_ptr to that *UpDownCounter*.

#### Parameters

- **name** – the name of the new *UpDownCounter*.
- **description** – a brief description of what the *UpDownCounter* is used for.
- **unit** – the unit of metric values following <https://unitsofmeasure.org/ucum.html>.

**Returns** a shared pointer to the created *UpDownCounter*.

```
inline virtual nostd::unique_ptr<UpDownCounter<double>> CreateDoubleUpDownCounter(nostd::string_view
                                                                              name,
                                                                              nostd::string_view
                                                                              description =
                                                                              "",
                                                                              nostd::string_view
                                                                              unit = "")
                                                                              noexcept
                                                                              override
```

```
inline virtual nostd::shared_ptr<ObservableInstrument> CreateInt64ObservableUpDownCounter(nostd::string_view name, nostd::string_view description = "", nostd::string_view unit = "") noexcept override
```

Creates a Asynchronous (Observable) *UpDownCounter* with the passed characteristics and returns a *shared\_ptr* to that Observable *UpDownCounter*

#### Parameters

- **name** – the name of the new Observable *UpDownCounter*.
- **description** – a brief description of what the Observable *UpDownCounter* is used for.
- **unit** – the unit of metric values following <https://unitsofmeasure.org/ucum.html>.

```
inline virtual nostd::shared_ptr<ObservableInstrument> CreateDoubleObservableUpDownCounter(nostd::string_view name, nostd::string_view description = "", nostd::string_view unit = "") noexcept override
```

### Class NoopMeterProvider

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_api_include_opentelemetry_metrics_noop.h`

### Inheritance Relationships

#### Base Type

- `public opentelemetry::metrics::MeterProvider` (Class *MeterProvider*)

## Class Documentation

class **NoopMeterProvider** : public opentelemetry::metrics::*MeterProvider*

No-op implementation of a *MeterProvider*.

### Public Functions

inline **NoopMeterProvider**()

inline virtual nostd::shared\_ptr<*Meter*> **GetMeter**(nostd::string\_view, nostd::string\_view, nostd::string\_view) noexcept override

Gets or creates a named *Meter* instance.

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

## Class NoopObservableInstrument

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_latest\_api\_include\_opentelemetry\_metrics\_noop.h

## Inheritance Relationships

### Base Type

- public opentelemetry::metrics::ObservableInstrument

## Class Documentation

class **NoopObservableInstrument** : public opentelemetry::metrics::*ObservableInstrument*

### Public Functions

inline **NoopObservableInstrument**(nostd::string\_view, nostd::string\_view, nostd::string\_view) noexcept

inline void **AddCallback**(*ObservableCallbackPtr*, void\*) noexcept override

inline void **RemoveCallback**(*ObservableCallbackPtr*, void\*) noexcept override

## Template Class NoopUpDownCounter

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_api_include_opentelemetry_metrics_noop.h`

## Inheritance Relationships

### Base Type

- `public opentelemetry::metrics::UpDownCounter< T >` (*Template Class UpDownCounter*)

## Class Documentation

template<class T>

class **NoopUpDownCounter** : public opentelemetry::metrics::*UpDownCounter*<T>

### Public Functions

inline **NoopUpDownCounter**(nostd::string\_view, nostd::string\_view, nostd::string\_view) noexcept

**~NoopUpDownCounter**() override = default

inline void **Add**(T) noexcept override

inline void **Add**(T, const opentelemetry::context::Context&) noexcept override

inline void **Add**(T, const common::KeyValueIterable&) noexcept override

inline void **Add**(T, const common::KeyValueIterable&, const opentelemetry::context::Context&) noexcept override

## Class ObservableInstrument

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_api_include_opentelemetry_metrics_async_instruments.h`

## Inheritance Relationships

### Derived Types

- `public opentelemetry::metrics::NoopObservableInstrument` (*Class NoopObservableInstrument*)
- `public opentelemetry::sdk::metrics::ObservableInstrument` (*Class ObservableInstrument*)

## Class Documentation

### class **ObservableInstrument**

Subclassed by *opentelemetry::metrics::NoopObservableInstrument*, *opentelemetry::sdk::metrics::ObservableInstrument*

#### Public Functions

**ObservableInstrument**() = default

virtual **~ObservableInstrument**() = default

virtual void **AddCallback**(*ObservableCallbackPtr*, void \*state) noexcept = 0

Sets up a function that will be called whenever a metric collection is initiated.

virtual void **RemoveCallback**(*ObservableCallbackPtr*, void \*state) noexcept = 0

Remove a function that was configured to be called whenever a metric collection is initiated.

### Template Class **ObserverResultT**

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_api_include_opentelemetry_metrics_observer_result.h`

## Inheritance Relationships

### Derived Type

- public `opentelemetry::sdk::metrics::ObserverResultT< T >` (*Template Class ObserverResultT*)

## Class Documentation

template<class **T**>

### class **ObserverResultT**

*ObserverResultT* class is necessary for the callback recording asynchronous instrument use.

Subclassed by *opentelemetry::sdk::metrics::ObserverResultT< T >*

#### Public Functions

virtual **~ObserverResultT**() = default

virtual void **Observe**(*T* value) noexcept = 0

virtual void **Observe**(*T* value, const common::KeyValueIterable &attributes) noexcept = 0

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

inline void **Observe**(*T* value, const *U* &attributes) noexcept

inline void **Observe**(*T* value, std::initializer\_list<std::pair<nstd::string\_view, common::AttributeValue>> attributes) noexcept

## Class Provider

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_api_include_opentelemetry_metrics_provider.h`

## Class Documentation

### class **Provider**

Stores the singleton global *MeterProvider*.

### Public Static Functions

static inline `nstd::shared_ptr<MeterProvider> GetMeterProvider()` noexcept

Returns the singleton *MeterProvider*.

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

static inline void `SetMeterProvider(nstd::shared_ptr<MeterProvider> tp)` noexcept

Changes the singleton *MeterProvider*.

## Class SynchronousInstrument

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_api_include_opentelemetry_metrics_sync_instruments.h`

## Inheritance Relationships

### Derived Types

- `public opentelemetry::metrics::Counter< double >` (*Template Class Counter*)
- `public opentelemetry::metrics::Histogram< double >` (*Template Class Histogram*)
- `public opentelemetry::metrics::Counter< T >` (*Template Class Counter*)
- `public opentelemetry::metrics::Histogram< T >` (*Template Class Histogram*)
- `public opentelemetry::metrics::UpDownCounter< T >` (*Template Class UpDownCounter*)
- `public opentelemetry::metrics::UpDownCounter< double >` (*Template Class UpDownCounter*)
- `public opentelemetry::metrics::UpDownCounter< int64_t >` (*Template Class UpDownCounter*)

## Class Documentation

### class `SynchronousInstrument`

Subclassed by `opentelemetry::metrics::Counter< double >`, `opentelemetry::metrics::Histogram< double >`, `opentelemetry::metrics::Counter< T >`, `opentelemetry::metrics::Histogram< T >`, `opentelemetry::metrics::UpDownCounter< T >`, `opentelemetry::metrics::UpDownCounter< double >`, `opentelemetry::metrics::UpDownCounter< int64_t >`

#### Public Functions

`SynchronousInstrument()` = default

virtual `~SynchronousInstrument()` = default

### Template Class `UpDownCounter`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_api_include_opentelemetry_metrics_sync_instruments.h`

## Inheritance Relationships

### Base Type

- public `opentelemetry::metrics::SynchronousInstrument`

### Derived Type

- public `opentelemetry::metrics::NoopUpDownCounter< T >` (*Template Class `NoopUpDownCounter`*)

## Class Documentation

template<class `T`>

class `UpDownCounter` : public `opentelemetry::metrics::SynchronousInstrument`

An up-down-counter instrument that adds or reduce values.

Subclassed by `opentelemetry::metrics::NoopUpDownCounter< T >`

#### Public Functions

virtual void `Add(T value)` noexcept = 0

Adds a value.

**Parameters** `value` – The amount of the measurement.

virtual void `Add(T value, const opentelemetry::context::Context &context)` noexcept = 0

virtual void **Add**(*T* value, const common::*KeyValueIterable* &attributes) noexcept = 0

Add a value with a set of attributes.

#### Parameters

- **value** – The increment amount. May be positive, negative or zero.
- **attributes** – A set of attributes to associate with the count.

virtual void **Add**(*T* value, const common::*KeyValueIterable* &attributes, const opentelemetry::context::*Context* &context) noexcept = 0

template<class **U**, nostd::enable\_if\_t<common::detail::is\_key\_value\_iterable<*U*::value>\* = nullptr>  
inline void **Add**(*T* value, const *U* &attributes) noexcept

template<class **U**, nostd::enable\_if\_t<common::detail::is\_key\_value\_iterable<*U*::value>\* = nullptr>  
inline void **Add**(*T* value, const *U* &attributes, const opentelemetry::context::*Context* &context) noexcept

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

inline void **Add**(*T* value, std::initializer\_list<std::pair<nostd::string\_view, common::*AttributeValue*>>  
attributes, const opentelemetry::context::*Context* &context) noexcept

## Class InstrumentationScope

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_latest\_sdk\_include\_opentelemetry\_sdk\_instrumentationscope\_instrumentation\_scope.h

## Class Documentation

class **InstrumentationScope**

### Public Functions

**InstrumentationScope**(const *InstrumentationScope*&) = default

inline std::size\_t **HashCode**() const noexcept

inline bool **operator==**(const *InstrumentationScope* &other) const

Compare 2 instrumentation libraries.

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

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

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

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

#### Parameters

- **name** – name of the instrumentation scope to compare.
- **version** – version of the instrumentation scope to compare.



- **schema\_url** – schema url of the telemetry emitted by the scope.

**Returns** true if name and version in this instrumentation scope 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<InstrumentationScope> Create(nostd::string_view name, nostd::string_view version = "", nostd::string_view schema_url = "")
```

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

### Parameters

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

**Returns** the newly created InstrumentationScope.

## Class Aggregation

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_aggregation_aggregation.h`

## Inheritance Relationships

### Derived Types

- `public opentelemetry::sdk::metrics::DoubleHistogramAggregation` (*Class DoubleHistogramAggregation*)
- `public opentelemetry::sdk::metrics::DoubleLastValueAggregation` (*Class DoubleLastValueAggregation*)
- `public opentelemetry::sdk::metrics::DoubleSumAggregation` (*Class DoubleSumAggregation*)
- `public opentelemetry::sdk::metrics::DropAggregation` (*Class DropAggregation*)
- `public opentelemetry::sdk::metrics::LongHistogramAggregation` (*Class LongHistogramAggregation*)
- `public opentelemetry::sdk::metrics::LongLastValueAggregation` (*Class LongLastValueAggregation*)
- `public opentelemetry::sdk::metrics::LongSumAggregation` (*Class LongSumAggregation*)

## Class Documentation

### class **Aggregation**

Subclassed by *opentelemetry::sdk::metrics::DoubleHistogramAggregation*, *opentelemetry::sdk::metrics::DoubleLastValueAggregation*, *opentelemetry::sdk::metrics::DoubleSumAggregation*, *opentelemetry::sdk::metrics::DropAggregation*, *opentelemetry::sdk::metrics::LongHistogramAggregation*, *opentelemetry::sdk::metrics::LongLastValueAggregation*, *opentelemetry::sdk::metrics::LongSumAggregation*

### Public Functions

virtual void **Aggregate**(int64\_t value, const *PointAttributes* &attributes = {}) noexcept = 0

virtual void **Aggregate**(double value, const *PointAttributes* &attributes = {}) noexcept = 0

virtual std::unique\_ptr<*Aggregation*> **Merge**(const *Aggregation* &delta) const noexcept = 0

Returns the result of the merge of the two aggregations.

This should always assume that the aggregations do not overlap and merge together for a new cumulative report.

**Parameters** *delta* – the newly captured (delta) aggregation

**Returns** the result of the merge of the given aggregation.

virtual std::unique\_ptr<*Aggregation*> **Diff**(const *Aggregation* &next) const noexcept = 0

Returns a new delta aggregation by comparing two cumulative measurements.

**Parameters** *next* – the newly captured (cumulative) aggregation.

**Returns** The resulting delta aggregation.

virtual *PointType* **ToPoint**() const noexcept = 0

Returns the point data that the aggregation will produce.

**Returns** *PointType*

virtual **~Aggregation**() = default

### Class **AggregationConfig**

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_aggregation_aggregation_config.h`

### Inheritance Relationships

#### Derived Type

- public `opentelemetry::sdk::metrics::HistogramAggregationConfig` (*Class HistogramAggregationConfig*)

## Class Documentation

### class **AggregationConfig**

Subclassed by *opentelemetry::sdk::metrics::HistogramAggregationConfig*

#### Public Functions

virtual **~AggregationConfig**() = default

### Class AlwaysSampleFilter

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_exemplar_always_sample_filter.h`

## Inheritance Relationships

### Base Type

- public `opentelemetry::sdk::metrics::ExemplarFilter` (*Class ExemplarFilter*)

## Class Documentation

class **AlwaysSampleFilter** : public `opentelemetry::sdk::metrics::ExemplarFilter`

#### Public Functions

inline bool **ShouldSampleMeasurement**(int64\_t, const *MetricAttributes*&, const `opentelemetry::context::Context`&) noexcept override

inline bool **ShouldSampleMeasurement**(double, const *MetricAttributes*&, const `opentelemetry::context::Context`&) noexcept override

explicit **AlwaysSampleFilter**() = default

### Class AsyncMetricStorage

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_state_async_metric_storage.h`

## Inheritance Relationships

### Base Types

- public opentelemetry::sdk::metrics::MetricStorage
- public opentelemetry::sdk::metrics::AsyncWritableMetricStorage

### Class Documentation

class **AsyncMetricStorage** : public opentelemetry::sdk::metrics::MetricStorage, public opentelemetry::sdk::metrics::AsyncWritableMetricStorage

#### Public Functions

inline **AsyncMetricStorage**(*InstrumentDescriptor* instrument\_descriptor, const *AggregationType* aggregation\_type, const *AggregationConfig* \*aggregation\_config)

template<class T>

inline void **Record**(const std::unordered\_map<*MetricAttributes*, T, *AttributeHashGenerator*> &measurements, opentelemetry::common::SystemTimestamp) noexcept

inline void **RecordLong**(const std::unordered\_map<*MetricAttributes*, int64\_t, *AttributeHashGenerator*> &measurements, opentelemetry::common::SystemTimestamp observation\_time) noexcept override

inline void **RecordDouble**(const std::unordered\_map<*MetricAttributes*, double, *AttributeHashGenerator*> &measurements, opentelemetry::common::SystemTimestamp observation\_time) noexcept override

inline bool **Collect**(*CollectorHandle* \*collector, nostd::span<std::shared\_ptr<*CollectorHandle*>> collectors, opentelemetry::common::SystemTimestamp sdk\_start\_ts, opentelemetry::common::SystemTimestamp collection\_ts, nostd::function\_ref<bool(*MetricData*)> metric\_collection\_callback) noexcept override

### Class AsyncMultiMetricStorage

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_latest\_sdk\_include\_opentelemetry\_sdk\_metrics\_state\_multi\_metric\_storage.h

## Inheritance Relationships

### Base Type

- public opentelemetry::sdk::metrics::AsyncWritableMetricStorage

## Class Documentation

class **AsyncMultiMetricStorage** : public `opentelemetry::sdk::metrics::AsyncWritableMetricStorage`

### Public Functions

inline void **AddStorage**(std::shared\_ptr<`AsyncWritableMetricStorage`> storage)

inline void **RecordLong**(const std::unordered\_map<`MetricAttributes`, int64\_t, `AttributeHashGenerator`> &measurements, `opentelemetry::common::SystemTimestamp` observation\_time)  
noexcept override

inline void **RecordDouble**(const std::unordered\_map<`MetricAttributes`, double, `AttributeHashGenerator`> &measurements, `opentelemetry::common::SystemTimestamp` observation\_time)  
noexcept override

## Class AsyncWritableMetricStorage

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_state_metric_storage.h`

## Inheritance Relationships

### Derived Types

- public `opentelemetry::sdk::metrics::AsyncMetricStorage` (*Class `AsyncMetricStorage`*)
- public `opentelemetry::sdk::metrics::AsyncMultiMetricStorage` (*Class `AsyncMultiMetricStorage`*)
- public `opentelemetry::sdk::metrics::NoopAsyncWritableMetricStorage` (*Class `NoopAsyncWritableMetricStorage`*)

## Class Documentation

class **AsyncWritableMetricStorage**

Subclassed by `opentelemetry::sdk::metrics::AsyncMetricStorage`, `opentelemetry::sdk::metrics::AsyncMultiMetricStorage`, `opentelemetry::sdk::metrics::NoopAsyncWritableMetricStorage`

### Public Functions

**AsyncWritableMetricStorage**() = default

virtual **~AsyncWritableMetricStorage**() = default

virtual void **RecordLong**(const std::unordered\_map<`MetricAttributes`, int64\_t, `AttributeHashGenerator`> &measurements, `opentelemetry::common::SystemTimestamp` observation\_time)  
noexcept = 0

```
virtual void RecordDouble(const std::unordered_map<MetricAttributes, double, AttributeHashGenerator>
    &measurements, opentelemetry::common::SystemTimestamp observation_time)
    noexcept = 0
```

### Class *AttributeHashGenerator*

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_state_attributes_hashmap.h`

### Class Documentation

class ***AttributeHashGenerator***

#### Public Functions

```
inline size_t operator() (const MetricAttributes &attributes) const
```

### Class *AttributesHashMap*

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_state_attributes_hashmap.h`

### Class Documentation

class ***AttributesHashMap***

#### Public Functions

```
inline Aggregation *Get(const MetricAttributes &attributes) const
```

```
inline bool Has(const MetricAttributes &attributes) const
```

**Returns** check if key is present in hash

```
inline Aggregation *GetOrSetDefault(const MetricAttributes &attributes,
    std::function<std::unique_ptr<Aggregation>()>
    aggregation_callback)
```

**Returns** the pointer to value for given key if present. If not present, it uses the provided callback to generate value and store in the hash

```
inline void Set(const MetricAttributes &attributes, std::unique_ptr<Aggregation> value)
```

Set the value for given key, overwriting the value if already present

```
inline bool GetAllEntries(nostd::function_ref<bool(const MetricAttributes&, Aggregation&)> callback)
    const
```

Iterate the hash to yield key and value stored in hash.

```
inline size_t Size()
```

Return the size of hash.

## Class AttributesProcessor

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_view_attributes_processor.h`

## Inheritance Relationships

### Derived Types

- `public opentelemetry::sdk::metrics::DefaultAttributesProcessor` (*Class DefaultAttributesProcessor*)
- `public opentelemetry::sdk::metrics::FilteringAttributesProcessor` (*Class FilteringAttributesProcessor*)

## Class Documentation

### class AttributesProcessor

The *AttributesProcessor* is responsible for customizing which attribute(s) are to be reported as metrics dimension(s).

Subclassed by `opentelemetry::sdk::metrics::DefaultAttributesProcessor`, `opentelemetry::sdk::metrics::FilteringAttributesProcessor`

### Public Functions

virtual *MetricAttributes* **process**(const opentelemetry::common::KeyValueType &attributes) const  
noexcept = 0

virtual **~AttributesProcessor**() = default

## Class CollectorHandle

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_state_metric_collector.h`

## Inheritance Relationships

### Derived Type

- `public opentelemetry::sdk::metrics::MetricCollector` (*Class MetricCollector*)

## Class Documentation

class **CollectorHandle**

Subclassed by *opentelemetry::sdk::metrics::MetricCollector*

### Public Functions

**CollectorHandle**() = default

virtual **~CollectorHandle**() = default

virtual *AggregationTemporality* **GetAggregationTemporality**(*InstrumentType* instrument\_type) noexcept = 0

## Class DefaultAggregation

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_aggregation_default_aggregation.h`

## Class Documentation

class **DefaultAggregation**

### Public Static Functions

static inline std::unique\_ptr<*Aggregation*> **CreateAggregation**(const opentelemetry::sdk::metrics::*InstrumentDescriptor* &instrument\_descriptor, const *AggregationConfig* \*aggregation\_config)

static inline std::unique\_ptr<*Aggregation*> **CreateAggregation**(*AggregationType* aggregation\_type, *InstrumentDescriptor* instrument\_descriptor, const *AggregationConfig* \*aggregation\_config = nullptr)

static inline std::unique\_ptr<*Aggregation*> **CloneAggregation**(*AggregationType* aggregation\_type, *InstrumentDescriptor* instrument\_descriptor, const *Aggregation* &to\_copy)



## Class `DefaultAttributesProcessor`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_view_attributes_processor.h`

## Inheritance Relationships

### Base Type

- `public opentelemetry::sdk::metrics::AttributesProcessor` (*Class `AttributesProcessor`*)

## Class Documentation

class **DefaultAttributesProcessor** : public `opentelemetry::sdk::metrics::AttributesProcessor`  
*DefaultAttributesProcessor* returns copy of input instrument attributes without any modification.

## Class `DoubleCounter`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_sync_instruments.h`

## Inheritance Relationships

### Base Types

- `public opentelemetry::sdk::metrics::Synchronous`
- `public opentelemetry::metrics::Counter< double >`

## Class Documentation

class **DoubleCounter** : public `opentelemetry::sdk::metrics::Synchronous`, public `opentelemetry::metrics::Counter<double>`

### Public Functions

**DoubleCounter**(*InstrumentDescriptor* instrument\_descriptor, `std::unique_ptr<SyncWritableMetricStorage>` storage)

void **Add**(double value, const `opentelemetry::common::KeyValueIterable` &attributes) noexcept override

void **Add**(double value, const `opentelemetry::common::KeyValueIterable` &attributes, const `opentelemetry::context::Context` &context) noexcept override

void **Add**(double value) noexcept override

void **Add**(double value, const `opentelemetry::context::Context` &context) noexcept override

## Class DoubleHistogram

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_sync_instruments.h`

## Inheritance Relationships

### Base Types

- `public opentelemetry::sdk::metrics::Synchronous`
- `public opentelemetry::metrics::Histogram< double >` (*Template Class Histogram*)

## Class Documentation

class **DoubleHistogram** : public `opentelemetry::sdk::metrics::Synchronous`, public `opentelemetry::metrics::Histogram<double>`

### Public Functions

**DoubleHistogram**(*InstrumentDescriptor* instrument\_descriptor,  
std::unique\_ptr<*SyncWritableMetricStorage*> storage)

void **Record**(double value, const `opentelemetry::common::KeyValueIterable` &attributes, const `opentelemetry::context::Context` &context) noexcept override

void **Record**(double value, const `opentelemetry::context::Context` &context) noexcept override

## Class DoubleHistogramAggregation

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_aggregation_histogram_aggregation.h`

## Inheritance Relationships

### Base Type

- `public opentelemetry::sdk::metrics::Aggregation`

## Class Documentation

class **DoubleHistogramAggregation** : public opentelemetry::sdk::metrics::Aggregation

### Public Functions

**DoubleHistogramAggregation**(const *AggregationConfig* \*aggregation\_config = nullptr)

**DoubleHistogramAggregation**(*HistogramPointData*&&)

**DoubleHistogramAggregation**(const *HistogramPointData*&)

inline void **Aggregate**(int64\_t, const *PointAttributes*&) noexcept override

void **Aggregate**(double value, const *PointAttributes* &attributes = {}) noexcept override

std::unique\_ptr<Aggregation> **Merge**(const *Aggregation* &delta) const noexcept override

std::unique\_ptr<Aggregation> **Diff**(const *Aggregation* &next) const noexcept override

*PointType* **ToPoint**() const noexcept override

## Class DoubleLastValueAggregation

- Defined in file `__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_aggregation_lastvalue_aggregation.h`

## Inheritance Relationships

### Base Type

- public opentelemetry::sdk::metrics::Aggregation

## Class Documentation

class **DoubleLastValueAggregation** : public opentelemetry::sdk::metrics::Aggregation

### Public Functions

**DoubleLastValueAggregation**()

**DoubleLastValueAggregation**(*LastValuePointData*&&)

**DoubleLastValueAggregation**(const *LastValuePointData*&)

inline void **Aggregate**(int64\_t, const *PointAttributes*&) noexcept override

void **Aggregate**(double value, const *PointAttributes* &attributes = {}) noexcept override

virtual std::unique\_ptr<*Aggregation*> **Merge**(const *Aggregation* &delta) const noexcept override

virtual std::unique\_ptr<*Aggregation*> **Diff**(const *Aggregation* &next) const noexcept override

*PointType* **ToPoint**() const noexcept override

### Class DoubleSumAggregation

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_latest\_sdk\_include\_opentelemetry\_sdk\_metrics\_aggregation\_sum\_aggregation.h

### Inheritance Relationships

#### Base Type

- public opentelemetry::sdk::metrics::Aggregation

### Class Documentation

class **DoubleSumAggregation** : public opentelemetry::sdk::metrics::*Aggregation*

#### Public Functions

**DoubleSumAggregation**(bool is\_monotonic)

**DoubleSumAggregation**(*SumPointData*&&)

**DoubleSumAggregation**(const *SumPointData*&)

inline void **Aggregate**(int64\_t, const *PointAttributes*&) noexcept override

void **Aggregate**(double value, const *PointAttributes* &attributes = {}) noexcept override

std::unique\_ptr<*Aggregation*> **Merge**(const *Aggregation* &delta) const noexcept override

std::unique\_ptr<*Aggregation*> **Diff**(const *Aggregation* &next) const noexcept override

*PointType* **ToPoint**() const noexcept override

### Class DoubleUpDownCounter

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_latest\_sdk\_include\_opentelemetry\_sdk\_metrics\_sync\_instruments.h

## Inheritance Relationships

### Base Types

- `public opentelemetry::sdk::metrics::Synchronous`
- `public opentelemetry::metrics::UpDownCounter< double >` (*Template Class UpDownCounter*)

### Class Documentation

class **DoubleUpDownCounter** : public `opentelemetry::sdk::metrics::Synchronous`, public `opentelemetry::metrics::UpDownCounter<double>`

#### Public Functions

**DoubleUpDownCounter**(*InstrumentDescriptor* instrument\_descriptor, `std::unique_ptr<SyncWritableMetricStorage>` storage)

void **Add**(double value, const `opentelemetry::common::KeyValueIterable` &attributes) noexcept override

void **Add**(double value, const `opentelemetry::common::KeyValueIterable` &attributes, const `opentelemetry::context::Context` &context) noexcept override

void **Add**(double value) noexcept override

void **Add**(double value, const `opentelemetry::context::Context` &context) noexcept override

### Class DropAggregation

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_aggregation_drop_aggregation.h`

## Inheritance Relationships

### Base Type

- `public opentelemetry::sdk::metrics::Aggregation`

### Class Documentation

class **DropAggregation** : public `opentelemetry::sdk::metrics::Aggregation`

A null Aggregation which denotes no aggregation should occur.

### Public Functions

**DropAggregation**() = default

inline **DropAggregation**(const *DropPointData*&)

inline void **Aggregate**(int64\_t, const *PointAttributes*&) noexcept override

inline void **Aggregate**(double, const *PointAttributes*&) noexcept override

inline std::unique\_ptr<*Aggregation*> **Merge**(const *Aggregation*&) const noexcept override

inline std::unique\_ptr<*Aggregation*> **Diff**(const *Aggregation*&) const noexcept override

inline *PointType* **ToPoint**() const noexcept override

### Class DropPointData

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_data_point_data.h`

### Class Documentation

class **DropPointData**

#### Public Functions

**DropPointData**(*DropPointData*&&) = default

**DropPointData**(const *DropPointData*&) = default

**DropPointData**() = default

*DropPointData* &**operator=**(*DropPointData*&&) = default

### Class ExactPredicate

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_view_predicate.h`

### Inheritance Relationships

#### Base Type

- public `opentelemetry::sdk::metrics::Predicate`

## Class Documentation

class **ExactPredicate** : public opentelemetry::sdk::metrics::Predicate

### Public Functions

inline **ExactPredicate**(opentelemetry::nostd::string\_view pattern)

inline bool **Match**(opentelemetry::nostd::string\_view str) const noexcept override

## Class ExemplarData

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_data_exemplar_data.h`

## Class Documentation

class **ExemplarData**

A sample input measurement.

Exemplars also hold information about the environment when the measurement was recorded, for example the span and trace ID of the active span when the exemplar was recorded.

### Public Functions

inline *MetricAttributes* **GetFilteredAttributes**()

The set of key/value pairs that were filtered out by the aggregator, but recorded alongside the original measurement. Only key/value pairs that were filtered out by the aggregator should be included

inline opentelemetry::common::SystemTimestamp **GetEpochNanos**()

Returns the timestamp in nanos when measurement was collected.

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

Returns the SpanContext associated with this exemplar. If the exemplar was not recorded inside a sampled trace, the Context will be invalid.

### Public Static Functions

static inline *ExemplarData* **Create**(std::shared\_ptr<trace::SpanContext> context, const opentelemetry::common::SystemTimestamp &timestamp, const *PointDataAttributes* &point\_data\_attr)

static inline *PointType* **CreateSumPointData**(*ValueType* value)

static inline *PointType* **CreateLastValuePointData**(*ValueType* value)

static inline *PointType* **CreateDropPointData**()

## Class ExemplarFilter

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_exemplar_filter.h`

## Inheritance Relationships

### Derived Types

- `public opentelemetry::sdk::metrics::AlwaysSampleFilter` (*Class AlwaysSampleFilter*)
- `public opentelemetry::sdk::metrics::NeverSampleFilter` (*Class NeverSampleFilter*)
- `public opentelemetry::sdk::metrics::WithTraceSampleFilter` (*Class WithTraceSampleFilter*)

## Class Documentation

### class **ExemplarFilter**

Exemplar filters are used to pre-filter measurements before attempting to store them in a reservoir.

Subclassed by `opentelemetry::sdk::metrics::AlwaysSampleFilter`, `opentelemetry::sdk::metrics::NeverSampleFilter`, `opentelemetry::sdk::metrics::WithTraceSampleFilter`

### Public Functions

virtual bool **ShouldSampleMeasurement**(int64\_t value, const *MetricAttributes* &attributes, const opentelemetry::context::Context &context) noexcept = 0

virtual bool **ShouldSampleMeasurement**(double value, const *MetricAttributes* &attributes, const opentelemetry::context::Context &context) noexcept = 0

virtual ~**ExemplarFilter**() = default

### Public Static Functions

static std::shared\_ptr<*ExemplarFilter*> **GetNeverSampleFilter**() noexcept

static std::shared\_ptr<*ExemplarFilter*> **GetAlwaysSampleFilter**() noexcept

static std::shared\_ptr<*ExemplarFilter*> **GetWithTraceSampleFilter**() noexcept

## Class ExemplarReservoir

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_exemplar_reservoir.h`



## Inheritance Relationships

### Derived Types

- `public opentelemetry::sdk::metrics::FilteredExemplarReservoir` (Class *FilteredExemplarReservoir*)
- `public opentelemetry::sdk::metrics::FixedSizeExemplarReservoir` (Class *FixedSizeExemplarReservoir*)
- `public opentelemetry::sdk::metrics::NoExemplarReservoir` (Class *NoExemplarReservoir*)

### Class Documentation

#### class **ExemplarReservoir**

An interface for an exemplar reservoir of samples.

This represents a reservoir for a specific “point” of metric data.

Subclassed by `opentelemetry::sdk::metrics::FilteredExemplarReservoir`, `opentelemetry::sdk::metrics::FixedSizeExemplarReservoir`, `opentelemetry::sdk::metrics::NoExemplarReservoir`

#### Public Functions

virtual `~ExemplarReservoir()` = default

virtual void **OfferMeasurement**(int64\_t value, const *MetricAttributes* &attributes, const `opentelemetry::context::Context` &context, const `opentelemetry::common::SystemTimestamp` &timestamp) noexcept = 0

Offers a long measurement to be sampled.

virtual void **OfferMeasurement**(double value, const *MetricAttributes* &attributes, const `opentelemetry::context::Context` &context, const `opentelemetry::common::SystemTimestamp` &timestamp) noexcept = 0

Offers a double measurement to be sampled.

virtual `std::vector<std::shared_ptr<ExemplarData>>` **CollectAndReset**(const *MetricAttributes* &pointAttributes) noexcept = 0

Builds vector of Exemplars for exporting from the current reservoir.

Additionally, clears the reservoir for the next sampling period.

**Parameters** `pointAttributes` – the Attributes associated with the metric point. Exemplar-Datas should filter these out of their final data state.

**Returns** A vector of sampled exemplars for this point. Implementers are expected to filter out `pointAttributes` from the original recorded attributes.

## Public Static Functions

```
static nstd::shared_ptr<ExemplarReservoir> GetFilteredExemplarReservoir(std::shared_ptr<ExemplarFilter>
                                                                    filter,
                                                                    std::shared_ptr<ExemplarReservoir>
                                                                    reservoir)
```

```
static nstd::shared_ptr<ExemplarReservoir> GetHistogramExemplarReservoir(size_t size,
                                                                    std::shared_ptr<ReservoirCellSelector>
                                                                    reservoir_cell_selector,
                                                                    std::shared_ptr<ExemplarData>
                                                                    (ReservoirCell::*
                                                                    map_and_reset_cell)(const
                                                                    com-
                                                                    mon::OrderedAttributeMap
                                                                    &attributes))
```

```
static nstd::shared_ptr<ExemplarReservoir> GetNoExemplarReservoir()
```

## Class FilteredExemplarReservoir

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_latest\_sdk\_include\_opentelemetry\_sdk\_metrics\_exemplar\_filtered\_exemplar\_reservoir.h

## Inheritance Relationships

### Base Type

- public opentelemetry::sdk::metrics::ExemplarReservoir (Class *ExemplarReservoir*)

## Class Documentation

```
class FilteredExemplarReservoir : public opentelemetry::sdk::metrics::ExemplarReservoir
```

### Public Functions

```
inline FilteredExemplarReservoir(std::shared_ptr<ExemplarFilter> filter,
                                std::shared_ptr<ExemplarReservoir> reservoir)
```

```
inline void OfferMeasurement(int64_t value, const MetricAttributes &attributes, const
                                opentelemetry::context::Context &context, const
                                opentelemetry::common::SystemTimestamp &timestamp) noexcept override
```

```
inline void OfferMeasurement(double value, const MetricAttributes &attributes, const
                                opentelemetry::context::Context &context, const
                                opentelemetry::common::SystemTimestamp &timestamp) noexcept override
```

```
inline std::vector<std::shared_ptr<ExemplarData>> CollectAndReset(const MetricAttributes
                                                                    &pointAttributes) noexcept override
```

## Class `FilteringAttributesProcessor`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_view_attributes_processor.h`

## Inheritance Relationships

### Base Type

- `public opentelemetry::sdk::metrics::AttributesProcessor` (*Class `AttributesProcessor`*)

## Class Documentation

class **`FilteringAttributesProcessor`** : `public opentelemetry::sdk::metrics::AttributesProcessor`

*FilteringAttributesProcessor* filters by allowed attribute names and drops any names that are not in the allow list.

### Public Functions

```
inline FilteringAttributesProcessor(const std::unordered_map<std::string, bool>
                                   allowed_attribute_keys = {})
```

```
inline virtual MetricAttributes process(const opentelemetry::common::KeyValueIterable &attributes) const
    noexcept override
```

## Class `FixedSizeExemplarReservoir`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_exemplar_fixed_size_exemplar_reservoir.h`

## Inheritance Relationships

### Base Type

- `public opentelemetry::sdk::metrics::ExemplarReservoir` (*Class `ExemplarReservoir`*)

### Derived Type

- `public opentelemetry::sdk::metrics::HistogramExemplarReservoir` (*Class `HistogramExemplarReservoir`*)

## Class Documentation

class **FixedSizeExemplarReservoir** : public opentelemetry::sdk::metrics::*ExemplarReservoir*

Subclassed by *opentelemetry::sdk::metrics::HistogramExemplarReservoir*

### Public Functions

```
inline FixedSizeExemplarReservoir(size_t size, std::shared_ptr<ReservoirCellSelector>
    reservoir_cell_selector, std::shared_ptr<ExemplarData>
    (ReservoirCell::* map_and_reset_cell)(const
    common::OrderedAttributeMap &attributes))
```

```
inline void OfferMeasurement(int64_t value, const MetricAttributes &attributes, const
    opentelemetry::context::Context &context, const
    opentelemetry::common::SystemTimestamp&) noexcept override
```

```
inline void OfferMeasurement(double value, const MetricAttributes &attributes, const
    opentelemetry::context::Context &context, const
    opentelemetry::common::SystemTimestamp&) noexcept override
```

```
inline std::vector<std::shared_ptr<ExemplarData>> CollectAndReset(const MetricAttributes
    &pointAttributes) noexcept override
```

## Class HistogramAggregationConfig

- Defined in file `__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_aggregation_aggregation_config.h`

## Inheritance Relationships

### Base Type

- public opentelemetry::sdk::metrics::AggregationConfig

## Class Documentation

class **HistogramAggregationConfig** : public opentelemetry::sdk::metrics::*AggregationConfig*

## Public Members

std::vector<double> **boundaries\_**

bool **record\_min\_max\_** = true

## Class HistogramExemplarReservoir

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_exemplar_histogram_exemplar_reservoir.h`

## Nested Relationships

### Nested Types

- *Class HistogramExemplarReservoir::HistogramCellSelector*

## Inheritance Relationships

### Base Type

- public `opentelemetry::sdk::metrics::FixedSizeExemplarReservoir`

## Class Documentation

class **HistogramExemplarReservoir** : public `opentelemetry::sdk::metrics::FixedSizeExemplarReservoir`

### Public Functions

```
inline HistogramExemplarReservoir(size_t size, std::shared_ptr<ReservoirCellSelector>
    reservoir_cell_selector, std::shared_ptr<ExemplarData>
    (ReservoirCell::* map_and_reset_cell)(const
    common::OrderedAttributeMap &attributes))
```

### Public Static Functions

```
static inline std::shared_ptr<ReservoirCellSelector> GetHistogramCellSelector(const std::vector<double>
    &boundaries =
    std::vector<double>{1.0,
    2.0, 3.0, 4.0, 5.0})
```

class **HistogramCellSelector** : public `ReservoirCellSelector`

### Public Functions

inline **HistogramCellSelector**(const std::vector<double> &boundaries)

inline int **ReservoirCellIndexFor**(const std::vector<*ReservoirCell*> &cells, int64\_t value, const *MetricAttributes* &attributes, const opentelemetry::context::Context &context) override

inline int **ReservoirCellIndexFor**(const std::vector<*ReservoirCell*>&, double value, const *MetricAttributes*&, const opentelemetry::context::Context&) override

### Class HistogramExemplarReservoir::HistogramCellSelector

- Defined in file `__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_exemplar_histogram_exemplar_reservoir.h`

### Nested Relationships

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

### Inheritance Relationships

#### Base Type

- public `ReservoirCellSelector`

### Class Documentation

class **HistogramCellSelector** : public `ReservoirCellSelector`

#### Public Functions

inline **HistogramCellSelector**(const std::vector<double> &boundaries)

inline int **ReservoirCellIndexFor**(const std::vector<*ReservoirCell*> &cells, int64\_t value, const *MetricAttributes* &attributes, const opentelemetry::context::Context &context) override

inline int **ReservoirCellIndexFor**(const std::vector<*ReservoirCell*>&, double value, const *MetricAttributes*&, const opentelemetry::context::Context&) override

## Class HistogramPointData

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_data_point_data.h`

## Class Documentation

class **HistogramPointData**

### Public Functions

**HistogramPointData**(*HistogramPointData*&&) = default

*HistogramPointData* &**operator=**(*HistogramPointData*&&) = default

**HistogramPointData**(const *HistogramPointData*&) = default

**HistogramPointData**() = default

inline **HistogramPointData**(std::vector<double> &boundaries)

### Public Members

std::vector<double> **boundaries\_** = {}

*ValueType* **sum\_** = {}

*ValueType* **min\_** = {}

*ValueType* **max\_** = {}

std::vector<uint64\_t> **counts\_** = {}

uint64\_t **count\_** = {}

bool **record\_min\_max\_** = true

## Class InstrumentMetaDataValidator

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_instrument_metadata_validator.h`

## Class Documentation

class **InstrumentMetaDataValidator**

### Public Functions

**InstrumentMetaDataValidator**()

bool **ValidateName**(nostd::string\_view name) const

bool **ValidateUnit**(nostd::string\_view unit) const

bool **ValidateDescription**(nostd::string\_view description) const

## Class InstrumentSelector

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_view_instrument_selector.h`

## Class Documentation

class **InstrumentSelector**

### Public Functions

inline **InstrumentSelector**(opentelemetry::sdk::metrics::InstrumentType instrument\_type, nostd::string\_view name)

inline const opentelemetry::sdk::metrics::Predicate \***GetNameFilter**() const

inline InstrumentType **GetInstrumentType**()

## Class LastValuePointData

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_data_point_data.h`

## Class Documentation

class **LastValuePointData**



### Public Functions

**LastValuePointData**(*LastValuePointData*&&) = default

**LastValuePointData**(const *LastValuePointData*&) = default

*LastValuePointData* &**operator=**(*LastValuePointData*&&) = default

**LastValuePointData**() = default

### Public Members

*ValueType* **value\_** = {}

bool **is\_lastvalue\_valid\_** = {}

opentelemetry::common::SystemTimestamp **sample\_ts\_** = {}

### Template Class LongCounter

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_latest\_sdk\_include\_opentelemetry\_sdk\_metrics\_sync\_instruments.h

### Inheritance Relationships

#### Base Types

- public opentelemetry::sdk::metrics::Synchronous
- public opentelemetry::metrics::Counter< T >

### Class Documentation

template<typename T>

class **LongCounter** : public opentelemetry::sdk::metrics::Synchronous, public opentelemetry::metrics::Counter<T>

#### Public Functions

inline **LongCounter**(*InstrumentDescriptor* instrument\_descriptor,  
std::unique\_ptr<*SyncWritableMetricStorage*> storage)

inline void **Add**(*T* value, const opentelemetry::common::KeyValueType &attributes) noexcept override

inline void **Add**(*T* value, const opentelemetry::common::KeyValueType &attributes, const  
opentelemetry::context::Context &context) noexcept override

inline void **Add**(*T* value) noexcept override

inline void **Add**(*T* value, const opentelemetry::context::Context &context) noexcept override

## Template Class LongHistogram

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_sync_instruments.h`

## Inheritance Relationships

### Base Types

- `public opentelemetry::sdk::metrics::Synchronous`
- `public opentelemetry::metrics::Histogram< T >` (*Template Class Histogram*)

## Class Documentation

```
template<typename T>
```

```
class LongHistogram : public opentelemetry::sdk::metrics::Synchronous, public  
opentelemetry::metrics::Histogram<T>
```

### Public Functions

```
inline LongHistogram(InstrumentDescriptor instrument_descriptor,  
std::unique_ptr<SyncWritableMetricStorage> storage)
```

```
inline void Record(T value, const opentelemetry::common::KeyValueIterable &attributes, const  
opentelemetry::context::Context &context) noexcept override
```

```
inline void Record(T value, const opentelemetry::context::Context &context) noexcept override
```

## Class LongHistogramAggregation

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_aggregation_histogram_aggregation.h`

## Inheritance Relationships

### Base Type

- `public opentelemetry::sdk::metrics::Aggregation`

## Class Documentation

class **LongHistogramAggregation** : public opentelemetry::sdk::metrics::Aggregation

### Public Functions

**LongHistogramAggregation**(const *AggregationConfig* \*aggregation\_config = nullptr)

**LongHistogramAggregation**(*HistogramPointData*&&)

**LongHistogramAggregation**(const *HistogramPointData*&)

void **Aggregate**(int64\_t value, const *PointAttributes* &attributes = {}) noexcept override

inline void **Aggregate**(double, const *PointAttributes*&) noexcept override

std::unique\_ptr<Aggregation> **Merge**(const *Aggregation* &delta) const noexcept override

std::unique\_ptr<Aggregation> **Diff**(const *Aggregation* &next) const noexcept override

*PointType* **ToPoint**() const noexcept override

## Class LongLastValueAggregation

- Defined in file `__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_aggregation_lastvalue_aggregation.h`

## Inheritance Relationships

### Base Type

- public opentelemetry::sdk::metrics::Aggregation

## Class Documentation

class **LongLastValueAggregation** : public opentelemetry::sdk::metrics::Aggregation

### Public Functions

**LongLastValueAggregation**()

**LongLastValueAggregation**(*LastValuePointData*&&)

**LongLastValueAggregation**(const *LastValuePointData*&)

void **Aggregate**(int64\_t value, const *PointAttributes* &attributes = {}) noexcept override

inline void **Aggregate**(double, const *PointAttributes*&) noexcept override

std::unique\_ptr<Aggregation> **Merge**(const *Aggregation* &delta) const noexcept override

std::unique\_ptr<Aggregation> **Diff**(const *Aggregation* &next) const noexcept override

*PointType* **ToPoint**() const noexcept override

## Class LongSumAggregation

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_latest\_sdk\_include\_opentelemetry\_sdk\_metrics\_aggregation\_sum\_aggregation.h

## Inheritance Relationships

### Base Type

- public opentelemetry::sdk::metrics::Aggregation

## Class Documentation

class **LongSumAggregation** : public opentelemetry::sdk::metrics::Aggregation

### Public Functions

**LongSumAggregation**(bool is\_monotonic)

**LongSumAggregation**(*SumPointData*&&)

**LongSumAggregation**(const *SumPointData*&)

void **Aggregate**(int64\_t value, const *PointAttributes* &attributes = {}) noexcept override

inline void **Aggregate**(double, const *PointAttributes*&) noexcept override

std::unique\_ptr<Aggregation> **Merge**(const *Aggregation* &delta) const noexcept override

std::unique\_ptr<Aggregation> **Diff**(const *Aggregation* &next) const noexcept override

*PointType* **ToPoint**() const noexcept override

## Class LongUpDownCounter

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_latest\_sdk\_include\_opentelemetry\_sdk\_metrics\_sync\_instruments.h

## Inheritance Relationships

### Base Types

- `public opentelemetry::sdk::metrics::Synchronous`
- `public opentelemetry::metrics::UpDownCounter< int64_t >` (*Template Class UpDownCounter*)

### Class Documentation

class **LongUpDownCounter** : public `opentelemetry::sdk::metrics::Synchronous`, public `opentelemetry::metrics::UpDownCounter<int64_t>`

#### Public Functions

**LongUpDownCounter**(*InstrumentDescriptor* instrument\_descriptor,  
std::unique\_ptr<*SyncWritableMetricStorage*> storage)

void **Add**(int64\_t value, const `opentelemetry::common::KeyValueIterable` &attributes) noexcept override

void **Add**(int64\_t value, const `opentelemetry::common::KeyValueIterable` &attributes, const `opentelemetry::context::Context` &context) noexcept override

void **Add**(int64\_t value) noexcept override

void **Add**(int64\_t value, const `opentelemetry::context::Context` &context) noexcept override

### Class MatchEverythingPattern

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_view_predicate.h`

## Inheritance Relationships

### Base Type

- `public opentelemetry::sdk::metrics::Predicate`

### Class Documentation

class **MatchEverythingPattern** : public `opentelemetry::sdk::metrics::Predicate`

## Class MatchNothingPattern

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_latest\_sdk\_include\_opentelemetry\_sdk\_metrics\_view\_predicate.h

## Inheritance Relationships

### Base Type

- public opentelemetry::sdk::metrics::Predicate

## Class Documentation

class **MatchNothingPattern** : public opentelemetry::sdk::metrics::Predicate

## Class Meter

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_latest\_sdk\_include\_opentelemetry\_sdk\_metrics\_meter.h

## Inheritance Relationships

### Base Type

- public opentelemetry::metrics::Meter (Class Meter)

## Class Documentation

class **Meter** : public opentelemetry::metrics::Meter

### Public Functions

```
explicit Meter(std::weak_ptr<sdk::metrics::MeterContext> meter_context,  
              std::unique_ptr<opentelemetry::sdk::instrumentationscope::InstrumentationScope> scope =  
              opentelemetry::sdk::instrumentationscope::InstrumentationScope::Create("")) noexcept
```

Construct a new Meter with the given pipeline.

```
nostd::unique_ptr<opentelemetry::metrics::Counter<uint64_t>> CreateUInt64Counter(nostd::string_view  
                                                                                name,  
                                                                                nostd::string_view  
                                                                                description = "",  
                                                                                nostd::string_view  
                                                                                unit = "") noexcept  
                                                                                override
```

```

nostd::unique_ptr<opentelemetry::metrics::Counter<double>> CreateDoubleCounter(nostd::string_view
                                                                    name,
                                                                    nostd::string_view
                                                                    description = "",
                                                                    nostd::string_view
                                                                    unit = "") noexcept
                                                                    override

nostd::shared_ptr<opentelemetry::metrics::ObservableInstrument> CreateInt64ObservableCounter(nostd::string_view
                                                                                             name,
                                                                                             nostd::string_view
                                                                                             de-
                                                                                             scrip-
                                                                                             tion
                                                                                             = "",
                                                                                             nostd::string_view
                                                                                             unit
                                                                                             = "")
                                                                                             noex-
                                                                                             cept
                                                                                             override

nostd::shared_ptr<opentelemetry::metrics::ObservableInstrument> CreateDoubleObservableCounter(nostd::string_view
                                                                                             name,
                                                                                             nostd::string_view
                                                                                             de-
                                                                                             scrip-
                                                                                             tion
                                                                                             =
                                                                                             "",
                                                                                             nostd::string_view
                                                                                             unit
                                                                                             =
                                                                                             "")
                                                                                             noex-
                                                                                             cept
                                                                                             override

nostd::unique_ptr<opentelemetry::metrics::Histogram<uint64_t>> CreateUInt64Histogram(nostd::string_view
                                                                                       name,
                                                                                       nostd::string_view
                                                                                       description =
                                                                                       "",
                                                                                       nostd::string_view
                                                                                       unit = "")
                                                                                       noexcept
                                                                                       override

nostd::unique_ptr<opentelemetry::metrics::Histogram<double>> CreateDoubleHistogram(nostd::string_view
                                                                                       name,
                                                                                       nostd::string_view
                                                                                       description = "",
                                                                                       nostd::string_view
                                                                                       unit = "")
                                                                                       noexcept
                                                                                       override

```

```
nostd::shared_ptr<opentelemetry::metrics::ObservableInstrument> CreateInt64ObservableGauge(nostd::string_view
                                                                                          name,
                                                                                          nostd::string_view
                                                                                          de-
                                                                                          scrip-
                                                                                          tion =
                                                                                          "",
                                                                                          nostd::string_view
                                                                                          unit =
                                                                                          "")
                                                                                          noexcept
                                                                                          override

nostd::shared_ptr<opentelemetry::metrics::ObservableInstrument> CreateDoubleObservableGauge(nostd::string_view
                                                                                          name,
                                                                                          nostd::string_view
                                                                                          de-
                                                                                          scrip-
                                                                                          tion =
                                                                                          "",
                                                                                          nostd::string_view
                                                                                          unit =
                                                                                          "")
                                                                                          noexcept
                                                                                          override

nostd::unique_ptr<opentelemetry::metrics::UpDownCounter<int64_t>> CreateInt64UpDownCounter(nostd::string_view
                                                                                          name,
                                                                                          nostd::string_view
                                                                                          de-
                                                                                          scrip-
                                                                                          tion
                                                                                          = "",
                                                                                          nostd::string_view
                                                                                          unit
                                                                                          = "")
                                                                                          noexcept
                                                                                          override
```



```

nostd::unique_ptr<opentelemetry::metrics::UpDownCounter<double>> CreateDoubleUpDownCounter(nostd::string_view
name,
nostd::string_view
de-
scrip-
tion
= "",
nostd::string_view
unit
=
"")
noex-
cept
override

```

```

nostd::shared_ptr<opentelemetry::metrics::ObservableInstrument> CreateInt64ObservableUpDownCounter(nostd::string_v
name,
nostd::string_v
de-
scrip-
tion
=
"",
nostd::string_v
unit
=
"")
noex-
cept
override

```

```

nostd::shared_ptr<opentelemetry::metrics::ObservableInstrument> CreateDoubleObservableUpDownCounter(nostd::string_v
name,
nostd::string_v
de-
scrip-
tion
=
"",
nostd::string_v
unit
=
"")
noex-
cept
override

```

```

const sdk::instrumentationscope::InstrumentationScope *GetInstrumentationScope() const noexcept
Returns the associated instrumentation scope

```

```

inline const sdk::instrumentationscope::InstrumentationScope *GetInstrumentationLibrary() const
noexcept

```

```

std::vector<MetricData> Collect(CollectorHandle *collector, opentelemetry::common::SystemTimestamp
collect_ts) noexcept

```

collect metrics across all the instruments configured for the meter

## Class MeterContext

- Defined in file `_home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_meter_context.h`

## Inheritance Relationships

### Base Type

- `public std::enable_shared_from_this< MeterContext >`

## Class Documentation

class **MeterContext** : public `std::enable_shared_from_this<MeterContext>`

A class which stores the MeterProvider context.

### Public Functions

**MeterContext**(`std::unique_ptr<ViewRegistry> views = std::unique_ptr<ViewRegistry>(new ViewRegistry()),  
opentelemetry::sdk::resource::Resource resource =  
opentelemetry::sdk::resource::Resource::Create({})`) noexcept

Initialize a new meter provider

#### Parameters

- **readers** – The readers to be configured with meter context.
- **views** – The views to be configured with meter context.
- **resource** – The resource for this meter context.

`const opentelemetry::sdk::resource::Resource &GetResource()` const noexcept

Obtain the resource associated with this meter context.

**Returns** The resource for this meter context

`ViewRegistry *GetViewRegistry()` const noexcept

Obtain the *View* Registry configured

**Returns** The reference to view registry

`bool ForEachMeter`(`nostd::function_ref<bool(std::shared_ptr<Meter> &meter)> callback`) noexcept

NOTE - INTERNAL method, can change in future. Process callback for each meter in thread-safe manner

`nostd::span<std::shared_ptr<Meter>> GetMeters()` noexcept

NOTE - INTERNAL method, can change in future. Get the configured meters. This method is NOT thread safe, and only called through MeterProvider

`nostd::span<std::shared_ptr<CollectorHandle>> GetCollectors()` noexcept

Obtain the configured collectors.

opentelemetry::common::SystemTimestamp **GetSDKStartTime**() noexcept  
GET SDK Start time

void **AddMetricReader**(std::shared\_ptr<MetricReader> reader) noexcept

Attaches a metric reader to list of configured readers for this Meter context.

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

**Parameters reader** – The metric reader for this meter context. This must not be a nullptr.

void **AddView**(std::unique\_ptr<InstrumentSelector> instrument\_selector, std::unique\_ptr<MeterSelector> meter\_selector, std::unique\_ptr<View> view) noexcept

Attaches a *View* to list of configured Views for this Meter context.

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

**Parameters view** – The Views for this meter context. This must not be a nullptr.

void **AddMeter**(std::shared\_ptr<Meter> meter)

NOTE - INTERNAL method, can change in future. Adds a meter to the list of configured meters in thread safe manner.

**Parameters meter** –

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

Force all active Collectors to flush any buffered meter data within the given timeout.

bool **Shutdown**() noexcept

Shutdown the Collectors associated with this meter provider.

## Class MeterProvider

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_latest\_sdk\_include\_opentelemetry\_sdk\_metrics\_meter\_provider.h

## Inheritance Relationships

### Base Type

- public opentelemetry::metrics::MeterProvider (*Class MeterProvider*)

## Class Documentation

class **MeterProvider** : public opentelemetry::metrics::*MeterProvider*

### Public Functions

**MeterProvider**(std::unique\_ptr<*ViewRegistry*> views = std::unique\_ptr<*ViewRegistry*>(new *ViewRegistry*()), sdk::resource::Resource resource = sdk::resource::Resource::Create({})) noexcept

Initialize a new meter provider

#### Parameters

- **views** – The views for this meter provider
- **resource** – The resources for this meter provider.

explicit **MeterProvider**(std::shared\_ptr<sdk::metrics::*MeterContext*> context) noexcept

Initialize a new meter provider with a specified context

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

nostd::shared\_ptr<opentelemetry::metrics::*Meter*> **GetMeter**(nostd::string\_view name, nostd::string\_view version = "", nostd::string\_view schema\_url = "") noexcept override

const sdk::resource::Resource &**GetResource**() const noexcept

Obtain the resource associated with this meter provider.

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

void **AddMetricReader**(std::shared\_ptr<*MetricReader*> reader) noexcept

Attaches a metric reader to list of configured readers for this Meter providers.

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

**Parameters reader** – The metric reader for this meter provider. This must not be a nullptr.

void **AddView**(std::unique\_ptr<*InstrumentSelector*> instrument\_selector, std::unique\_ptr<*MeterSelector*> meter\_selector, std::unique\_ptr<*View*> view) noexcept

Attaches a *View* to list of configured Views for this Meter provider.

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

**Parameters view** – The Views for this meter provider. This must not be a nullptr.

bool **Shutdown**() noexcept

Shutdown the meter provider.

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

Force flush the meter provider.

**~MeterProvider**() override

## Class MeterSelector

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_view_meter_selector.h`

## Class Documentation

class **MeterSelector**

### Public Functions

inline **MeterSelector**(opentelemetry::nostd::string\_view name, opentelemetry::nostd::string\_view version, opentelemetry::nostd::string\_view schema)

inline const opentelemetry::sdk::metrics::Predicate \***GetNameFilter**() const

inline const opentelemetry::sdk::metrics::Predicate \***GetVersionFilter**() const

inline const opentelemetry::sdk::metrics::Predicate \***GetSchemaFilter**() const

## Class MetricCollector

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_state_metric_collector.h`

## Inheritance Relationships

### Base Types

- public opentelemetry::sdk::metrics::MetricProducer (*Class MetricProducer*)
- public opentelemetry::sdk::metrics::CollectorHandle

## Class Documentation

class **MetricCollector** : public opentelemetry::sdk::metrics::MetricProducer, public opentelemetry::sdk::metrics::CollectorHandle

An internal opaque interface that the *MetricReader* receives as *MetricProducer*. It acts as the storage key to the internal metric stream state for each *MetricReader*.

## Public Functions

**MetricCollector**(*MetricContext* \*context, std::shared\_ptr<*MetricReader*> metric\_reader)

~**MetricCollector**() override = default

*AggregationTemporality* **GetAggregationTemporality**(*InstrumentType* instrument\_type) noexcept override

virtual bool **Collect**(nostd::function\_ref<bool(*ResourceMetrics* &metric\_data)> callback) noexcept override

The callback to be called for each metric exporter. This will only be those metrics that have been produced since the last time this method was called.

**Returns** a status of completion of method.

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

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

## Class MetricData

- Defined in file\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_latest\_sdk\_include\_opentelemetry\_sdk\_metrics\_data\_metric\_data.h

## Class Documentation

class **MetricData**

### Public Members

*InstrumentDescriptor* instrument\_descriptor

*AggregationTemporality* aggregation\_temporality

opentelemetry::common::SystemTimestamp start\_ts

opentelemetry::common::SystemTimestamp end\_ts

std::vector<*PointDataAttributes*> point\_data\_attr\_

## Class MetricProducer

- Defined in file\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_latest\_sdk\_include\_opentelemetry\_sdk\_metrics\_export\_metric\_producer.h

## Inheritance Relationships

### Derived Type

- `public opentelemetry::sdk::metrics::MetricCollector` (*Class `MetricCollector`*)

### Class Documentation

#### class `MetricProducer`

*MetricProducer* is the interface that is used to make metric data available to the OpenTelemetry exporters. Implementations should be stateful, in that each call to `Collect` will return any metric generated since the last call was made.

Implementations must be thread-safe.

Subclassed by *`opentelemetry::sdk::metrics::MetricCollector`*

#### Public Functions

`MetricProducer()` = default

virtual `~MetricProducer()` = default

virtual bool `Collect`(`nostd::function_ref<bool(ResourceMetrics &metric_data)>` callback) noexcept = 0

The callback to be called for each metric exporter. This will only be those metrics that have been produced since the last time this method was called.

**Returns** a status of completion of method.

#### Class `MetricReader`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_metric_reader.h`

## Inheritance Relationships

### Derived Type

- `public opentelemetry::sdk::metrics::PeriodicExportingMetricReader` (*Class `PeriodicExportingMetricReader`*)

## Class Documentation

### class **MetricReader**

*MetricReader* defines the interface to collect metrics from SDK

Subclassed by *opentelemetry::sdk::metrics::PeriodicExportingMetricReader*

#### Public Functions

##### **MetricReader()**

void **SetMetricProducer**(*MetricProducer* \*metric\_producer)

bool **Collect**(nostd::function\_ref<bool(*ResourceMetrics* &metric\_data)> callback) noexcept

Collect the metrics from SDK.

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

virtual *AggregationTemporality* **GetAggregationTemporality**(*InstrumentType* instrument\_type) const  
noexcept = 0

Get the AggregationTemporality for given Instrument Type for this reader.

**Returns** AggregationTemporality

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

Shutdown the meter reader.

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

Force flush the metric read by the reader.

virtual **~MetricReader**() = default

#### Protected Functions

bool **IsShutdown**() const noexcept

### Class **MetricStorage**

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_state_metric_storage.h`

## Inheritance Relationships

### Derived Types

- public `opentelemetry::sdk::metrics::AsyncMetricStorage` (*Class AsyncMetricStorage*)
- public `opentelemetry::sdk::metrics::NoopMetricStorage` (*Class NoopMetricStorage*)
- public `opentelemetry::sdk::metrics::SyncMetricStorage` (*Class SyncMetricStorage*)



## Class Documentation

### class **MetricStorage**

Subclassed by *opentelemetry::sdk::metrics::AsyncMetricStorage*, *opentelemetry::sdk::metrics::NoopMetricStorage*, *opentelemetry::sdk::metrics::SyncMetricStorage*

#### Public Functions

**MetricStorage**() = default

virtual **~MetricStorage**() = default

virtual bool **Collect**(*CollectorHandle* \*collector, nostd::span<std::shared\_ptr<*CollectorHandle*>> collectors, opentelemetry::common::SystemTimestamp sdk\_start\_ts, opentelemetry::common::SystemTimestamp collection\_ts, nostd::function\_ref<bool(*MetricData*)> callback) noexcept = 0

### Class **NeverSampleFilter**

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_exemplar_never_sample_filter.h`

## Inheritance Relationships

### Base Type

- public opentelemetry::sdk::metrics::ExemplarFilter (*Class ExemplarFilter*)

## Class Documentation

class **NeverSampleFilter** : public opentelemetry::sdk::metrics::ExemplarFilter

#### Public Functions

inline bool **ShouldSampleMeasurement**(int64\_t, const *MetricAttributes*&, const opentelemetry::context::Context&) noexcept override

inline bool **ShouldSampleMeasurement**(double, const *MetricAttributes*&, const opentelemetry::context::Context&) noexcept override

explicit **NeverSampleFilter**() = default

## Class NoExemplarReservoir

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_exemplar_no_exemplar_reservoir.h`

## Inheritance Relationships

### Base Type

- `public opentelemetry::sdk::metrics::ExemplarReservoir` (*Class ExemplarReservoir*)

## Class Documentation

class **NoExemplarReservoir** : public `opentelemetry::sdk::metrics::ExemplarReservoir`

### Public Functions

`inline void OfferMeasurement`(`int64_t`, const *MetricAttributes*&, const `opentelemetry::context::Context`&, const `opentelemetry::common::SystemTimestamp`&) noexcept override

`inline void OfferMeasurement`(`double`, const *MetricAttributes*&, const `opentelemetry::context::Context`&, const `opentelemetry::common::SystemTimestamp`&) noexcept override

`inline std::vector<std::shared_ptr<ExemplarData>>` **CollectAndReset**(const *MetricAttributes*&) noexcept override

`explicit NoExemplarReservoir`() = default

## Class NoopAsyncWritableMetricStorage

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_state_metric_storage.h`

## Inheritance Relationships

### Base Type

- `public opentelemetry::sdk::metrics::AsyncWritableMetricStorage`

## Class Documentation

class **NoopAsyncWritableMetricStorage** : public opentelemetry::sdk::metrics::*AsyncWritableMetricStorage*

### Public Functions

inline void **RecordLong**(const std::unordered\_map<*MetricAttributes*, int64\_t, *AttributeHashGenerator*>&, opentelemetry::common::*SystemTimestamp*) noexcept override

inline void **RecordDouble**(const std::unordered\_map<*MetricAttributes*, double, *AttributeHashGenerator*>&, opentelemetry::common::*SystemTimestamp*) noexcept override

## Class NoopMetricStorage

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_state_metric_storage.h`

## Inheritance Relationships

### Base Type

- public opentelemetry::sdk::metrics::MetricStorage

## Class Documentation

class **NoopMetricStorage** : public opentelemetry::sdk::metrics::*MetricStorage*

### Public Functions

inline bool **Collect**(*CollectorHandle*\*, nostd::span<std::shared\_ptr<*CollectorHandle*>>, opentelemetry::common::*SystemTimestamp*, opentelemetry::common::*SystemTimestamp*, nostd::function\_ref<bool(*MetricData*)> callback) noexcept override

## Class NoopWritableMetricStorage

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_state_metric_storage.h`

## Inheritance Relationships

### Base Type

- public opentelemetry::sdk::metrics::SyncWritableMetricStorage

### Class Documentation

class **NoopWritableMetricStorage** : public opentelemetry::sdk::metrics::SyncWritableMetricStorage

#### Public Functions

virtual void **RecordLong**(int64\_t value, const opentelemetry::context::Context &context) noexcept override = 0

inline void **RecordLong**(int64\_t, const opentelemetry::common::KeyValueIterable&, const opentelemetry::context::Context&) noexcept override

inline void **RecordDouble**(double, const opentelemetry::context::Context&) noexcept override

inline void **RecordDouble**(double, const opentelemetry::common::KeyValueIterable&, const opentelemetry::context::Context&) noexcept override

### Class ObservableInstrument

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_latest\_sdk\_include\_opentelemetry\_sdk\_metrics\_async\_instruments.h

## Inheritance Relationships

### Base Type

- public opentelemetry::metrics::ObservableInstrument

### Class Documentation

class **ObservableInstrument** : public opentelemetry::metrics::ObservableInstrument

## Public Functions

**ObservableInstrument**(*InstrumentDescriptor* instrument\_descriptor,  
std::unique\_ptr<*AsyncWritableMetricStorage*> storage,  
std::shared\_ptr<*ObservableRegistry*> observable\_registry)

void **AddCallback**(opentelemetry::metrics::*ObservableCallbackPtr* callback, void \*state) noexcept override

void **RemoveCallback**(opentelemetry::metrics::*ObservableCallbackPtr* callback, void \*state) noexcept override

const *InstrumentDescriptor* &**GetInstrumentDescriptor**()

*AsyncWritableMetricStorage* \***GetMetricStorage**()

## Class ObservableRegistry

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_latest\_sdk\_include\_opentelemetry\_sdk\_metrics\_state\_observable\_registry.h

## Class Documentation

class **ObservableRegistry**

### Public Functions

void **AddCallback**(opentelemetry::metrics::*ObservableCallbackPtr* callback, void \*state,  
opentelemetry::metrics::*ObservableInstrument* \*instrument)

void **RemoveCallback**(opentelemetry::metrics::*ObservableCallbackPtr* callback, void \*state,  
opentelemetry::metrics::*ObservableInstrument* \*instrument)

void **Observe**(opentelemetry::common::*SystemTimestamp* collection\_ts)

## Template Class ObserverResultT

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_latest\_sdk\_include\_opentelemetry\_sdk\_metrics\_observer\_result.h

## Inheritance Relationships

### Base Type

- public opentelemetry::metrics::**ObserverResultT**< T > (*Template Class ObserverResultT*)

## Class Documentation

template<class T>

class **ObserverResultT** : public opentelemetry::metrics::*ObserverResultT*<T>

### Public Functions

inline explicit **ObserverResultT**(const *AttributesProcessor* \*attributes\_processor = nullptr)

**~ObserverResultT**() override = default

inline void **Observe**(T value) noexcept override

inline void **Observe**(T value, const opentelemetry::common::*KeyValueIterable* &attributes) noexcept override

inline const std::unordered\_map<*MetricAttributes*, T, *AttributeHashGenerator*> &**GetMeasurements**()

## Class PatternPredicate

- Defined in file `__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_view_predicate.h`

## Inheritance Relationships

### Base Type

- public opentelemetry::sdk::metrics::Predicate

## Class Documentation

class **PatternPredicate** : public opentelemetry::sdk::metrics::*Predicate*

### Public Functions

inline **PatternPredicate**(opentelemetry::nostd::string\_view pattern)

inline bool **Match**(opentelemetry::nostd::string\_view str) const noexcept override

## Class PeriodicExportingMetricReader

- Defined in file `__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_export_periodic_exporting_metric_reader.h`

## Inheritance Relationships

### Base Type

- public opentelemetry::sdk::metrics::MetricReader (*Class MetricReader*)

### Class Documentation

class **PeriodicExportingMetricReader** : public opentelemetry::sdk::metrics::MetricReader

#### Public Functions

**PeriodicExportingMetricReader**(std::unique\_ptr<PushMetricExporter> exporter, const *PeriodicExportingMetricReaderOptions* &option)

*AggregationTemporality* **GetAggregationTemporality**(InstrumentType instrument\_type) const noexcept  
override

### Class Predicate

- Defined in file\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_latest\_sdk\_include\_opentelemetry\_sdk\_metrics\_view\_predicate.h

## Inheritance Relationships

### Derived Types

- public opentelemetry::sdk::metrics::ExactPredicate (*Class ExactPredicate*)
- public opentelemetry::sdk::metrics::MatchEverythingPattern (*Class MatchEverythingPattern*)
- public opentelemetry::sdk::metrics::MatchNothingPattern (*Class MatchNothingPattern*)
- public opentelemetry::sdk::metrics::PatternPredicate (*Class PatternPredicate*)

### Class Documentation

class **Predicate**

Subclassed by *opentelemetry::sdk::metrics::ExactPredicate*, *opentelemetry::sdk::metrics::MatchEverythingPattern*, *opentelemetry::sdk::metrics::MatchNothingPattern*, *opentelemetry::sdk::metrics::PatternPredicate*

### Public Functions

virtual **~Predicate()** = default

virtual bool **Match**(opentelemetry::nostd::string\_view string) const noexcept = 0

### Class PredicateFactory

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_view_predicate_factory.h`

### Class Documentation

class **PredicateFactory**

#### Public Static Functions

static inline std::unique\_ptr<*Predicate*> **GetPredicate**(opentelemetry::nostd::string\_view pattern, *PredicateType* type)

### Class PushMetricExporter

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_push_metric_exporter.h`

### Class Documentation

class **PushMetricExporter**

*PushMetricExporter* defines the interface to be used by metrics libraries to push metrics data to the OpenTelemetry exporters.

#### Public Functions

virtual **~PushMetricExporter()** = default

virtual opentelemetry::sdk::common::ExportResult **Export**(const *ResourceMetrics* &data) noexcept = 0

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

**Parameters** data – metrics data

virtual *AggregationTemporality* **GetAggregationTemporality**(*InstrumentType* instrument\_type) const noexcept = 0

Get the AggregationTemporality for given Instrument Type for this exporter.

**Returns** AggregationTemporality



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

Force flush the exporter.

```
virtual bool Shutdown(std::chrono::microseconds timeout = std::chrono::microseconds(0)) noexcept = 0
```

Shut down the metric exporter.

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

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

## Class ReservoirCell

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_exemplar_reservoir_cell.h`

## Class Documentation

class **ReservoirCell**

A Reservoir cell pre-allocated memories for Exemplar data.

### Public Functions

**ReservoirCell**() = default

```
inline void RecordLongMeasurement(int64_t value, const MetricAttributes &attributes, const
    opentelemetry::context::Context &context)
```

Record the long measurement to the cell.

```
inline void RecordDoubleMeasurement(double value, const MetricAttributes &attributes, const
    opentelemetry::context::Context &context)
```

Record the long measurement to the cell.

```
inline std::shared_ptr<ExemplarData> GetAndResetLong(const MetricAttributes &point_attributes)
```

Retrieve the cell's *ExemplarData*.

Must be used in tandem with `recordLongMeasurement(int64_t, Attributes, Context)`.

```
inline std::shared_ptr<ExemplarData> GetAndResetDouble(const MetricAttributes &point_attributes)
```

Retrieve the cell's *ExemplarData*.

Must be used in tandem with `recordDoubleMeasurement(double, Attributes, Context)`.

```
inline void reset()
```

## Class SumPointData

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_data_point_data.h`

## Class Documentation

class **SumPointData**

### Public Functions

**SumPointData**(*SumPointData*&&) = default

**SumPointData**(const *SumPointData*&) = default

*SumPointData* &**operator=**(*SumPointData*&&) = default

**SumPointData**() = default

### Public Members

*ValueType* **value\_** = {}

bool **is\_monotonic\_** = true

## Class Synchronous

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_sync_instruments.h`

## Inheritance Relationships

### Derived Types

- public `opentelemetry::sdk::metrics::DoubleCounter` (*Class DoubleCounter*)
- public `opentelemetry::sdk::metrics::DoubleHistogram` (*Class DoubleHistogram*)
- public `opentelemetry::sdk::metrics::DoubleUpDownCounter` (*Class DoubleUpDownCounter*)
- public `opentelemetry::sdk::metrics::LongCounter< T >` (*Template Class LongCounter*)
- public `opentelemetry::sdk::metrics::LongHistogram< T >` (*Template Class LongHistogram*)
- public `opentelemetry::sdk::metrics::LongUpDownCounter` (*Class LongUpDownCounter*)

## Class Documentation

### class **Synchronous**

Subclassed by *opentelemetry::sdk::metrics::DoubleCounter*, *opentelemetry::sdk::metrics::DoubleHistogram*, *opentelemetry::sdk::metrics::DoubleUpDownCounter*, *opentelemetry::sdk::metrics::LongCounter< T >*, *opentelemetry::sdk::metrics::LongHistogram< T >*, *opentelemetry::sdk::metrics::LongUpDownCounter*

#### Public Functions

```
inline Synchronous(InstrumentDescriptor instrument_descriptor,
                    std::unique_ptr<SyncWritableMetricStorage> storage)
```

#### Protected Attributes

*InstrumentDescriptor* **instrument\_descriptor\_**

std::unique\_ptr<*SyncWritableMetricStorage*> **storage\_**

### Class SyncMetricStorage

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_state_sync_metric_storage.h`

## Inheritance Relationships

### Base Types

- public `opentelemetry::sdk::metrics::MetricStorage`
- public `opentelemetry::sdk::metrics::SyncWritableMetricStorage`

## Class Documentation

```
class SyncMetricStorage : public opentelemetry::sdk::metrics::MetricStorage, public
opentelemetry::sdk::metrics::SyncWritableMetricStorage
```

## Public Functions

```
inline SyncMetricStorage (InstrumentDescriptor instrument_descriptor,  
const AggregationType aggregation_type,  
const AttributesProcessor *attributes_processor,  
nstd::shared_ptr< ExemplarReservoir > &&exemplar_reservoir OPENTELEMETRY_MAYBE_UNUSED,  
const AggregationConfig *aggregation_config)  
  
inline void RecordLong (int64_t value,  
const opentelemetry::context::Context &context OPENTELEMETRY_MAYBE_UNUSED) noexcept override  
  
inline void RecordLong (int64_t value,  
const opentelemetry::common::KeyValueIterable &attributes,  
const opentelemetry::context::Context &context OPENTELEMETRY_MAYBE_UNUSED) noexcept override  
  
inline void RecordDouble (double value,  
const opentelemetry::context::Context &context OPENTELEMETRY_MAYBE_UNUSED) noexcept override  
  
inline void RecordDouble (double value,  
const opentelemetry::common::KeyValueIterable &attributes,  
const opentelemetry::context::Context &context OPENTELEMETRY_MAYBE_UNUSED) noexcept override  
  
bool Collect (CollectorHandle *collector, nstd::span<std::shared_ptr<CollectorHandle>> collectors,  
opentelemetry::common::SystemTimestamp sdk_start_ts,  
opentelemetry::common::SystemTimestamp collection_ts,  
nstd::function_ref<bool(MetricData)> callback) noexcept override
```

## Class SyncMultiMetricStorage

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_state_multi_metric_storage.h`

## Inheritance Relationships

### Base Type

- public `opentelemetry::sdk::metrics::SyncWritableMetricStorage`

## Class Documentation

```
class SyncMultiMetricStorage : public opentelemetry::sdk::metrics::SyncWritableMetricStorage
```

## Public Functions

```
inline void AddStorage(std::shared_ptr<SyncWritableMetricStorage> storage)
```

```
inline virtual void RecordLong(int64_t value, const opentelemetry::context::Context &context) noexcept  
override
```

```
inline virtual void RecordLong(int64_t value, const opentelemetry::common::KeyValueIterable &attributes,  
const opentelemetry::context::Context &context) noexcept override
```

```
inline virtual void RecordDouble(double value, const opentelemetry::context::Context &context) noexcept  
override
```

```
inline virtual void RecordDouble(double value, const opentelemetry::common::KeyValueIterable &attributes,  
const opentelemetry::context::Context &context) noexcept override
```

## Class SyncWritableMetricStorage

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_state_metric_storage.h`

## Inheritance Relationships

### Derived Types

- public `opentelemetry::sdk::metrics::NoopWritableMetricStorage` (Class *NoopWritableMetricStorage*)
- public `opentelemetry::sdk::metrics::SyncMetricStorage` (Class *SyncMetricStorage*)
- public `opentelemetry::sdk::metrics::SyncMultiMetricStorage` (Class *SyncMultiMetricStorage*)

## Class Documentation

```
class SyncWritableMetricStorage
```

```
Subclassed by opentelemetry::sdk::metrics::NoopWritableMetricStorage, opentelemetry::sdk::metrics::SyncMetricStorage, opentelemetry::sdk::metrics::SyncMultiMetricStorage
```

### Public Functions

```
virtual void RecordLong(int64_t value, const opentelemetry::context::Context &context) noexcept = 0
```

```
virtual void RecordLong(int64_t value, const opentelemetry::common::KeyValueIterable &attributes, const  
opentelemetry::context::Context &context) noexcept = 0
```

```
virtual void RecordDouble(double value, const opentelemetry::context::Context &context) noexcept = 0
```

```
virtual void RecordDouble(double value, const opentelemetry::common::KeyValueIterable &attributes, const  
opentelemetry::context::Context &context) noexcept = 0
```

```
virtual ~SyncWritableMetricStorage() = default
```

## Class TemporalMetricStorage

- Defined in file `_home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_state_temporal_metric_storage.h`

## Class Documentation

class **TemporalMetricStorage**

### Public Functions

**TemporalMetricStorage**(*InstrumentDescriptor* instrument\_descriptor, *AggregationType* aggregation\_type, const *AggregationConfig* \*aggregation\_config)

bool **buildMetrics**(*CollectorHandle* \*collector, nostd::span<std::shared\_ptr<*CollectorHandle*>> collectors, opentelemetry::common::SystemTimestamp sdk\_start\_ts, opentelemetry::common::SystemTimestamp collection\_ts, std::shared\_ptr<*AttributesHashMap*> delta\_metrics, nostd::function\_ref<bool(*MetricData*)> callback) noexcept

## Class View

- Defined in file `_home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_view_view.h`

## Class Documentation

class **View**

*View* defines the interface to allow SDK user to customize the metrics before exported.

### Public Functions

inline **View**(const std::string &name, const std::string &description = "", *AggregationType* aggregation\_type = *AggregationType::kDefault*, std::shared\_ptr<*AggregationConfig*> aggregation\_config = nullptr, std::unique\_ptr<opentelemetry::sdk::metrics::*AttributesProcessor*> attributes\_processor = std::unique\_ptr<opentelemetry::sdk::metrics::*AttributesProcessor*>(new opentelemetry::sdk::metrics::*DefaultAttributesProcessor*()))

virtual **~View**() = default

inline virtual std::string **GetName**() const noexcept

inline virtual std::string **GetDescription**() const noexcept

inline virtual *AggregationType* **GetAggregationType**() const noexcept

inline virtual *AggregationConfig* \***GetAggregationConfig**() const noexcept

inline virtual const opentelemetry::sdk::metrics::*AttributesProcessor* &**GetAttributesProcessor**() const noexcept

## Class ViewRegistry

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_view_view_registry.h`

## Class Documentation

class **ViewRegistry**

### Public Functions

```
inline void AddView(std::unique_ptr<opentelemetry::sdk::metrics::InstrumentSelector> instrument_selector,
                    std::unique_ptr<opentelemetry::sdk::metrics::MeterSelector> meter_selector,
                    std::unique_ptr<opentelemetry::sdk::metrics::View> view)
```

```
inline bool FindViews(const opentelemetry::sdk::metrics::InstrumentDescriptor &instrument_descriptor,
                      const opentelemetry::sdk::instrumentationscope::InstrumentationScope
                      &instrumentation_scope, nostd::function_ref<bool(const View&)> callback) const
```

```
~ViewRegistry() = default
```

## Class WithTraceSampleFilter

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_exemplar_with_trace_sample_filter.h`

## Inheritance Relationships

### Base Type

- public `opentelemetry::sdk::metrics::ExemplarFilter` (Class *ExemplarFilter*)

## Class Documentation

class **WithTraceSampleFilter** : public `opentelemetry::sdk::metrics::ExemplarFilter`

### Public Functions

```
inline bool ShouldSampleMeasurement(int64_t, const MetricAttributes&, const
                                     opentelemetry::context::Context &context) noexcept override
```

```
inline bool ShouldSampleMeasurement(double, const MetricAttributes&, const
                                     opentelemetry::context::Context &context) noexcept override
```

```
explicit WithTraceSampleFilter() = default
```

## Class `OTELResourceDetector`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_resource_resource_detector.h`

## Inheritance Relationships

### Base Type

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

## Class Documentation

class `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_latest_sdk_include_opentelemetry_sdk_resource_resource.h`

## Class Documentation

class `Resource`

### Public Functions

`Resource(const Resource&)` = default

const `ResourceAttributes &GetAttributes()` const noexcept

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

`Resource Merge(const Resource &other)` const noexcept

Returns a new, merged Resource by merging the current Resource with the other Resource. In case of a collision, the other 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_latest_sdk_include_opentelemetry_sdk_resource_resource_detector.h`

## Inheritance Relationships

### Derived Type

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

## Class Documentation

class **ResourceDetector**

Interface for a Resource Detector

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

### Public Functions

**ResourceDetector**() = default

virtual **~ResourceDetector**() = default

virtual *Resource Detect*() = 0

### Class AlwaysOffSampler

- Defined in file `__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_trace_samplers_always_off.h`

### Inheritance Relationships

#### Base Type

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

### Class Documentation

class **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 AlwaysOffSamplerFactory

- Defined in file `__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_trace_samplers_always_off_factory.h`

## Class Documentation

class **AlwaysOffSamplerFactory**

Factory class for *AlwaysOffSampler*.

### Public Static Functions

static std::unique\_ptr<*Sampler*> **Create**()

Create an *AlwaysOffSampler*.

## Class AlwaysOnSampler

- Defined in file `__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_trace_samplers_always_on.h`

## Inheritance Relationships

### Base Type

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

## Class Documentation

class **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`, `nstd::string_view`, `opentelemetry::trace::SpanKind`, const `opentelemetry::common::KeyValueIterable`&, const `opentelemetry::trace::SpanContextKeyValueIterable`&) noexcept override

**Returns** Always return `Decision RECORD_AND_SAMPLE`

inline virtual `nstd::string_view` **GetDescription**() const noexcept override

**Returns** Description MUST be *AlwaysOnSampler*

## Class AlwaysOnSamplerFactory

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_trace_samplers_always_on_factory.h`

## Class Documentation

class **AlwaysOnSamplerFactory**

Factory class for *AlwaysOnSampler*.

### Public Static Functions

static `std::unique_ptr<Sampler>` **Create()**

Create an *AlwaysOnSampler*.

## Class BatchSpanProcessor

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_trace_batch_span_processor.h`

## Nested Relationships

### Nested Types

- *Struct* `BatchSpanProcessor::SynchronizationData`

## Inheritance Relationships

### Base Type

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

## Class Documentation

class **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**() override

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).

### Protected Functions

void **DoBackgroundWork**()

The background routine performed by the worker thread.

virtual void **Export**()

Exports all ended spans to the configured exporter.

void **DrainQueue**()

Called when *Shutdown()* is invoked. Completely drains the queue of all its ended spans and passes them to the exporter.

void **GetWaitAdjustedTime**(std::chrono::microseconds &timeout,  
std::chrono::time\_point<std::chrono::system\_clock> &start\_time)

### Protected Attributes

std::unique\_ptr<*SpanExporter*> **exporter\_**

const size\_t **max\_queue\_size\_**

const std::chrono::milliseconds **schedule\_delay\_millis\_**

const size\_t **max\_export\_batch\_size\_**

common::CircularBuffer<*Recordable*> **buffer\_**

std::shared\_ptr<*SynchronizationData*> **synchronization\_data\_**

std::thread **worker\_thread\_**

### Protected Static Functions

static void **NotifyCompletion**(bool notify\_force\_flush, const std::shared\_ptr<*SynchronizationData*>  
&synchronization\_data)

Notify completion of shutdown and force flush. This may be called from the any thread at any time.

#### Parameters

- **notify\_force\_flush** – Flag to indicate whether to notify force flush completion.
- **synchronization\_data** – Synchronization data to be notified.

struct **SynchronizationData**

## Public Members

std::condition\_variable **cv**

std::condition\_variable **force\_flush\_cv**

std::mutex **cv\_m**

std::mutex **force\_flush\_cv\_m**

std::mutex **shutdown\_m**

std::atomic<bool> **is\_force\_wakeup\_background\_worker**

std::atomic<bool> **is\_force\_flush\_pending**

std::atomic<bool> **is\_force\_flush\_notified**

std::atomic<bool> **is\_shutdown**

## Class BatchSpanProcessorFactory

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_trace_batch_span_processor_factory.h`

## Class Documentation

class **BatchSpanProcessorFactory**

Factory class for *BatchSpanProcessor*.

### Public Static Functions

static std::unique\_ptr<*SpanProcessor*> **Create**(std::unique\_ptr<*SpanExporter*> &&exporter, const *BatchSpanProcessorOptions* &options)

Create a *BatchSpanProcessor*.

## Class IdGenerator

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_trace_id_generator.h`

## Inheritance Relationships

### Derived Type

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

## Class Documentation

### class 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_latest_sdk_include_opentelemetry_sdk_trace_multi_recordable.h`

## Inheritance Relationships

### Base Type

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



## Class Documentation

class **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 **SetInstrumentationScope**(const InstrumentationScope &instrumentation\_scope) noexcept override

### Class MultiSpanProcessor

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_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 **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() override
```

## Class ParentBasedSampler

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_trace_samplers_parent.h`

## Inheritance Relationships

### Base Type

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

## Class Documentation

```
class 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 ParentBasedSamplerFactory

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_trace_samplers_parent_factory.h`

## Class Documentation

class **ParentBasedSamplerFactory**

Factory class for *ParentBasedSampler*.

### Public Static Functions

static `std::unique_ptr<Sampler>` **Create**(`std::shared_ptr<Sampler>` delegate\_sampler)

Create a *ParentBasedSampler*.

## Class RandomIdGenerator

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_trace_random_id_generator.h`

## Inheritance Relationships

### Base Type

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

## Class Documentation

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

### Public Functions

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

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

## Class RandomIdGeneratorFactory

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_trace_random_id_generator_factory.h`

## Class Documentation

### class **RandomIdGeneratorFactory**

Factory class for RandomIdGenerator.

#### Public Static Functions

static std::unique\_ptr<IdGenerator> **Create**()

Create a RandomIdGenerator.

### Class Recordable

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_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 **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(nostd::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(nostd::string_view name)
```

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

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

```
inline void AddEvent(nostd::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

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

Add an event to a span.

**Parameters**

- **name** – the name of the event
- **attributes** – the attributes associated with 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, nostd::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**(nostd::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

inline virtual explicit **operator** *SpanData*\*() const

Get the *SpanData* object for this *Recordable*.

**Returns** *SpanData*\*

virtual void **SetInstrumentationScope**(const InstrumentationScope &instrumentation\_scope) noexcept = 0

Set the instrumentation scope of the span.

**Parameters** **instrumentation\_scope** – the instrumentation scope to set

inline void **SetInstrumentationLibrary**(const InstrumentationScope &instrumentation\_scope) noexcept

## Class Sampler

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_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 **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_latest_sdk_include_opentelemetry_sdk_trace_simple_processor.h`

## Inheritance Relationships

### Base Type

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

## Class Documentation

class **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*&, const `opentelemetry::trace::SpanContext`&) 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) 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**() override

### Class SimpleSpanProcessorFactory

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

### Class Documentation

class **SimpleSpanProcessorFactory**

Factory class for *SimpleSpanProcessor*.

#### Public Static Functions

static std::unique\_ptr<*SpanProcessor*> **Create**(std::unique\_ptr<*SpanExporter*> &&exporter)

Create a *SimpleSpanProcessor*.

### Class SpanData

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

## Inheritance Relationships

### Base Type

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

### Class Documentation

class **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::InstrumentationScope &GetInstrumentationScope() const noexcept  
Get the attributes associated with the resource  
**Returns** the attributes associated with the resource configured for TracerProvider

inline const opentelemetry::sdk::trace::InstrumentationScope &GetInstrumentationLibrary() const noexcept

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 **SetInstrumentationScope**(const InstrumentationScope &instrumentation\_scope)  
noexcept override

Set the instrumentation scope of the span.

**Parameters** `instrumentation_scope` – the instrumentation scope to set

### Class `SpanDataEvent`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_trace_span_data.h`

### Class Documentation

class `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_latest_sdk_include_opentelemetry_sdk_trace_span_data.h`

## Class Documentation

### class `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_latest_sdk_include_opentelemetry_sdk_trace_exporter.h`

## Class Documentation

### class `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
                                         nostd::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_latest_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 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`, and `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\_latest\_sdk\_include\_opentelemetry\_sdk\_trace\_samplers\_trace\_id\_ratio.h

## Inheritance Relationships

### Base Type

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

## Class Documentation

class **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 TraceIdRatioBasedSamplerFactory

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_latest\_sdk\_include\_opentelemetry\_sdk\_trace\_samplers\_trace\_id\_ratio\_factory.h

### Class Documentation

class **TraceIdRatioBasedSamplerFactory**

Factory class for *TraceIdRatioBasedSampler*.

#### Public Static Functions

static std::unique\_ptr<*Sampler*> **Create**(double ratio)

Create a *TraceIdRatioBasedSampler*.

### Class Tracer

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_latest\_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 **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<InstrumentationScope> instrumentation\_scope = InstrumentationScope::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 InstrumentationScope &**GetInstrumentationScope**() const noexcept

Returns the associated instrumentation scope

inline const InstrumentationScope &**GetInstrumentationLibrary**() const noexcept

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\_latest\_sdk\_include\_opentelemetry\_sdk\_trace\_tracer\_context.h

## Class Documentation

class **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 TracerContextFactory

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_trace_tracer_context_factory.h`

## Class Documentation

### class **TracerContextFactory**

Factory class for *TracerContext*.

#### Public Static Functions

static std::unique\_ptr<*TracerContext*> **Create**(std::vector<std::unique\_ptr<*SpanProcessor*>> &&processors)  
Create a *TracerContext*.

static std::unique\_ptr<*TracerContext*> **Create**(std::vector<std::unique\_ptr<*SpanProcessor*>> &&processors,  
const opentelemetry::sdk::resource::Resource &resource)  
Create a *TracerContext*.

static std::unique\_ptr<*TracerContext*> **Create**(std::vector<std::unique\_ptr<*SpanProcessor*>> &&processors,  
const opentelemetry::sdk::resource::Resource &resource,  
std::unique\_ptr<*Sampler*> sampler)  
Create a *TracerContext*.

static std::unique\_ptr<*TracerContext*> **Create**(std::vector<std::unique\_ptr<*SpanProcessor*>> &&processors,  
const opentelemetry::sdk::resource::Resource &resource,  
std::unique\_ptr<*Sampler*> sampler,  
std::unique\_ptr<*IdGenerator*> id\_generator)  
Create a *TracerContext*.

### Class **TracerProvider**

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

## Inheritance Relationships

### Base Type

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

## Class Documentation

class **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.

**~TracerProvider()** override

```
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.

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 TracerProviderFactory

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_trace_tracer_provider_factory.h`

## Class Documentation

class **TracerProviderFactory**

Factory class for TracerProvider. See TracerProvider.

### Public Static Functions

static std::unique\_ptr<opentelemetry::trace::TracerProvider> **Create**(std::unique\_ptr<SpanProcessor> processor)

static std::unique\_ptr<opentelemetry::trace::TracerProvider> **Create**(std::unique\_ptr<SpanProcessor> processor, const opentelemetry::sdk::resource::Resource &resource)

static std::unique\_ptr<opentelemetry::trace::TracerProvider> **Create**(std::unique\_ptr<SpanProcessor> processor, const opentelemetry::sdk::resource::Resource &resource, std::unique\_ptr<Sampler> sampler)

static std::unique\_ptr<opentelemetry::trace::TracerProvider> **Create**(std::unique\_ptr<SpanProcessor> processor, const opentelemetry::sdk::resource::Resource &resource, std::unique\_ptr<Sampler> sampler, std::unique\_ptr<IdGenerator> id\_generator)

static std::unique\_ptr<opentelemetry::trace::TracerProvider> **Create**(std::vector<std::unique\_ptr<SpanProcessor>> &&processors)

static std::unique\_ptr<opentelemetry::trace::TracerProvider> **Create**(std::vector<std::unique\_ptr<SpanProcessor>> &&processors, const opentelemetry::sdk::resource::Resource &resource)

static std::unique\_ptr<opentelemetry::trace::TracerProvider> **Create**(std::vector<std::unique\_ptr<SpanProcessor>> &&processors, const opentelemetry::sdk::resource::Resource &resource, std::unique\_ptr<Sampler> sampler)

```
static std::unique_ptr<opentelemetry::trace::TracerProvider> Create(std::vector<std::unique_ptr<SpanProcessor>>
                                                                    &&processors, const
                                                                    opentelemetry::sdk::resource::Resource
                                                                    &resource, std::unique_ptr<Sampler>
                                                                    sampler, std::unique_ptr<IdGenerator>
                                                                    id_generator)

static std::unique_ptr<opentelemetry::trace::TracerProvider> Create(std::shared_ptr<sdk::trace::TracerContext>
                                                                    context)
```

## Class DefaultSpan

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

## Inheritance Relationships

### Base Type

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

## Class Documentation

class **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 override

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

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, const common::KeyValueIterable&) 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 nostd::string\_view **ToString**() const noexcept

inline **DefaultSpan**(*SpanContext* span\_context) noexcept

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

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

### Public Static Functions

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

### Class NoopSpan

- Defined in file\_\_home\_docs\_checkouts\_readthedocs.org\_user\_builds\_opentelemetry-cpp\_checkouts\_latest\_api\_include\_opentelemetry\_trace\_noop.h

### Inheritance Relationships

#### Base Type

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

### Class Documentation

class **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, const common::*KeyValueIterable*&) 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\_latest\_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 **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_latest_api_include_opentelemetry_trace_noop.h`

## Inheritance Relationships

### Base Type

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

## Class Documentation

class **NoopTracerProvider** : public `opentelemetry::trace::TracerProvider`

No-op implementation of a *TracerProvider*.

### Public Functions

`inline NoopTracerProvider()` noexcept

`inline virtual nostd::shared_ptr<opentelemetry::trace::Tracer> GetTracer`(`nostd::string_view`,  
`nostd::string_view`,  
`nostd::string_view`) noexcept  
 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_latest_api_include_opentelemetry_trace_span_context_kv_iterable.h`

## Inheritance Relationships

### Base Type

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

## Class Documentation

class **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_latest_api_include_opentelemetry_trace_propagation_b3_propagator.h`

## Inheritance Relationships

### Base Type

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

## Class Documentation

class **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_latest_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 **B3PropagatorExtractor** : public `opentelemetry::context::propagation::TextMapPropagator`

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

### Public Functions

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

### Public Static Functions

static inline `TraceId` **TraceIdFromHex**(`nstd::string_view` trace\_id)

static inline `SpanId` **SpanIdFromHex**(`nstd::string_view` span\_id)

static inline `TraceFlags` **TraceFlagsFromHex**(`nstd::string_view` trace\_flags)

## Class B3PropagatorMultiHeader

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_api_include_opentelemetry_trace_propagation_b3_propagator.h`

## Inheritance Relationships

### Base Type

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

### Class Documentation

class **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\_latest\_api\_include\_opentelemetry\_trace\_propagation\_http\_trace\_context.h

## Inheritance Relationships

### Base Type

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

### Class Documentation

class **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_latest_api_include_opentelemetry_trace_propagation_jaeger.h`

## Inheritance Relationships

### Base Type

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

## Class Documentation

class **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_latest_api_include_opentelemetry_trace_provider.h`

## Class Documentation

class **Provider**

Stores the singleton global *TracerProvider*.

## Public Static Functions

static inline nostd::shared\_ptr<*TracerProvider*> **GetTracerProvider**() noexcept

Returns the singleton *TracerProvider*.

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

static inline void **SetTracerProvider**(nostd::shared\_ptr<*TracerProvider*> tp) noexcept

Changes the singleton *TracerProvider*.

## Class Scope

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_api_include_opentelemetry_trace_scope.h`

## Class Documentation

class **Scope**

Controls how long a span is active.

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

## Public Functions

inline **Scope**(const nostd::shared\_ptr<*Span*> &span) noexcept

Initialize a new scope.

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

## Class Span

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_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 **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_latest_api_include_opentelemetry_trace_span_context.h`

## Class Documentation

class `SpanContext`

### Public Functions

inline `SpanContext`(bool `sampled_flag`, bool `is_remote`) noexcept

inline `SpanContext`(`TraceId` `trace_id`, `SpanId` `span_id`, `TraceFlags` `trace_flags`, bool `is_remote`,  
nstd::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 nstd::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()` noexcept

## Class `SpanContextKeyValueIterable`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_api_include_opentelemetry_trace_span_context_kv_iterable.h`

## Inheritance Relationships

### Derived Type

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

## Class Documentation

### class `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_latest_api_include_opentelemetry_trace_span_id.h`

## Class Documentation

### class `SpanId`

### Public Functions

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

### Public Static Attributes

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

### Class TraceFlags

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_api_include_opentelemetry_trace_trace_flags.h`

### Class Documentation

class **TraceFlags**

### Public Functions

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

## Public Static Attributes

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

## Class TraceId

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_api_include_opentelemetry_trace_trace_id.h`

## Class Documentation

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

Starts a span.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## Public Static Functions

```
static inline 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_latest_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 **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 = "") noexcept = 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_latest_api_include_opentelemetry_trace_trace_state.h`

## Class Documentation

### class **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**() const noexcept

Creates a w3c tracestate header from *TraceState* object

inline bool **Get**(nstd::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 nstd::shared\_ptr<*TraceState*> **Set**(const nstd::string\_view &key, const nstd::string\_view &value) noexcept

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

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

## Public Static Functions

**static inline OPENTELEMETRY\_API\_SINGLETON nstd::shared\_ptr< TraceState > GetDefault ()**

static inline nstd::shared\_ptr<*TraceState*> **FromHeader**(nstd::string\_view header) noexcept

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**(nstd::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**(nstd::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** = ','

## 3.2.3 Enums

### Enum AggregationTemporality

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_instruments.h`

### Enum Documentation

enum opentelemetry::sdk::metrics::AggregationTemporality

*Values:*

enumerator **kUnspecified**

enumerator **kDelta**

enumerator **kCumulative**

### Enum AggregationType

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_instruments.h`

### Enum Documentation

enum opentelemetry::sdk::metrics::AggregationType

*Values:*

enumerator **kDrop**

enumerator **kHistogram**

enumerator **kLastValue**

enumerator **kSum**

enumerator **kDefault**

### Enum InstrumentClass

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_instruments.h`

### Enum Documentation

enum `opentelemetry::sdk::metrics::InstrumentClass`

*Values:*

enumerator **kSync**

enumerator **kAsync**

### Enum InstrumentType

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_instruments.h`

### Enum Documentation

enum `opentelemetry::sdk::metrics::InstrumentType`

*Values:*

enumerator **kCounter**

enumerator **kHistogram**

enumerator **kUpDownCounter**

enumerator **kObservableCounter**

enumerator **kObservableGauge**

enumerator **kObservableUpDownCounter**

### Enum InstrumentValueType

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_instruments.h`

### Enum Documentation

enum opentelemetry::sdk::metrics::InstrumentValueType

*Values:*

enumerator **kInt**

enumerator **kLong**

enumerator **kFloat**

enumerator **kDouble**

### Enum PredicateType

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_view_predicate_factory.h`

### Enum Documentation

enum opentelemetry::sdk::metrics::PredicateType

*Values:*

enumerator **kPattern**

enumerator **kExact**

### Enum Decision

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_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_latest_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_latest_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_latest_api_include_opentelemetry_trace_span_metadata.h`

## Enum Documentation

enum opentelemetry::trace::StatusCode

*Values:*

enumerator **kUnset**

enumerator **kOk**

enumerator **kError**

## 3.2.4 Functions

### Function opentelemetry::baggage::GetBaggage

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

### Function Documentation

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

### Function opentelemetry::baggage::SetBaggage

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

### Function Documentation

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



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

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_api_include_opentelemetry_context_runtime_context.h`

### Function Documentation

static *RuntimeContextStorage* \*`opentelemetry::context::GetDefaultStorage()` noexcept

Construct and return the default *RuntimeContextStorage*

**Returns** a `ThreadLocalContextStorage`

### Template Function `opentelemetry::sdk::metrics::BucketBinarySearch`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_aggregation_histogram_aggregation.h`

### Function Documentation

template<class `T`>

size\_t `opentelemetry::sdk::metrics::BucketBinarySearch`(*T* value, const std::vector<double> &boundaries)

### Template Function `opentelemetry::sdk::metrics::HistogramDiff`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_aggregation_histogram_aggregation.h`

### Function Documentation

template<class `T`>

void `opentelemetry::sdk::metrics::HistogramDiff`(*HistogramPointData* &current, *HistogramPointData* &next, *HistogramPointData* &diff)

### Template Function `opentelemetry::sdk::metrics::HistogramMerge`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_aggregation_histogram_aggregation.h`

## Function Documentation

```
template<class T>
void opentelemetry::sdk::metrics::HistogramMerge(HistogramPointData &current, HistogramPointData
&delta, HistogramPointData &merge)
```

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

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_api_include_opentelemetry_trace_context.h`

## Function Documentation

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

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

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_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_latest_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_latest_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\_latest\_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\_latest\_api\_include\_opentelemetry\_trace\_context.h

## Function Documentation

inline *context::Context* opentelemetry::trace::SetSpan(opentelemetry::context::Context &context, nostd::shared\_ptr<Span> span) noexcept

## 3.2.5 Variables

### Variable opentelemetry::baggage::kBaggageHeader

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

## Variable Documentation

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



**Variable `opentelemetry::trace::propagation::kB3CombinedHeader`**

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_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_latest_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_latest_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_latest_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_latest_api_include_opentelemetry_trace_propagation_jaeger.h`

### Variable Documentation

```
static const nstd::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_latest_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_latest_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_latest_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_latest_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_latest_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_latest_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_latest_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_latest_api_include_opentelemetry_trace_propagation_http_trace_context.h`

### Variable Documentation

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

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

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_api_include_opentelemetry_trace_propagation_http_trace_context.h`

### Variable Documentation

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

## 3.2.6 Defines

### Define `OPENTELEMETRY_API_SINGLETON`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_api_include_opentelemetry_common_macros.h`

### Define Documentation

`OPENTELEMETRY_API_SINGLETON`

### Define `OPENTELEMETRY_DEPRECATED`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_api_include_opentelemetry_common_macros.h`

### Define Documentation

`OPENTELEMETRY_DEPRECATED`



**Define OPENTELEMETRY\_DEPRECATED\_MESSAGE**

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

**Define Documentation**

**OPENTELEMETRY\_DEPRECATED\_MESSAGE**(msg)

**Define OPENTELEMETRY\_HAVE\_WORKING\_REGEX**

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

**Define Documentation**

**OPENTELEMETRY\_HAVE\_WORKING\_REGEX**

**Define OPENTELEMETRY\_LIKELY\_IF**

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

**Define Documentation**

**OPENTELEMETRY\_LIKELY\_IF**(...)

**Define OPENTELEMETRY\_MAYBE\_UNUSED**

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

**Define Documentation**

**OPENTELEMETRY\_MAYBE\_UNUSED**

Declare variable as maybe unused usage: `OPENTELEMETRY_MAYBE_UNUSED int a; class OPENTELEMETRY_MAYBE_UNUSED a; OPENTELEMETRY_MAYBE_UNUSED int a();`.

## 3.2.7 Typedefs

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

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_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_latest_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::metrics::ObservableCallbackPtr**

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_api_include_opentelemetry_metrics_async_instruments.h`

**Typedef Documentation**

```
using opentelemetry::metrics::ObservableCallbackPtr = void (*)(ObserverResult, void*)
```

**Typedef opentelemetry::metrics::ObserverResult**

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_api_include_opentelemetry_metrics_observer_result.h`

**Typedef Documentation**

```
using opentelemetry::metrics::ObserverResult =
nostd::variant<nostd::shared_ptr<ObserverResultT<int64_t>>, nostd::shared_ptr<ObserverResultT<double>>>
```

**Typedef opentelemetry::sdk::instrumentationlibrary::InstrumentationLibrary**

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_instrumentationlibrary_instrumentation_library.h`

**Typedef Documentation**

```
using opentelemetry::sdk::instrumentationlibrary::InstrumentationLibrary =
instrumentationscope::InstrumentationScope
```

**Typedef opentelemetry::sdk::metrics::AggregationTemporalitySelector**

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_instruments.h`

**Typedef Documentation**

```
using opentelemetry::sdk::metrics::AggregationTemporalitySelector =
std::function<AggregationTemporality(InstrumentType)>
```

### Typedef `opentelemetry::sdk::metrics::MetricAttributes`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_data_exemplar_data.h`

### Typedef Documentation

```
typedef opentelemetry::sdk::common::OrderedAttributeMap
opentelemetry::sdk::metrics::MetricAttributes
```

### Typedef `opentelemetry::sdk::metrics::PointAttributes`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_data_metric_data.h`

### Typedef Documentation

```
using opentelemetry::sdk::metrics::PointAttributes =
opentelemetry::sdk::common::OrderedAttributeMap
```

### Typedef `opentelemetry::sdk::metrics::PointType`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_data_metric_data.h`

### Typedef Documentation

```
using opentelemetry::sdk::metrics::PointType = opentelemetry::nstd::variant<SumPointData,
HistogramPointData, LastValuePointData, DropPointData>
```

### Typedef `opentelemetry::sdk::metrics::ValueType`

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_metrics_data_point_data.h`

### Typedef Documentation

```
using opentelemetry::sdk::metrics::ValueType = nstd::variant<int64_t, double>
```

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

- Defined in `file__home_docs_checkouts_readthedocs.org_user_builds_opentelemetry-cpp_checkouts_latest_sdk_include_opentelemetry_sdk_resource_resource.h`

### Typedef Documentation

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



## PERFORMANCE TESTS - BENCHMARKS

Click [here](#) to view the latest performance benchmarks for packages in this repo.

Please note that the fluctuation in the results are mainly because [machines with different CPUs](#) are used for tests.





## 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

### O

- opentelemetry::baggage::Baggage (C++ class), 32
- opentelemetry::baggage::Baggage::Baggage (C++ function), 32
- opentelemetry::baggage::Baggage::Delete (C++ function), 32
- opentelemetry::baggage::Baggage::FromHeader (C++ function), 32
- opentelemetry::baggage::Baggage::GetAllEntries (C++ function), 32
- opentelemetry::baggage::Baggage::GetValue (C++ function), 32
- opentelemetry::baggage::Baggage::kKeyValueSeparator (C++ member), 32
- opentelemetry::baggage::Baggage::kMaxKeyValuePairs (C++ member), 32
- opentelemetry::baggage::Baggage::kMaxKeyValueSize (C++ member), 32
- opentelemetry::baggage::Baggage::kMaxSize (C++ member), 32
- opentelemetry::baggage::Baggage::kMembersSeparator (C++ member), 32
- opentelemetry::baggage::Baggage::kMetadataSeparator (C++ member), 32
- opentelemetry::baggage::Baggage::Set (C++ function), 32
- opentelemetry::baggage::Baggage::ToHeader (C++ function), 32
- opentelemetry::baggage::GetBaggage (C++ function), 172
- opentelemetry::baggage::kBaggageHeader (C++ member), 175
- opentelemetry::baggage::propagation::BaggagePropagator (C++ class), 33
- opentelemetry::baggage::propagation::BaggagePropagator::Extract (C++ function), 33
- opentelemetry::baggage::propagation::BaggagePropagator::Fields (C++ function), 33
- opentelemetry::baggage::propagation::BaggagePropagator::Inject (C++ function), 33
- opentelemetry::baggage::SetBaggage (C++ function), 172
- opentelemetry::common::AttributeValue (C++ type), 182
- opentelemetry::common::DurationUtil (C++ class), 33
- opentelemetry::common::DurationUtil::AdjustWaitForTimeout (C++ function), 34
- opentelemetry::common::KeyValueIterable (C++ class), 34
- opentelemetry::common::KeyValueIterable::~~KeyValueIterable (C++ function), 34
- opentelemetry::common::KeyValueIterable::ForEachKeyValue (C++ function), 34
- opentelemetry::common::KeyValueIterable::size (C++ function), 34
- opentelemetry::common::NoopKeyValueIterable (C++ class), 35
- opentelemetry::common::NoopKeyValueIterable::~~NoopKeyValueIterable (C++ function), 35
- opentelemetry::common::NoopKeyValueIterable::ForEachKeyValue (C++ function), 35
- opentelemetry::common::NoopKeyValueIterable::size (C++ function), 35
- opentelemetry::common::SteadyTimestamp (C++ class), 35
- opentelemetry::common::SteadyTimestamp::operator std::chrono::steady\_clock::time\_point (C++ function), 36
- opentelemetry::common::SteadyTimestamp::operator!= (C++ function), 36
- opentelemetry::common::SteadyTimestamp::operator== (C++ function), 36
- opentelemetry::common::SteadyTimestamp::SteadyTimestamp (C++ function), 36
- opentelemetry::common::SteadyTimestamp::time\_since\_epoch (C++ function), 36
- opentelemetry::common::SystemTimestamp (C++ class), 36
- opentelemetry::common::SystemTimestamp::operator std::chrono::system\_clock::time\_point (C++ function), 37
- opentelemetry::common::SystemTimestamp::operator!= (C++ function), 37

opentelemetry::common::SystemTimestamp::operator== (C++ function), 40  
 (C++ function), 37  
 opentelemetry::common::SystemTimestamp::SystemTimestamp (C++ function), 40  
 (C++ function), 37  
 opentelemetry::common::SystemTimestamp::time\_since\_epoch (C++ class), 40  
 (C++ function), 37  
 opentelemetry::context::Context (C++ class), 37  
 opentelemetry::context::Context::Context (C++ function), 37  
 opentelemetry::context::Context::GetValue (C++ function), 38  
 opentelemetry::context::Context::HasKey (C++ function), 38  
 opentelemetry::context::Context::operator== (C++ function), 38  
 opentelemetry::context::Context::SetValue (C++ function), 38  
 opentelemetry::context::Context::SetValues (C++ function), 37  
 opentelemetry::context::ContextValue (C++ type), 182  
 opentelemetry::context::GetDefaultStorage (C++ function), 173  
 opentelemetry::context::propagation::CompositedPropagator (C++ class), 38  
 opentelemetry::context::propagation::CompositedPropagator::CompositedPropagator (C++ function), 38  
 opentelemetry::context::propagation::CompositedPropagator::Extract (C++ function), 38  
 opentelemetry::context::propagation::CompositedPropagator::Fields (C++ function), 38  
 opentelemetry::context::propagation::CompositedPropagator::Inject (C++ function), 38  
 opentelemetry::context::propagation::GlobalTextMapPropagator (C++ class), 39  
 opentelemetry::context::propagation::GlobalTextMapPropagator::GetGlobalPropagator (C++ function), 39  
 opentelemetry::context::propagation::GlobalTextMapPropagator::SetGlobalPropagator (C++ function), 39  
 opentelemetry::context::propagation::NoOpPropagator (C++ class), 39  
 opentelemetry::context::propagation::NoOpPropagator::Extract (C++ function), 39  
 opentelemetry::context::propagation::NoOpPropagator::Fields (C++ function), 39  
 opentelemetry::context::propagation::NoOpPropagator::Inject (C++ function), 39  
 opentelemetry::context::propagation::TextMapCarrier (C++ class), 40  
 opentelemetry::context::propagation::TextMapCarrier::TextMapCarrier (C++ function), 40  
 opentelemetry::context::propagation::TextMapCarrier::Keys (C++ function), 40  
 opentelemetry::context::propagation::TextMapCarrier::operator== (C++ function), 40  
 opentelemetry::context::propagation::TextMapCarrier::Set (C++ function), 40  
 opentelemetry::context::propagation::TextMapPropagator (C++ class), 40  
 opentelemetry::context::propagation::TextMapPropagator::~TextMapPropagator (C++ function), 41  
 opentelemetry::context::propagation::TextMapPropagator::Extract (C++ function), 41  
 opentelemetry::context::propagation::TextMapPropagator::Fields (C++ function), 41  
 opentelemetry::context::propagation::TextMapPropagator::Inject (C++ function), 41  
 opentelemetry::context::RuntimeContext (C++ class), 41  
 opentelemetry::context::RuntimeContext::Attach (C++ function), 41  
 opentelemetry::context::RuntimeContext::Detach (C++ function), 41  
 opentelemetry::context::RuntimeContext::GetConstRuntimeContext (C++ function), 41  
 opentelemetry::context::RuntimeContext::GetCurrent (C++ function), 41  
 opentelemetry::context::RuntimeContext::GetValue (C++ function), 41  
 opentelemetry::context::RuntimeContext::SetRuntimeContextS (C++ function), 41  
 opentelemetry::context::RuntimeContext::SetValue (C++ function), 41  
 opentelemetry::context::RuntimeContextStorage (C++ class), 42  
 opentelemetry::context::RuntimeContextStorage::~RuntimeContextStorage (C++ function), 42  
 opentelemetry::context::RuntimeContextStorage::Attach (C++ function), 42  
 opentelemetry::context::RuntimeContextStorage::CreateToken (C++ function), 43  
 opentelemetry::context::RuntimeContextStorage::Detach (C++ function), 42  
 opentelemetry::context::RuntimeContextStorage::GetCurrent (C++ function), 42  
 opentelemetry::context::ThreadLocalContextStorage (C++ class), 43  
 opentelemetry::context::ThreadLocalContextStorage::Attach (C++ function), 43  
 opentelemetry::context::ThreadLocalContextStorage::Detach (C++ function), 43  
 opentelemetry::context::ThreadLocalContextStorage::GetCurrent (C++ function), 43  
 opentelemetry::context::ThreadLocalContextStorage::ThreadL (C++ function), 43  
 opentelemetry::context::Token (C++ class), 43  
 opentelemetry::context::Token::~Token (C++ function), 44

opentelemetry::context::Token::operator== (C++ function), 44  
 opentelemetry::metrics::Counter (C++ class), 44  
 opentelemetry::metrics::Counter::Add (C++ function), 44, 45  
 opentelemetry::metrics::Histogram (C++ class), 45  
 opentelemetry::metrics::Histogram::Record (C++ function), 46  
 opentelemetry::metrics::Meter (C++ class), 46  
 opentelemetry::metrics::Meter::~Meter (C++ function), 47  
 opentelemetry::metrics::Meter::CreateDoubleCounter (C++ function), 47  
 opentelemetry::metrics::Meter::CreateDoubleHistogram (C++ function), 48  
 opentelemetry::metrics::Meter::CreateDoubleObservableCounter (C++ function), 47  
 opentelemetry::metrics::Meter::CreateDoubleObservableGauge (C++ function), 48  
 opentelemetry::metrics::Meter::CreateDoubleObservableUpDownCounter (C++ function), 49  
 opentelemetry::metrics::Meter::CreateDoubleUpDownCounter (C++ function), 48  
 opentelemetry::metrics::Meter::CreateInt64ObservableCounter (C++ function), 47  
 opentelemetry::metrics::Meter::CreateInt64ObservableGauge (C++ function), 48  
 opentelemetry::metrics::Meter::CreateInt64ObservableUpDownCounter (C++ function), 48  
 opentelemetry::metrics::Meter::CreateInt64UpDownCounter (C++ function), 48  
 opentelemetry::metrics::Meter::CreateUInt64Counter (C++ function), 47  
 opentelemetry::metrics::Meter::CreateUInt64Histogram (C++ function), 47  
 opentelemetry::metrics::MeterProvider (C++ class), 49  
 opentelemetry::metrics::MeterProvider::~MeterProvider (C++ function), 50  
 opentelemetry::metrics::MeterProvider::GetMeter (C++ function), 50  
 opentelemetry::metrics::NoopCounter (C++ class), 50  
 opentelemetry::metrics::NoopCounter::Add (C++ function), 50  
 opentelemetry::metrics::NoopCounter::NoopCounter (C++ function), 50  
 opentelemetry::metrics::NoopHistogram (C++ class), 51  
 opentelemetry::metrics::NoopHistogram::NoopHistogram (C++ function), 51  
 opentelemetry::metrics::NoopHistogram::Record (C++ function), 51  
 opentelemetry::metrics::NoopMeter (C++ class), 51  
 opentelemetry::metrics::NoopMeter::CreateDoubleCounter (C++ function), 52  
 opentelemetry::metrics::NoopMeter::CreateDoubleHistogram (C++ function), 52  
 opentelemetry::metrics::NoopMeter::CreateDoubleObservableCounter (C++ function), 52  
 opentelemetry::metrics::NoopMeter::CreateDoubleObservableGauge (C++ function), 53  
 opentelemetry::metrics::NoopMeter::CreateDoubleObservableUpDownCounter (C++ function), 54  
 opentelemetry::metrics::NoopMeter::CreateDoubleUpDownCounter (C++ function), 53  
 opentelemetry::metrics::NoopMeter::CreateInt64ObservableCounter (C++ function), 52  
 opentelemetry::metrics::NoopMeter::CreateInt64ObservableGauge (C++ function), 53  
 opentelemetry::metrics::NoopMeter::CreateInt64ObservableUpDownCounter (C++ function), 53  
 opentelemetry::metrics::NoopMeter::CreateInt64UpDownCounter (C++ function), 53  
 opentelemetry::metrics::NoopMeter::CreateUInt64Counter (C++ function), 51  
 opentelemetry::metrics::NoopMeter::CreateUInt64Histogram (C++ function), 52  
 opentelemetry::metrics::NoopMeterProvider (C++ class), 55  
 opentelemetry::metrics::NoopMeterProvider::GetMeter (C++ function), 55  
 opentelemetry::metrics::NoopMeterProvider::NoopMeterProvider (C++ function), 55  
 opentelemetry::metrics::NoopObservableInstrument (C++ class), 55  
 opentelemetry::metrics::NoopObservableInstrument::AddCallback (C++ function), 55  
 opentelemetry::metrics::NoopObservableInstrument::NoopObservableInstrument (C++ function), 55  
 opentelemetry::metrics::NoopObservableInstrument::RemoveCallback (C++ function), 55  
 opentelemetry::metrics::NoopUpDownCounter (C++ class), 56  
 opentelemetry::metrics::NoopUpDownCounter::~NoopUpDownCounter (C++ function), 56  
 opentelemetry::metrics::NoopUpDownCounter::Add (C++ function), 56  
 opentelemetry::metrics::NoopUpDownCounter::NoopUpDownCounter (C++ function), 56  
 opentelemetry::metrics::ObservableCallbackPtr (C++ type), 183  
 opentelemetry::metrics::ObservableInstrument (C++ class), 57  
 opentelemetry::metrics::ObservableInstrument::~ObservableInstrument (C++ function), 57

opentelemetry::metrics::ObservableInstrument::AddCallback: sdk::metrics::Aggregation::Aggregate (C++ function), 57

opentelemetry::metrics::ObservableInstrument::AddCallback: sdk::metrics::Aggregation::Aggregate (C++ function), 62

opentelemetry::metrics::ObservableInstrument::ObservableInstrument: sdk::metrics::Aggregation::Diff (C++ function), 57

opentelemetry::metrics::ObservableInstrument::ObservableInstrument: sdk::metrics::Aggregation::Diff (C++ function), 62

opentelemetry::metrics::ObservableInstrument::RemoveCallback: sdk::metrics::Aggregation::Merge (C++ function), 57

opentelemetry::metrics::ObservableInstrument::RemoveCallback: sdk::metrics::Aggregation::Merge (C++ function), 62

opentelemetry::metrics::ObserverResult (C++ type), 183

opentelemetry::metrics::ObserverResult: sdk::metrics::Aggregation::ToPoint (C++ function), 62

opentelemetry::metrics::ObserverResultT (C++ class), 57

opentelemetry::metrics::ObserverResultT: sdk::metrics::AggregationConfig (C++ class), 63

opentelemetry::metrics::ObserverResultT::~ObserverResultT: sdk::metrics::AggregationConfig::~AggregationConfig (C++ function), 57

opentelemetry::metrics::ObserverResultT::~ObserverResultT: sdk::metrics::AggregationConfig::~AggregationConfig (C++ function), 63

opentelemetry::metrics::ObserverResultT::ObserverResultT: sdk::metrics::AggregationTemporality (C++ function), 57

opentelemetry::metrics::ObserverResultT::ObserverResultT: sdk::metrics::AggregationTemporality (C++ enum), 166

opentelemetry::metrics::Provider (C++ class), 58

opentelemetry::metrics::Provider: sdk::metrics::AggregationTemporality::kCumulative (C++ enumerator), 166

opentelemetry::metrics::Provider::GetMeterProvider: sdk::metrics::AggregationTemporality::kDelta (C++ enumerator), 166

opentelemetry::metrics::Provider::SetMeterProvider: sdk::metrics::AggregationTemporality::kUnspecified (C++ enumerator), 166

opentelemetry::metrics::SynchronousInstrument (C++ class), 59

opentelemetry::metrics::SynchronousInstrument: sdk::metrics::AggregationTemporalitySelector (C++ type), 183

opentelemetry::metrics::SynchronousInstrument::SynchronousInstrument: sdk::metrics::AggregationType (C++ function), 59

opentelemetry::metrics::SynchronousInstrument::SynchronousInstrument: sdk::metrics::AggregationType (C++ enum), 166

opentelemetry::metrics::SynchronousInstrument::SynchronousInstrument: sdk::metrics::AggregationType::kDefault (C++ function), 59

opentelemetry::metrics::SynchronousInstrument::SynchronousInstrument: sdk::metrics::AggregationType::kDefault (C++ enumerator), 167

opentelemetry::metrics::UpDownCounter (C++ class), 59

opentelemetry::metrics::UpDownCounter: sdk::metrics::AggregationType::kDrop (C++ enumerator), 166

opentelemetry::metrics::UpDownCounter::Add: sdk::metrics::AggregationType::kHistogram (C++ function), 59, 60

opentelemetry::metrics::UpDownCounter::Add: sdk::metrics::AggregationType::kHistogram (C++ enumerator), 166

opentelemetry::sdk::instrumentationlibrary::InstrumentationLibrary: sdk::metrics::AggregationType::kLastValue (C++ type), 183

opentelemetry::sdk::instrumentationlibrary::InstrumentationLibrary: sdk::metrics::AggregationType::kLastValue (C++ enumerator), 167

opentelemetry::sdk::instrumentationscope::InstrumentationScope: sdk::metrics::AggregationType::kSum (C++ class), 60

opentelemetry::sdk::instrumentationscope::InstrumentationScope: sdk::metrics::AggregationType::kSum (C++ enumerator), 167

opentelemetry::sdk::instrumentationscope::InstrumentationScope::Create: sdk::AlwaysSampleFilter (C++ function), 61

opentelemetry::sdk::instrumentationscope::InstrumentationScope::Create: sdk::AlwaysSampleFilter (C++ class), 63

opentelemetry::sdk::instrumentationscope::InstrumentationScope::Equal: sdk::AlwaysSampleFilter::AlwaysSampleFilter (C++ function), 60

opentelemetry::sdk::instrumentationscope::InstrumentationScope::Equal: sdk::AlwaysSampleFilter::AlwaysSampleFilter (C++ function), 63

opentelemetry::sdk::instrumentationscope::InstrumentationScope::GetName: sdk::AlwaysSampleFilter::ShouldSample (C++ function), 61

opentelemetry::sdk::instrumentationscope::InstrumentationScope::GetName: sdk::AlwaysSampleFilter::ShouldSample (C++ function), 63

opentelemetry::sdk::instrumentationscope::InstrumentationScope::GetSchemaURL: sdk::AsyncMetricStorage (C++ function), 61

opentelemetry::sdk::instrumentationscope::InstrumentationScope::GetSchemaURL: sdk::AsyncMetricStorage (C++ class), 64

opentelemetry::sdk::instrumentationscope::InstrumentationScope::GetVersion: sdk::AsyncMetricStorage::AsyncMetricStorage (C++ function), 61

opentelemetry::sdk::instrumentationscope::InstrumentationScope::GetVersion: sdk::AsyncMetricStorage::AsyncMetricStorage (C++ function), 64

opentelemetry::sdk::instrumentationscope::InstrumentationScope::HashCode: sdk::AsyncMetricStorage::Collect (C++ function), 60

opentelemetry::sdk::instrumentationscope::InstrumentationScope::HashCode: sdk::AsyncMetricStorage::Collect (C++ function), 64

opentelemetry::sdk::instrumentationscope::InstrumentationScope::InstrumentationScope: sdk::AsyncMetricStorage::Record (C++ function), 60

opentelemetry::sdk::instrumentationscope::InstrumentationScope::InstrumentationScope: sdk::AsyncMetricStorage::Record (C++ function), 64

opentelemetry::sdk::instrumentationscope::InstrumentationScope::operator=: sdk::AsyncMetricStorage::RecordDou (C++ function), 60

opentelemetry::sdk::instrumentationscope::InstrumentationScope::operator=: sdk::AsyncMetricStorage::RecordDou (C++ function), 64

opentelemetry::sdk::metrics::Aggregation: sdk::metrics::AsyncMetricStorage::RecordLong (C++ class), 62

opentelemetry::sdk::metrics::Aggregation: sdk::metrics::AsyncMetricStorage::RecordLong (C++ function), 64

opentelemetry::sdk::metrics::Aggregation::~Aggregation: sdk::metrics::AsyncMultiMetricStorage (C++ function), 62

opentelemetry::sdk::metrics::Aggregation::~Aggregation: sdk::metrics::AsyncMultiMetricStorage (C++ class), 65

`opentelemetry::sdk::metrics::AsyncMultiMetricStorage::AddStorage` (C++ function), 65  
`opentelemetry::sdk::metrics::AsyncMultiMetricStorage::GetStorage` (C++ function), 65  
`opentelemetry::sdk::metrics::AsyncMultiMetricStorage::RemoveStorage` (C++ function), 65  
`opentelemetry::sdk::metrics::AsyncWritableMetricStorage::Add` (C++ class), 65  
`opentelemetry::sdk::metrics::AsyncWritableMetricStorage::Get` (C++ function), 65  
`opentelemetry::sdk::metrics::AsyncWritableMetricStorage::Remove` (C++ function), 65  
`opentelemetry::sdk::metrics::AsyncWritableMetricStorage::Record` (C++ function), 65  
`opentelemetry::sdk::metrics::AsyncWritableMetricStorage::RecordLong` (C++ function), 65  
`opentelemetry::sdk::metrics::AttributeHashGenerator` (C++ class), 66  
`opentelemetry::sdk::metrics::AttributeHashGenerator::GetDefault` (C++ function), 66  
`opentelemetry::sdk::metrics::AttributesHashMap` (C++ class), 66  
`opentelemetry::sdk::metrics::AttributesHashMap::Get` (C++ function), 66  
`opentelemetry::sdk::metrics::AttributesHashMap::GetAllEntries` (C++ function), 66  
`opentelemetry::sdk::metrics::AttributesHashMap::GetOldSetDefault` (C++ function), 66  
`opentelemetry::sdk::metrics::AttributesHashMap::Has` (C++ function), 66  
`opentelemetry::sdk::metrics::AttributesHashMap::Set` (C++ function), 66  
`opentelemetry::sdk::metrics::AttributesHashMap::Size` (C++ function), 66  
`opentelemetry::sdk::metrics::AttributesProcessor` (C++ class), 67  
`opentelemetry::sdk::metrics::AttributesProcessor::AlterAttributesProcessor` (C++ function), 67  
`opentelemetry::sdk::metrics::AttributesProcessor::Process` (C++ function), 67  
`opentelemetry::sdk::metrics::BucketBinarySearch` (C++ function), 173  
`opentelemetry::sdk::metrics::CollectorHandle` (C++ class), 68  
`opentelemetry::sdk::metrics::CollectorHandle::GetCollectorHandle` (C++ function), 68  
`opentelemetry::sdk::metrics::CollectorHandle::GetDefaultAggregation` (C++ function), 68  
`opentelemetry::sdk::metrics::DefaultAggregation` (C++ class), 68  
`opentelemetry::sdk::metrics::DefaultAggregation::GetAggregation` (C++ function), 68  
`opentelemetry::sdk::metrics::DefaultAggregation::CreateAggregation` (C++ function), 68  
`opentelemetry::sdk::metrics::DefaultAttributesProcessor` (C++ class), 69  
`opentelemetry::sdk::metrics::DefaultAttributesProcessor::Process` (C++ function), 69  
`opentelemetry::sdk::metrics::DoubleCounter` (C++ class), 69  
`opentelemetry::sdk::metrics::DoubleCounter::Add` (C++ function), 69  
`opentelemetry::sdk::metrics::DoubleCounter::DoubleCounter` (C++ function), 69  
`opentelemetry::sdk::metrics::DoubleHistogram` (C++ class), 70  
`opentelemetry::sdk::metrics::DoubleHistogram::DoubleHistogram` (C++ function), 70  
`opentelemetry::sdk::metrics::DoubleHistogram::Record` (C++ function), 70  
`opentelemetry::sdk::metrics::DoubleHistogramAggregation` (C++ class), 71  
`opentelemetry::sdk::metrics::DoubleHistogramAggregation::Add` (C++ function), 71  
`opentelemetry::sdk::metrics::DoubleHistogramAggregation::Delete` (C++ function), 71  
`opentelemetry::sdk::metrics::DoubleHistogramAggregation::Merge` (C++ function), 71  
`opentelemetry::sdk::metrics::DoubleHistogramAggregation::ToPoint` (C++ function), 71  
`opentelemetry::sdk::metrics::DoubleLastValueAggregation` (C++ class), 71  
`opentelemetry::sdk::metrics::DoubleLastValueAggregation::Add` (C++ function), 71  
`opentelemetry::sdk::metrics::DoubleLastValueAggregation::Delete` (C++ function), 72  
`opentelemetry::sdk::metrics::DoubleLastValueAggregation::Merge` (C++ function), 72  
`opentelemetry::sdk::metrics::DoubleLastValueAggregation::ToPoint` (C++ function), 72  
`opentelemetry::sdk::metrics::DoubleSumAggregation` (C++ class), 72  
`opentelemetry::sdk::metrics::DoubleSumAggregation::Aggregate` (C++ function), 72  
`opentelemetry::sdk::metrics::DoubleSumAggregation::Diff` (C++ function), 72  
`opentelemetry::sdk::metrics::DoubleSumAggregation::DoubleSumAggregation` (C++ function), 72  
`opentelemetry::sdk::metrics::DoubleSumAggregation::Merge` (C++ function), 72  
`opentelemetry::sdk::metrics::DoubleSumAggregation::ToPoint` (C++ function), 72  
`opentelemetry::sdk::metrics::DoubleUpDownCounter` (C++ class), 73

opentelemetry::sdk::metrics::DoubleUpDownCounter (C++ function), 73

opentelemetry::sdk::metrics::DoubleUpDownCounter::Delta (C++ function), 73

opentelemetry::sdk::metrics::DropAggregation (C++ class), 73

opentelemetry::sdk::metrics::DropAggregation::Aggregate (C++ function), 74

opentelemetry::sdk::metrics::DropAggregation::Diff (C++ function), 74

opentelemetry::sdk::metrics::DropAggregation::Drop (C++ function), 74

opentelemetry::sdk::metrics::DropAggregation::Merge (C++ function), 74

opentelemetry::sdk::metrics::DropAggregation::DropPointData (C++ class), 74

opentelemetry::sdk::metrics::DropAggregation::DropPointData::DropPointData (C++ function), 74

opentelemetry::sdk::metrics::DropAggregation::operator (C++ function), 74

opentelemetry::sdk::metrics::ExactPredicate (C++ class), 75

opentelemetry::sdk::metrics::ExactPredicate::ExactPredicate (C++ function), 75

opentelemetry::sdk::metrics::ExactPredicate::Match (C++ function), 75

opentelemetry::sdk::metrics::ExemplarData (C++ class), 75

opentelemetry::sdk::metrics::ExemplarData::Create (C++ function), 75

opentelemetry::sdk::metrics::ExemplarData::CreateDropPointData (C++ function), 75

opentelemetry::sdk::metrics::ExemplarData::CreateHistogramPointData (C++ function), 75

opentelemetry::sdk::metrics::ExemplarData::CreateSumPointData (C++ function), 75

opentelemetry::sdk::metrics::ExemplarData::GetAttributes (C++ function), 75

opentelemetry::sdk::metrics::ExemplarData::GetIntegerAttributes (C++ function), 75

opentelemetry::sdk::metrics::ExemplarData::GetStringAttributes (C++ function), 75

opentelemetry::sdk::metrics::ExemplarFilter (C++ class), 76

opentelemetry::sdk::metrics::ExemplarFilter::~ExemplarFilter (C++ function), 76

opentelemetry::sdk::metrics::ExemplarFilter::GetAttributesFilter (C++ function), 76

opentelemetry::sdk::metrics::ExemplarFilter::GetNeverSampleFilter (C++ function), 76

opentelemetry::sdk::metrics::ExemplarFilter::GetIntelligenceSampleFilter (C++ function), 76

opentelemetry::sdk::metrics::ExemplarFilter::ShouldSample (C++ function), 76

opentelemetry::sdk::metrics::ExemplarReservoir (C++ class), 77

opentelemetry::sdk::metrics::ExemplarReservoir::~ExemplarFilter (C++ function), 77

opentelemetry::sdk::metrics::ExemplarReservoir::CollectAndReset (C++ function), 77

opentelemetry::sdk::metrics::ExemplarReservoir::GetFilter (C++ function), 78

opentelemetry::sdk::metrics::ExemplarReservoir::GetHistogram (C++ function), 78

opentelemetry::sdk::metrics::ExemplarReservoir::GetNoExemplars (C++ function), 78

opentelemetry::sdk::metrics::ExemplarReservoir::OfferMeasurements (C++ function), 77

opentelemetry::sdk::metrics::FilteredExemplarReservoir (C++ class), 78

opentelemetry::sdk::metrics::FilteredExemplarReservoir::CollectAndReset (C++ function), 78

opentelemetry::sdk::metrics::FilteredExemplarReservoir::Filter (C++ function), 78

opentelemetry::sdk::metrics::FilteredExemplarReservoir::OfferMeasurements (C++ function), 78

opentelemetry::sdk::metrics::FilteringAttributesProcessor (C++ class), 79

opentelemetry::sdk::metrics::FilteringAttributesProcessor::Match (C++ function), 79

opentelemetry::sdk::metrics::FilteringAttributesProcessor::operator (C++ function), 79

opentelemetry::sdk::metrics::FixedSizeExemplarReservoir (C++ class), 80

opentelemetry::sdk::metrics::FixedSizeExemplarReservoir::CollectAndReset (C++ function), 80

opentelemetry::sdk::metrics::FixedSizeExemplarReservoir::Filter (C++ function), 80

opentelemetry::sdk::metrics::FixedSizeExemplarReservoir::OfferMeasurements (C++ function), 80

opentelemetry::sdk::metrics::HistogramAggregationConfig (C++ class), 80

opentelemetry::sdk::metrics::HistogramAggregationConfig::buckets (C++ member), 81

opentelemetry::sdk::metrics::HistogramAggregationConfig::range (C++ member), 81

opentelemetry::sdk::metrics::HistogramDiff (C++ function), 173

opentelemetry::sdk::metrics::HistogramExemplarReservoir (C++ class), 81

opentelemetry::sdk::metrics::HistogramExemplarReservoir::CollectAndReset (C++ function), 81

opentelemetry::sdk::metrics::HistogramExemplarReservoir::Filter (C++ class), 81, 82

opentelemetry::sdk::metrics::HistogramExemplarReservoir::FilteringAttributesProcessor (C++ function), 82



opentelemetry::sdk::metrics::HistogramExemplarResetHintHistogramGetSelectorInstrumentSelector::GetInstrumentSelector::GetInstrumentSelector (C++ function), 82

opentelemetry::sdk::metrics::HistogramExemplarResetHintHistogramExemplarResetHintInstrumentSelector::GetInstrumentSelector (C++ function), 81

opentelemetry::sdk::metrics::HistogramMerge opentelemetry::sdk::metrics::InstrumentSelector::GetNameFile (C++ function), 174

opentelemetry::sdk::metrics::HistogramPointData opentelemetry::sdk::metrics::InstrumentSelector::InstrumentSelector (C++ class), 83

opentelemetry::sdk::metrics::HistogramPointData opentelemetry::sdk::metrics::InstrumentType (C++ member), 83

opentelemetry::sdk::metrics::HistogramPointData opentelemetry::sdk::metrics::InstrumentType::kCounter (C++ member), 83

opentelemetry::sdk::metrics::HistogramPointData opentelemetry::sdk::metrics::InstrumentType::kHistogram (C++ member), 83

opentelemetry::sdk::metrics::HistogramPointData opentelemetry::sdk::metrics::InstrumentType::kObservableCounter (C++ function), 83

opentelemetry::sdk::metrics::HistogramPointData opentelemetry::sdk::metrics::InstrumentType::kObservableGauge (C++ member), 83

opentelemetry::sdk::metrics::HistogramPointData opentelemetry::sdk::metrics::InstrumentType::kObservableUpDownCounter (C++ member), 83

opentelemetry::sdk::metrics::HistogramPointData opentelemetry::sdk::metrics::InstrumentType::kUpDownCounter (C++ function), 83

opentelemetry::sdk::metrics::HistogramPointData opentelemetry::sdk::metrics::InstrumentValueType (C++ member), 83

opentelemetry::sdk::metrics::HistogramPointData opentelemetry::sdk::metrics::InstrumentValueType::kDouble (C++ member), 83

opentelemetry::sdk::metrics::InstrumentClass opentelemetry::sdk::metrics::InstrumentValueType::kFloat (C++ enum), 167

opentelemetry::sdk::metrics::InstrumentClass::kAsync opentelemetry::sdk::metrics::InstrumentValueType::kInt (C++ enumerator), 167

opentelemetry::sdk::metrics::InstrumentClass::kSync opentelemetry::sdk::metrics::InstrumentValueType::kLong (C++ enumerator), 167

opentelemetry::sdk::metrics::InstrumentDescriptor opentelemetry::sdk::metrics::kExportIntervalMillis (C++ struct), 24

opentelemetry::sdk::metrics::InstrumentDescriptor opentelemetry::sdk::metrics::kExportTimeOutMillis (C++ member), 24

opentelemetry::sdk::metrics::InstrumentDescriptor opentelemetry::sdk::metrics::LastReportedMetrics (C++ member), 24

opentelemetry::sdk::metrics::InstrumentDescriptor opentelemetry::sdk::metrics::LastReportedMetrics::attribute (C++ member), 24

opentelemetry::sdk::metrics::InstrumentDescriptor opentelemetry::sdk::metrics::LastReportedMetrics::collectible (C++ member), 24

opentelemetry::sdk::metrics::InstrumentDescriptor opentelemetry::sdk::metrics::LastValuePointData (C++ member), 24

opentelemetry::sdk::metrics::InstrumentMetaDatum opentelemetry::sdk::metrics::LastValuePointData::is\_last\_value (C++ class), 84

opentelemetry::sdk::metrics::InstrumentMetaDatum opentelemetry::sdk::metrics::LastValuePointData::LastValuePointData::LastValuePointData (C++ function), 84

opentelemetry::sdk::metrics::InstrumentMetaDatum opentelemetry::sdk::metrics::LastValuePointData::operator= (C++ function), 84

opentelemetry::sdk::metrics::InstrumentMetaDatum opentelemetry::sdk::metrics::LastValuePointData::sample\_timestamp (C++ function), 84

opentelemetry::sdk::metrics::InstrumentMetaDatum opentelemetry::sdk::metrics::LastValuePointData::value (C++ function), 84

`opentelemetry::sdk::metrics::LongCounter` (C++ class), 85  
`opentelemetry::sdk::metrics::LongCounter::Add` (C++ function), 85  
`opentelemetry::sdk::metrics::LongCounter::LongCounter` (C++ function), 85  
`opentelemetry::sdk::metrics::LongHistogram` (C++ class), 86  
`opentelemetry::sdk::metrics::LongHistogram::LongHistogram` (C++ function), 86  
`opentelemetry::sdk::metrics::LongHistogram::Record` (C++ function), 86  
`opentelemetry::sdk::metrics::LongHistogramAggregation` (C++ class), 87  
`opentelemetry::sdk::metrics::LongHistogramAggregation::Aggregate` (C++ function), 87  
`opentelemetry::sdk::metrics::LongHistogramAggregation::Diff` (C++ function), 87  
`opentelemetry::sdk::metrics::LongHistogramAggregation::Merge` (C++ function), 87  
`opentelemetry::sdk::metrics::LongHistogramAggregation::ForPoints` (C++ function), 87  
`opentelemetry::sdk::metrics::LongLastValueAggregation` (C++ class), 87  
`opentelemetry::sdk::metrics::LongLastValueAggregation::Aggregate` (C++ function), 87  
`opentelemetry::sdk::metrics::LongLastValueAggregation::Diff` (C++ function), 88  
`opentelemetry::sdk::metrics::LongLastValueAggregation::Merge` (C++ function), 87  
`opentelemetry::sdk::metrics::LongLastValueAggregation::ForPoints` (C++ function), 88  
`opentelemetry::sdk::metrics::LongSumAggregation` (C++ class), 88  
`opentelemetry::sdk::metrics::LongSumAggregation::Aggregate` (C++ function), 88  
`opentelemetry::sdk::metrics::LongSumAggregation::Diff` (C++ function), 88  
`opentelemetry::sdk::metrics::LongSumAggregation::Merge` (C++ function), 88  
`opentelemetry::sdk::metrics::LongSumAggregation::ForPoints` (C++ function), 88  
`opentelemetry::sdk::metrics::LongUpDownCounter` (C++ class), 89  
`opentelemetry::sdk::metrics::LongUpDownCounter::Add` (C++ function), 89  
`opentelemetry::sdk::metrics::LongUpDownCounter::LongUpDownCounter` (C++ function), 89  
`opentelemetry::sdk::metrics::MatchEverythingPattern` (C++ class), 89  
`opentelemetry::sdk::metrics::MatchNothingPattern` (C++ class), 90  
`opentelemetry::sdk::metrics::Meter` (C++ class), 90  
`opentelemetry::sdk::metrics::Meter::Collect` (C++ function), 93  
`opentelemetry::sdk::metrics::Meter::CreateDoubleCounter` (C++ function), 90  
`opentelemetry::sdk::metrics::Meter::CreateDoubleHistogram` (C++ function), 91  
`opentelemetry::sdk::metrics::Meter::CreateDoubleObservableCounter` (C++ function), 91  
`opentelemetry::sdk::metrics::Meter::CreateDoubleObservableUpDownCounter` (C++ function), 92  
`opentelemetry::sdk::metrics::Meter::CreateDoubleUpDownCounter` (C++ function), 92  
`opentelemetry::sdk::metrics::Meter::CreateInt64ObservableCounter` (C++ function), 91  
`opentelemetry::sdk::metrics::Meter::CreateInt64ObservableUpDownCounter` (C++ function), 92  
`opentelemetry::sdk::metrics::Meter::CreateUInt64Counter` (C++ function), 90  
`opentelemetry::sdk::metrics::Meter::CreateUInt64Histogram` (C++ function), 91  
`opentelemetry::sdk::metrics::Meter::GetInstrumentationLibrary` (C++ function), 93  
`opentelemetry::sdk::metrics::Meter::GetInstrumentationScope` (C++ function), 93  
`opentelemetry::sdk::metrics::Meter::Meter` (C++ function), 90  
`opentelemetry::sdk::metrics::MeterContext` (C++ class), 94  
`opentelemetry::sdk::metrics::MeterContext::AddMeter` (C++ function), 95  
`opentelemetry::sdk::metrics::MeterContext::AddMetricReader` (C++ function), 95  
`opentelemetry::sdk::metrics::MeterContext::AddView` (C++ function), 95  
`opentelemetry::sdk::metrics::MeterContext::ForceFlush` (C++ function), 95  
`opentelemetry::sdk::metrics::MeterContext::ForEachMeter` (C++ function), 94  
`opentelemetry::sdk::metrics::MeterContext::GetCollectors` (C++ function), 94  
`opentelemetry::sdk::metrics::MeterContext::GetMeters` (C++ function), 94

opentelemetry::sdk::metrics::MeterContext::GetResourceName opentelemetry::sdk::metrics::MetricData  
 (C++ function), 94 (C++ class), 98  
 opentelemetry::sdk::metrics::MeterContext::GetSDKStartTime opentelemetry::sdk::metrics::MetricData::aggregation\_tempo  
 (C++ function), 94 (C++ member), 98  
 opentelemetry::sdk::metrics::MeterContext::GetViewRegistry opentelemetry::sdk::metrics::MetricData::end\_ts  
 (C++ function), 94 (C++ member), 98  
 opentelemetry::sdk::metrics::MeterContext::MeterContext opentelemetry::sdk::metrics::MetricData::instrument\_descri  
 (C++ function), 94 (C++ member), 98  
 opentelemetry::sdk::metrics::MeterContext::Shutdown opentelemetry::sdk::metrics::MetricData::point\_data\_attr\_  
 (C++ function), 95 (C++ member), 98  
 opentelemetry::sdk::metrics::MeterProvider opentelemetry::sdk::metrics::MetricData::start\_ts  
 (C++ class), 96 (C++ member), 98  
 opentelemetry::sdk::metrics::MeterProvider::~~MeterProvider opentelemetry::sdk::metrics::MetricProducer  
 (C++ function), 96 (C++ class), 99  
 opentelemetry::sdk::metrics::MeterProvider::AddMetricReader opentelemetry::sdk::metrics::MetricProducer::~~MetricProdu  
 (C++ function), 96 (C++ function), 99  
 opentelemetry::sdk::metrics::MeterProvider::AddView opentelemetry::sdk::metrics::MetricProducer::Collect  
 (C++ function), 96 (C++ function), 99  
 opentelemetry::sdk::metrics::MeterProvider::ForceFlush opentelemetry::sdk::metrics::MetricProducer::MetricProdu  
 (C++ function), 96 (C++ function), 99  
 opentelemetry::sdk::metrics::MeterProvider::GetMeter opentelemetry::sdk::metrics::MetricReader  
 (C++ function), 96 (C++ class), 100  
 opentelemetry::sdk::metrics::MeterProvider::GetResourceName opentelemetry::sdk::metrics::MetricReader::~~MetricReader  
 (C++ function), 96 (C++ function), 100  
 opentelemetry::sdk::metrics::MeterProvider::MetricProvider opentelemetry::sdk::metrics::MetricReader::Collect  
 (C++ function), 96 (C++ function), 100  
 opentelemetry::sdk::metrics::MeterProvider::Shutdown opentelemetry::sdk::metrics::MetricReader::ForceFlush  
 (C++ function), 96 (C++ function), 100  
 opentelemetry::sdk::metrics::MeterSelector opentelemetry::sdk::metrics::MetricReader::GetAggregationT  
 (C++ class), 97 (C++ function), 100  
 opentelemetry::sdk::metrics::MeterSelector::GetMeter opentelemetry::sdk::metrics::MetricReader::IsShutdown  
 (C++ function), 97 (C++ function), 100  
 opentelemetry::sdk::metrics::MeterSelector::GetMetricReader opentelemetry::sdk::metrics::MetricReader::MetricReader  
 (C++ function), 97 (C++ function), 100  
 opentelemetry::sdk::metrics::MeterSelector::GetViewRegistry opentelemetry::sdk::metrics::MetricReader::SetMetricProdu  
 (C++ function), 97 (C++ function), 100  
 opentelemetry::sdk::metrics::MeterSelector::MetricSelector opentelemetry::sdk::metrics::MetricReader::Shutdown  
 (C++ function), 97 (C++ function), 100  
 opentelemetry::sdk::metrics::MetricAttributes opentelemetry::sdk::metrics::MetricStorage  
 (C++ type), 184 (C++ class), 101  
 opentelemetry::sdk::metrics::MetricCollector opentelemetry::sdk::metrics::MetricStorage::~~MetricStorage  
 (C++ class), 97 (C++ function), 101  
 opentelemetry::sdk::metrics::MetricCollector::Collect opentelemetry::sdk::metrics::MetricStorage::Collect  
 (C++ function), 98 (C++ function), 101  
 opentelemetry::sdk::metrics::MetricCollector::GetMeter opentelemetry::sdk::metrics::MetricStorage::MetricStorage  
 (C++ function), 98 (C++ function), 101  
 opentelemetry::sdk::metrics::MetricCollector::GetViewRegistry opentelemetry::sdk::metrics::NeverSampleFilter  
 (C++ function), 98 (C++ class), 101  
 opentelemetry::sdk::metrics::MetricCollector::GetAggregationTemporality opentelemetry::sdk::metrics::NeverSampleFilter::NeverSampl  
 (C++ function), 98 (C++ function), 101  
 opentelemetry::sdk::metrics::MetricCollector::MetricCollector opentelemetry::sdk::metrics::NeverSampleFilter::ShouldSampl  
 (C++ function), 98 (C++ function), 101  
 opentelemetry::sdk::metrics::MetricCollector::Shutdown opentelemetry::sdk::metrics::NoExemplarReservoir  
 (C++ function), 98 (C++ class), 102

opentelemetry::sdk::metrics::NoExemplarReservoir::CollectAndReset (C++ function), 102

opentelemetry::sdk::metrics::NoExemplarReservoir::CollectAndReset::metrics::ObserverResultT::GetMeasurement (C++ function), 106

opentelemetry::sdk::metrics::NoExemplarReservoir::CollectAndReset::metrics::ObserverResultT::Observe (C++ function), 102

opentelemetry::sdk::metrics::NoExemplarReservoir::CollectAndReset::metrics::ObserverResultT::ObserverResult (C++ function), 106

opentelemetry::sdk::metrics::NoopAsyncWritableMetricStorage::metrics::PatternPredicate (C++ class), 103

opentelemetry::sdk::metrics::NoopAsyncWritableMetricStorage::SdkCombiner::PatternPredicate::Match (C++ function), 103

opentelemetry::sdk::metrics::NoopAsyncWritableMetricStorage::SdkCombiner::PatternPredicate::PatternPredicate (C++ function), 103

opentelemetry::sdk::metrics::NoopMetricStorage::metrics::PeriodicExportingMetricReader (C++ class), 103

opentelemetry::sdk::metrics::NoopMetricStorage::Collect (C++ function), 103

opentelemetry::sdk::metrics::NoopMetricStorage::SdkCombiner::metrics::PeriodicExportingMetricReader (C++ function), 107

opentelemetry::sdk::metrics::NoopWritableMetricStorage::metrics::PeriodicExportingMetricReader (C++ function), 104

opentelemetry::sdk::metrics::NoopWritableMetricStorage::RecordDouble (C++ function), 104

opentelemetry::sdk::metrics::NoopWritableMetricStorage::RecordLong (C++ function), 104

opentelemetry::sdk::metrics::ObservableCallback::metrics::PeriodicExportingMetricReader (C++ struct), 25

opentelemetry::sdk::metrics::ObservableCallback::RecordCallback::metrics::PointAttributes (C++ member), 25

opentelemetry::sdk::metrics::ObservableCallback::RecordInstrument::metrics::PointDataAttributes (C++ member), 25

opentelemetry::sdk::metrics::ObservableCallback::RecordState::metrics::PointDataAttributes::attribut (C++ member), 25

opentelemetry::sdk::metrics::ObservableInstrument::metrics::PointDataAttributes::point\_da (C++ class), 104

opentelemetry::sdk::metrics::ObservableInstrument::RecordCallback::metrics::PointType (C++ (C++ function), 105

opentelemetry::sdk::metrics::ObservableInstrument::RecordInstrumentDescriptor::Predicate (C++ (C++ function), 105

opentelemetry::sdk::metrics::ObservableInstrument::RecordMetricStorage::metrics::Predicate::~Predicate (C++ function), 105

opentelemetry::sdk::metrics::ObservableInstrument::RecordObservableInstrument::metrics::Predicate::Match (C++ function), 105

opentelemetry::sdk::metrics::ObservableInstrument::RecordRemoveCallback::metrics::PredicateFactory (C++ function), 105

opentelemetry::sdk::metrics::ObservableRegistry::metrics::PredicateFactory::GetPredicate (C++ class), 105

opentelemetry::sdk::metrics::ObservableRegistry::RecordCallback::metrics::PredicateType (C++ function), 105

opentelemetry::sdk::metrics::ObservableRegistry::RecordObserver::metrics::PredicateType::kExact (C++ function), 105

opentelemetry::sdk::metrics::ObservableRegistry::RecordRemoveCallback::metrics::PredicateType::kPattern (C++ function), 105

opentelemetry::sdk::metrics::ObserverResultT opentelemetry::sdk::metrics::PushMetricExporter (C++ class), 106

opentelemetry::sdk::metrics::ObserverResultT::ObserverResultT::metrics::PushMetricExporter::~PushMetric (C++ function), 106

opentelemetry::sdk::metrics::PushMetricExporter ~~opentelemetry::sdk::metrics::Synchronous~~  
 (C++ function), 108 ~~(C++ class), 111~~

opentelemetry::sdk::metrics::PushMetricExporter ~~opentelemetry::sdk::metrics::Synchronous::instrument\_desc~~  
 (C++ function), 108 ~~(C++ member), 111~~

opentelemetry::sdk::metrics::PushMetricExporter ~~opentelemetry::sdk::metrics::Synchronous::storage\_~~  
 (C++ function), 108 ~~(C++ member), 111~~

opentelemetry::sdk::metrics::PushMetricExporter ~~opentelemetry::sdk::metrics::Synchronous::Synchronous~~  
 (C++ function), 109 ~~(C++ function), 111~~

opentelemetry::sdk::metrics::RegisteredView ~~opentelemetry::sdk::metrics::SyncMetricStorage~~  
 (C++ struct), 27 ~~(C++ class), 111~~

opentelemetry::sdk::metrics::RegisteredView::instrument\_selector ~~opentelemetry::sdk::metrics::SyncMetricStorage::Collect~~  
 (C++ member), 27 ~~(C++ function), 112~~

opentelemetry::sdk::metrics::RegisteredView::metric\_selector ~~opentelemetry::sdk::metrics::SyncMultiMetricStorage~~  
 (C++ member), 27 ~~(C++ class), 112~~

opentelemetry::sdk::metrics::RegisteredView::RegisteredView ~~opentelemetry::sdk::metrics::SyncMultiMetricStorage::AddSt~~  
 (C++ function), 27 ~~(C++ function), 113~~

opentelemetry::sdk::metrics::RegisteredView::view ~~opentelemetry::sdk::metrics::SyncMultiMetricStorage::Reco~~  
 (C++ member), 27 ~~(C++ function), 113~~

opentelemetry::sdk::metrics::ReservoirCell ~~opentelemetry::sdk::metrics::SyncMultiMetricStorage::Reco~~  
 (C++ class), 109 ~~(C++ function), 113~~

opentelemetry::sdk::metrics::ReservoirCell::GetAndResetDouble ~~opentelemetry::sdk::metrics::SyncWritableMetricStorage~~  
 (C++ function), 109 ~~(C++ class), 113~~

opentelemetry::sdk::metrics::ReservoirCell::GetAndResetLong ~~opentelemetry::sdk::metrics::SyncWritableMetricStorage::~S~~  
 (C++ function), 109 ~~(C++ function), 113~~

opentelemetry::sdk::metrics::ReservoirCell::RecordDoubleMeasurement ~~opentelemetry::sdk::metrics::SyncWritableMetricStorage::Re~~  
 (C++ function), 109 ~~(C++ function), 113~~

opentelemetry::sdk::metrics::ReservoirCell::RecordLongMeasurement ~~opentelemetry::sdk::metrics::SyncWritableMetricStorage::Re~~  
 (C++ function), 109 ~~(C++ function), 113~~

opentelemetry::sdk::metrics::ReservoirCell::Reset ~~opentelemetry::sdk::metrics::TemporalMetricStorage~~  
 (C++ function), 109 ~~(C++ class), 114~~

opentelemetry::sdk::metrics::ReservoirCell::reset ~~opentelemetry::sdk::metrics::TemporalMetricStorage::build~~  
 (C++ function), 109 ~~(C++ function), 114~~

opentelemetry::sdk::metrics::ResourceMetrics ~~opentelemetry::sdk::metrics::TemporalMetricStorage::Tempo~~  
 (C++ struct), 27 ~~(C++ function), 114~~

opentelemetry::sdk::metrics::ResourceMetrics::resource ~~opentelemetry::sdk::metrics::ValueType (C++~~  
 (C++ member), 27 ~~type), 184~~

opentelemetry::sdk::metrics::ResourceMetrics::scope ~~opentelemetry::sdk::metrics::View (C++ class),~~  
 (C++ member), 27 ~~114~~

opentelemetry::sdk::metrics::ScopeMetrics ~~opentelemetry::sdk::metrics::View::~~View~~  
 (C++ struct), 28 ~~(C++ function), 114~~

opentelemetry::sdk::metrics::ScopeMetrics::metric\_data ~~opentelemetry::sdk::metrics::View::GetAggregationConfig~~  
 (C++ member), 28 ~~(C++ function), 114~~

opentelemetry::sdk::metrics::ScopeMetrics::scope ~~opentelemetry::sdk::metrics::View::GetAggregationType~~  
 (C++ member), 28 ~~(C++ function), 114~~

opentelemetry::sdk::metrics::SumPointData ~~opentelemetry::sdk::metrics::View::GetAttributesProcessor~~  
 (C++ class), 110 ~~(C++ function), 114~~

opentelemetry::sdk::metrics::SumPointData::is\_monotonic ~~opentelemetry::sdk::metrics::View::GetDescription~~  
 (C++ member), 110 ~~(C++ function), 114~~

opentelemetry::sdk::metrics::SumPointData::operator ~~opentelemetry::sdk::metrics::View::GetName~~  
 (C++ function), 110 ~~(C++ function), 114~~

opentelemetry::sdk::metrics::SumPointData::SumPointData ~~opentelemetry::sdk::metrics::View::View~~  
 (C++ function), 110 ~~(C++ function), 114~~

opentelemetry::sdk::metrics::SumPointData::value ~~opentelemetry::sdk::metrics::ViewRegistry~~  
 (C++ member), 110 ~~(C++ class), 115~~

opentelemetry::sdk::metrics::ViewRegistry::~ViewRegistry opentelemetry::sdk::trace::AlwaysOnSampler::GetDescription  
 (C++ function), 115 (C++ function), 119

opentelemetry::sdk::metrics::ViewRegistry::AddView opentelemetry::sdk::trace::AlwaysOnSampler::ShouldSample  
 (C++ function), 115 (C++ function), 119

opentelemetry::sdk::metrics::ViewRegistry::FindViews opentelemetry::sdk::trace::AlwaysOnSamplerFactory  
 (C++ function), 115 (C++ class), 120

opentelemetry::sdk::metrics::WithTraceSampleFilter opentelemetry::sdk::trace::AlwaysOnSamplerFactory::Create  
 (C++ class), 115 (C++ function), 120

opentelemetry::sdk::metrics::WithTraceSampleFilter::ShouldSampleMeasurement opentelemetry::sdk::trace::BatchSpanProcessor  
 (C++ function), 115 (C++ class), 120

opentelemetry::sdk::metrics::WithTraceSampleFilter::WithTraceSampleFilter opentelemetry::sdk::trace::BatchSpanProcessor::~BatchSpanProcessor  
 (C++ function), 115 (C++ function), 121

opentelemetry::sdk::resource::OTELResourceDetector opentelemetry::sdk::trace::BatchSpanProcessor::BatchSpanProcessor  
 (C++ class), 116 (C++ function), 121

opentelemetry::sdk::resource::OTELResourceDetector::Detect opentelemetry::sdk::trace::BatchSpanProcessor::buffer\_  
 (C++ function), 116 (C++ member), 122

opentelemetry::sdk::resource::Resource (C++ opentelemetry::sdk::trace::BatchSpanProcessor::DoBackgroundWork  
 class), 116 (C++ function), 122

opentelemetry::sdk::resource::Resource::Create opentelemetry::sdk::trace::BatchSpanProcessor::DrainQueue  
 (C++ function), 117 (C++ function), 122

opentelemetry::sdk::resource::Resource::GetAttributes opentelemetry::sdk::trace::BatchSpanProcessor::Export  
 (C++ function), 116 (C++ function), 122

opentelemetry::sdk::resource::Resource::GetDefault opentelemetry::sdk::trace::BatchSpanProcessor::exporter\_  
 (C++ function), 117 (C++ member), 122

opentelemetry::sdk::resource::Resource::GetEmpty opentelemetry::sdk::trace::BatchSpanProcessor::ForceFlush  
 (C++ function), 117 (C++ function), 121

opentelemetry::sdk::resource::Resource::GetSchemaName opentelemetry::sdk::trace::BatchSpanProcessor::GetWaitAdjustment  
 (C++ function), 116 (C++ function), 122

opentelemetry::sdk::resource::Resource::Merge opentelemetry::sdk::trace::BatchSpanProcessor::MakeRecordable  
 (C++ function), 116 (C++ function), 121

opentelemetry::sdk::resource::Resource::Resource opentelemetry::sdk::trace::BatchSpanProcessor::max\_export\_  
 (C++ function), 116, 117 (C++ member), 122

opentelemetry::sdk::resource::ResourceAttributes opentelemetry::sdk::trace::BatchSpanProcessor::max\_queue\_size  
 (C++ type), 185 (C++ member), 122

opentelemetry::sdk::resource::ResourceDetector opentelemetry::sdk::trace::BatchSpanProcessor::NotifyComplete  
 (C++ class), 117 (C++ function), 122

opentelemetry::sdk::resource::ResourceDetector::Detect opentelemetry::sdk::trace::BatchSpanProcessor::OnEnd  
 (C++ function), 118 (C++ function), 121

opentelemetry::sdk::resource::ResourceDetector::Detect opentelemetry::sdk::trace::BatchSpanProcessor::OnStart  
 (C++ function), 118 (C++ function), 121

opentelemetry::sdk::resource::ResourceDetector::ResourceDetector opentelemetry::sdk::trace::BatchSpanProcessor::schedule\_delay  
 (C++ function), 118 (C++ member), 122

opentelemetry::sdk::trace::AlwaysOffSampler opentelemetry::sdk::trace::BatchSpanProcessor::Shutdown  
 (C++ class), 118 (C++ function), 121

opentelemetry::sdk::trace::AlwaysOffSampler::GetDescription opentelemetry::sdk::trace::BatchSpanProcessor::synchronize  
 (C++ function), 118 (C++ member), 122

opentelemetry::sdk::trace::AlwaysOffSampler::ShouldSample opentelemetry::sdk::trace::BatchSpanProcessor::Synchronize  
 (C++ function), 118 (C++ struct), 28, 122

opentelemetry::sdk::trace::AlwaysOffSamplerFactory opentelemetry::sdk::trace::BatchSpanProcessor::Synchronize  
 (C++ class), 119 (C++ member), 28, 123

opentelemetry::sdk::trace::AlwaysOffSamplerFactory::Create opentelemetry::sdk::trace::BatchSpanProcessor::Synchronize  
 (C++ function), 119 (C++ member), 28, 123

opentelemetry::sdk::trace::AlwaysOnSampler opentelemetry::sdk::trace::BatchSpanProcessor::Synchronize  
 (C++ class), 119 (C++ member), 28, 123

opentelemetry::sdk::trace::BatchSpanProcessor::SymbolizeDataForMultiRecordable::SetAttribute  
 (C++ member), 28, 123

opentelemetry::sdk::trace::BatchSpanProcessor::SymbolizeDataForMultiRecordable::SetDuration  
 (C++ member), 28, 123

opentelemetry::sdk::trace::BatchSpanProcessor::SymbolizeDataForMultiRecordable::SetIdentity  
 (C++ member), 28, 123

opentelemetry::sdk::trace::BatchSpanProcessor::SymbolizeDataForMultiRecordable::SetTimestamp  
 (C++ member), 28, 123

opentelemetry::sdk::trace::BatchSpanProcessor::SymbolizeDataForMultiRecordable::SetName  
 (C++ member), 28, 123

opentelemetry::sdk::trace::BatchSpanProcessor::SymbolizeDataForMultiRecordable::SetResource  
 (C++ member), 28, 123

opentelemetry::sdk::trace::BatchSpanProcessor::SymbolizeDataForMultiRecordable::SetSpanKind  
 (C++ member), 122

opentelemetry::sdk::trace::BatchSpanProcessor::SymbolizeDataForMultiRecordable::SetStartTime  
 (C++ member), 123

opentelemetry::sdk::trace::BatchSpanProcessor::SymbolizeDataForMultiRecordable::SetStatus  
 (C++ member), 123

opentelemetry::sdk::trace::BatchSpanProcessor::SymbolizeDataForMultiSpanProcessor  
 (C++ member), 29

opentelemetry::sdk::trace::BatchSpanProcessor::SymbolizeDataForMultiSpanProcessor::~MultiSpanProcessor  
 (C++ member), 29

opentelemetry::sdk::trace::BatchSpanProcessor::SymbolizeDataForMultiSpanProcessor::AddProcessor  
 (C++ member), 29

opentelemetry::sdk::trace::BatchSpanProcessor::SymbolizeDataForMultiSpanProcessor::ForceFlush  
 (C++ member), 29

opentelemetry::sdk::trace::Decision (C++ enum), 169

opentelemetry::sdk::trace::Decision::DROP (C++ enumerator), 169

opentelemetry::sdk::trace::Decision::RECORD\_AND\_SAMPLE (C++ enumerator), 169

opentelemetry::sdk::trace::Decision::RECORD\_ONLY (C++ enumerator), 169

opentelemetry::sdk::trace::IdGenerator (C++ class), 124

opentelemetry::sdk::trace::IdGenerator::~~IdGenerator (C++ function), 124

opentelemetry::sdk::trace::IdGenerator::GenerateSpanId (C++ function), 124

opentelemetry::sdk::trace::IdGenerator::GenerateTraceId (C++ function), 124

opentelemetry::sdk::trace::MultiRecordable (C++ class), 125

opentelemetry::sdk::trace::MultiRecordable::AddEvent (C++ function), 125

opentelemetry::sdk::trace::MultiRecordable::AddLink (C++ function), 125

opentelemetry::sdk::trace::MultiRecordable::AddParent (C++ function), 125

opentelemetry::sdk::trace::MultiRecordable::GetParent (C++ function), 125

opentelemetry::sdk::trace::MultiRecordable::RepeatRecordable (C++ function), 125

opentelemetry::sdk::trace::MultiSpanProcessor (C++ class), 126

opentelemetry::sdk::trace::MultiSpanProcessor::~MultiSpanProcessor (C++ function), 127

opentelemetry::sdk::trace::MultiSpanProcessor::AddProcessor (C++ function), 126

opentelemetry::sdk::trace::MultiSpanProcessor::ForceFlush (C++ function), 126

opentelemetry::sdk::trace::MultiSpanProcessor::MakeRecordable (C++ function), 126

opentelemetry::sdk::trace::MultiSpanProcessor::MultiSpanProcessor (C++ function), 126

opentelemetry::sdk::trace::MultiSpanProcessor::OnEnd (C++ function), 126

opentelemetry::sdk::trace::MultiSpanProcessor::OnStart (C++ function), 126

opentelemetry::sdk::trace::MultiSpanProcessor::ProcessorNoop (C++ struct), 29

opentelemetry::sdk::trace::MultiSpanProcessor::ProcessorNoop (C++ member), 30

opentelemetry::sdk::trace::MultiSpanProcessor::ProcessorNoop (C++ member), 30

opentelemetry::sdk::trace::MultiSpanProcessor::ProcessorNoop (C++ function), 30

opentelemetry::sdk::trace::MultiSpanProcessor::ProcessorNoop (C++ member), 30

opentelemetry::sdk::trace::MultiSpanProcessor::Shutdown (C++ function), 127

opentelemetry::sdk::trace::MultiSpanProcessorOptions (C++ struct), 30

opentelemetry::sdk::trace::ParentBasedSampler (C++ class), 127

opentelemetry::sdk::trace::ParentBasedSampler::GetDescriptor (C++ function), 127

opentelemetry::sdk::trace::ParentBasedSampler::ParentBasedSampler (C++ function), 127

opentelemetry::sdk::trace::ParentBasedSampler::ShouldSample (C++ function), 127

opentelemetry::sdk::trace::ParentBasedSamplerFactory::Create (C++ function), 128

opentelemetry::sdk::trace::ParentBasedSamplerFactory::Create (C++ function), 128

opentelemetry::sdk::trace::RandomIdGenerator (C++ class), 128

opentelemetry::sdk::trace::RandomIdGenerator::GenerateSpanId (C++ function), 128

opentelemetry::sdk::trace::RandomIdGenerator::GenerateTraceId (C++ function), 128

opentelemetry::sdk::trace::RandomIdGeneratorFactory::Create (C++ function), 129

opentelemetry::sdk::trace::RandomIdGeneratorFactory::Create (C++ function), 129

opentelemetry::sdk::trace::Recordable (C++ class), 129

opentelemetry::sdk::trace::Recordable::~~Recordable (C++ function), 129

opentelemetry::sdk::trace::Recordable::AddEvent (C++ function), 130

opentelemetry::sdk::trace::Recordable::AddLink (C++ function), 130

opentelemetry::sdk::trace::Recordable::operator SpanData\* (C++ function), 131

opentelemetry::sdk::trace::Recordable::SetAttribute (C++ function), 129

opentelemetry::sdk::trace::Recordable::SetDuration (C++ function), 131

opentelemetry::sdk::trace::Recordable::SetIdentity (C++ function), 129

opentelemetry::sdk::trace::Recordable::SetInstrumentationLibrary (C++ function), 131

opentelemetry::sdk::trace::Recordable::SetInstrumentationScope (C++ function), 131

opentelemetry::sdk::trace::Recordable::SetName (C++ function), 131

opentelemetry::sdk::trace::Recordable::SetResource (C++ function), 131

opentelemetry::sdk::trace::Recordable::SetSpanKind (C++ function), 131

opentelemetry::sdk::trace::Recordable::SetStartTime (C++ function), 131

opentelemetry::sdk::trace::Recordable::SetStatus (C++ function), 130

opentelemetry::sdk::trace::Sampler (C++ class), 132

opentelemetry::sdk::trace::Sampler::~~Sampler (C++ function), 132

opentelemetry::sdk::trace::Sampler::GetDescription (C++ function), 132

opentelemetry::sdk::trace::Sampler::ShouldSample (C++ function), 132

opentelemetry::sdk::trace::SamplingResult (C++ struct), 30

opentelemetry::sdk::trace::SamplingResult::attributes (C++ member), 31

opentelemetry::sdk::trace::SamplingResult::decision (C++ member), 31

opentelemetry::sdk::trace::SamplingResult::IsRecording (C++ function), 30

opentelemetry::sdk::trace::SamplingResult::IsSampled (C++ function), 30

opentelemetry::sdk::trace::SamplingResult::trace\_state (C++ member), 31

opentelemetry::sdk::trace::SimpleSpanProcessor (C++ class), 133

opentelemetry::sdk::trace::SimpleSpanProcessor::~SimpleSpanProcessor (C++ function), 134

opentelemetry::sdk::trace::SimpleSpanProcessor::ForceFlush (C++ function), 134

opentelemetry::sdk::trace::SimpleSpanProcessor::MakeRecording (C++ function), 133

opentelemetry::sdk::trace::SimpleSpanProcessor::OnEnd (C++ function), 133

opentelemetry::sdk::trace::SimpleSpanProcessor::OnStart (C++ function), 133

opentelemetry::sdk::trace::SimpleSpanProcessor::Shutdown (C++ function), 134

opentelemetry::sdk::trace::SimpleSpanProcessor::SimpleSpanProcessor (C++ function), 133

opentelemetry::sdk::trace::SimpleSpanProcessorFactory (C++ class), 134

opentelemetry::sdk::trace::SimpleSpanProcessorFactory::Create (C++ function), 134

opentelemetry::sdk::trace::SpanData (C++ class), 135

opentelemetry::sdk::trace::SpanData::AddEvent (C++ function), 136

opentelemetry::sdk::trace::SpanData::AddLink (C++ function), 137

opentelemetry::sdk::trace::SpanData::GetAttributes (C++ function), 136

opentelemetry::sdk::trace::SpanData::GetDescription (C++ function), 135

opentelemetry::sdk::trace::SpanData::GetDuration (C++ function), 136

opentelemetry::sdk::trace::SpanData::GetEvents (C++ function), 136

opentelemetry::sdk::trace::SpanData::GetInstrumentationLibrary (C++ function), 136

opentelemetry::sdk::trace::SpanData::GetInstrumentationScope (C++ function), 136

opentelemetry::sdk::trace::SpanData::GetLinks (C++ function), 136

opentelemetry::sdk::trace::SpanData::GetName (C++ function), 135



`opentelemetry::sdk::trace::SpanData::GetParentSpanId` `opentelemetry::sdk::trace::SpanExporter`  
 (C++ function), 135 (C++ class), 139  
`opentelemetry::sdk::trace::SpanData::GetResource` `opentelemetry::sdk::trace::SpanExporter::~~SpanExporter`  
 (C++ function), 135 (C++ function), 139  
`opentelemetry::sdk::trace::SpanData::GetSpanContext` `opentelemetry::sdk::trace::SpanExporter::Export`  
 (C++ function), 135 (C++ function), 139  
`opentelemetry::sdk::trace::SpanData::GetSpanId` `opentelemetry::sdk::trace::SpanExporter::MakeRecordable`  
 (C++ function), 135 (C++ function), 139  
`opentelemetry::sdk::trace::SpanData::GetSpanKind` `opentelemetry::sdk::trace::SpanExporter::Shutdown`  
 (C++ function), 135 (C++ function), 140  
`opentelemetry::sdk::trace::SpanData::GetStartTime` `opentelemetry::sdk::trace::SpanProcessor`  
 (C++ function), 136 (C++ class), 140  
`opentelemetry::sdk::trace::SpanData::GetStatus` `opentelemetry::sdk::trace::SpanProcessor::~~SpanProcessor`  
 (C++ function), 135 (C++ function), 140  
`opentelemetry::sdk::trace::SpanData::GetTraceId` `opentelemetry::sdk::trace::SpanProcessor::ForceFlush`  
 (C++ function), 135 (C++ function), 141  
`opentelemetry::sdk::trace::SpanData::SetAttribute` `opentelemetry::sdk::trace::SpanProcessor::MakeRecordable`  
 (C++ function), 136 (C++ function), 140  
`opentelemetry::sdk::trace::SpanData::SetDuration` `opentelemetry::sdk::trace::SpanProcessor::OnEnd`  
 (C++ function), 137 (C++ function), 141  
`opentelemetry::sdk::trace::SpanData::SetIdentity` `opentelemetry::sdk::trace::SpanProcessor::OnStart`  
 (C++ function), 136 (C++ function), 140  
`opentelemetry::sdk::trace::SpanData::SetInstrumentationScope` `opentelemetry::sdk::trace::SpanProcessor::Shutdown`  
 (C++ function), 137 (C++ function), 141  
`opentelemetry::sdk::trace::SpanData::SetName` `opentelemetry::sdk::trace::TraceIdRatioBasedSampler`  
 (C++ function), 137 (C++ class), 141  
`opentelemetry::sdk::trace::SpanData::SetResource` `opentelemetry::sdk::trace::TraceIdRatioBasedSampler::GetDe`  
 (C++ function), 137 (C++ function), 142  
`opentelemetry::sdk::trace::SpanData::SetSpanKind` `opentelemetry::sdk::trace::TraceIdRatioBasedSampler::Shoul`  
 (C++ function), 137 (C++ function), 141  
`opentelemetry::sdk::trace::SpanData::SetStartTime` `opentelemetry::sdk::trace::TraceIdRatioBasedSampler::Trace`  
 (C++ function), 137 (C++ function), 141  
`opentelemetry::sdk::trace::SpanData::SetStatus` `opentelemetry::sdk::trace::TraceIdRatioBasedSamplerFactory`  
 (C++ function), 137 (C++ class), 142  
`opentelemetry::sdk::trace::SpanData::SpanData` `opentelemetry::sdk::trace::TraceIdRatioBasedSamplerFactory`  
 (C++ function), 135 (C++ function), 142  
`opentelemetry::sdk::trace::SpanDataEvent` `opentelemetry::sdk::trace::Tracer (C++ class),`  
 (C++ class), 138 142  
`opentelemetry::sdk::trace::SpanDataEvent::GetAttributes` `opentelemetry::sdk::trace::Tracer::CloseWithMicroseconds`  
 (C++ function), 138 (C++ function), 143  
`opentelemetry::sdk::trace::SpanDataEvent::GetName` `opentelemetry::sdk::trace::Tracer::ForceFlushWithMicroseco`  
 (C++ function), 138 (C++ function), 143  
`opentelemetry::sdk::trace::SpanDataEvent::GetTimestamp` `opentelemetry::sdk::trace::Tracer::GetIdGenerator`  
 (C++ function), 138 (C++ function), 143  
`opentelemetry::sdk::trace::SpanDataEvent::SpanDataEvent` `opentelemetry::sdk::trace::Tracer::GetInstrumentationLibra`  
 (C++ function), 138 (C++ function), 143  
`opentelemetry::sdk::trace::SpanDataLink` `opentelemetry::sdk::trace::Tracer::GetInstrumentationScope`  
 (C++ class), 139 (C++ function), 143  
`opentelemetry::sdk::trace::SpanDataLink::GetAttributes` `opentelemetry::sdk::trace::Tracer::GetProcessor`  
 (C++ function), 139 (C++ function), 143  
`opentelemetry::sdk::trace::SpanDataLink::GetSpanContext` `opentelemetry::sdk::trace::Tracer::GetResource`  
 (C++ function), 139 (C++ function), 143  
`opentelemetry::sdk::trace::SpanDataLink::SpanDataLink` `opentelemetry::sdk::trace::Tracer::GetSampler`  
 (C++ function), 139 (C++ function), 143

`opentelemetry::sdk::trace::Tracer::StartSpan` (*C++ function*), 143  
`opentelemetry::sdk::trace::Tracer::Tracer` (*C++ function*), 143  
`opentelemetry::sdk::trace::TracerContext` (*C++ class*), 143  
`opentelemetry::sdk::trace::TracerContext::AddProcessor` (*C++ function*), 144  
`opentelemetry::sdk::trace::TracerContext::ForceFlush` (*C++ function*), 144  
`opentelemetry::sdk::trace::TracerContext::GetIdGenerator` (*C++ function*), 144  
`opentelemetry::sdk::trace::TracerContext::GetProcessor` (*C++ function*), 144  
`opentelemetry::sdk::trace::TracerContext::GetReporter` (*C++ function*), 144  
`opentelemetry::sdk::trace::TracerContext::GetSampler` (*C++ function*), 144  
`opentelemetry::sdk::trace::TracerContext::Shutdown` (*C++ function*), 144  
`opentelemetry::sdk::trace::TracerContext::TraceContext` (*C++ function*), 144  
`opentelemetry::sdk::trace::TracerContextFactory` (*C++ class*), 145  
`opentelemetry::sdk::trace::TracerContextFactory::Create` (*C++ function*), 145  
`opentelemetry::sdk::trace::TracerProvider` (*C++ class*), 145  
`opentelemetry::sdk::trace::TracerProvider::~~TracerProvider` (*C++ function*), 146  
`opentelemetry::sdk::trace::TracerProvider::AddProcessor` (*C++ function*), 146  
`opentelemetry::sdk::trace::TracerProvider::ForceFlush` (*C++ function*), 147  
`opentelemetry::sdk::trace::TracerProvider::GetResource` (*C++ function*), 146  
`opentelemetry::sdk::trace::TracerProvider::GetTracer` (*C++ function*), 146  
`opentelemetry::sdk::trace::TracerProvider::Shutdown` (*C++ function*), 146  
`opentelemetry::sdk::trace::TracerProvider::TraceProvider` (*C++ function*), 146  
`opentelemetry::sdk::trace::TracerProviderFactory` (*C++ class*), 147  
`opentelemetry::sdk::trace::TracerProviderFactory::Create` (*C++ function*), 147, 148  
`opentelemetry::trace::CanonicalCode` (*C++ enum*), 169  
`opentelemetry::trace::CanonicalCode::ABORTED` (*C++ enumerator*), 170  
`opentelemetry::trace::CanonicalCode::ALREADY_EXISTS` (*C++ enumerator*), 169  
`opentelemetry::trace::CanonicalCode::CANCELLED` (*C++ enumerator*), 169  
`opentelemetry::trace::CanonicalCode::DATA_LOSS` (*C++ enumerator*), 171  
`opentelemetry::trace::CanonicalCode::DEADLINE_EXCEEDED` (*C++ enumerator*), 169  
`opentelemetry::trace::CanonicalCode::FAILED_PRECONDITION` (*C++ enumerator*), 170  
`opentelemetry::trace::CanonicalCode::INTERNAL` (*C++ enumerator*), 170  
`opentelemetry::trace::CanonicalCode::INVALID_ARGUMENT` (*C++ enumerator*), 169  
`opentelemetry::trace::CanonicalCode::NOT_FOUND` (*C++ enumerator*), 169  
`opentelemetry::trace::CanonicalCode::OK` (*C++ enumerator*), 169  
`opentelemetry::trace::CanonicalCode::OUT_OF_RANGE` (*C++ enumerator*), 170  
`opentelemetry::trace::CanonicalCode::PERMISSION_DENIED` (*C++ enumerator*), 170  
`opentelemetry::trace::CanonicalCode::RESOURCE_EXHAUSTED` (*C++ enumerator*), 170  
`opentelemetry::trace::CanonicalCode::UNAUTHENTICATED` (*C++ enumerator*), 171  
`opentelemetry::trace::CanonicalCode::UNAVAILABLE` (*C++ enumerator*), 170  
`opentelemetry::trace::CanonicalCode::UNIMPLEMENTED` (*C++ enumerator*), 170  
`opentelemetry::trace::CanonicalCode::UNKNOWN` (*C++ enumerator*), 169  
`opentelemetry::trace::DefaultSpan` (*C++ class*), 148  
`opentelemetry::trace::DefaultSpan::AddEvent` (*C++ function*), 148  
`opentelemetry::trace::DefaultSpan::DefaultSpan` (*C++ function*), 149  
`opentelemetry::trace::DefaultSpan::End` (*C++ function*), 148  
`opentelemetry::trace::DefaultSpan::GetContext` (*C++ function*), 148  
`opentelemetry::trace::DefaultSpan::GetInvalid` (*C++ function*), 149  
`opentelemetry::trace::DefaultSpan::IsRecording` (*C++ function*), 148  
`opentelemetry::trace::DefaultSpan::SetAttribute` (*C++ function*), 148  
`opentelemetry::trace::DefaultSpan::SetStatus` (*C++ function*), 148  
`opentelemetry::trace::DefaultSpan::ToString` (*C++ function*), 149  
`opentelemetry::trace::DefaultSpan::UpdateName` (*C++ function*), 148  
`opentelemetry::trace::EndSpanOptions` (*C++ struct*), 31  
`opentelemetry::trace::EndSpanOptions::end_steady_time` (*C++ member*), 31



*(C++ member)*, 178  
 opentelemetry::trace::propagation::kTraceIdHexStrLength *(C++ member)*, 179  
 opentelemetry::trace::propagation::kTraceIdSize *(C++ member)*, 179  
 opentelemetry::trace::propagation::kTraceParent *(C++ member)*, 179  
 opentelemetry::trace::propagation::kTraceParentSize *(C++ member)*, 179  
 opentelemetry::trace::propagation::kTraceState *(C++ member)*, 180  
 opentelemetry::trace::propagation::kVersionSize *(C++ member)*, 180  
 opentelemetry::trace::Provider *(C++ class)*, 155  
 opentelemetry::trace::Provider::GetTracerProvider *(C++ function)*, 156  
 opentelemetry::trace::Provider::SetTracerProvider *(C++ function)*, 156  
 opentelemetry::trace::Scope *(C++ class)*, 156  
 opentelemetry::trace::Scope::Scope *(C++ function)*, 156  
 opentelemetry::trace::SetSpan *(C++ function)*, 175  
 opentelemetry::trace::Span *(C++ class)*, 157  
 opentelemetry::trace::Span::~~Span *(C++ function)*, 157  
 opentelemetry::trace::Span::AddEvent *(C++ function)*, 157  
 opentelemetry::trace::Span::End *(C++ function)*, 157  
 opentelemetry::trace::Span::GetContext *(C++ function)*, 157  
 opentelemetry::trace::Span::IsRecording *(C++ function)*, 158  
 opentelemetry::trace::Span::operator= *(C++ function)*, 157  
 opentelemetry::trace::Span::SetAttribute *(C++ function)*, 157  
 opentelemetry::trace::Span::SetStatus *(C++ function)*, 157  
 opentelemetry::trace::Span::Span *(C++ function)*, 157  
 opentelemetry::trace::Span::UpdateName *(C++ function)*, 157  
 opentelemetry::trace::SpanContext *(C++ class)*, 158  
 opentelemetry::trace::SpanContext::GetInvalid *(C++ function)*, 158  
 opentelemetry::trace::SpanContext::IsRemote *(C++ function)*, 158  
 opentelemetry::trace::SpanContext::IsSampled *(C++ function)*, 158  
 opentelemetry::trace::SpanContext::IsValid *(C++ function)*, 158  
 opentelemetry::trace::SpanContext::operator= *(C++ function)*, 158  
 opentelemetry::trace::SpanContext::operator== *(C++ function)*, 158  
 opentelemetry::trace::SpanContext::span\_id  
 opentelemetry::trace::SpanContext::SpanContext  
 opentelemetry::trace::SpanContext::trace\_flags  
 opentelemetry::trace::SpanContext::trace\_id  
 opentelemetry::trace::SpanContext::trace\_state *(C++ function)*, 158  
 opentelemetry::trace::SpanContext::trace\_state *(C++ function)*, 158  
 opentelemetry::trace::SpanContextKeyValueIterable  
 opentelemetry::trace::SpanContextKeyValueIterable::~SpanContextKeyValueIterable *(C++ function)*, 159  
 opentelemetry::trace::SpanContextKeyValueIterable::ForEach *(C++ function)*, 159  
 opentelemetry::trace::SpanContextKeyValueIterable::size *(C++ function)*, 159  
 opentelemetry::trace::SpanId *(C++ class)*, 159  
 opentelemetry::trace::SpanId::CopyBytesTo *(C++ function)*, 160  
 opentelemetry::trace::SpanId::Id *(C++ function)*, 160  
 opentelemetry::trace::SpanId::IsValid *(C++ function)*, 160  
 opentelemetry::trace::SpanId::kSize *(C++ member)*, 160  
 opentelemetry::trace::SpanId::operator!= *(C++ function)*, 160  
 opentelemetry::trace::SpanId::operator== *(C++ function)*, 160  
 opentelemetry::trace::SpanId::SpanId *(C++ function)*, 160  
 opentelemetry::trace::SpanId::ToLowerBase16 *(C++ function)*, 160  
 opentelemetry::trace::SpanKind *(C++ enum)*, 171  
 opentelemetry::trace::SpanKind::kClient *(C++ enumerator)*, 171  
 opentelemetry::trace::SpanKind::kConsumer *(C++ enumerator)*, 171  
 opentelemetry::trace::SpanKind::kInternal *(C++ enumerator)*, 171  
 opentelemetry::trace::SpanKind::kProducer *(C++ enumerator)*, 171  
 opentelemetry::trace::SpanKind::kServer *(C++ enumerator)*, 171  
 opentelemetry::trace::StartSpanOptions *(C++ struct)*, 31  
 opentelemetry::trace::StartSpanOptions::kind *(C++ member)*, 31

`opentelemetry::trace::StartSpanOptions::parent` (`opentelemetry::trace::Tracer::CloseWithMicroseconds` (C++ member), 31) (`opentelemetry::trace::Tracer::ForceFlush` (C++ function), 163)  
`opentelemetry::trace::StartSpanOptions::start_time` (`opentelemetry::trace::Tracer::ForceFlushWithMicroseconds` (C++ member), 31) (`opentelemetry::trace::Tracer::GetCurrentSpan` (C++ function), 163)  
`opentelemetry::trace::StartSpanOptions::start_system_time` (`opentelemetry::trace::Tracer::StartSpan` (C++ member), 31) (`opentelemetry::trace::Tracer::WithActiveSpan` (C++ function), 163)  
`opentelemetry::trace::StatusCode` (C++ enum), 172 (`opentelemetry::trace::TracerProvider` (C++ class), 164)  
`opentelemetry::trace::StatusCode::kError` (C++ enumerator), 172 (`opentelemetry::trace::TracerProvider::~~TracerProvider` (C++ function), 164)  
`opentelemetry::trace::StatusCode::kOk` (C++ enumerator), 172 (`opentelemetry::trace::TracerProvider::GetTracer` (C++ function), 164)  
`opentelemetry::trace::StatusCode::kUnset` (C++ enumerator), 172 (`opentelemetry::trace::TraceState` (C++ class), 164)  
`opentelemetry::trace::TraceFlags` (C++ class), 160 (`opentelemetry::trace::TraceState::Delete` (C++ function), 165)  
`opentelemetry::trace::TraceFlags::CopyBytesTo` (C++ function), 160 (`opentelemetry::trace::TraceState::Empty` (C++ function), 165)  
`opentelemetry::trace::TraceFlags::flags` (C++ function), 160 (`opentelemetry::trace::TraceState::FromHeader` (C++ function), 165)  
`opentelemetry::trace::TraceFlags::IsSampled` (C++ function), 160 (`opentelemetry::trace::TraceState::Get` (C++ function), 165)  
`opentelemetry::trace::TraceFlags::kIsSampled` (C++ member), 161 (`opentelemetry::trace::TraceState::GetAllEntries` (C++ function), 165)  
`opentelemetry::trace::TraceFlags::operator!=` (C++ function), 160 (`opentelemetry::trace::TraceState::IsValidKey` (C++ function), 165)  
`opentelemetry::trace::TraceFlags::operator==` (C++ function), 160 (`opentelemetry::trace::TraceState::IsValidValue` (C++ function), 165)  
`opentelemetry::trace::TraceFlags::ToLowerBase16` (C++ function), 160 (`opentelemetry::trace::TraceState::kKeyMaxSize` (C++ member), 166)  
`opentelemetry::trace::TraceFlags::TraceFlags` (C++ function), 160 (`opentelemetry::trace::TraceState::kKeyValueSeparator` (C++ member), 166)  
`opentelemetry::trace::TraceId` (C++ class), 161 (`opentelemetry::trace::TraceState::kMaxKeyValuePairs` (C++ member), 166)  
`opentelemetry::trace::TraceId::CopyBytesTo` (C++ function), 161 (`opentelemetry::trace::TraceState::kMembersSeparator` (C++ member), 166)  
`opentelemetry::trace::TraceId::Id` (C++ function), 161 (`opentelemetry::trace::TraceState::kValueMaxSize` (C++ member), 166)  
`opentelemetry::trace::TraceId::IsValid` (C++ function), 161 (`opentelemetry::trace::TraceState::Set` (C++ function), 165)  
`opentelemetry::trace::TraceId::kSize` (C++ member), 161 (`opentelemetry::trace::TraceState::ToHeader` (C++ function), 165)  
`opentelemetry::trace::TraceId::operator!=` (C++ function), 161 (`OPENTELEMETRY_API_SINGLETON` (C macro), 180)  
`opentelemetry::trace::TraceId::operator==` (C++ function), 161 (`OPENTELEMETRY_DEPRECATED` (C macro), 180)  
`opentelemetry::trace::TraceId::ToLowerBase16` (C++ function), 161 (`OPENTELEMETRY_DEPRECATED_MESSAGE` (C macro), 181)  
`opentelemetry::trace::TraceId::TraceId` (C++ function), 161 (`OPENTELEMETRY_HAVE_WORKING_REGEX` (C macro), 181)  
`opentelemetry::trace::Tracer` (C++ class), 162 (`OPENTELEMETRY_LIKELY_IF` (C macro), 181)  
`opentelemetry::trace::Tracer::~~Tracer` (C++ function), 162 (`OPENTELEMETRY_MAYBE_UNUSED` (C macro), 181)  
`opentelemetry::trace::Tracer::Close` (C++ function), 163