
Table of Contents

Introduction	1.1
Kafka Streams — Stream Processing Library on Apache Kafka	1.2
Why Kafka Streams	1.3
Multi-Instance Kafka Streams Applications	1.4
Exactly-Once Support (EOS)	1.5
KafkaStreams, StreamThreads, StreamTasks and StandbyTasks	1.6

Demos

Creating Topology with State Store with Logging Enabled	2.1
---	-----

Stateful Stream Processing

Stateful Stream Processing	3.1
StateStore	3.2
KeyValueStore	3.2.1
SessionStore	3.2.2
WindowStore	3.2.3
Stores	3.3
Materialized	3.4
StoreSupplier	3.5
KeyValueBytesStoreSupplier	3.5.1
SessionBytesStoreSupplier	3.5.2
WindowBytesStoreSupplier	3.5.3
StoreBuilder	3.6
TimestampedKeyValueStore	3.7
ReadOnlyKeyValueStore	3.8

Stream Processing with Kafka Streams

KafkaStreams	4.1
Topology	4.2
TopologyDescription	4.3
Node	4.3.1
Processor	4.3.2
Sink	4.3.3
Source	4.3.4
StreamsConfig	4.4
Configuration Properties	4.5
Logging	4.6
KafkaClientSupplier	4.7

High-Level Streams DSL

High-Level Streams DSL	5.1
StreamsBuilder	5.2
KStream	5.3
KTable	5.4
GlobalKTable	5.5
KGroupedStream	5.6
SessionWindowedKStream	5.6.1
TimeWindowedKStream	5.6.2
KGroupedTable	5.7
Consumed	5.8
Produced	5.9
Grouped	5.10
Joined	5.11
Printed	5.12
KeyValueMapper	5.13
ValueJoiner	5.14
ValueTransformer	5.15
ValueTransformerSupplier	5.15.1
ValueTransformerWithKey	5.16
Transformer	5.17

TransformerSupplier	5.18
Windows	5.19
JoinWindows	5.19.1
TimeWindows	5.19.2
UnlimitedWindows	5.19.3
Window	5.20
WindowedSerdes	5.21
Windowed	5.21.1
TimeWindowedSerializer	5.21.2
Scala API for Kafka Streams	5.22
ImplicitConversions	5.22.1
Serdes	5.22.2
Consumed	5.22.3
Produced	5.22.4
Grouped	5.22.5
Materialized	5.22.6

Low-Level Processor API

Low-Level Processor API	6.1
Processor Contract — Stream Processing Node	6.2
AbstractProcessor — Base for Stream Processors	6.2.1
ProcessorContext	6.3
TaskId	6.3.1
To	6.3.2
Punctuator Contract — Scheduled Periodic Actions	6.4
Cancellable	6.5
ProcessorSupplier Contract	6.6
StreamPartitioner	6.7
TopicNameExtractor — Dynamic Routing of Output Records	6.8
RecordContext — Record Metadata	6.8.1
TimestampExtractor Contract	6.9
WallclockTimestampExtractor	6.9.1
ExtractRecordMetadataTimestamp	6.9.2

FailOnInvalidTimestamp	6.9.3
PartitionGrouper Contract	6.10
DefaultPartitionGrouper	6.10.1
StateRestoreCallback	6.11
BatchingStateRestoreCallback	6.11.1
StateRestoreListener	6.12
AbstractNotifyingBatchingRestoreCallback	6.12.1
AbstractNotifyingRestoreCallback	6.12.2

Monitoring Kafka Streams Applications

StateListener — KafkaStreams State Listener	7.1
CacheFlushListener	7.2
StreamsMetrics	7.3
StreamsMetricsImpl	7.3.1
StreamsMetricsThreadImpl	7.3.2
TaskMetrics	7.4
StoreChangeLogger	7.5
DelegatingStateRestoreListener	7.6

Testing

TopologyTestDriver	8.1
------------------------------------	-----

Internals of Kafka Streams

Low-Level Stream Processing Graph

InternalTopologyBuilder	10.1
AbstractNode	10.2
Processor	10.2.1
Sink	10.2.2
Source	10.2.3

NodeFactory	10.3
ProcessorNodeFactory	10.3.1
SinkNodeFactory	10.3.2
SourceNodeFactory	10.3.3
InternalTopologyBuilder.TopologyDescription	10.4
GlobalStore	10.5
StateStoreFactory	10.6

High-Level Stream Processing Graph

InternalStreamsBuilder	11.1
StreamsGraphNode	11.2
BaseJoinProcessorNode	11.2.1
BaseRepartitionNode	11.2.2
ProcessorParameters	11.3
GlobalStoreNode	11.4
GroupedTableOperationRepartitionNode	11.5
GroupedTableOperationRepartitionNodeBuilder	11.5.1
KTableKTableJoinNode	11.6
OptimizableRepartitionNode	11.7
ProcessorGraphNode	11.8
StatefulProcessorNode	11.9
StateStoreNode	11.10
StreamSinkNode	11.11
StreamSourceNode	11.12
StreamStreamJoinNode	11.13
StreamTableJoinNode	11.14
TableProcessorNode	11.15
TableSourceNode	11.16

Processors and ProcessorSuppliers

KStreamAggProcessorSupplier	12.1
KStreamBranch	12.2

KStreamFilterProcessor	12.3
KStreamFilter	12.3.1
KStreamJoinWindowProcessor	12.4
KStreamPeek	12.5
KStreamPassThrough	12.6
KStreamSessionWindowAggregateProcessor	12.7
KStreamSessionWindowAggregate	12.7.1
KStreamTransformProcessor	12.8
KStreamTransform	12.8.1
KStreamTransformValuesProcessor	12.9
KStreamTransformValues	12.9.1
KStreamWindowAggregateProcessor	12.10
KStreamWindowAggregate	12.10.1
KTableSourceProcessor	12.11
KTableSource	12.11.1
KTableSuppressProcessor	12.12
KTableValueGetter	12.13
KTableValueGetterSupplier	12.14
KTableMaterializedValueGetterSupplier	12.14.1
KTableSourceValueGetterSupplier	12.14.2
KTableKTableAbstractJoinValueGetterSupplier	12.14.3

Internals of State Stores

WrappedStateStore	13.1
CachingKeyValueStore	13.1.1
CachingSessionStore	13.1.2
CachingWindowStore	13.1.3
ChangeLoggingKeyValueBytesStore	13.1.4
ChangeLoggingSessionBytesStore	13.1.5
ChangeLoggingWindowBytesStore	13.1.6
MeteredKeyValueStore	13.1.7
MeteredSessionStore	13.1.8
MeteredWindowStore	13.1.9

RocksDBSessionStore	13.1.10
RocksDBWindowStore	13.1.11
InMemoryKeyValueStore	13.1.12
InMemorySessionStore	13.1.13
InMemoryWindowStore	13.1.14
MemoryLRUCache	13.1.15
MeteredKeyValueBytesStore	13.1.16
RocksDBStore	13.1.17
RocksDBTimestampedStore	13.1.18
AbstractStoreBuilder	13.2
KeyValueStoreBuilder	13.2.1
SessionStoreBuilder	13.2.2
TimestampedKeyValueStoreBuilder	13.2.3
TimestampedWindowStoreBuilder	13.2.4
WindowStoreBuilder	13.2.5
InMemorySessionBytesStoreSupplier	13.3
RocksDbKeyValueBytesStoreSupplier	13.4
RocksDbSessionBytesStoreSupplier	13.5
RocksDbWindowBytesStoreSupplier	13.6
KeyValueToTimestampedKeyValueByteStoreAdapter	13.7
WindowToTimestampedWindowByteStoreAdapter	13.8
SegmentedBytesStore	13.9
AbstractRocksDBSegmentedBytesStore	13.9.1
RocksDBSegmentedBytesStore	13.9.2
TimeOrderedKeyValueBuffer	13.10
InMemoryTimeOrderedKeyValueBuffer	13.10.1
CachedStateStore	13.11

Logical Plan of Stream Processing Execution

ProcessorNode	14.1
SourceNode	14.1.1

SinkNode	14.1.2
NodeMetrics	14.2
InternalTopicConfig	14.3
WindowedChangelogTopicConfig	14.3.1
UnwindowedChangelogTopicConfig	14.3.2
WindowedStreamPartitioner	14.4
WindowedSerializer	14.5
DefaultKafkaClientSupplier	14.6
SessionWindow	14.7
TimeWindow	14.8
UnlimitedWindow	14.9
AbstractStream	14.10
GlobalKTableImpl	14.10.1
KGroupedStreamImpl	14.10.2
KGroupedTableImpl	14.10.3
KStreamAggregate	14.10.4
KStreamImpl	14.10.5
KTableImpl	14.10.6
SessionWindowedKStreamImpl — Default SessionWindowedKStream	14.10.7
TimeWindowedKStreamImpl	14.10.8
MaterializedInternal	14.11
KeyValueStoreMaterializer	14.12
InternalNameProvider Contract	14.13
GroupedStreamAggregateBuilder	14.14
KStreamImplJoin	14.15
StaticTopicNameExtractor	14.16
ConsumedInternal	14.17
ProducedInternal	14.18
QuickUnion	14.19
TopicsInfo	14.20

Physical Plan of Stream Processing Execution

ProcessorTopology	15.1
Task	15.2
AbstractTask	15.2.1
StandbyTask	15.2.2
StreamTask	15.2.3
ProcessorContextImpl	15.3
ProducerSupplier	15.4
AssignedTasks	15.5
AssignedStandbyTasks	15.5.1
AssignedStreamsTasks	15.5.2
ProcessorNodePunctuator	15.6

Kafka Streams Execution Engine

TaskManager	16.1
TaskCreator	16.1.1
StandbyTaskCreator	16.1.2
AbstractTaskCreator	16.1.3
StreamThread	16.2
RebalanceListener	16.3
StateListener	16.4
StreamsMetadataState	16.5
StreamsPartitionAssignor	16.6
InternalTopicManager	16.6.1
AssignmentInfo	16.6.2
SubscriptionInfo	16.6.3
ClientMetadata	16.6.4
TaskAssignor Contract	16.6.5
StickyTaskAssignor	16.6.5.1
InternalProcessorContext Contract	16.7
AbstractProcessorContext	16.7.1
GlobalProcessorContextImpl	16.7.2
StandbyContextImpl	16.7.3
ForwardingDisabledProcessorContext	16.8

GlobalStreamThread	16.9
StateConsumer	16.9.1
GlobalStateManager	16.10
GlobalStateUpdateTask	16.10.1
RecordCollector	16.11
RecordCollectorImpl	16.11.1
ThreadCache	16.12
NamedCache	16.12.1
Stamped	16.13
PartitionGroup	16.14
RecordInfo	16.14.1
RecordQueue	16.15
StampedRecord	16.15.1
PunctuationQueue	16.16
PunctuationSchedule	16.16.1
QueryableStoreProvider	16.17
StateStoreProvider	16.18
StreamThreadStateStoreProvider	16.18.1
GlobalStateStoreProvider	16.18.2
WrappingStoreProvider	16.18.3
RecordDeserializer	16.19
StateDirectory	16.20
ProcessorRecordContext	16.21
CopartitionedTopicsValidator	16.22

State (Store) Management

StateManager	17.1
AbstractStateManager	17.1.1
ProcessorStateManager	17.2
GlobalStateManager	17.3
GlobalStateManagerImpl	17.3.1
Checkpointable	17.4
OffsetCheckpoint	17.5

ChangelogReader	17.6
StoreChangelogReader	17.6.1
StateRestorer	17.7
RestoringTasks	17.8
RecordBatchingStateRestoreCallback	17.9
CompositeRestoreListener	17.9.1
StateRestoreCallbackAdapter	17.10

The Internals of Kafka Streams 2.3.0

Welcome to **The Internals of Kafka Streams** gitbook! I'm very excited to have you here and hope you will enjoy exploring the internals of Kafka Streams as much as I have.

I write to discover what I know.

— Flannery O'Connor

I'm [Jacek Laskowski](#), a freelance IT consultant, software engineer and technical instructor specializing in [Apache Spark](#), [Apache Kafka](#) and [Kafka Streams](#) (with [Scala](#) and [sbt](#)).

I offer software development and consultancy services with hands-on in-depth workshops and mentoring. Reach out to me at jacek@japila.pl or [@jaceklaskowski](https://twitter.com/jaceklaskowski) to discuss opportunities.

Consider joining me at [Warsaw Scala Enthusiasts](#) and [Warsaw Spark](#) meetups in Warsaw, Poland.

Tip

I'm also writing other books in the "The Internals of" series about [Apache Kafka](#), [Apache Spark](#), [Spark SQL](#), and [Spark Structured Streaming](#).

Expect text and code snippets from a variety of public sources. Attribution follows.

Now, let me introduce you to [Kafka Streams](#).

Kafka Streams — Stream Processing Library on Apache Kafka

Kafka Streams is a library for developing applications for processing records from topics in Apache Kafka.

A Kafka Streams application processes record streams through a [topology](#) (of stream processors). A Kafka Streams developer defines the processing logic using either the [high-level Streams DSL](#) (with streams and tables) or [low-level Processor API](#) (with stream processing nodes).

Once you have defined the processing logic in the form of a Kafka Stream topology, you should define the execution environment of the application using [KafkaStreams API](#) that you [start](#) in the end.

```

object KafkaStreamsApp extends App {

    // Step 0. Imports for Scala API for Kafka Streams
    import org.apache.kafka.streams.scala._
    import ImplicitConversions._
    import Serdes._

    // Step 1. Describe Topology
    // Consume records from input topic and produce records to upper topic
    val builder = new StreamsBuilder

    val uppers = builder
        .stream[String, String]("input") // Consume records as a stream
        .mapValues(_.toUpperCase)         // Transform (map) the values

    // Produce records to "upper" topic
    uppers.to("upper")

    // Print out records to stdout for debugging purposes
    import org.apache.kafka.streams.kstream.Printed
    val sysout = Printed
        .toSysOut[String, String]
        .withLabel("stdout")
    uppers.print(sysout)

    // Step 2. Build Topology
    val topology = builder.build

    // You can describe the topology and just finish
    println(topology.describe())

    // That finishes the "declaration" part of developing a Kafka Stream application
    // Nothing is executed at this time (no threads have started yet)

    // Step 3. Specify Configuration
    import java.util.Properties
    val props = new Properties()
    import org.apache.kafka.streams.StreamsConfig
    val appId = this.getClass.getSimpleName.replace("$", "")
    props.put(StreamsConfig.APPLICATION_ID_CONFIG, appId)
    props.put(StreamsConfig.CLIENT_ID_CONFIG, appId)
    props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, ":9092")

    // Step 4. Create Kafka Streams Client
    import org.apache.kafka.streams.KafkaStreams
    val ks = new KafkaStreams(topology, props)

    // Step 5. Start Stream Processing, i.e. consuming, processing and producing records
    ks.start
}

```

Since Kafka Streams is a library, you should define it as the dependency of your Kafka Streams application.

```
val kafkaVer = 2.3.0
libraryDependencies += "org.apache.kafka" % "kafka-streams" % kafkaVer
libraryDependencies += "org.apache.kafka" %% "kafka-streams-scala" % kafkaVer
```

On the outside, a typical Kafka Streams application is made up of two main objects, i.e. [Topology](#) and [KafkaStreams](#) (aka *streams client*).

On the inside, the `KafkaStreams` object creates a [StateDirectory](#), a `Metric` (with a `JmxReporter`), a [StreamsMetadataState](#), a [GlobalStreamThread](#) and one or more [StreamThreads](#).

`KafkaStreams` uses `num.stream.threads` configuration property for the number of `StreamThreads` to create (default: `1` processing thread).

A single `StreamThread` creates a restore Kafka Consumer (`restoreConsumer`), a [StoreChangelogReader](#), a [StreamsMetricsThreadImpl](#), a [ThreadCache](#), a [TaskCreator](#) (for [StreamTasks](#)), a [StandbyTaskCreator](#) (for [StandbyTasks](#)) and the [TaskManager](#).

It is through the [TaskManager](#) that a `StreamThread` manages the [active stream tasks](#) and processes records.

A `streamThread` creates also a [RebalanceListener](#) for...FIXME

Eventually, `KafkaStreams` is [started](#) and the Kafka Streams application starts consuming, processing, and producing records (as described by the [Topology](#)).

When [started](#), `KafkaStreams` starts the [GlobalStreamThread](#) and the [StreamThreads](#).

A `streamThread` runs the main record processing loop that uses a [KafkaConsumer](#) to subscribe to the [source topics](#) of the topology (with the [ConsumerRebalanceListener](#) for dynamically assigned partitions).

A `streamThread` then uses the [KafkaConsumer](#) to fetch records (from the [source topics](#)).

`StreamThread` uses `poll.ms` configuration property for the poll time (default: `100L` milliseconds) or `0` when in `PARTITIONS_ASSIGNED` state.

If there are records to process and there are [active streams tasks](#), `StreamThread` ...FIXME

Why Kafka Streams

In this section you can find reasons that could make Kafka Streams a viable contender among the solutions for your stream processing project.

1. [Apache Kafka as the Data Storage](#)
2. [Writing Kafka ConsumerRebalanceListener Is Tricky](#)

Apache Kafka as the Data Storage

Kafka Streams is a library for developing applications for record stream processing where the data is in topics in Apache Kafka.

Note	Kafka Streams supports consuming from and producing records to a single Apache Kafka cluster only (unless you use <code>foreach</code> operator for producing records to another Kafka cluster which means that you are in charge of managing a Kafka Producer to that cluster).
------	--

Writing Kafka ConsumerRebalanceListener Is Tricky

Quoting the documentation of

[org.apache.kafka.clients.consumer.ConsumerRebalanceListener](#):

ConsumerRebalanceListener is a callback interface that the user can implement to trigger custom actions when the set of partitions assigned to the consumer changes.

When Kafka is managing the group membership, a partition re-assignment will be triggered any time the members of the group change or the subscription of the members changes. This can occur when processes die, new process instances are added or old instances come back to life after failure. Rebalances can also be triggered by changes affecting the subscribed topics (e.g. when the number of partitions is administratively adjusted).

Multi-Instance Kafka Streams Applications

A single Kafka Streams application can be executed in a group (of individual [stream processing nodes](#) that are identified by the same [application.id](#)).

The stream processing clients can be run on the same physical machine or separate nodes.

From the perspective of a Apache Kafka cluster the instances all together act as a **consumer group** (and, by the rules of consumer group, they share the partitions in such a way that no two instances access a partition).

If a topology uses local state stores, they are owned exclusively (and not shared) by the instances themselves that have an exclusive access to the stores (that hold state based on the records in the exclusive set of partitions assigned to them).

In such a stream processing group, every `KafkaStreams` streams client can however expose a user-defined endpoint (as a pair of host and port using [application.server](#) configuration property) that allows for [discovering the location of and connecting to the state stores and the partitions available on the instance](#).

The KafkaStreams instances become **discoverable** as a feature of KafkaStreams library not some external discovery framework.

That distributed yet interconnected Kafka Streams application allows for developing APIs and services that could use the state distributed across stream processing nodes (that span over multiple machines).

Tip

Read up more on the feature in the initial code drop as part of [KAFKA-3914: Global discovery of state stores](#) and [KIP-67: Queryable state for Kafka Streams](#).

Exactly-Once Support (EOS)

Exactly-Once Support (EOS) (aka Exactly-Once Processing Guarantee) is...FIXME

`StreamsConfig` defines `eosEnabled` internal flag that is enabled (`true`) when `StreamsConfig.PROCESSING_GUARANTEE_CONFIG` is `StreamsConfig.EXACTLY_ONCE` (`exactly_once`).

`StreamsConfig.PROCESSING_GUARANTEE_CONFIG` can be one of the two values:

- `AT_LEAST_ONCE` (default)
- `EXACTLY_ONCE`

With EOS enabled, the frequency with which to save the position of a processor (`StreamsConfig.COMMIT_INTERVAL_MS_CONFIG`) is always `100L`.

Exactly-once processing requires a cluster of at least three brokers (by default) what is the recommended setting for production. For development you can change this, by adjusting broker setting `transaction.state.log.replication.factor` and `transaction.state.log.min_isr`.

Notes

- Committing and suspending StreamTasks are sensitive to EOS (per `eosEnabled` flag)
- With Exactly-once support enabled `StreamTask` uses `StreamTask.initializeTransactions` when created and requested to resume

KafkaStreams, StreamThreads, StreamTasks and StandbyTasks

KafkaStreams, StreamThreads and StreamsConfig.NUM_STREAM_THREADS_CONFIG

A [KafkaStreams](#) instance uses [StreamThreads](#) for stream processing.

The number of `StreamThreads` is controlled by `StreamsConfig.NUM_STREAM_THREADS_CONFIG` (`num.stream.threads`) configuration property.

StreamThread, Kafka Consumer and Consumer Group

`StreamThread` is given a [Kafka Consumer](#) (`consumer<byte[], byte[]>`) using [KafkaClientSupplier](#) when created.

`StreamThread` [uses the Kafka consumer](#) to act as a regular Kafka consumer (e.g. subscribes to the source topics of a topology or poll records) and simply becomes a Kafka client.

When `KafkaClientSupplier` is requested for a [Kafka Consumer](#), `StreamThread` passes on the [consumer configuration](#) with `ConsumerConfig.GROUP_ID_CONFIG` for consumer group ID and `CommonClientConfigs.CLIENT_ID_CONFIG` for client ID.

`ConsumerConfig.GROUP_ID_CONFIG` is exactly `StreamsConfig.APPLICATION_ID_CONFIG`.

In summary, using the required `StreamsConfig.APPLICATION_ID_CONFIG` for group ID among all `StreamThreads` of a Kafka Streams application creates a consumer group. Dynamic partition assignment is controlled using Kafka protocol via [StreamsPartitionAssignor](#) among all `StreamThreads`.

StreamThread, TaskManager and StreamTasks

Every `StreamThread` manages its own [TaskManager](#) (with the factories to create `stream` and `standby` tasks).

When requested to [poll records once and process them using active stream tasks](#), `StreamThread` requests the `TaskManager` for [active stream tasks](#) for every partition with records (see `StreamThread.addRecordsToTasks`).

A `StreamTask` is responsible for processing records (one at a time) from the assigned partitions (using record buffers (partition queues of `RecordQueues` per `TopicPartition`)).

When created, `StreamTask` creates a `PartitionGroup` (with `RecordQueues`). A `RecordQueue` is created with the `SourceNode` of the `ProcessorTopology` for the topic of the partition.

A `RecordQueue` is simply a partition and a `SourceNode`.

When requested to process a single record, `StreamTask` requests the `PartitionGroup` for the next stamped record (record with timestamp) and the `RecordQueue` and simply requests the corresponding `SourceNode` to process the record.

When requested to process a record, a `SourceNode` simply requests the `ProcessorContext` to forward it (down the topology) and so the record goes (*visits / is pushed down to*) every node in the topology.

`StreamThread` uses `TaskManager` for processing records using `AssignedStreamsTasks` (that uses the running `StreamTasks`).

Every loop of `StreamThread` (through the `TaskManager` and the `AssignedStreamsTasks`), every running `StreamTask` is requested to process a single record (the next `StampedRecord` from the `PartitionGroup`).

When requested for a `StreamTask`, `TaskCreator` is given partitions.

Note	Explore the relationship between topic groups, partitions and tasks (<code>builder.build(taskId.topicGroupId)</code> while creating a <code>StreamTask</code>) which is <code>TaskManager.addStreamTask</code> .
------	---

The number of `StreamTask` is exactly the number of partitions assigned (as `streamTasks` simply create a consumer group and so every partition can only be consumed by one and exactly one consumer group member).

- Number of StreamTasks = # `TaskCreator.createTask` = # `AbstractTaskCreator.createTasks` = `TaskManager.addStreamTasks`

`StreamsConfig.NUM_STANDBY_REPLICAS_CONFIG` (`num.standby.replicas`) is the number of standby replicas for each task.

- `StickyTaskAssignor.assignActive` = number of tasks per StreamThread
- `StickyTaskAssignor.assign`

`StreamsPartitionAssignor.assign` —> `TaskManager.builder().topicGroups()` —> `PartitionGrouper.partitionGroups(sourceTopicsByGroup, fullMetadata)`

At this step = `Map<TaskId, Set<TopicPartition>>` `partitionsForTask`

Enable DEBUG logging level for
org.apache.kafka.streams.processor.internals.StreamsPartitionAssignor

DEBUG Assigning tasks {} to clients {} with number of replicas {}

Step 1. StreamsPartitionAssignor.onAssignment(final Assignment assignment) —>
TaskManager.setAssignmentMetadata(Map<TaskId, Set<TopicPartition>> activeTasks,
Map<TaskId, Set<TopicPartition>> standbyTasks)

Step 2. RebalanceListener.onPartitionsAssigned(Collection<TopicPartition> assignment) —>
TaskManager.createTasks(assignment)

`StreamsPartitionAssignor` makes sure that the [number of assigned partitions to a Kafka Streams application instance is exactly the same as number of active tasks](#).

Demo: Creating Topology with State Store with Logging Enabled

The following demo shows the internals of [state stores with logging enabled](#).

Logging / log4j.properties

The following logging configuration may help understand the internals. Use the following in `log4j.properties` :

```
log4j.rootLogger=OFF, stdout
log4j.appenders.stdout=org.apache.log4j.ConsoleAppender
log4j.appenders.stdout.target=System.out
log4j.appenders.stdout.layout=org.apache.log4j.PatternLayout
log4j.appenders.stdout.layout.ConversionPattern=[%d] %p %m (%c)%n

log4j.logger.org.apache.kafka.streams=INFO
log4j.logger.org.apache.kafka.streams.processor.internals.StreamTask=ALL
log4j.logger.org.apache.kafka.streams.processor.internals.StandbyTask=ALL
```

Refer to [Application Logging Using log4j](#).

Kafka Setup

Before you run the demo application, use `./bin/kafka-topics.sh --create` to create the source topic.

```
./bin/kafka-topics.sh \
--bootstrap-server :9092 \
--create \
--topic StateStoreLoggingEnabledDemo-input \
--partitions 1 \
--replication-factor 1
```

Use `--partitions 1` as that helps keeping the log messages at the minimum, just to understand the internals. [More partitions would simply create many tasks](#) (and add nothing but more log messages).

After the demo application is up and running, use `./bin/kafka-topics.sh --list` to list the topics used in the topology.

```
$ ./bin/kafka-topics.sh --bootstrap-server :9092 --list
StateStoreLoggingEnabledDemo-StateStoreLoggingEnabledDemo-in-memory-key-value-store-ch
angelog
StateStoreLoggingEnabledDemo-StateStoreLoggingEnabledDemo-in-memory-key-value-store-re
partition
StateStoreLoggingEnabledDemo-input
```

After the demo, you may also want to reset the Kafka Streams demo application (in order to reprocess its data from scratch) using `./bin/kafka-streams-application-reset.sh` script.

```
./bin/kafka-streams-application-reset.sh \
--application-id StateStoreLoggingEnabledDemo \
--input-topics StateStoreLoggingEnabledDemo-input \
--execute
```

Demo Application

```
val appId = this.getClass.getSimpleName.replace("$", "")
val storeName = s"$appId-in-memory-key-value-store"
val source = s"$appId-input"
val bootstrapServers = ":9092"

println(s"Application ID: $appId")
println(s"Source topics: $source")
println("Make sure that the source topics are available and press ENTER")
/*
 * ./bin/kafka-topics.sh \
 *   --bootstrap-server :9092 \
 *   --create \
 *   --topic StateStoreLoggingEnabledDemo-input \
 *   --partitions 1 \
 *   --replication-factor 1
 */
System.in.read()

// Using Scala API for Kafka Streams
import org.apache.kafka.streams.scala._
import ImplicitConversions._
import Serdes._

import org.apache.kafka.streams.state.Stores
val storeSupplier = Stores.inMemoryKeyValueStore(storeName)

import org.apache.kafka.streams.scala.kstream.Materialized
import scala.collection.JavaConverters._
val materialized: Materialized[String, Long, ByteArrayKeyValueStore] = Materialized
  .as[String, Long](storeSupplier)
  .withLoggingEnabled(Map.empty[String, String].asJava)
```

```
import org.apache.kafka.streams.kstream.Printed
val sysout = Printed.toSysOut[String, Long].withLabel("sysout")

val builder = new StreamsBuilder
builder
  .stream[String, String](source)
  .selectKey { case (_, v) => v.split(",").head }
  .groupByKey
  .count()(materialized)
  .toStream
  .print(sysout)

val topology = builder.build

println(topology.describe)

import org.apache.kafka.streams.StreamsConfig
val config = new java.util.Properties()
// You'd usually use APPLICATION_ID_CONFIG, but it is simply too long
config.put(StreamsConfig.APPLICATION_ID_CONFIG, appId)
// Using CLIENT_ID_CONFIG allows for explicit application ID
config.put(StreamsConfig.CLIENT_ID_CONFIG, appId)
config.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers)

import org.apache.kafka.streams.KafkaStreams
val ks = new KafkaStreams(topology, config)

// TIP: Copy and paste the above code using :paste mode in sbt console / Scala REPL
// So all the INFO messages go away
// So you can focus on the log messages after starting the KafkaStreams instance

ks.start()
```

Stateful Stream Processing

Stateful Stream Processing is...FIXME

Note

A `StateStore` can be **local** or **global** (although it is a property of a [ProcessorTopology](#)).

Kafka Streams developers use [Stores](#) utility for creating state stores.

Note

A `StateStore` can be backed by a changelog topic in the Kafka cluster for fault-tolerance, i.e. with logging enabled (although it is an implementation detail, and not part of the `StateStore` contract). Kafka Streams developers use [Materialized](#) or [StoreBuilder](#) APIs to control logging.

Note

A `StateStore` can be cached for performance, i.e. with caching enabled (although it is an implementation detail, and not part of the `StateStore` contract). Kafka Streams developers use [Materialized](#) or [StoreBuilder](#) APIs to control logging.

StateStore Contract — State Storage Engines

`StateStore` is the [contract](#) of [state storage engines](#) (*state stores*) that store a state.

`StateStore` can be [persistent](#) or not (i.e. [in-memory](#)).

Table 1. StateStore Contract

Method	Description
<code>close</code>	<pre>void close()</pre> <p>Closes the state store</p> <p>Used when:</p> <ul style="list-style-type: none"> • <code>AbstractStateManager</code> is requested to reinitializeStateStoresForPartitions • <code>GlobalStateManagerImpl</code> and <code>ProcessorStateManager</code> are requested to close • <code>RocksDBStore</code> is requested to <code>toggleDbForBulkLoading</code> • <code>Segments</code> is requested to close and cleanup
<code>flush</code>	<pre>void flush()</pre> <p>Flushes cached data</p> <p>Used when:</p> <ul style="list-style-type: none"> • <code>GlobalStateManagerImpl</code> and <code>ProcessorStateManager</code> are requested to flush their state stores • <code>CachingKeyValueStore</code>, <code>CachingSessionStore</code> and <code>CachingWindowStore</code> are requested to close state stores • <code>RocksDBSegmentedBytesStore</code> is requested to initialize the state store • <code>Segments</code> is requested to <code>flush</code>
<code>init</code>	<pre>void init(ProcessorContext context, StateStore root)</pre> <p>Initializes the state store</p> <p>Used when:</p>

	<ul style="list-style-type: none"> • <code>AbstractStateManager</code> is requested to reinitializeStateStoresForPartitions • <code>AbstractTask</code> is requested to registerStateStores • <code>GlobalStateManagerImpl</code> is requested to initialize
<code>isOpen</code>	<pre>boolean isOpen()</pre> <p>Indicates whether the state store is open or not</p> <p>Used when:</p> <ul style="list-style-type: none"> • <code>GlobalStateStoreProvider</code> and <code>StreamThreadStateStoreProvider</code> are requested to <code>stores</code> • <code>SegmentIterator</code> is requested to <code>hasNext</code> • <code>Segments</code> is requested to <code>segments</code> and <code>allSegments</code>
<code>name</code>	<pre>String name()</pre> <p>Name of the state store (for identification purposes)</p>
<code>persistent</code>	<pre>boolean persistent()</pre> <p>Indicates whether the state store is persistent (<code>true</code>) or not (<code>false</code>)</p> <p>Used when:</p> <ul style="list-style-type: none"> • <code>GlobalStateManagerImpl</code> is created • <code>ProcessorStateManager</code> is requested to register a state store and checkpoint

Table 2. StateStores (Direct Extensions)

StateStore	Description
KeyValueStore	
SegmentedBytesStore	
SessionStore	
TimeOrderedKeyValueBuffer	
WindowStore	
WrappedStateStore	

KeyValueStore Contract — Key-Value State Stores with Range Queries

`KeyValueStore` (`KeyValueStore<K, V>`) is the extension of the [StateStore contract](#) for [key-value state stores](#) that allow for [inserting](#), [updating](#) and [deleting](#) key-value pairs.

`KeyValueStore` is also a [ReadOnlyKeyValueStore](#) that allows for range queries.

Table 1. KeyValueStore Contract

Method	Description
<code>delete</code>	<pre>V delete(K key)</pre> <p>Deletes the value for the non-<code>null</code> key from the store (if there was one) and gives the old value or <code>null</code> if there was no such key.</p> <p>The key must not be <code>null</code> or an <code>NullPointerException</code> is thrown.</p>
<code>put</code>	<pre>void put(K key, V value)</pre> <p>Inserts or updates the value associated with the key</p>
<code>putAll</code>	<pre>void putAll(List<KeyValue<K, V>> entries)</pre>
<code>putIfAbsent</code>	<pre>V putIfAbsent(K key, V value)</pre>

Table 2. KeyValueStores (Direct Implementations and Extensions Only)

KeyValueStore	Description
CachingKeyValueStore	
ChangeLoggingKeyValueBytesStore	
InMemoryKeyValueStore	
InMemoryTimestampedKeyValueStoreMarker	
KeyValueToTimestampedKeyValueByteStoreAdapter	
MemoryLRUCache	
MeteredKeyValueStore	
RocksDBStore	
Segment	
TimestampedKeyValueStore	

SessionStore

`SessionStore` is the contract of state stores that [FIXME](#).

```
package org.apache.kafka.streams.state;

interface SessionStore<K, AGG> extends StateStore, ReadOnlySessionStore<K, AGG> {
    KeyValueIterator<Windowed<K>, AGG> findSessions(final K key, long earliestSessionEndTime,
                                                       final long latestSessionStartTime);
    KeyValueIterator<Windowed<K>, AGG> findSessions(final K keyFrom, final K keyTo, long earliestSessionEndTime,
                                                       final long latestSessionStartTime);
    void put(final Windowed<K> sessionKey, final AGG aggregate);
    void remove(final Windowed<K> sessionKey);
}
```

Table 1. SessionStore Contract

Method	Description
<code>findSessions</code>	Used when... FIXME
<code>put</code>	<p>Storing an aggregated value for a session key</p> <p>Used when:</p> <ul style="list-style-type: none"> • <code>KStreamSessionWindowAggregateProcessor</code> is requested to process a key-value record • <code>CachingSessionStore</code> is requested to remove a session key and putAndMaybeForward • <code>ChangeLoggingSessionBytesStore</code> is requested to store an aggregated value for a session key • <code>MeteredSessionStore</code> is requested to store an aggregated value for a session key
<code>remove</code>	Used when... FIXME

Table 2. SessionStores

SessionStore	Description
CachingSessionStore	
RocksDBSessionStore	
ChangeLoggingSessionBytesStore	
MeteredSessionStore	

WindowStore

`WindowStore` is the [extension](#) of the [StateStore contract](#) for [state stores](#) that are [ReadOnlyWindowStores](#) and can [put](#) a record with an optional timestamp.

Table 1. WindowStore Contract

Method	Description
<code>put</code>	<pre>void put(K key, V value) (1) void put(K key, V value, long timestamp)</pre> <p>1. Uses the current wall-clock time as the timestamp Used when KStreamJoinWindowProcessor and KStreamWindowAggregateProcessor are requested to process a record</p>

Table 2. WindowStores

WindowStore	Description
CachingWindowStore	
ChangeLoggingWindowBytesStore	
MeteredWindowStore	
RocksDBWindowStore	

Stores Utility — Factory of State Stores

`Stores` is a utility for creating state stores (via `StoreSuppliers` or `StoreBuilders`).

```
import org.apache.kafka.streams.state.Stores
val kvStore = Stores.inMemoryKeyValueStore("store-name")
scala> :type kvStore
org.apache.kafka.streams.state.KeyValueBytesStoreSupplier
```

Table 1. Stores API (Static Methods)

Method	Description
<code>inMemoryKeyValueStore</code>	<pre>KeyValueBytesStoreSupplier inMemoryKeyValueStore(String name)</pre> <p>Creates a <code>KeyValueBytesStoreSupplier</code> of <code>KeyValueInMemoryKeyValueStore</code>)</p>
<code>inMemorySessionStore</code>	<pre>SessionBytesStoreSupplier inMemorySessionStore(String name, Duration retentionPeriod)</pre> <p>Creates a <code>SessionBytesStoreSupplier</code> of <code>SessionSInMemorySessionBytesStoreSupplier</code>)</p>
<code>inMemoryWindowStore</code>	<pre>WindowBytesStoreSupplier inMemoryWindowStore(String name, Duration retentionPeriod, Duration windowSize, boolean retainDuplicates)</pre> <p>Creates a <code>WindowBytesStoreSupplier</code> of <code>WindowSInMemoryWindowBytesStoreSupplier</code>)</p>
<code>keyValueStoreBuilder</code>	<pre>StoreBuilder<KeyValueStore<K, V>> keyValueStoreBuilder(KeyValueBytesStoreSupplier supplier, Serde<K> keySerde, Serde<V> valueSerde)</pre> <p>Creates a <code>KeyValueStoreBuilder</code> of <code>KeyValueStore</code></p>
<code>lruMap</code>	<pre>KeyValueBytesStoreSupplier lruMap(String name, int maxCacheSize)</pre>

	<p>Creates a KeyValueBytesStoreSupplier of KeyValueMemoryNavigableLRUCache)</p>
<code>persistentKeyValueStore</code>	<pre>KeyValueBytesStoreSupplier persistentKeyValueStore(String name)</pre> <p>Creates a KeyValueBytesStoreSupplier of KeyValueRocksDbKeyValueBytesStoreSupplier)</p>
<code>persistentSessionStore</code>	<pre>SessionBytesStoreSupplier persistentSessionStore(String name, Duration retentionPeriod)</pre> <p>Creates a SessionBytesStoreSupplier of SessionSRocksDbSessionBytesStoreSupplier)</p>
<code>persistentTimestampedKeyValueStore</code>	<pre>KeyValueBytesStoreSupplier persistentTimestampedKeyValueStore(String name)</pre> <p>Creates a KeyValueBytesStoreSupplier of KeyValueRocksDbKeyValueBytesStoreSupplier)</p>
<code>persistentTimestampedWindowStore</code>	<pre>WindowBytesStoreSupplier persistentTimestampedWindowStore(String name, Duration retentionPeriod, Duration windowSize, boolean retainDuplicates)</pre> <p>Creates a WindowBytesStoreSupplier of WindowSRocksDbWindowBytesStoreSupplier)</p>
<code>persistentWindowStore</code>	<pre>WindowBytesStoreSupplier persistentWindowStore(String name, Duration retentionPeriod, Duration windowSize, boolean retainDuplicates) WindowBytesStoreSupplier persistentWindowStore(String name, Duration retentionPeriod, Duration windowSize, boolean retainDuplicates, boolean timestampedStore) WindowBytesStoreSupplier persistentWindowStore(String name, long retentionPeriod, long windowSize, boolean retainDuplicates, long segmentInterval, boolean timestampedStore)</pre> <p>Creates a WindowBytesStoreSupplier of WindowSRocksDbWindowBytesStoreSupplier)</p>

sessionStoreBuilder	<pre>StoreBuilder<SessionStore<K, V>> sessionStoreBuilderSupplier(sessionBytesStoreSupplier, Serde<K> keySerde, Serde<V> valueSerde)</pre>
timestampedKeyValueStoreBuilder	<pre>StoreBuilder<TimestampedKeyValueStore<K, V>> timestampedKeyValueStoreBuilderSupplier(KeyValueBytesStoreSupplier supplier, Serde<K> keySerde, Serde<V> valueSerde)</pre> <p>Creates a StoreBuilder of TimestampedKeyValueStores (TimestampedKeyValueStoreBuilder)</p>
timestampedWindowStoreBuilder	<pre>StoreBuilder<TimestampedWindowStore<K, V>> timestampedWindowStoreBuilderSupplier(WindowBytesStoreSupplier supplier, Serde<K> keySerde, Serde<V> valueSerde)</pre> <p>Creates a StoreBuilder of TimestampedWindowStores (TimestampedWindowStoreBuilder)</p>
windowStoreBuilder	<pre>StoreBuilder<WindowStore<K, V>> windowStoreBuilderSupplier(WindowBytesStoreSupplier supplier, Serde<K> keySerde, Serde<V> valueSerde)</pre> <p>Creates a StoreBuilder of WindowStores (WindowStoreBuilder)</p>

Materialized — Metadata for State Store Materialized View

`Materialized` provides optional parameters that describe how to materialize a state store (as a [KTable](#)):

- Name of the [state store](#)
- [StoreSupplier](#)

`Materialized` is used for the following:

- In [High-Level Stream Processing DSL](#) to create [KTables](#) and [global KTables](#)
- [KTable API](#)
- For `count`, `reduce` and `aggregate` operators in [KGroupedStream](#), [KGroupedTable](#), [SessionWindowedKStream](#), [TimeWindowedKStream](#)

`Materialized` is [created](#) using the [factory methods](#).

```
Materialized<K, V, S> as(
    String storeName)
Materialized<K, V, KeyValueStore<Bytes, byte[]>> as(
    KeyValueBytesStoreSupplier supplier)
Materialized<K, V, SessionStore<Bytes, byte[]>> as(
    SessionBytesStoreSupplier supplier)
Materialized<K, V, WindowStore<Bytes, byte[]>> as(
    WindowBytesStoreSupplier supplier)
Materialized<K, V, S> with(
    Serde<K> keySerde,
    Serde<V> valueSerde)
```

A `Materialized` instance can further be configured using the ["with"](#) methods.

Table 1. Materialized's with Methods

Method	Description
withCachingDisabled	<code>Materialized<K, V, S> withCachingDisabled()</code> Creates a <code>Materialized</code> with <code>caching disabled</code>
withCachingEnabled	<code>Materialized<K, V, S> withCachingEnabled()</code> Creates a <code>Materialized</code> with <code>caching enabled</code>
withKeySerde	<code>Materialized<K, V, S> withKeySerde(Serde<K> keySerde)</code> Creates a <code>Materialized</code> with the <code>Serde for keys</code>
withLoggingDisabled	<code>Materialized<K, V, S> withLoggingDisabled()</code> Creates a <code>Materialized</code> with <code>logging disabled</code>
withLoggingEnabled	<code>Materialized<K, V, S> withLoggingEnabled(Map<String, String> config)</code> Creates a <code>Materialized</code> with <code>logging enabled</code>
withRetention	<code>Materialized<K, V, S> withRetention(Duration retention)</code>
WithValueSerde	<code>Materialized<K, V, S> withValueSerde(Serde<V> valueSerde)</code> Creates a <code>Materialized</code> with the <code>Serde for values</code>

Scala API for Kafka Streams

Scala API for Kafka Streams makes the optional `Materialized` metadata an implicit parameter in the `KStream` API.

Moreover, `ImplicitConversions` object defines `materializedFromSerde` implicit method that creates a `Materialized` instance with the key and value `Serde` objects available in implicit scope.

Scala API for Kafka Streams also defines `Materialized` object with ``with`` and `as` factory methods that use implicit key and value `Serde` objects.

StoreSupplier Contract — State Store Suppliers for High-Level Streams DSL

`StoreSupplier` is the [abstraction](#) of state store suppliers that can [create state stores](#) in the [Streams DSL — High-Level Stream Processing DSL](#).

Kafka Streams developers usually create a `StoreSupplier` using [Stores](#) utility. The `StoreSupplier` can then be further customized using [Materialized](#).

Note	When using the Low-Level Processor API , Kafka Streams developers use StoreBuilders instead that are then attached to Processors .
------	--

When using the [Low-Level Processor API](#), Kafka Streams developers use [StoreBuilders](#) instead that are then attached to [Processors](#).

Table 1. StoreSupplier Contract

Method	Description
get	<p><code>T get()</code></p> <p>Supplies a state store (as <code>T</code>)</p> <p>Used when StoreBuilders (i.e. KeyValueStoreBuilder, SessionStoreBuilder and WindowStoreBuilder) are requested to build a KeyValueStore</p>
metricsScope	<p><code>String metricsScope()</code></p> <p>Metric scope for the metrics recorded by Metered stores (e.g. MeteredKeyValueBytesStore, MeteredSessionStore and MeteredWindowStore)</p> <p>Used when:</p> <ul style="list-style-type: none"> • StoreBuilders (i.e. KeyValueStoreBuilder, SessionStoreBuilder and WindowStoreBuilder) are requested to build a KeyValueStore • RocksDbSessionBytesStoreSupplier is requested for a SessionStore • RocksDbWindowBytesStoreSupplier is requested for a WindowStore
name	<p><code>String name()</code></p> <p>Name of the state store (for identification purposes)</p> <p>Used when:</p> <ul style="list-style-type: none"> • MaterializedInternal is requested for the storeName • StoreBuilders (i.e. KeyValueStoreBuilder, SessionStoreBuilder and WindowStoreBuilder) are created

Table 2. StoreSuppliers (Interfaces)

StoreSupplier	Description
KeyValueBytesStoreSupplier	
SessionBytesStoreSupplier	
WindowBytesStoreSupplier	

KeyValueBytesStoreSupplier Contract

`KeyValueBytesStoreSupplier` is the [extension](#) of the [StoreSupplier contract](#) for store suppliers of [KeyValueStores](#) (of `Bytes` keys and `byte[]` values, i.e. `StoreSupplier<KeyValueStore<Bytes, byte[]>>`).

`KeyValueBytesStoreSupplier` offers no new methods and simply specifies the type of the [StateStores](#) to [supply](#), i.e. `KeyValueStore<Bytes, byte[]>`.

Table 1. KeyValueBytesStoreSuppliers

KeyValueBytesStoreSupplier	Description
Stores.inMemoryKeyValueStore()	
Stores.lruMap()	
RocksDbKeyValueBytesStoreSupplier	

SessionBytesStoreSupplier Contract

`SessionBytesStoreSupplier` is the [extension](#) of the [StoreSupplier contract](#) for store suppliers that [FIXME](#).

Table 1. SessionBytesStoreSuppliers

SessionBytesStoreSupplier	Description
InMemorySessionBytesStoreSupplier	
RocksDbSessionBytesStoreSupplier	

WindowBytesStoreSupplier Contract — StoreSuppliers of WindowStores

`WindowBytesStoreSupplier` is the extension of the `StoreSupplier` contract for state store suppliers that supply `WindowStores` (i.e. `WindowStores` of type `<Bytes, byte[]>`).

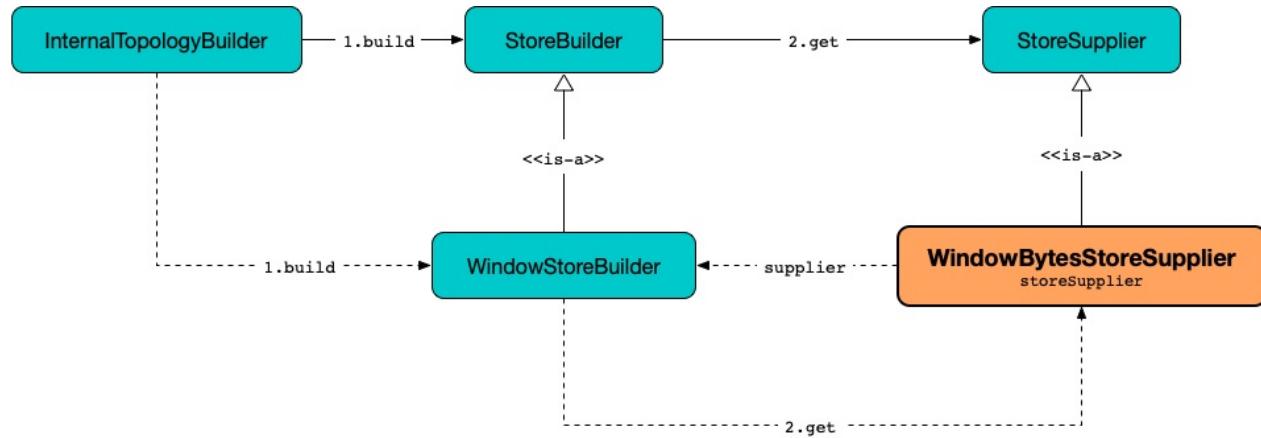


Figure 1. `WindowBytesStoreSupplier` et al.

Note

`RocksDbWindowBytesStoreSupplier` is the one and only known implementation of the `WindowBytesStoreSupplier` Contract.

Table 1. WindowBytesStoreSupplier Contract

Property	Description
<code>retainDuplicates</code>	<pre data-bbox="620 287 997 321"><code>boolean retainDuplicates()</code></pre> <p data-bbox="605 377 1378 451">Used exclusively when <code>WindowStoreBuilder</code> is requested to maybeWrapLogging.</p>
<code>retentionPeriod</code>	<pre data-bbox="636 527 946 561"><code>long retentionPeriod()</code></pre> <p data-bbox="605 617 1378 691">Used exclusively when <code>WindowStoreBuilder</code> is requested for the retentionPeriod.</p>
<code>segmentIntervalMs</code>	<pre data-bbox="636 768 970 801"><code>long segmentIntervalMs()</code></pre> <p data-bbox="605 857 1378 932">Used exclusively when <code>WindowStoreBuilder</code> is requested to maybeWrapCaching.</p>
<code>windowSize</code>	<pre data-bbox="636 1008 874 1042"><code>long windowSize()</code></pre> <p data-bbox="605 1098 1378 1172">Used exclusively when <code>WindowStoreBuilder</code> is requested to maybeWrapCaching.</p>

StoreBuilder Contract — State Store Builders

`StoreBuilder` is the abstraction of state store builders that can build a state store (with optional caching and logging).

Note

`AbstractStoreBuilder` is the one and only known base implementation of the StoreBuilder Contract.

`StoreBuilder` is used to add a local or global state store to a topology using the following APIs:

- StreamsBuilder (i.e. `StreamsBuilder.addStateStore` and `StreamsBuilder.addGlobalStore`)
- Topology (i.e. `Topology.addStateStore` and `Topology.addGlobalStore`).

Table 1. StoreBuilder Contract

Method	Description
<code>build</code>	<pre>T build()</pre> <p>Builds (<i>creates</i>) a state store</p> <p>Used when:</p> <ul style="list-style-type: none"> • <code>InternalTopologyBuilder</code> is requested to <code>rewriteTopology</code> (with any <code>StoreBuilders</code> for global state stores) • <code>StateStoreFactory</code> is requested to <code>build a StateStore</code> (when <code>InternalTopologyBuilder</code> is requested to <code>buildProcessorNode</code>)
<code>logConfig</code>	<pre>Map<String, String> logConfig()</pre> <p>Configuration of the changelog for a state store</p> <p>Used when...FIXME</p>
<code>loggingEnabled</code>	<pre>boolean loggingEnabled()</pre> <p>Indicates whether change-logging should be enabled (<code>true</code>) or not (<code>false</code>) on the state store</p> <p>Used when:</p> <ul style="list-style-type: none"> • <code>InternalTopologyBuilder</code> is requested to <code>addGlobalStore</code>

name	<code>String name()</code> Name of state stores to build (for identification purposes)
withCachingDisabled	<code>StoreBuilder<T> withCachingDisabled()</code> Disables caching on state stores
withCachingEnabled	<code>StoreBuilder<T> withCachingEnabled()</code> Enables caching on state stores
withLoggingDisabled	<code>StoreBuilder<T> withLoggingDisabled()</code> Disables change-logging on state stores (i.e. a changelog for any changes made to the stores)
withLoggingEnabled	<code>StoreBuilder<T> withLoggingEnabled(Map<String, String> config)</code> Enables change-logging on state stores (i.e. a changelog for any changes made to the stores)

TimestampedKeyValueStore

TimestampedKeyValueStore is...FIXME

ReadOnlyKeyValueStore

ReadOnlyKeyValueStore is...FIXME

KafkaStreams — Streams Client

`KafkaStreams` is the [interface](#) for managing and inspecting the execution environment of the [processing topology](#) of a Kafka Streams application.

Note

A `KafkaStreams` instance is also referred by the name **streams client** (esp. in logs).

A `kafkaStreams` instance (process) can be [started](#) or [closed](#) (*shut down*). The current state is available using [state](#).

`KafkaStreams` is running when the [state](#) is either `RUNNING` or `REBALANCING`.

There could be many `KafkaStreams` instances running simultaneously (e.g. as separate JVM processes each with its own [StreamThreads](#)). While a `KafkaStreams` instance is running, it allows for inspecting [streams metadata](#) using `allMetadata`, `allMetadataForStore`, and `metadataForKey` methods.

A `kafkaStreams` instance exposes [monitoring metrics](#) (incl. the Kafka Producer, Consumers, and AdminClient).

Only when in `CREATED` state, a `KafkaStreams` instance can be registered with [StateRestoreListeners](#), [StateListeners](#), and [UncaughtExceptionHandlers](#).

Table 1. KafkaStreams API

Method	Description
<code>allMetadata</code>	<code>Collection<StreamsMetadata> allMetadata()</code> <code>StreamsMetadatas</code> for all instances in a multi-instance Kafka Streams application
<code>allMetadataForStore</code>	<code>Collection<StreamsMetadata> allMetadataForStore(String storeName)</code>
<code>cleanUp</code>	<code>void cleanUp()</code> Cleans up the local directory with state stores
<code>close</code>	<code>void close() (1) boolean close(Duration timeout)</code>

	<p>1. Uses <code>Long.MAX_VALUE</code> for the timeout Closes the <code>KafkaStreams</code> instance</p>
<code>localThreadsMetadata</code>	<code>Set<ThreadMetadata> localThreadsMetadata()</code>
<code>metadataForKey</code>	<pre>StreamsMetadata metadataForKey(String storeName, K key, Serializer<K> keySerializer) StreamsMetadata metadataForKey(String storeName, K key, StreamPartitioner<? super K, ?> partitioner)</pre>
<code>metrics</code>	<code>Map<MetricName, ? extends Metric> metrics()</code> <p>Kafka monitoring metrics of the <code>KafkaStreams</code> instance (incl. the Kafka Producer, Consumers, and AdminClient used by the StreamThreads)</p>
<code>setGlobalStateRestoreListener</code>	<code>void setGlobalStateRestoreListener(StateRestoreListener globalStateRestoreListener)</code> <p>Registers a global <code>StateRestoreListener</code></p>
<code>setStateListener</code>	<code>void setStateListener(KafkaStreams.StateListener listener)</code> <p>Registers a <code>KafkaStreams.StateListener</code></p>
<code>setUncaughtExceptionHandler</code>	<code>void setUncaughtExceptionHandler(Thread.UncaughtExceptionHandler eh)</code> <p>Registers a <code>java.lang.UncaughtExceptionHandler</code></p>
<code>start</code>	<code>void start()</code> <p>Starts the <code>KafkaStreams</code> instance (and the <code>Topology</code>)</p>
<code>state</code>	<code>State state()</code> <p>The current state of the <code>KafkaStreams</code> instance</p>

<pre>store</pre>	<pre>T store(String storeName, QueryableStoreType<T> queryableStoreType)</pre>
------------------	---

`KafkaStreams` is simply a [Kafka client](#) that consumes messages from and produces the processing results to Kafka topics (abstracted as [SourceNodes](#) and [SinkNodes](#), respectively).

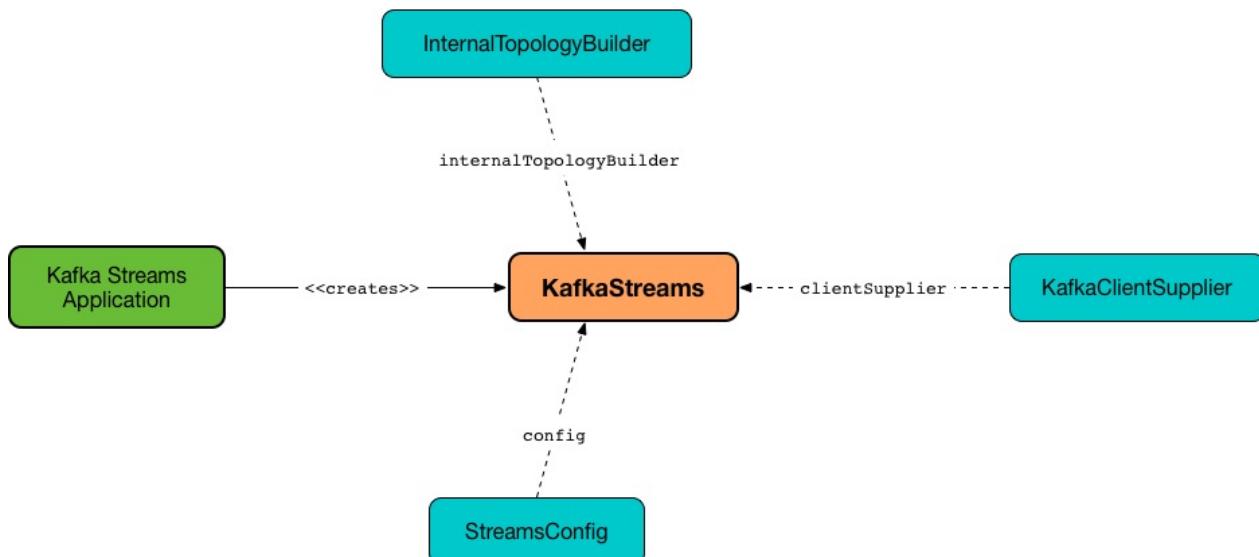


Figure 1. KafkaStreams

Note

A Kafka Streams developer describes the processing logic using a [Topology](#) directly (that is a graph of [processors](#)) or indirectly through a [StreamsBuilder](#) that provides the high-level DSL to define transformations and build a stream processing topology.

```

val topology: Topology = ...
val config: StreamsConfig = ...

import org.apache.kafka.streams.KafkaStreams
val ks = new KafkaStreams(topology, config)
  
```

Once [created](#), `KafkaStreams` is [started up](#) to start consuming, processing, and producing records (as described by a [Topology](#)).

```
ks.start
```

(only when in [CREATED state](#)) `KafkaStreams` can be given a [StateRestoreListener](#) to be informed about the state-related events: [onRestoreStart](#), [onBatchRestored](#) and [onRestoreEnd](#) (through [DelegatingStateRestoreListener](#)).

```
import org.apache.kafka.streams.processor.StateRestoreListener
val userRestoreListener: StateRestoreListener = ???
ks.setGlobalStateRestoreListener(userRestoreListener)
```

KafkaStreams uses a [InternalTopologyBuilder](#) for the following:

- Creating a [StreamsMetadataState](#), [StreamThreads](#) and [QueryableStoreProvider](#)
- Requesting a [global processor topology](#) (for a [GlobalStreamThread](#))

KafkaStreams uses **stream-client [client.id]** for the log prefix (with the [clientId](#)).

Tip Enable ALL logging level for `org.apache.kafka.streams.KafkaStreams` logger to see what happens inside.

Add the following line to `log4j.properties`:

Tip

```
log4j.logger.org.apache.kafka.streams.KafkaStreams=ALL
```

Refer to [Application Logging Using log4j](#).

Cleaning Up Local Directory for State Stores — `cleanUp` Method

```
void cleanUp()
```

`cleanUp` simply requests [StateDirectory](#) to [clean](#) when `KafkaStreams` is not [running](#).

Note

`cleanUp` can only be executed before `KafkaStreams` will be [started](#) or after has been [closed](#).

`cleanUp` reports a [IllegalStateException](#) when `KafkaStreams` is [running](#).

Cannot clean up while running.

Closing KafkaStreams — `close` Method

```
void close() (1)
synchronized boolean close(final long timeout, final TimeUnit timeUnit)
```

1. Calls `close(final long timeout, final TimeUnit timeUnit)` with 0 timeout

`close ...FIXME`

Important	Always execute <code>close</code> on a <code>KafkaStreams</code> instance even if you never call <code>start</code> to avoid resource leaks.
-----------	--

Creating KafkaStreams Instance

```
// public API
KafkaStreams(
    final Topology topology,
    final Properties props) (1)

// public API (mostly for testing)
KafkaStreams(
    final Topology topology,
    final Properties props,
    final KafkaClientSupplier clientSupplier) (3)

KafkaStreams(
    final Topology topology,
    final Properties props,
    final Time time) (4)

// private/internal API
KafkaStreams(
    final InternalTopologyBuilder internalTopologyBuilder,
    final StreamsConfig config,
    final KafkaClientSupplier clientSupplier) (5)

KafkaStreams(
    final InternalTopologyBuilder internalTopologyBuilder,
    final StreamsConfig config,
    final KafkaClientSupplier clientSupplier,
    final Time time) (6)
```

1. Calls the internal `KafkaStreams` (5) with a new `DefaultKafkaClientSupplier`
2. Calls the internal `KafkaStreams` (6) with `SystemTime`

`KafkaStreams` takes the following to be created:

- `InternalTopologyBuilder`
- `StreamsConfig`
- `KafkaClientSupplier`
- `Time`

`KafkaStreams` initializes the `internal properties`.

While being created, `KafkaStreams` ...FIXME

`KafkaStreams` requests the input `KafkaClientSupplier` for a `Kafka AdminClient` (for the `AdminClient configuration` for the `clientId`).

setRunningFromCreated Internal Method

```
boolean setRunningFromCreated()
```

`setRunningFromCreated` ...FIXME

Note	<code>setRunningFromCreated</code> is used exclusively when <code>KafkaStreams</code> is <code>started</code> .
------	---

Starting KafkaStreams — start Method

```
synchronized void start()
throws IllegalStateException, StreamsException
```

`start` starts the `Topology` (that in turn starts consuming, processing, and producing records).

Internally, `start` prints out the following DEBUG message to the logs:

```
Starting Streams client
```

`start` marks KafkaStreams as running (i.e. transitions from CREATED to RUNNING state and notifies `StateListeners`).

`start` starts `global stream thread` if defined (which is when...FIXME)

`start` starts `stream threads`.

`start` schedules a thread that requests `StateDirectory` to `cleanRemovedTasks` every `state.cleanup.delay.ms` milliseconds.

You should see the following DEBUG message in the logs:

```
Started Streams client
```

In case the changing state to running fails, `start` merely prints out the following ERROR message to the logs:

```
Already stopped, cannot re-start
```

Registering Global StateRestoreListener

— setGlobalStateRestoreListener Method

```
void setGlobalStateRestoreListener(final StateRestoreListener globalStateRestoreListener)
```

setGlobalStateRestoreListener registers a [StateRestoreListener](#) (in a Kafka Streams application).

Internally, setGlobalStateRestoreListener simply sets the [globalStateRestoreListener](#) internal property to be the input [StateRestoreListener](#) (only when in [CREATED](#) state).

setGlobalStateRestoreListener throws a [IllegalStateException](#) when not in [CREATED](#) state:

```
Can only set GlobalStateRestoreListener in CREATED state. Current state is: [state]
```

allMetadata Method

```
Collection<StreamsMetadata> allMetadata()
```

allMetadata makes sure that KafkaStreams is running and requests the [StreamsMetadataState](#) for metadata.

Making Sure That KafkaStreams Is Running

— validateIsRunning Internal Method

```
void validateIsRunning()
```

validateIsRunning throws a [IllegalStateException](#) when KafkaStreams is not running. Otherwise, validateIsRunning does nothing.

```
KafkaStreams is not running. State is [state].
```

Note

validateIsRunning is used when KafkaStreams is requested to [allMetadata](#), [allMetadataForStore](#), [metadataForKey](#), [metadataForKey](#), [store](#), and [localThreadsMetadata](#).

Internal Properties

Name	Description
<code>clientId</code>	<p>Client ID</p> <p>Used for the following:</p> <ul style="list-style-type: none"> • logging • metrics • Global thread ID • Requesting the KafkaClientSupplier for the global Kafka Consumer (when creating the GlobalStreamThread) • Requesting the StreamsConfig for the KafkaAdmin configuration (when creating the AdminClient) • Creating StreamThreads • The name of the ScheduledExecutorService thread
<code>globalStateRestoreListener</code>	<p>A user-defined global StateRestoreListener to be notified about the state-related events: onRestoreStart, onBatchRestored and onRestoreEnd (through DelegatingStateRestoreListener)</p> <p>Set using setGlobalStateRestoreListener method</p>
<code>globalStreamThread</code>	<p>GlobalStreamThread</p> <ul style="list-style-type: none"> • Initialized exclusively when InternalTopologyBuilder could build a global ProcessorTopology • Started when <code>KafkaStreams</code> is being started • Set to <code>null</code> while <code>KafkaStreams</code> is being closed
<code>stateDirCleaner</code>	<p>A single-threaded executor (<code>java.util.concurrent.ScheduledExecutorService</code>) that uses a single daemon thread with the name as <code>clientId</code> followed by <code>-CleanupThread</code></p> <p>Used to schedule a periodic action that requests the StateDirectory to cleanRemovedTasks after every <code>state.cleanup.delay.ms</code> milliseconds (and only when the <code>state</code> is <code>RUNNING</code>)</p> <p>Initialized when <code>KafkaStreams</code> is created and shut down when requested to close</p>
	Kafka AdminClient (that allows for managing and inspecting topics, brokers, configurations and ACLs)

<code>adminClient</code>	<ul style="list-style-type: none"> Initialized when <code>KafkaStreams</code> is created for the only purpose of creating StreamThreads (that simply use it to create a TaskManager) Closed when <code>KafkaStreams</code> is closed 		
<code>clientId</code>	<p>Client ID that is initialized when <code>KafkaStreams</code> is created as follows:</p> <ul style="list-style-type: none"> <code>client.id</code> if defined <code>application.id</code> followed by <code>-</code> and the <code>processId</code> 		
<code>queryableStoreProvider</code>	QueryableStoreProvider		
<code>stateDirectory</code>	StateDirectory		
<code>stateLock</code>	Object lock for... FIXME		
<code>streamsMetadataState</code>	<p>StreamsMetadataState (with the InternalTopologyBuilder and <code>application.server</code> configuration property)</p> <p><code>KafkaStreams</code> is simply a public facade to expose the StreamsMetadataState using the following methods:</p> <ul style="list-style-type: none"> <code>allMetadata</code> <code>allMetadataForStore</code> <code>metadataForKey</code> <p>Initialized when <code>KafkaStreams</code> is created to create StreamThreads</p>		
<code>threads</code>	<p>Stream processor threads</p> <table border="1"> <tr> <td>Note</td><td>The number of stream processor threads per KafkaStreams instance is controlled by <code>num.stream.threads</code> configuration property (default: 1 processing thread).</td></tr> </table> <ul style="list-style-type: none"> Created when <code>KafkaStreams</code> is created Started when <code>KafkaStreams</code> is started Shut down when <code>KafkaStreams</code> is closed 	Note	The number of stream processor threads per KafkaStreams instance is controlled by <code>num.stream.threads</code> configuration property (default: 1 processing thread).
Note	The number of stream processor threads per KafkaStreams instance is controlled by <code>num.stream.threads</code> configuration property (default: 1 processing thread).		

Topology — Directed Acyclic Graph of Stream Processing Nodes

`Topology` is a **directed acyclic graph of stream processing nodes** that represents the stream processing logic of a Kafka Streams application.

`Topology` can be [created](#) directly (as part of [Low-Level Processor API](#)) or indirectly using [Streams DSL — High-Level Stream Processing DSL](#).

`Topology` provides the [fluent API](#) to add [local](#) and [global](#) state stores, [sources](#), [processors](#) and [sinks](#) to build advanced stream processing graphs.

`Topology` takes no arguments when created.

```
// Created directly
import org.apache.kafka.streams.Topology
val topology = new Topology

// Created using Streams DSL (StreamsBuilder API)
// Scala API for Kafka Streams
import org.apache.kafka.streams.scala._
import ImplicitConversions._
import Serdes._

val builder = new StreamsBuilder
val topology = builder.build
scala> :type topology
org.apache.kafka.streams.Topology
```

Once [created](#), `Topology` can be extended with [sources](#), [processors](#) (optionally connected to one or more [state stores](#)), [sinks](#), with [local](#) and [global](#) state stores.

```

scala> :type topology
org.apache.kafka.streams.Topology

import org.apache.kafka.streams.state.Stores
val storeBuilder = Stores
  .keyValueStoreBuilder[String, String]()
  Stores.inMemoryKeyValueStore("in-memory-key-value-store"),
  Serdes.String,
  Serdes.String)
  .withLoggingDisabled // this is for a global table
val sourceName = "demo-source-processor"
val timestampExtractor = null
val keyDeserializer = Serdes.String.deserializer
val valueDeserializer = Serdes.String.deserializer
val topic = "demo-topic"
val processorName = "demo-processor-supplier"
import org.apache.kafka.streams.kstream.internals.KTableSource
val stateUpdateSupplier = new KTableSource[String, String]("store-name")
topology.addGlobalStore(
  storeBuilder,
  sourceName,
  timestampExtractor,
  keyDeserializer,
  valueDeserializer,
  topic,
  processorName,
  stateUpdateSupplier)

```

Topology can be described.

```

scala> :type topology
org.apache.kafka.streams.Topology

scala> println(topology.describe)
Topologies:
  Sub-topology: 0 for global store (will not generate tasks)
    Source: demo-source-processor (topics: [demo-topic])
      --> demo-processor-supplier
    Processor: demo-processor-supplier (stores: [in-memory-key-value-store])
      --> none
      <-- demo-source-processor

```

Topology is a logical representation of a ProcessorTopology.

Table 1. Topology API / Methods

Method	Description

	<pre><code>Topology addGlobalStore(StoreBuilder storeBuilder, String sourceName, Deserializer keyDeserializer, Deserializer valueDeserializer, String topic, String processorName, ProcessorSupplier stateUpdateSupplier) Topology addGlobalStore(StoreBuilder storeBuilder, String sourceName, TimestampExtractor timestampExtractor, Deserializer keyDeserializer, Deserializer valueDeserializer, String topic, String processorName, ProcessorSupplier stateUpdateSupplier)</code></pre>
addGlobalStore	<p>Adds a global StateStore (with the StoreBuilder, ProcessorSupplier and optional TimestampExtractor) to the topology.</p> <p>Internally, <code>addGlobalStore</code> simply requests the InternalTopologyBuilder to add a global store.</p>
addProcessor	<pre><code>Topology addProcessor(String name, ProcessorSupplier supplier, String... parentNames)</code></pre> <p>Adds a new processor node (with the ProcessorSupplier) to the topology</p> <p>Internally, <code>addProcessor</code> simply requests the InternalTopologyBuilder to add a processor.</p>

```

Topology addSink(
    String name,
    String topic,
    Serializer<K> keySerializer,
    Serializer<V> valueSerializer,
    StreamPartitioner<? super K, ? super V> partitioner,
    String... parentNames)
Topology addSink(
    String name,
    String topic,
    Serializer<K> keySerializer,
    Serializer<V> valueSerializer,
    String... parentNames)
Topology addSink(
    String name,
    String topic,
    StreamPartitioner<? super K, ? super V> partitioner,
    String... parentNames)
Topology addSink(
    String name,
    String topic,
    String... parentNames)
Topology addSink(
    String name,
    TopicNameExtractor<K, V> topicExtractor,
    Serializer<K> keySerializer,
    Serializer<V> valueSerializer,
    StreamPartitioner<? super K, ? super V> partitioner,
    String... parentNames)
Topology addSink(
    String name,
    TopicNameExtractor<K, V> topicExtractor,
    Serializer<K> keySerializer,
    Serializer<V> valueSerializer,
    String... parentNames)
Topology addSink(
    String name,
    TopicNameExtractor<K, V> topicExtractor,
    StreamPartitioner<? super K, ? super V> partitioner,
    String... parentNames)
Topology addSink(
    String name,
    TopicNameExtractor<K, V> topicExtractor,
    String... parentNames)

```

Adds a new [sink node](#) (with the optional [TopicNameExtractor](#) and [StreamPartitioner](#)) to the topology.

Internally, `addSink` simply requests the [InternalTopologyBuilder](#) to [add a sink](#).

```
Topology addSource(  
    AutoOffsetReset offsetReset,  
    String name,  
    Deserializer keyDeserializer,  
    Deserializer valueDeserializer,  
    Pattern topicPattern)  
Topology addSource(  
    AutoOffsetReset offsetReset,  
    String name,  
    Deserializer keyDeserializer,  
    Deserializer valueDeserializer,  
    String... topics)  
Topology addSource(  
    AutoOffsetReset offsetReset,  
    String name,  
    Pattern topicPattern)  
Topology addSource(  
    AutoOffsetReset offsetReset,  
    String name,  
    String... topics)  
Topology addSource(  
    AutoOffsetReset offsetReset,  
    String name,  
    TimestampExtractor timestampExtractor,  
    Deserializer keyDeserializer,  
    Deserializer valueDeserializer,  
    Pattern topicPattern)  
Topology addSource(  
    AutoOffsetReset offsetReset,  
    String name,  
    TimestampExtractor timestampExtractor,  
    Deserializer keyDeserializer,  
    Deserializer valueDeserializer,  
    String... topics)  
Topology addSource(  
    AutoOffsetReset offsetReset,  
    TimestampExtractor timestampExtractor,  
    String name,  
    Pattern topicPattern)  
Topology addSource(  
    AutoOffsetReset offsetReset,  
    TimestampExtractor timestampExtractor,  
    String name,  
    String... topics)  
Topology addSource(  
    String name,  
    Deserializer keyDeserializer,  
    Deserializer valueDeserializer,  
    Pattern topicPattern)  
Topology addSource(  
    String name,  
    Deserializer keyDeserializer,  
    Deserializer valueDeserializer,  
    String... topics)  
Topology addSource(  
    String name,  
    Pattern topicPattern)  
Topology addSource(  
    String name,  
    String... topics)  
Topology addSource(  
    TimestampExtractor timestampExtractor,  
    String name,  
    Pattern topicPattern)  
Topology addSource(  
    TimestampExtractor timestampExtractor,  
    String name,  
    String... topics)
```

	<p>Adds a new source node (with the optional <code>AutoOffsetReset</code> and <code>TimestampExtractor</code>) to the topology.</p> <p>Internally, <code>addSource</code> simply requests the <code>InternalTopologyBuilder</code> to add a source.</p>
<code>addStateStore</code>	<pre>Topology addStateStore(StoreBuilder storeBuilder, String... processorNames)</pre> <p>Adds a new state store (as a <code>StoreBuilder</code>) to the topology and associates it with processors</p> <p>Internally, <code>addStateStore</code> simply requests the <code>InternalTopologyBuilder</code> to add a state store.</p>
<code>connectProcessorAndStateStores</code>	<pre>Topology connectProcessorAndStateStores(String processorName, String... stateStoreNames)</pre> <p>Connects the processor node with state stores (by name).</p> <p>Internally, <code>connectProcessorAndStateStores</code> simply requests the <code>InternalTopologyBuilder</code> to connect a processor with state stores.</p>
<code>describe</code>	<pre>TopologyDescription describe()</pre> <p>Describes the topology via <code>TopologyDescription</code> (<i>meta representation</i>)</p> <p>Internally, <code>describe</code> simply requests the <code>InternalTopologyBuilder</code> to describe a topology.</p>

Internally, `Topology` uses an `InternalTopologyBuilder` for all the **methods** and is simply a thin layer atop (that aims at making Kafka Streams developers' life simpler).



Figure 1. Topology and InternalTopologyBuilder

`Topology` defines **offset reset policy** (`AutoOffsetReset`) that can be one of the following values:

- `EARLIEST`

- LATEST

TopologyDescription — Meta Representation of Topology

`TopologyDescription` is the abstraction of meta representations (*topology descriptions*) that Kafka Streams developers use to know (*describe*) the global stores and subtopologies of a topology.

Note

`InternalTopologyBuilder.TopologyDescription` is the only available implementation of the `TopologyDescription Contract` in Kafka Streams.

`TopologyDescription` is available using `Topology.describe` method.

```
// Scala API for Kafka Streams
import org.apache.kafka.streams.scala._
import ImplicitConversions._
import Serdes._

val builder = new StreamsBuilder
builder
  .stream[String, String]("records")
  .groupByKey
  .count
  .toStream
  .to("counts")
val topology = builder.build

val meta = topology.describe
scala> :type meta
org.apache.kafka.streams.TopologyDescription

scala> println(meta.globalStores)
[]

scala> println(meta.subtopologies.size)
1
```

Table 1. TopologyDescription Contract

Method	Description
globalStores	<code>Set<GlobalStore> globalStores()</code> The global stores of a topology
subtopologies	<code>Set<Subtopology> subtopologies()</code> The subtopologies of a topology

TopologyDescription.Node

`TopologyDescription.Node` is the abstraction of topology nodes that are identified by `name`, and have `predecessors` and `successors` nodes.

Table 1. `TopologyDescription.Node` Contract

Method	Description
<code>name</code>	<code>String name()</code> Used when...FIXME
<code>predecessors</code>	<code>Set<Node> predecessors()</code> Used when...FIXME
<code>successors</code>	<code>Set<Node> successors()</code> Used when...FIXME

Table 2. `TopologyDescription.Nodes` (Direct Implementations and Extensions)

<code>TopologyDescription.Node</code>	Description
<code>AbstractNode</code>	
<code>Processor</code>	
<code>Sink</code>	
<code>Source</code>	

TopologyDescription.Processor

`TopologyDescription.Processor` is the [extension](#) of the [Node contract](#) for [processor nodes](#) that can have one or more connected [stores](#).

Table 1. `TopologyDescription.Processor` Contract

Method	Description
<code>stores</code>	<code>Set<String> stores()</code> Names of the connected stores Used when...FIXME
Note	InternalTopologyBuilder.Processor is the default and only known implementation of the TopologyDescription.Processor Contract in Kafka Streams.

TopologyDescription.Sink

`TopologyDescription.Sink` is the [extension](#) of the [Node contract](#) for [sink nodes](#) that can have a [topic](#) and [TopicNameExtractor](#).

Table 1. TopologyDescription.Sink Contract

Method	Description
<code>topic</code>	<code>String topic()</code> Used when...FIXME
<code>topicNameExtractor</code>	<code>TopicNameExtractor topicNameExtractor()</code> Used when...FIXME
Note	<code>InternalTopologyBuilder.Sink</code> is the default and only known implementation of the TopologyDescription.Sink Contract in Kafka Streams.

TopologyDescription.Source

`TopologyDescription.Source` is the [extension](#) of the [Node contract](#) for [source nodes](#) that can have [comma-separated topic names](#), [topic names](#), and [topic pattern](#).

Table 1. `TopologyDescription.Source` Contract

Method	Description
<code>topicPattern</code>	<code>Pattern topicPattern()</code> Used when...FIXME
<code>topics</code>	<code>String topics()</code> Used when...FIXME
<code>topicSet</code>	<code>Set<String> topicSet()</code> Used when...FIXME
Note	InternalTopologyBuilder.Source is the default and only known implementation of the TopologyDescription.Source Contract in Kafka Streams.

StreamsConfig — Configuration Properties for Kafka Clients

`StreamsConfig` is a Apache Kafka [AbstractConfig](#) with the [configuration properties](#) for a Kafka Streams application.

`StreamsConfig` is used to reference the [properties names](#) (e.g. to avoid any typos or a better type safety).

```
import org.apache.kafka.streams.StreamsConfig
val conf = new java.util.Properties()
conf.put(StreamsConfig.APPLICATION_ID_CONFIG, "appId")
conf.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, ":9092,localhost:9192")
```

`StreamsConfig` uses **admin.** prefix for the configuration properties that are meant to be used for a Kafka [AdminClient](#). Use [adminClientPrefix](#) method to add the prefix to a `AdminClient` property.

```
import org.apache.kafka.clients.admin.AdminClientConfig
scala> println(AdminClientConfig.RETRIES_CONFIG)
retries

import org.apache.kafka.streams.StreamsConfig.adminClientPrefix
scala> println(adminClientPrefix(AdminClientConfig.RETRIES_CONFIG))
admin.retries
```

Table 1. Constants for Configuration Properties

Name	Property
APPLICATION_ID_CONFIG	application.id
APPLICATION_SERVER_CONFIG	application.server
BOOTSTRAP_SERVERS_CONFIG	Kafka Client's bootstrap.servers
BUFFERED_RECORDS_PER_PARTITION_CONFIG	buffered.records.per.partition
CACHE_MAX_BYTES_BUFFERING_CONFIG	cache.max.bytes.buffering
CLIENT_ID_CONFIG	client.id
COMMIT_INTERVAL_MS_CONFIG	commit.interval.ms

DEFAULT_WINDOWED_KEY_SERDE_INNER_CLASS	default.windowed.key.serde.in
DEFAULT_WINDOWED_VALUE_SERDE_INNER_CLASS	default.windowed.value.serde
DEFAULT_TIMESTAMP_EXTRACTOR_CLASS_CONFIG	default.timestamp.extractor
MAX_TASK_IDLE_MS_CONFIG	max.task.idle.ms
METRICS_RECORDING_LEVEL_CONFIG	metrics.recording.level
NUM_STANDBY_REPLICAS_CONFIG	num.standby.replicas
NUM_STREAM_THREADS_CONFIG	num.stream.threads
PARTITION_GROUPER_CLASS_CONFIG	partition.grouper
POLL_MS_CONFIG	poll.ms
PROCESSING_GUARANTEE_CONFIG	processing.guarantee
REPLICATION_FACTOR_CONFIG	replication.factor
STATE_CLEANUP_DELAY_MS_CONFIG	state.cleanup.delay.ms
STATE_DIR_CONFIG	state.dir
UPGRADE_FROM_CONFIG	
WINDOW_STORE_CHANGE_LOG_ADDITIONAL_RETENTION_MS_CONFIG	windowstore.changelog.additional.retention.ms

`StreamsConfig` is also used for the configuration for the following Kafka clients (that Kafka Streams uses under the covers):

- Consumer
- Producer
- AdminClient
- Restore Consumer

```

import org.apache.kafka.streams.StreamsConfig
val props = new java.util.Properties()
// ...
val conf = new StreamsConfig(props)

// Consumer Properties
val consumerConfigs = conf.getMainConsumerConfigs("groupId", "clientId")
import collection.JavaConverters._
scala> consumerConfigs.asScala.map { case (key, value) => s"$key -> $value" }.foreach(
println)
replication.factor -> 1
num.standby.replicas -> 0
max.poll.records -> 1000
group.id -> groupId
partition.assignment.strategy -> org.apache.kafka.streams.processor.internals.StreamsP
artitionAssignor
bootstrap.servers -> localhost:8082
enable.auto.commit -> false
admin.retries -> 5
application.server ->
max.poll.interval.ms -> 2147483647
auto.offset.reset -> earliest
windowstore.changelog.additional.retention.ms -> 86400000
internal.leave.group.on.close -> false
application.id -> groupId
client.id -> clientId-consumer

```

`StreamsConfig` does not allow users to configure certain Kafka configurations (e.g. for `consumer`) that are simply [removed](#) (with a `WARN` message in the logs).

Table 2. Kafka Consumer Non-Overridable Configurations

Name	Value	Description
max.poll.records	1000	
auto.offset.reset	earliest	
buffered.records.per.partition	1000	The maximum number of records to buffer per partition
enable.auto.commit	false	
internal.leave.group.on.close	false	
max.poll.interval.ms	Integer.MAX_VALUE	

`StreamsConfig` uses `consumer` prefix for custom Kafka configurations of a Kafka consumer.

`StreamsConfig` defines the `InternalConfig` inner class with the [internal properties](#).

Note

`InternalStreamsConfig` is an extension of `StreamsConfig` that is used in `StreamsPartitionAssignor` to specify custom configuration properties and turn the `doLog` flag off.

StreamsConfig.EXACTLY_ONCE for Exactly-Once Support (EOS)

```
String EXACTLY_ONCE = "exactly_once"
```

`StreamsConfig` defines `StreamsConfig.EXACTLY_ONCE` (`exactly_once`) constant value that is used for the following:

- `eosEnabled` internal flag
- `postProcessParsedConfig`
- `checkIfUnexpectedUserSpecifiedConsumerConfig`
- `AbstractTask` is created
- `GlobalStateManagerImpl` is created
- `StreamThread` is created

postProcessParsedConfig Method

```
Map<String, Object> postProcessParsedConfig(
    Map<String, Object> parsedValues)
```

`postProcessParsedConfig ...FIXME`

Note

`postProcessParsedConfig` is used when...FIXME

eosEnabled Internal Flag for Exactly-Once Support (EOS)

`StreamsConfig` defines `eosEnabled` internal flag that is enabled (`true`) when `StreamsConfig.PROCESSING_GUARANTEE_CONFIG` is `EXACTLY_ONCE`.

`eosEnabled` is used for the following:

- ...FIXME

getProducerConfigs Method

```
Map<String, Object> getProducerConfigs(final String clientId)
```

getProducerConfigs ...FIXME

Note

getProducerConfigs is used when...FIXME

Getting Configuration for (Creating) Kafka AdminClient — getAdminConfigs Method

```
Map<String, Object> getAdminConfigs(final String clientId)
```

getAdminConfigs firstly finds the client properties for a Kafka AdminClient (with admin. prefix).

getAdminConfigs takes the getClientCustomProps and copies the AdminClient properties over.

In the end, getAdminConfigs adds the clientId with -admin suffix as the client.id configuration property.

```
import org.apache.kafka.streams.StreamsConfig
val props = new java.util.Properties()
// required configurations
props.put(StreamsConfig.APPLICATION_ID_CONFIG, "demo")
props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, ":9092")

// Define a custom configuration with admin prefix
import org.apache.kafka.clients.admin.AdminClientConfig
props.put("admin." + AdminClientConfig.METADATA_MAX_AGE_CONFIG, "10")
val streamsConf = new StreamsConfig(props)
val adminConfigs = streamsConf.getAdminConfigs("my-client-id")

import scala.collection.JavaConverters._
scala> adminConfigs.asScala.map { case (k,v) => s"$k = $v" }.foreach(println)
bootstrap.servers = :9092
metadata.max.age.ms = 10
client.id = my-client-id-admin
```

Note	<p><code>getAdminConfigs</code> is used when:</p> <ul style="list-style-type: none"> • <code>KafkaStreams</code> is created • <code>InternalTopicManager</code> is created
-------------	--

Collecting Properties per Key — `clientProps` Internal Method

```
Map<String, Object> clientProps(
    final Set<String> configNames,
    final Map<String, Object> originals)
```

`clientProps` collects the configuration properties from `originals` that have their names in the input `configNames`, i.e. includes the properties that have been listed in `configNames`.

Note	<p><code>clientProps</code> is used exclusively when <code>StreamsConfig</code> is requested to getClientPropsWithPrefix.</p>
-------------	---

Getting Subset of User Configuration by Given Names and Prefix — `getClientPropsWithPrefix` Internal Method

```
Map<String, Object> getClientPropsWithPrefix(
    final String prefix,
    final Set<String> configNames)
```

`getClientPropsWithPrefix` takes only the properties (as passed in by a user) that have their keys in `configNames` and adds all properties with the given `prefix`.

Internally, `getClientPropsWithPrefix` collects the configuration properties from the original values of Kafka properties as passed in by a user that have their names in `configNames`.

`getClientPropsWithPrefix` then copies all original settings with the given `prefix` (stripping the prefix before adding them) to the collected properties (and possibly overwriting some).

Note	<p><code>getClientPropsWithPrefix</code> uses <code>AbstractConfig.originals</code> to get the original values of Kafka properties as passed in by the user.</p>
Note	<p><code>getClientPropsWithPrefix</code> is used when <code>streamsConfig</code> is requested for getAdminConfigs, getCommonConsumerConfigs, getMainConsumerConfigs and getProducerConfigs.</p>

Getting Common Consumer Configuration

— `getCommonConsumerConfigs` Internal Method

```
Map<String, Object> getCommonConsumerConfigs()
```

`getCommonConsumerConfigs` gets a subset of user configuration for a Kafka consumer as well as the properties with `consumer` prefix.

Note	<code>getCommonConsumerConfigs</code> uses <code>ConsumerConfig.configNames</code> for the list of the Kafka Consumer-specific configuration keys.
------	--

Caution	FIXME
---------	-------

Note	<code>getCommonConsumerConfigs</code> is used when <code>streamsConfig</code> is requested for <code>getMainConsumerConfigs</code> and <code>getRestoreConsumerConfigs</code> .
------	---

Removing "Illegal" User-Defined Configuration Properties

— `checkIfUnexpectedUserSpecifiedConsumerConfig` Internal Method

```
void checkIfUnexpectedUserSpecifiedConsumerConfig(
    Map<String, Object> clientProvidedProps,
    String[] nonConfigurableConfigs)
```

`checkIfUnexpectedUserSpecifiedConsumerConfig` removes non-configurable configuration properties (`nonConfigurableConfigs`) from user-defined configurations (`clientProvidedProps`) and prints out a warning for any violation.

Internally, `checkIfUnexpectedUserSpecifiedConsumerConfig` iterates over `nonConfigurableConfigs` ...FIXME

Note	<code>checkIfUnexpectedUserSpecifiedConsumerConfig</code> is used when <code>StreamsConfig</code> is requested for <code>getCommonConsumerConfigs</code> and <code>getProducerConfigs</code> .
------	--

`getRestoreConsumerConfigs` Method

```
Map<String, Object> getRestoreConsumerConfigs(final String clientId)
```

`getRestoreConsumerConfigs` ...FIXME

Note	<code>getRestoreConsumerConfigs</code> is used when...FIXME
------	---

Configuration for Kafka Consumer

— `getMainConsumerConfigs` Method

```
Map<String, Object> getMainConsumerConfigs(
    final String groupId,
    final String clientId)
```

`getMainConsumerConfigs` gets the base configuration for a Kafka Consumer first.

`getMainConsumerConfigs` then...FIXME

Note	<code>getMainConsumerConfigs</code> is used exclusively when <code>streamThread</code> is requested to create a <code>StreamThread</code> instance (and requests the <code>KafkaClientSupplier</code> for a Kafka Consumer).
------	--

defaultValueSerde Method

```
Serde defaultValueSerde()
```

`defaultValueSerde` ...FIXME

Note	<code>defaultValueSerde</code> is used when...FIXME
------	---

defaultKeySerde Method

```
Serde defaultKeySerde()
```

`defaultKeySerde` ...FIXME

Note	<code>defaultKeySerde</code> is used when...FIXME
------	---

originalsWithPrefix Method

```
Map<String, Object> originalsWithPrefix(String prefix)
```

`originalsWithPrefix` ...FIXME

Note	<code>originalsWithPrefix</code> is used when...FIXME
------	---

adminClientPrefix Static Method

```
static String adminClientPrefix(final String adminClientProp)
```

`adminClientPrefix` simply adds the `admin.` prefix to a given `adminClientProp`.

Creating StreamsConfig Instance

`StreamsConfig` takes the following to be created:

- Configuration properties
- `doLog` flag

`StreamsConfig` initializes the `eosEnabled` internal property.

Internal Configuration Properties

Name	Value
<code>ASSIGNMENT_ERROR_CODE</code>	<code>_assignment.error.code_</code>
<code>TASK_MANAGER_FOR_PARTITION_ASSIGNOR</code>	<code>_task.manager.instance_</code> Used to associate the <code>TaskManager</code> (of <code>StreamThread</code>) with <code>StreamsPartitionAssignor</code> (when configured)

Configuration Properties

Table 1. Configuration Properties

Name	Description
application.id	<p>Application ID that is the required identifier for the processing application</p> <p>Default: (empty)</p> <p><code>application.id</code> must be unique within the namespace for the default client-id prefix management, and the prefix for internal state stores creates internally).</p> <p><code>application.id</code> is used as a Kafka consumer group identifier. Check out using <code>kafka-consumer-groups</code>.</p> <p>Use StreamsConfig.APPLICATION_ID to reference the property.</p>
application.server	<p>A <code>host:port</code> pair of a user-defined endpoint for a specific Kafka Streams application that contains the locations of local state stores</p> <p>Default: (empty)</p> <p>Used exclusively when <code>StreamPartitioner</code> is <code>null</code>.</p> <p>Use StreamsConfig.APPLICATION_SERVER to reference the property.</p>
buffered.records.per.partition	<p>Maximum number of records to buffer per partition</p> <p>Default: 1000</p> <p>Used exclusively when <code>streamTask</code> is running <code>read record</code> and <code>buffer new records</code> to resume processing respectively</p> <p>Use StreamsConfig.BUFFERED_RECORDS_PER_PARTITION to reference the property.</p>
cache.max.bytes.buffering	<p>Maximum number of memory bytes to buffer per thread (StreamThreads and GlobalThreads)</p> <p>Default: <code>10 * 1024 * 1024L</code> (i.e. 10MB)</p> <p>Use StreamsConfig.CACHE_MAX_BYT to reference the property.</p>

	<p>Client ID of internal consumer, produce pattern <code><client.id>-StreamThread-<thread-id>-<consumer,producer,restore-consumer></code></p> <p><code>client.id</code></p>	<p>Default: (empty)</p> <p>Used when:</p> <ul style="list-style-type: none"> • <code>KafkaStreams</code> is created (and initialized) • <code>StreamsPartitionAssignor</code> is requested <p>Use StreamsConfig.CLIENT_ID_CONFIG</p>
	<p><code>commit.interval.ms</code></p>	<p>Flush interval or commit interval (in milliseconds) that <code>KafkaStreams</code> attempts to commit the position of processed records.</p> <p>Default:</p> <ul style="list-style-type: none"> • <code>100L</code> when <code>processing.guarantee</code> is <code>Guaranteed</code> (the default) • <code>30000L</code> (30s) otherwise <p>Must be at least <code>0L</code></p> <p>Use StreamsConfig.COMMIT_INTERVAL_MS_CONFIG property.</p>
	<p><code>default.windowed.key.serde.inner</code></p>	<p>Inner serde class that implements the <code>org.apache.kafka.common.serialization.Serde</code> interface.</p> <p>Default: (empty)</p> <p>Use StreamsConfig.DEFAULT_WINDOWED_KEY_SERDE_CONFIG to reference the property.</p>
	<p><code>default.windowed.value.serde.inner</code></p>	<p>Inner serde class that implements the <code>org.apache.kafka.common.serialization.Serde</code> interface.</p> <p>Default: (empty)</p> <p>Use StreamsConfig.DEFAULT_WINDOWED_VALUE_SERDE_CONFIG to reference the property.</p>
	<p><code>default.timestamp.extractor</code></p>	<p>Default timestamp extractor class that implements the <code>org.apache.kafka.streams.processor.TimestampExtractor</code> interface.</p> <p>Default: <code>FailOnInvalidTimestamp</code></p>
	<p><code>max.task.idle.ms</code></p>	<p>How long a <code>stream task</code> will stay idle without receiving new records, to avoid potential out-of-order processing of multiple input streams.</p> <p>Default: <code>0L</code></p>

	Use StreamsConfig.MAX_TASK_IDLE_PROPERTY .
<code>metrics.recording.level</code>	<p>The highest recording level for metrics</p> <p>Default: <code>INFO</code></p> <p>Possible values are:</p> <ul style="list-style-type: none"> • <code>INFO</code> • <code>DEBUG</code> <p>Use StreamsConfig.METRICS_RECORDING_LEVEL reference the property.</p>
<code>num.standby.replicas</code>	<p>The number of standby replicas per processor</p> <p>Default: <code>0</code></p> <ul style="list-style-type: none"> • Used when StreamsPartitionAssignmentStrategy is <code>RoundRobin</code> <p>Use StreamsConfig.NUM_STANDBY_REPLICAS reference the property.</p>
<code>num.stream.threads</code>	<p>The number of stream processor threads (used for stream processing)</p> <p>Default: <code>1</code></p> <p>Use StreamsConfig.NUM_STREAM_THREADS reference the property.</p>
<code>partition.grouper</code>	
<code>poll.ms</code>	<p>Polling interval (in milliseconds), i.e. the time between <code>Consumer.poll</code> (unless data is available immediately) or <code>poll()</code> returns immediately with any records than the buffer, else returns empty. Must not be negative.</p> <p>Default: <code>100</code></p> <p>Used when:</p> <ul style="list-style-type: none"> • <code>GlobalStreamThread</code> is requested to poll when <code>KafkaStreams</code> has been requested to poll by <code>StateConsumer</code> • <code>StreamThread</code> is created <p>Use StreamsConfig.POLL_MS_CONFIG reference the property.</p>
	<p>Processing guarantee (aka <i>Exactly-Once</i>)</p> <p>Default: <code>at_least_once</code></p> <p>Possible values are:</p>

	<ul style="list-style-type: none"> • at_least_once • exactly_once
processing.guarantee	<p>Note exactly-once processing guarantees at least once delivery with at least three brokers (with at least one broker in each partition). For development you can change this behavior by setting <code>transaction.state.log.cleanup</code>.</p>
	<p>Use StreamsConfig.PROCESSING_GUARANTEE_CONFIG to set the property.</p>
replication.factor	<p>The replication factor for changelog topics created by a stream processing application</p> <p>Default: 1</p> <p>Use StreamsConfig.REPLICATION_FACTOR_CONFIG to set the property.</p>
state.cleanup.delay.ms	<p>The amount of time (in milliseconds) to wait before a partition has migrated. Only state directories created for at least one partition will be cleaned up.</p> <p>Default: 10 * 60 * 1000 (i.e. 10 mins)</p> <p>Used exclusively when <code>KafkaStreams</code> is created.</p> <p>Use StreamsConfig.STATE_CLEANUP_DELAY_MS_CONFIG to reference the property.</p>
state.dir	<p>Path to the base directory for a state store.</p> <p>Default: <code>/tmp/kafka-streams</code></p> <p>Used when <code>StateDirectory</code> is created.</p> <p>Use StreamsConfig.STATE_DIR_CONFIG to set the property.</p>
windowstore.changelog.additional.retention.ms	<p>Added to a Window's <code>maintainMs</code> to ensure it is not closed prematurely. Allows for clock drift.</p> <p>Default: 24 * 60 * 60 * 1000L (i.e. 1 day)</p>

Logging

Application Logging Using log4j — `log4j.properties` Logging Configuration File

`log4j.properties`

```
log4j.rootLogger=INFO, stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=[%d] %p %m (%c)%n

log4j.logger.org.apache.kafka.clients.consumer.ConsumerConfig=DEBUG
```

Tip

Save `log4j.properties` in `src/main/resources` in your Kafka application's project.

Kafka uses [Simple Logging Facade for Java \(SLF4J\)](#) for logging.

Use `slf4j-simple` library dependency in Scala applications (in `build.sbt`) for basic logging where messages of level `INFO` and higher are printed to `System.err`.

`build.sbt`

```
libraryDependencies += "org.slf4j" % "slf4j-simple" % "1.8.0-alpha2"
```

Tip

Replace slf4j's simple binding to switch between logging frameworks (e.g. `slf4j-log4j12` for log4j).

`build.sbt`

```
val logback = "1.2.3"
libraryDependencies += "ch.qos.logback" % "logback-core" % logback
libraryDependencies += "ch.qos.logback" % "logback-classic" % logback
```

With logback's configuration (as described in the [above tip](#)) you may see the following output:

```
SLF4J: Class path contains multiple SLF4J bindings.  
SLF4J: Found binding in [jar:file:/Users/jacek/.ivy2/cache/org.slf4j/slf4j-logback/1.7.25/jar/slf4j-logback-1.7.25.jar!/org/slf4j/impl/Log4jLoggerFactory.class]  
SLF4J: Found binding in [jar:file:/Users/jacek/.ivy2/cache/ch.qos.logback/logback-classic/1.2.3/jar/logback-classic-1.2.3.jar!/org/slf4j/impl/Log4jLoggerFactory.class]  
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation  
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
```

Note Commenting out `logback-classic` in `build.sbt` resolves it.

`build.sbt`

```
val logback = "1.2.3"  
libraryDependencies += "ch.qos.logback" % "logback-core" % logback  
//libraryDependencies += "ch.qos.logback" % "logback-classic" % logback
```

FIXME: Explain why the commenting out is required?

KafkaClientSupplier Contract — Suppliers of Kafka AdminClient, Consumers and Producers

`KafkaClientSupplier` is the [contract](#) of [Kafka client suppliers](#) that provide (*supply*) Kafka clients, i.e. [AdminClient](#), Kafka Consumers ([getConsumer](#), [getGlobalConsumer](#) and [getRestoreConsumer](#)) and a Kafka [Producer](#).

Table 1. KafkaClientSupplier Contract

Method	Description
<code>getAdminClient</code>	<pre>AdminClient getAdminClient(final Map<String, Object> config)</pre> <p>A Kafka AdminClient (for KafkaStreams) Used exclusively when <code>KafkaStreams</code> is created</p>
<code>getConsumer</code>	<pre>Consumer<byte[], byte[]> getConsumer(final Map<String, Object> config)</pre> <p>A Kafka Consumer (for TaskManager and StreamThread) Used exclusively when <code>StreamThread</code> is created (that also sets the Kafka Consumer for the TaskManager)</p>
<code>getGlobalConsumer</code>	<pre>Consumer<byte[], byte[]> getGlobalConsumer(final Map<String, Object> config)</pre> <p>A Kafka Consumer (for GlobalStreamThread) Used exclusively when <code>KafkaStreams</code> is created (for the only purpose of creating a GlobalStreamThread)</p>
<code>getProducer</code>	<pre>Producer<byte[], byte[]> getProducer(final Map<String, Object> config)</pre> <p>A Kafka Producer (for StreamThreads and TaskCreator) Used when:</p> <ul style="list-style-type: none"> • <code>StreamThread</code> is created (with <code>processing.guarantee</code> as <code>at_least_once</code>) • <code>TaskCreator</code> is requested to createProducer (with <code>processing.guarantee</code> as <code>exactly-once</code>)
<code>getRestoreConsumer</code>	<pre>Consumer<byte[], byte[]> getRestoreConsumer(final Map<String, Object> config)</pre> <p>A Kafka Consumer (for StoreChangelogReader, TaskManager and StreamThread) Used exclusively when <code>StreamThread</code> is created</p>

Note	<p>DefaultKafkaClientSupplier is the one and only known implementation of the KafkaClientSupplier Contract.</p>
------	---

Streams DSL — High-Level Stream Processing DSL

Kafka Streams offers **Streams DSL** with high-level data stream abstractions for streams, tables, state stores, topologies.

Streams DSL includes:

- [StreamsBuilder](#) for defining a topology
- [KStream](#) for working with record streams
- [KTable](#) for working with changelog streams
- [GlobalKTable](#) for working with global changelog streams

Streams DSL also includes the following for streaming aggregations and joins:

- [KGroupedStream](#)
- [KGroupedTable](#)

Streams DSL is designed for Java developers primarily, but also offers [Scala API for Kafka Streams](#) for Scala developers.

StreamsBuilder — The Entry Point to High-Level Streams DSL

`StreamsBuilder` is the entry point to the [Streams DSL — High-Level Stream Processing DSL](#).

`StreamsBuilder` provides the [operators](#) to build a processor topology of [local](#) and [global](#) state stores, [global tables](#), [streams](#), and [tables](#).

Tip	Use Scala API for Kafka Streams to make your Kafka Streams development more pleasant if Scala is the programming language of your choice.
-----	---

Table 1. StreamsBuilder API / Operators

Operator	Description
addGlobalStore	<pre style="font-family: monospace; background-color: #f0f0f0; padding: 5px;"><code>StreamsBuilder addGlobalStore(StoreBuilder storeBuilder, String topic, Consumed consumed, ProcessorSupplier stateUpdateSupplier)</code></pre> <p>Adds a global StateStore (given StoreBuilder, Consumed and ProcessorSupplier) to the topology.</p>
addStateStore	<pre style="font-family: monospace; background-color: #f0f0f0; padding: 5px;"><code>StreamsBuilder addStateStore(StoreBuilder builder)</code></pre> <p>Adds a StateStore to the topology (given StoreBuilder).</p>
build	<pre style="font-family: monospace; background-color: #f0f0f0; padding: 5px;"><code>Topology build() (1) Topology build(Properties props)</code></pre> <p>1. Uses <code>null</code> for the Properties</p> <p>Builds the topology.</p>

globalTable	<pre>GlobalKTable<K, V> globalTable(String topic) GlobalKTable<K, V> globalTable(String topic, Consumed<K, V> consumed) GlobalKTable<K, V> globalTable(String topic, Materialized<K, V, KeyValueStore<Bytes, byte[]>> materialized) GlobalKTable<K, V> globalTable(String topic, Consumed<K, V> consumed, Materialized<K, V, KeyValueStore<Bytes, byte[]>> materialized)</pre>	Creates a GlobalKTable for the given topic (and Consumed , Materialized or both).
stream	<pre>KStream<K, V> stream(Collection<String> topics) KStream<K, V> stream(Collection<String> topics, Consumed<K, V> consumed) KStream<K, V> stream(Pattern topicPattern) KStream<K, V> stream(Pattern topicPattern, Consumed<K, V> consumed) KStream<K, V> stream(String topic) KStream<K, V> stream(String topic, Consumed<K, V> consumed)</pre>	Creates a KStream for the given topic(s) (and Consumed).
table	<pre>KTable<K, V> table(String topic) KTable<K, V> table(String topic, Consumed<K, V> consumed) KTable<K, V> table(String topic, Materialized<K, V, KeyValueStore<Bytes, byte[]>> materialized) KTable<K, V> table(String topic, Consumed<K, V> consumed, Materialized<K, V, KeyValueStore<Bytes, byte[]>> materialized)</pre>	Creates a KTable for the given topic (and Consumed , Materialized or both).

`StreamsBuilder` takes no arguments when created.

```
import org.apache.kafka.streams.StreamsBuilder
val builder = new StreamsBuilder
```

A typical Kafka Streams application (that uses [Streams DSL](#) and [Scala API for Kafka Streams](#)) looks as follows:

```
// Using Scala API for Kafka Streams
import org.apache.kafka.streams.scala._
import ImplicitConversions._
import Serdes._

val builder = new StreamsBuilder

// Add a KStream if needed
// K and V are the types of keys and values, accordingly
builder.stream[K, V](...)

// Add a KTable if needed
builder.table[K, V](...)

// Add a global store if needed
builder.addGlobalStore(...)

// Add a global store if needed
builder.globalTable[K, V](...)

// In the end, build a topology
val topology = builder.build
```

When [created](#), `StreamsBuilder` creates an empty [Topology](#) (that you enrich using the [operators](#)). The topology is immediately requested for the [InternalTopologyBuilder](#) that is in turn used to create an [InternalStreamsBuilder](#).

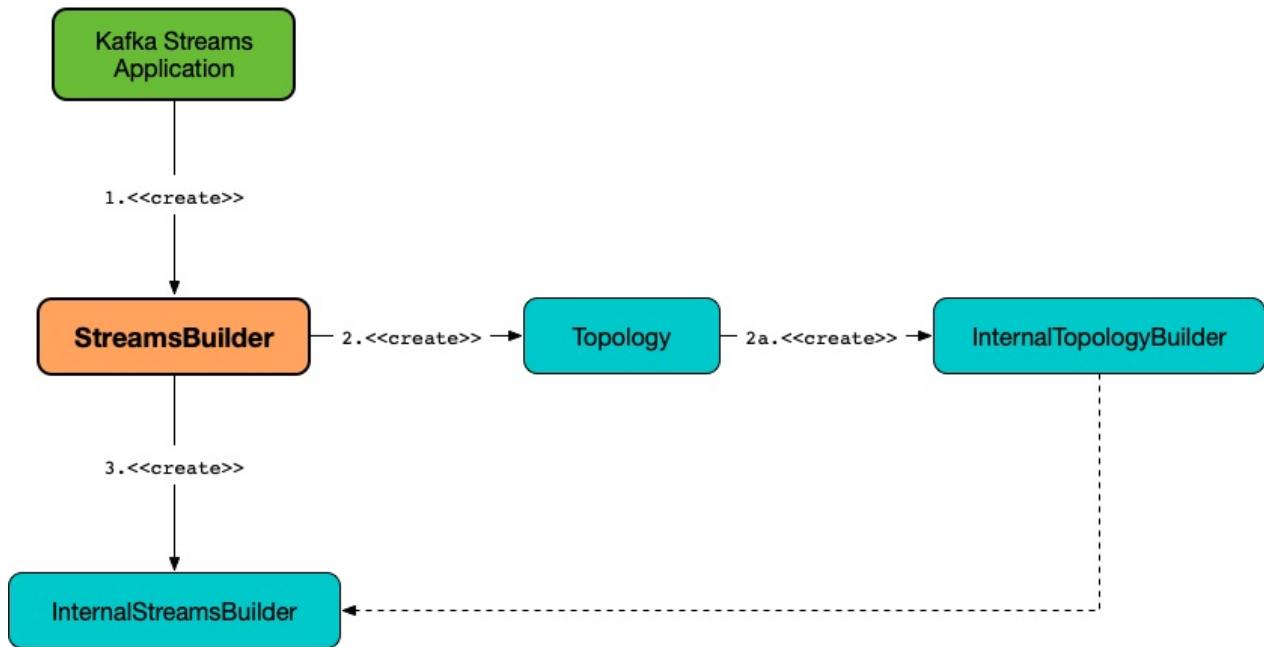


Figure 1. StreamsBuilder, Topology and InternalStreamsBuilder

All [operators](#) use the [InternalStreamsBuilder](#) behind the scenes. In other words, `StreamsBuilder` offers a more [developer-friendly high-level API](#) for developing Kafka Streams applications than using the [InternalStreamsBuilder](#) API directly (and is a façade of `InternalStreamsBuilder`).

Creating KTable for Topic — `table` Method

```

KTable<K, V> table(
    String topic)
KTable<K, V> table(
    String topic,
    Consumed<K, V> consumed)
KTable<K, V> table(
    String topic,
    Materialized<K, V, KeyValueStore<Bytes, byte[]>> materialized)
KTable<K, V> table(
    String topic,
    Consumed<K, V> consumed,
    Materialized<K, V, KeyValueStore<Bytes, byte[]>> materialized)
  
```

`table ...FIXME`

Adding GlobalKTable to Topology — `globalTable` Method

```

GlobalKTable<K, V> globalTable(
    String topic)
GlobalKTable<K, V> globalTable(
    String topic,
    Consumed<K, V> consumed)
GlobalKTable<K, V> globalTable(
    String topic,
    Materialized<K, V, KeyValueStore<Bytes, byte[]>> materialized)
GlobalKTable<K, V> globalTable(
    String topic,
    Consumed<K, V> consumed,
    Materialized<K, V, KeyValueStore<Bytes, byte[]>> materialized)

```

`globalTable` creates an `ConsumedInternal` for the given `Consumed`.

`globalTable` creates a new `MaterializedInternal` (with a new `Materialized` with the `keySerde` and the `valueSerde` of the `consumedInternal`).

Note	The new <code>MaterializedInternal</code> uses <code>KeyValueStore<Bytes, byte[]></code> <code>StateStore</code> .
------	--

`globalTable` requests the `MaterializedInternal` to `generateStoreNameIfNeeded` (with the `InternalStreamsBuilder` and the input `topic name`).

In the end, `globalTable` requests the `InternalStreamsBuilder` to `add a GlobalKTable to the topology` (with the `topic name`, the `ConsumedInternal` and the `MaterializedInternal`).

Demo: Non-queryable GlobalKTable

```
import org.apache.kafka.streams.scala._  
import ImplicitConversions._  
import Serdes._  
  
import org.apache.kafka.streams.scala.StreamsBuilder  
val builder = new StreamsBuilder  
  
val globalTable = builder.globalTable[String, String](topic = "global-table")  
scala> :type globalTable  
org.apache.kafka.streams.kstream.GlobalKTable[String, String]  
  
assert(globalTable.queryableStoreName == null)  
  
val topology = builder.build  
scala> println(topology.describe)  
Topologies:  
  Sub-topology: 0 for global store (will not generate tasks)  
    Source: KTABLE-SOURCE-0000000001 (topics: [global-table])  
      --> KTABLE-SOURCE-0000000002  
    Processor: KTABLE-SOURCE-0000000002 (stores: [global-table-STATE-STORE-0000000000])  
  )  
    --> none  
  <-- KTABLE-SOURCE-0000000001
```

Demo: Queryable GlobalKTable

```

import org.apache.kafka.streams.scala._
import ImplicitConversions._
import Serdes._

import org.apache.kafka.streams.scala.StreamsBuilder
val builder = new StreamsBuilder

import org.apache.kafka.streams.state.Stores
val supplier = Stores.inMemoryKeyValueStore("queryable-store-name")

import org.apache.kafka.streams.scala.kstream.Materialized
val materialized = Materialized.as[String, String](supplier)
val zipCodes = builder.globalTable[String, String](topic = "zip-codes", materialized)

scala> :type zipCodes
org.apache.kafka.streams.kstream.GlobalKTable[String, String]

assert(zipCodes.queryableStoreName == "queryable-store-name")

val topology = builder.build
scala> println(topology.describe)
Topologies:
  Sub-topology: 0 for global store (will not generate tasks)
  Source: KTABLE-SOURCE-0000000000 (topics: [zip-codes])
    --> KTABLE-SOURCE-0000000001
  Processor: KTABLE-SOURCE-0000000001 (stores: [queryable-store-name])
    --> none
    <-- KTABLE-SOURCE-0000000000

```

Registering Global State Store (in Topology)

— addGlobalStore Method

```

StreamsBuilder addGlobalStore(
  StoreBuilder storeBuilder,
  String topic,
  Consumed consumed,
  ProcessorSupplier stateUpdateSupplier)

```

addGlobalStore ...FIXME

addStateStore Method

```

StreamsBuilder addStateStore(StoreBuilder builder)

```

addStateStore ...FIXME

Creating KStream (of Records from One or Many Topics) — `stream` Method

```
KStream<K, V> stream(
    Collection<String> topics)
KStream<K, V> stream(
    Collection<String> topics,
    Consumed<K, V> consumed)
KStream<K, V> stream(
    Pattern topicPattern)
KStream<K, V> stream(
    Pattern topicPattern,
    Consumed<K, V> consumed)
KStream<K, V> stream(
    String topic)
KStream<K, V> stream(
    String topic,
    Consumed<K, V> consumed)
```

`stream` creates a `KStream` (of keys of type `k` and values of type `v`) for the defined topic(s) and the parameters in the input `Consumed`.

```
scala> :type builder
org.apache.kafka.streams.StreamsBuilder

// Create a KStream to read records from the input topic
// Keys and values of the records are of String type
val input = builder.stream[String, String]("input")

scala> :type input
org.apache.kafka.streams.kstream.KStream[String, String]
```

Internally, `stream` creates a `ConsumedInternal` (for the input `Consumed`) and requests the `InternalStreamsBuilder` to create a `KStream` (for the input `topics` and the `ConsumedInternal`).

Building Topology — `build` Method

```
Topology build()
Topology build(Properties props)
```

`build` requests the `InternalStreamsBuilder` to `buildAndOptimizeTopology` (with the given `Properties`) and returns the underlying `topology`.

KStream API — Record Stream

`KStream` is the [abstraction](#) of a **record stream** (of key-value pairs).

`KStream` can be created directly from one or many Kafka topics (using `StreamsBuilder.stream` operator) or as a result of [transformations](#) on an existing `KStream`.

Tip	Use Scala API for Kafka Streams to make your Kafka Streams development more pleasant if Scala is your programming language.
-----	---

```
// Scala API for Kafka Streams
import org.apache.kafka.streams.scala._
import ImplicitConversions._
import Serdes._

val builder = new StreamsBuilder

// Use type annotation to describe the stream, i.e. stream[String, String]
// Else...Scala type inferencer gives us a stream of "nothing", i.e. KStream[Nothing, Nothing]
val input = builder.stream[String, String]("input")

scala> :type input
org.apache.kafka.streams.scala.kstream.KStream[String, String]
```

`KStream` comes with a rich set of [operators](#) (aka *KStream API*) that allow for building topologies to consume, process and produce key-value records.

Table 1. KStream API / Operators

Operator	Description
branch	<code>KStream<K, V>[] branch(Predicate<? super K, ? super V>... predicates)</code>
filter	<code>KStream<K, V> filter(Predicate<? super K, ? super V> predicate)</code>
filterNot	<code>KStream<K, V> filterNot(Predicate<? super K, ? super V> predicate)</code>

flatMap	<pre>KStream<KR, VR> flatMap(KeyValueMapper< ? super K, ? super V, ? extends Iterable<? extends KeyValue<? extends KR, ? extends VR>>> mapper)</pre>
flatMapValues	<pre>KStream<K, VR> flatMapValues(ValueMapper< ? super V, ? extends Iterable<? extends VR>>> mapper) KStream<K, VR> flatMapValues(ValueMapperWithKey< ? super K, ? super V, ? extends Iterable<? extends VR>>> mapper)</pre>
flatTransform	<pre>KStream<K1, V1> flatTransform(TransformerSupplier< ? super K, ? super V, Iterable<KeyValue<K1, V1>>> transformerSupplier, String... stateStoreNames)</pre>
flatTransformValues	<pre>KStream<K, VR> flatTransformValues(ValueTransformerSupplier< ? super V, Iterable<VR>>> valueTransformerSupplier, String... stateStoreNames) KStream<K, VR> flatTransformValues(ValueTransformerWithKeySupplier< ? super K, ? super V, Iterable<VR>>> valueTransformerSupplier, String... stateStoreNames)</pre>
foreach	<pre>void foreach(ForeachAction<? super K, ? super V> action)</pre>
groupBy	<pre>KGroupedStream<KR, V> groupBy(KeyValueMapper<? super K, ? super V, KR> selector) KGroupedStream<KR, V> groupBy(KeyValueMapper<? super K, ? super V, KR> selector, Grouped<KR, V> grouped)</pre>
Creates a KGroupedStream with a given KeyValueMapper	
groupByKey	<pre>KGroupedStream<K, V> groupByKey() KGroupedStream<K, V> groupByKey(Grouped<K, V> grouped)</pre>

	Creates a KGroupedStream
join	<pre>KStream<K, RV> join(GlobalKTable<GK, GV> globalKTable, KeyValueMapper<? super K, ? super V, ? extends GK> keyValueMapper, ValueJoiner<? super V, ? super GV, ? extends RV> joiner) KStream<K, VR> join(KStream<K, VO> otherStream, ValueJoiner<? super V, ? super VO, ? extends VR> joiner, JoinWindows windows) KStream<K, VR> join(KStream<K, VO> otherStream, ValueJoiner<? super V, ? super VO, ? extends VR> joiner, JoinWindows windows, Joined<K, V, VO> joined) KStream<K, VR> join(KTable<K, VT> table, ValueJoiner<? super V, ? super VT, ? extends VR> joiner) KStream<K, VR> join(KTable<K, VT> table, ValueJoiner<? super V, ? super VT, ? extends VR> joiner, Joined<K, V, VT> joined)</pre>
leftJoin	<pre>KStream<K, RV> leftJoin(GlobalKTable<GK, GV> globalKTable, KeyValueMapper<? super K, ? super V, ? extends GK> keyValueMapper, ValueJoiner<? super V, ? super GV, ? extends RV> valueJoiner) KStream<K, VR> leftJoin(KStream<K, VO> otherStream, ValueJoiner<? super V, ? super VO, ? extends VR> joiner, JoinWindows windows) KStream<K, VR> leftJoin(KStream<K, VO> otherStream, ValueJoiner<? super V, ? super VO, ? extends VR> joiner, JoinWindows windows, Joined<K, V, VO> joined) KStream<K, VR> leftJoin(KTable<K, VT> table, ValueJoiner<? super V, ? super VT, ? extends VR> joiner) KStream<K, VR> leftJoin(KTable<K, VT> table, ValueJoiner<? super V, ? super VT, ? extends VR> joiner, Joined<K, V, VT> joined)</pre>
map	<pre>KStream<KR, VR> map(KeyValueMapper< ? super K, ? super V, ? extends KeyValue<? extends KR, ? extends VR>> mapper)</pre>
mapValues	<pre>KStream<K, VR> mapValues(ValueMapper<? super V, ? extends VR> mapper) KStream<K, VR> mapValues(ValueMapperWithKey<? super K, ? super V, ? extends VR> mapper)</pre>

merge	<pre>KStream<K, V> merge(KStream<K, V> stream)</pre>
outerJoin	<pre>KStream<K, VR> outerJoin(KStream<K, VO> otherStream, ValueJoiner<? super V, ? super VO, ? extends VR> joiner, JoinWindows windows) KStream<K, VR> outerJoin(KStream<K, VO> otherStream, ValueJoiner<? super V, ? super VO, ? extends VR> joiner, JoinWindows windows, Joined<K, V, VO> joined)</pre>
peek	<pre>KStream<K, V> peek(ForeachAction<? super K, ? super V> action)</pre>
print	<pre>void print(Printed<K, V> printed)</pre>
process	<pre>void process(ProcessorSupplier<? super K, ? super V> processorSupplier, String... stateStoreNames)</pre>
selectKey	<pre>KStream<KR, V> selectKey(KeyValueMapper< ? super K, ? super V, ? extends KR> mapper)</pre>
through	<pre>KStream<K, V> through(String topic) KStream<K, V> through(String topic, Produced<K, V> produced)</pre>
	<p>Materializes the stream to a given topic (<i>passes it through</i>) and creates a <code>KStream</code> from the topic (using the <code>Produced</code> for configuration)</p>

<p><code>to</code></p>	<pre>void to(String topic) void to(String topic, Produced<K, V> produced) void to(TopicNameExtractor<K, V> topicExtractor) void to(TopicNameExtractor<K, V> topicExtractor, Produced<K, V> produced)</pre> <p>Produces records to a given topic or using dynamic routing based on TopicNameExtractor</p> <table border="1" data-bbox="531 617 1416 707"> <tr> <td data-bbox="531 617 659 707">Note</td><td data-bbox="659 617 1416 707">Topics should be created manually before the Kafka Streams application is started.</td></tr> </table>	Note	Topics should be created manually before the Kafka Streams application is started.
Note	Topics should be created manually before the Kafka Streams application is started.		
<p><code>transform</code></p>	<pre>KStream<K1, V1> transform(TransformerSupplier< ? super K, ? super V, KeyValue<K1, V1>> transformerSupplier, String... stateStoreNames)</pre>		
	<p>Stateful record transformation</p>		
<p><code>transformValues</code></p>	<pre>KStream<K, VR> transformValues(ValueTransformerSupplier< ? super V, ? extends VR> valueTransformerSupplier, String... stateStoreNames) KStream<K, VR> transformValues(ValueTransformerWithKeySupplier< ? super K, ? super V, ? extends VR> valueTransformerSupplier, String... stateStoreNames)</pre> <p>Stateful record-by-record value transformation</p> <p><code>transformValues</code> uses ValueTransformerSupplier to create a ValueTransformer. This is used for a stateful transformation of record values in a stream.</p>		
<p>Note</p> <p>KStreamImpl is the one and only known implementation of the KStream Contract in Kafka Streams.</p>			

KTable API—Changelog Stream

`KTable` is the [abstraction](#) of a [changelog stream](#) from a primary-keyed table. Each record in the changelog stream is an update on the primary-keyed table with the record key as the primary key.

`KTable` assumes that records from the source topic that have `null` keys are simply dropped.

`KTable` can be created directly from a Kafka topic (using [StreamsBuilder.table](#) operator), as a result of [transformations](#) on an existing `KTable`, or aggregations (`aggregate`, `count`, and `reduce`) of the following abstractions:

- [KGroupedStream](#) (`KStream.groupBy` and `KStream.groupByKey` aggregate stream operators)
- [KGroupedTable](#)
- [SessionWindowedKStream](#) (`KGroupedStream.windowedBy` stream operator)
- [TimeWindowedKStream](#) (`KGroupedStream.windowedBy` stream operator with a `TimeWindows` window specification)

Tip

Use [Scala API for Kafka Streams](#) to make your Kafka Streams development more pleasant if Scala is your programming language.

Table 1. KTable API / Operators

Operator	Description
<code>filter</code>	<pre>KTable<K, V> filter(Predicate<? super K, ? super V> predicate) KTable<K, V> filter(Predicate<? super K, ? super V> predicate, Materialized<K, V, KeyValueStore<Bytes, byte[]>> mate rialized)</pre>
<code>filterNot</code>	<pre>KTable<K, V> filterNot(Predicate<? super K, ? super V> predicate) KTable<K, V> filterNot(Predicate<? super K, ? super V> predicate, Materialized<K, V, KeyValueStore<Bytes, byte[]>> mate rialized)</pre>

	<code>KGroupedTable<KR, VR> groupBy(KeyValueMapper<? super K, ? super V, KeyValue<KR, VR> > selector) KGroupedTable<KR, VR> groupBy(KeyValueMapper<? super K, ? super V, KeyValue<KR, VR> > selector, Grouped<KR, VR> grouped)</code>
join	<code>KTable<K, VR> join(KTable<K, V0> other, ValueJoiner<? super V, ? super V0, ? extends VR> join er) KTable<K, VR> join(KTable<K, V0> other, ValueJoiner<? super V, ? super V0, ? extends VR> join er, Materialized<K, VR, KeyValueStore<Bytes, byte[]>> mat erialized)</code>
leftJoin	<code>KTable<K, VR> leftJoin(KTable<K, V0> other, ValueJoiner<? super V, ? super V0, ? extends VR> join er) KTable<K, VR> leftJoin(KTable<K, V0> other, ValueJoiner<? super V, ? super V0, ? extends VR> join er, Materialized<K, VR, KeyValueStore<Bytes, byte[]>> mat erialized)</code>
mapValues	<code>KTable<K, VR> mapValues(ValueMapper<? super V, ? extends VR> mapper) KTable<K, VR> mapValues(ValueMapper<? super V, ? extends VR> mapper, Materialized<K, VR, KeyValueStore<Bytes, byte[]>> mat erialized) KTable<K, VR> mapValues(ValueMapperWithKey<? super K, ? super V, ? extends VR > mapper) KTable<K, VR> mapValues(ValueMapperWithKey<? super K, ? super V, ? extends VR > mapper, Materialized<K, VR, KeyValueStore<Bytes, byte[]>> mat erialized)</code>
outerJoin	<code>KTable<K, VR> outerJoin(KTable<K, V0> other, ValueJoiner<? super V, ? super V0, ? extends VR> join er) KTable<K, VR> outerJoin(KTable<K, V0> other, ValueJoiner<? super V, ? super V0, ? extends VR> join er, Materialized<K, VR, KeyValueStore<Bytes, byte[]>> mat erialized)</code>

queryableStoreName	<code>String queryableStoreName()</code>
suppress	<code>KTable<K, V> suppress(Suppressed<? super K> suppressed)</code>
toStream	<code>KStream<K, V> toStream() KStream<KR, VR> toStream(KeyValueMapper<? super K, ? super V, ? extends KR> mapper)</code>
transformValues	<code>KTable<K, VR> transformValues(ValueTransformerWithKeySupplier<? super K, ? super V, ? extends VR> transformerSupplier, Materialized<K, VR, KeyValueStore<Bytes, byte[]>> materialized, String... stateStoreNames) KTable<K, VR> transformValues(ValueTransformerWithKeySupplier<? super K, ? super V, ? extends VR> transformerSupplier, String... stateStoreNames)</code>

GlobalKTable — Global Changelog Stream

`GlobalKTable` is the [abstraction](#) of [changelog streams](#) that are global and have a local state store registered under the optional [queryableStoreName](#).

Table 1. GlobalKTable Contract

Method	Description
<code>queryableStoreName</code>	<code>String queryableStoreName()</code> Name of the local state store that can be used in interactive queries. If <code>null</code> , the local state store of the <code>GlobalKTable</code> cannot be queried (and so the <code>GlobalKTable</code> is not queryable).
Note	<code>GlobalKTable</code> is an Evolving contract which means that compatibility may be broken at a minor release.
Note	GlobalKTableImpl is the one and only known implementation of GlobalKTable Contract in Kafka Streams.

KGroupedStream — Basic Stream Aggregations

`KGroupedStream` is the [abstraction](#) of a [grouped record stream](#) that allows Kafka Streams developers for [aggregate](#), [count](#), [reduce](#) and [windowedBy](#) stream aggregations.

`KGroupedStream` is the result of the following aggregate stream operators:

- [KStream.groupBy](#)
- [KStream.groupByKey](#)

Tip

Use [Scala API for Kafka Streams](#) to make your Kafka Streams development more pleasant if Scala is your programming language.

```
import org.apache.kafka.streams.scala.StreamsBuilder
import org.apache.kafka.streams.scala._
import ImplicitConversions._
import Serdes._
val builder = new StreamsBuilder
val groupedKStream = builder
  .stream[String, String]("events")
  .groupByKey
scala> :type groupedKStream
org.apache.kafka.streams.scala.kstream.KGroupedStream[String, String]
```

Table 1. KGroupedStream API / Aggregation Operators

Method	Description
aggregate	<pre>KTable<K, VR> aggregate(final Initializer<VR> initializer, final Aggregator<? super K, ? super V, VR> aggregator) KTable<K, VR> aggregate(final Initializer<VR> initializer, final Aggregator<? super K, ? super V, VR> aggregator, final Materialized<K, VR, KeyValueStore<Bytes, byte[]>> materialized)</pre> <p>Creates a KTable with a given <code>Initializer</code>, <code>Aggregator</code> and Materialized KeyValueStore)</p>
count	<pre>KTable<K, Long> count() KTable<K, Long> count(final Materialized<K, Long, KeyValueStore<Bytes, byte[]>> materialized)</pre> <p>Creates a KTable with a given Materialized (view of a KeyValueStore)</p>
reduce	<pre>KTable<K, V> reduce(final Reducer<V> reducer) KTable<K, V> reduce(final Reducer<V> reducer, final Materialized<K, V, KeyValueStore<Bytes, byte[]>> materialized)</pre> <p>Creates a KTable with a given <code>Reducer</code> and Materialized (view of a KeyValueStore)</p>
windowedBy	<pre>SessionWindowedKStream<K, V> windowedBy(final SessionWindows windows)</pre> <p>Creates a SessionWindowedKStream with a given <code>SessionWindows</code> window specification</p>
windowedBy	<pre><W extends Window> TimeWindowedKStream<K, V> windowedBy(final Windows<W> windows)</pre> <p>Creates a TimeWindowedKStream with a given <code>Windows</code> window specification</p>
Note	KGroupedStreamImpl is the one and only known implementation of the KGroupedStream Contract in Kafka Streams 2.3.0.

SessionWindowedKStream — Session-Windowed Stream Aggregations

`SessionWindowedKStream` is the [contract](#) of [KStreams](#) that allows Kafka Streams developers for [aggregate](#), [count](#) and [reduce](#) aggregations over a session-windowed record stream.

`SessionWindowedKStream` is created using [KGroupedStream.windowedBy](#) method.

```
package org.apache.kafka.streams.kstream;

interface SessionWindowedKStream<K, V> {
    <VR> KTable<Windowed<K>, VR> aggregate(
        final Initializer<VR> initializer,
        final Aggregator<? super K, ? super V, VR> aggregator,
        final Merger<? super K, VR> sessionMerger);
    <VR> KTable<Windowed<K>, VR> aggregate(
        final Initializer<VR> initializer,
        final Aggregator<? super K, ? super V, VR> aggregator,
        final Merger<? super K, VR> sessionMerger,
        final Materialized<K, VR, SessionStore<Bytes, byte[]>> materialized);
    KTable<Windowed<K>, Long> count();
    KTable<Windowed<K>, Long> count(
        final Materialized<K, Long, SessionStore<Bytes, byte[]>> materialized);
    KTable<Windowed<K>, V> reduce(final Reducer<V> reducer);
    KTable<Windowed<K>, V> reduce(
        final Reducer<V> reducer,
        final Materialized<K, V, SessionStore<Bytes, byte[]>> materializedAs);
}
```

Table 1. SessionWindowedKStream Contract

Method	Description
aggregate	
count	
reduce	

Note	SessionWindowedKStreamImpl is the one and only known implementation of SessionWindowedKStream Contract in Kafka Streams 2.3.0.
------	--

TimeWindowedKStream — Time-Windows Streaming Aggregations

`TimeWindowedKStream` is the [abstraction](#) of a [windowed record stream](#) that allows Kafka Streams developers for [aggregate](#), [count](#), [reduce](#) aggregations.

`TimeWindowedKStream` is the result of [KGroupedStream.windowedBy](#) stream operator with a [TimeWindows](#) window specification.

Tip

Use [Scala API for Kafka Streams](#) to make your Kafka Streams development more pleasant if Scala is your programming language.

```
import org.apache.kafka.streams.kstream.TimeWindows
import scala.concurrent.duration._
val everyMinute = TimeWindows.of(1.minute.toMillis)

import org.apache.kafka.streams.scala.StreamsBuilder
import org.apache.kafka.streams.scala._
import ImplicitConversions._
import Serdes._
val builder = new StreamsBuilder
val windowedKStream = builder
    .stream[String, String]("events")
    .groupByKey
    .windowedBy(everyMinute)
scala> :type windowedKStream
org.apache.kafka.streams.scala.kstream.TimeWindowedKStream[String, String]

// Print out counts over 1-minute time windows to stdout
import org.apache.kafka.streams.kstream.Printed
import org.apache.kafka.streams.kstream.Windowed
val stdout = Printed
    .toSysOut[Windowed[String], Long]
    .withLabel("[time-windows]")
windowedKStream
    .count
    .toStream
    .print(stdout)

// Use println(builder.build.describe) to see the whole topology
```

Table 1. TimeWindowedKStream API / Windowed Operators

Method	Description
aggregate	<pre>KTable<Windowed<K>, VR> aggregate(final Initializer<VR> initializer, final Aggregator<? super K, ? super V, VR> aggregator) KTable<Windowed<K>, VR> aggregate(final Initializer<VR> initializer, final Aggregator<? super K, ? super V, VR> aggregator, final Materialized<K, VR, WindowStore<Bytes, byte[]>> materialized)</pre> <p>Creates a KTable with a given <code>Initializer</code>, <code>Aggregator</code> and <code>Materialized</code> (view of a WindowStore)</p>
count	<pre>KTable<Windowed<K>, Long> count() KTable<Windowed<K>, Long> count(final Materialized<K, Long, WindowStore<Bytes, byte[]>> materialized)</pre> <p>Creates a KTable with a given <code>Materialized</code> (view of a WindowStore)</p>
reduce	<pre>KTable<Windowed<K>, V> reduce(final Reducer<V> reducer) KTable<Windowed<K>, V> reduce(final Reducer<V> reducer, final Materialized<K, V, WindowStore<Bytes, byte[]>> materialized)</pre> <p>Combining record stream (by a grouped key)</p> <p>Creates a KTable with a given <code>Reducer</code> and <code>Materialized</code> (view of a WindowStore)</p>
Note	<p>TimeWindowedKStreamImpl is the one and only known implementation of the TimeWindowedKStream Contract in Kafka Streams 2.3.0.</p>

KGroupedTable Contract

KGroupedTable is...FIXME

Consumed — Metadata for Consuming Streams

`Consumed` provides the optional parameters that describe how to consume streams in the [High-Level Stream Processing DSL](#):

- **Key Serde** (Apache Kafka's [Serde](#) for record keys)
- **Value Serde** (Apache Kafka's [Serde](#) for record values)
- [TimestampExtractor](#)
- **Offset reset policy** ([Topology.AutoOffsetReset](#))

`Consumed` is used as an optional parameter in the following `streamsBuilder` operators:

- [StreamsBuilder.stream](#)
- [StreamsBuilder.table](#)
- [StreamsBuilder.globalTable](#)
- [StreamsBuilder.addGlobalStore](#)

A `consumed` instance is [created](#) using one of the available [with](#) factory methods.

```
Consumed<K, V> with(
    final Serde<K> keySerde,
    final Serde<V> valueSerde)
Consumed<K, V> with(final TimestampExtractor timestampExtractor)
Consumed<K, V> with(final Topology.AutoOffsetReset resetPolicy)
Consumed<K, V> with(
    final Serde<K> keySerde,
    final Serde<V> valueSerde,
    final TimestampExtractor timestampExtractor,
    final Topology.AutoOffsetReset resetPolicy)
```

A `consumed` instance can further be configured using the "with" methods.

Table 1. Consumed's "with" Methods

Method	Description
withKeySerde	<code>Consumed<K, V> withKeySerde(final Serde<K> keySerde)</code>
withOffsetResetPolicy	<code>Consumed<K, V> withOffsetResetPolicy(final Topology.AutoOffsetReset resetPolicy)</code>
withTimestampExtractor	<code>Consumed<K, V> withTimestampExtractor(final TimestampExtractor timestampExtractor)</code>
WithValueSerde	<code>Consumed<K, V> withValueSerde(final Serde<V> valueSerde)</code>

Scala API for Kafka Streams

Scala API for Kafka Streams makes the optional `Consumed` metadata an implicit parameter in the `StreamsBuilder` API.

Moreover, `ImplicitConversions` object defines `consumedFromSerde` implicit method that creates a `Consumed` instance with the key and value `Serde` objects available in implicit scope.

And the last but not least, Scala API for Kafka Streams defines `Consumed` object with ``with`` factory methods that use implicit key and value `Serde` objects.

Produced — Metadata for Producing Streams

`Produced` provides the optional parameters that describe how to produce streams in the [High-Level Stream Processing DSL](#):

- **Key Serde** (Apache Kafka's [Serde](#) for record keys)
- **Value Serde** (Apache Kafka's [Serde](#) for record values)
- [StreamPartitioner](#)

`Produced` is used as an optional parameter in the following `kstream` operators:

- [KStream.through](#)
- [KStream.to](#)

A `Produced` instance is [created](#) using one of the available [factory methods](#).

```
Produced<K, V> keySerde(  
    Serde<K> keySerde)  
Produced<K, V> streamPartitioner(  
    StreamPartitioner<? super K, ? super V> partitioner)  
Produced<K, V> valueSerde(  
    Serde<V> valueSerde)  
Produced<K, V> with(  
    Serde<K> keySerde,  
    Serde<V> valueSerde)  
Produced<K, V> with(  
    Serde<K> keySerde,  
    Serde<V> valueSerde,  
    StreamPartitioner<? super K, ? super V> partitioner)
```

A `Produced` instance can further be configured using the "[with](#)" methods.

Table 1. Produced's "with" Methods

Method	Description
withKeySerde	<code>Produced<K, V> withKeySerde(final Serde<K> keySerde)</code>
withStreamPartitioner	<code>Produced<K, V> withStreamPartitioner(final StreamPartitioner<? super K, ? super V> partitioner)</code>
WithValueSerde	<code>Produced<K, V>WithValueSerde(final Serde<V> valueSerde)</code>

Scala API for Kafka Streams

Scala API for Kafka Streams makes the optional `Produced` metadata an implicit parameter in the `KStream` API.

Moreover, `ImplicitConversions` object defines `producedFromSerde` implicit method that creates a `Produced` instance with the key and value `Serde` objects available in implicit scope.

And the last but not least, Scala API for Kafka Streams defines `Produced` object with ``with`` factory methods that use implicit key and value `Serde` objects.

Grouped — Metadata for Aggregating Streams

`Grouped` provides the optional parameters that describe how to aggregate streams in the [High-Level Stream Processing DSL](#):

- **Repartition topic name**
- **Key Serde** (Apache Kafka's [Serde](#) for record keys)
- **Value Serde** (Apache Kafka's [Serde](#) for record values)

`Grouped` is used as an optional parameter in the following streaming aggregation operators:

- [KStream.groupByKey](#)
- [KStream.groupBy](#)
- [KTable.groupBy](#)

A `Grouped` instance is [created](#) using one of the [factory methods](#):

```
static <K, V> Grouped<K, V> as(final String name)
static <K> Grouped keySerde(final Serde<K> keySerde)
static <V> Grouped valueSerde(final Serde<V> valueSerde)
static <K, V> Grouped<K, V> with(
    final Serde<K> keySerde,
    final Serde<V> valueSerde)
static <K, V> Grouped<K, V> with(
    final String name,
    final Serde<K> keySerde,
    final Serde<V> valueSerde)
```

A `Grouped` instance can further be configured using the ["with"](#) methods.

Table 1. Grouped's "with" Methods

Method	Description
<code>withKeySerde</code>	<code>Grouped<K, V> withKeySerde(final Serde<K> keySerde)</code>
<code>withName</code>	<code>Grouped<K, V> withName(final String name)</code>
<code>WithValueSerde</code>	<code>Grouped<K, V> withValueSerde(final Serde<V> valueSerde)</code>

Scala API for Kafka Streams

Scala API for Kafka Streams makes the optional `Grouped` metadata an implicit parameter in the `KStream` and `KTable` APIs.

Moreover, `ImplicitConversions` object defines `groupedFromSerde` implicit conversion that creates a `Grouped` instance with the key and value `Serde` objects available in implicit scope.

And the last but not least, Scala API for Kafka Streams defines `Grouped` object with ``with`` factory methods that use implicit key and value `Serde` objects.

Joined — Metadata for Joined Streams

Joined is...FIXME

Printed

Printed is...FIXME

Table 1. Printed Operators

Operator	Description
withKeyValueMapper	Printed<K, V> withKeyValueMapper(final KeyValueMapper<? super K, ? super V, String> mapper)

withKeyValueMapper Method

```
Printed<K, V> withKeyValueMapper(final KeyValueMapper<? super K, ? super V, String> mapper)
```

withKeyValueMapper ...FIXME

Note	withKeyValueMapper is used when...FIXME
------	---

KeyValueMapper

`KeyValueMapper` is the [contract](#) of **key-value mappers** that [map a record to a new value](#).

`KeyValueMapper` is the input argument of the following operators:

- [KStream.selectKey](#)
- [KStream.map](#)
- [KStream.flatMap](#)
- [KStream.groupBy](#)
- [KStream.join](#)
- [KStream.leftJoin](#)
- [KTable.toStream](#)
- [KTable.groupBy](#)
- [Printed.withKeyValueMapper](#)

```
package org.apache.kafka.streams.kstream;

interface KeyValueMapper<K, V, VR> {
    VR apply(final K key, final V value);
}
```

Table 1. `KeyValueMapper` Contract

Method	Description
<code>apply</code>	Used when...FIXME

ValueJoiner

`ValueJoiner` is...FIXME

ValueTransformer — Stateful Record-by-Record Value Transformation

`ValueTransformer` is the [contract](#) in Kafka Streams for **stateful mapping** of a value to a new value of an arbitrary type.

```
package org.apache.kafka.streams.kstream;

interface ValueTransformer<V, VR> {
    void close();
    void init(final ProcessorContext context);
    VR transform(final V value);
    // deprecated
    VR punctuate(final long timestamp);
}
```

`ValueTransformer` is used when a stream (`AbstractStream`) is requested to `toInternalValueTransformerSupplier`.

Table 1. ValueTransformer Contract

Method	Description
<code>close</code>	Used when...FIXME
<code>init</code>	Used when...FIXME
<code>transform</code>	Used when...FIXME
<code>punctuate</code>	DEPRECATED Use Punctuator interface

ValueTransformerSupplier — ValueTransformers Object Factory

`ValueTransformerSupplier` is a [single-method interface](#) in Kafka Streams for **object factories** (aka *suppliers*) that [can create one or many ValueTransformers](#).

```
package org.apache.kafka.streams.kstream;

interface ValueTransformerSupplier<V, VR> {
    ValueTransformer<V, VR> get();
}
```

`ValueTransformerSupplier` is used exclusively in [KStream.transformValues](#) stateful transformation (of record values in a stream) to get a new [ValueTransformer](#) where the transformation happens.

```
FIXME Example of KStream.transformValues
```

```
get ...FIXME
```

ValueTransformerWithKey

ValueTransformerWithKey is...FIXME

Transformer — Record-by-Record Stateful Transformation

`Transformer` is the [contract](#) that allows for a [stateful transformation](#) of every record in the input stream into zero or more output records (both key and value type can be altered arbitrarily).

In order to emit (produce) no record, `transform` should simply return `null`.

In order to use a state, a state store has to be registered beforehand (using `addStateStore` or `addGlobalStore`) before they can be connected to the `Transformer`.

You can access a state store using the [ProcessorContext](#) that is given when a transformer is [initialized](#).

Tip

You could emit more output records for a single record using the `ProcessorContext` and its [forward](#) method.

```
// FIXME: Demo
```

`Transformer` is created (*supplied*) using a corresponding [TransformerSupplier](#).

Property	Description
<code>close</code>	<pre data-bbox="609 249 806 280"><code>void close()</code></pre> <p>Closes the transformer Used exclusively when <code>KStreamTransformProcessor</code> is requested to close</p>
<code>init</code>	<pre data-bbox="632 541 1203 572"><code>void init(final ProcessorContext context)</code></pre> <p>Initializes the transformer in the given ProcessorContext Used exclusively when <code>KStreamTransformProcessor</code> is requested to initialize itself</p>
<code>transform</code>	<pre data-bbox="632 846 1176 878"><code>R transform(final K key, final V value)</code></pre> <p>Transforms a single record (into a <code>KeyValue</code> object) In order to emit (produce) no record, <code>transform</code> should return <code>null</code>. Used exclusively when <code>KStreamTransformProcessor</code> is requested to process a record</p>

`Transformer` is used to create a [KStreamTransformProcessor](#).

TransformerSupplier

TransformerSupplier is...FIXME

Windows — Window Specification For Time-Windowed Streaming Aggregations

`Windows` is the abstraction of window specifications that describe windows of fixed size and grace period.

`Windows` are used in `KGroupedStream.windowedBy` streaming operator (for time-windowed streaming aggregations).

Table 1. Windows API / Operators

Method	Description
<code>gracePeriodMs</code>	<pre>long gracePeriodMs()</pre> <p>Grace period (in milliseconds) New events are added to windows until their grace period ends</p>
<code>size</code>	<pre>long size()</pre> <p>Window size (in milliseconds)</p>
<code>windowsFor</code>	<pre>Map<Long, W> windowsFor(final long timestamp)</pre> <p>Returns all the windows that contain the given timestamp, indexed by non-negative window start timestamps. Used when <code>KStreamWindowAggregateProcessor</code> is requested to process a record</p>

Table 2. Windows

Windows	Description
<code>JoinWindows</code>	
<code>TimeWindows</code>	Time-bound window specification for tumbling or hopping windows
<code>UnlimitedWindows</code>	

`Windows` takes no arguments when created and simply initializes the internal registries and counters.

Note

`Windows` is a Java abstract class and cannot be created directly. It is created indirectly when the concrete Windows are.

`Windows` defines a **window maintain duration** (aka **retention time**). The default window maintain duration is `1 day`.

JoinWindows — Window Specification for Streaming Joins

JoinWindows is a [window specification](#) that is used for streaming joins.

TimeWindows — Time-Bound Window Specification

TimeWindows is a [window specification of time windows](#).

TimeWindows is described by the following properties:

- **Window duration** (aka *window size*) (in millis)
- **Advance interval** (in millis)
- **Grace period** for late events
- **Maintain duration** (in millis)

Note

Maintain duration is no longer in use.

TimeWindows can be created only using the [of](#) factory method.

```
static TimeWindows of(final Duration size)
```

of is used to create a time specification of **tumbling windows** which are fixed-sized, gapless, non-overlapping windows (and simply sets the [sizeMs](#) and [advanceMs](#) internal properties to the given value).

```
import org.apache.kafka.streams.kstream.TimeWindows
import java.time.Duration
val timeWindows = TimeWindows.of(Duration.ofMinutes(1))
scala> println(timeWindows)
TimeWindows{maintainDurationMs=86400000, sizeMs=60000, advanceMs=60000, grace=null, segments=3}
```

TimeWindows can be further configured using the [advanceBy](#) and [grace](#) methods.

```
TimeWindows advanceBy(final Duration advance)
```

advanceBy allows for a time specification of **hopping windows** which are fixed-sized, overlapping windows (and simply sets the [advanceMs](#) internal property).

```
import org.apache.kafka.streams.kstream.TimeWindows
import java.time.Duration
val timeWindows = TimeWindows
.of(Duration.ofMinutes(1))
.advanceBy(Duration.ofSeconds(30))
scala> println(timeWindows)
TimeWindows{maintainDurationMs=86400000, sizeMs=60000, advanceMs=30000, grace=null, segments=3}
```

```
TimeWindows grace(final Duration afterWindowEnd)
```

`grace` specifies how long to wait for **late events** to be included in a time window (and simply sets the `grace` internal property).

```
import org.apache.kafka.streams.kstream.TimeWindows
import java.time.Duration
val timeWindows = TimeWindows
.of(Duration.ofMinutes(1))
.grace(Duration.ofMinutes(2)) // 1 minute late after a time window has elapsed
scala> println(timeWindows)
TimeWindows{maintainDurationMs=86400000, sizeMs=60000, advanceMs=60000, grace=PT2M, segments=3}
```

windowsFor Method

```
Map<Long, TimeWindow> windowsFor(final long timestamp)
```

Note

`windowsFor` is part of the [Windows Contract](#) to...FIXME.

`windowsFor` ...FIXME

```
import collection.JavaConverters.-
val windows = timeWindows.windowsFor(1.minute.toMillis).asScala
scala> windows.foreach(println)
(30000,Window{start=30000, end=90000})
(60000,Window{start=60000, end=120000})
```

UnlimitedWindows

UnlimitedWindows is a window specification of UnlimitedWindows.

Window

`Window` is the base of single window instances that are described by the `start` and `end` timestamps and can be checked if they overlap.

```
package org.apache.kafka.streams.kstream;

abstract class Window {
    // only required methods that have no implementation
    // the others follow
    public abstract boolean overlap(final Window other);
}
```

Table 1. (Subset of) Window Contract

Method	Description
<code>overlap</code>	Checks whether a window overlaps with another.

Use `start` method to access the `start timestamp` of a window.

```
long start()
```

Use `end` method to access the `end timestamp` of a window.

```
long end()
```

`Window` has the following text representation:

```
Window{start=[startMs], end=[endMs]}
```

Table 2. Windows

Window	Description
<code>SessionWindow</code>	
<code>TimeWindow</code>	
<code>UnlimitedWindow</code>	

Creating Window Instance

`Window` takes the following when created:

- The start timestamp of the window (in milliseconds)
- The end timestamp of the window (in milliseconds)

`Window` initializes the [internal registries and counters](#).

`Window` is [created](#) when...MEFIXME

`Window` is [created](#) along with...MEFIXME

Note

`Window` is a Java/Scala abstract class and cannot be [created](#) directly. It is created indirectly when the [concrete FIXMEs](#) are.

WindowedSerdes — SessionWindowedSerde and TimeWindowedSerde

`WindowedSerdes` is a class that acts as a namespace for two static classes:

- `SessionWindowedSerde` for session-windowed aggregations
- `TimeWindowedSerde` for time-windowed aggregation

`WindowedSerdes` also defines two [factory methods](#) to create instances of `TimeWindowedSerde` and `SessionWindowedSerde` for the specified inner class type.

Table 1. WindowedSerdes's Factory Methods

Method	Description
<code>sessionWindowedSerdeFrom</code>	<code>Serde<Windowed<T>> sessionWindowedSerdeFrom(final Class<T> type)</code>
<code>timeWindowedSerdeFrom</code>	<code>Serde<Windowed<T>> timeWindowedSerdeFrom(final Class<T> type)</code>

Scala API for Kafka Streams

`Serdes` object defines the following implicit conversions to create instances of `TimeWindowedSerde` and `SessionWindowedSerde` (without an explicit inner class type):

- `sessionWindowedSerde` for a `WindowedSerdes.SessionWindowedSerde[T]` instance
- `timeWindowedSerde` for a `WindowedSerdes.TimeWindowedSerde[T]` instance

Windowed

`Windowed` is the result key type of a windowed stream aggregation.

`Windowed` is created when:

- `KStreamSessionWindowAggregateProcessor` and `KStreamWindowAggregateProcessor` processors are requested to process a record
- `CachingSessionStore` state store is requested to `putAndMaybeForward`
- `MergedSortedCacheSessionStoreIterator` is requested to `deserializeCacheKey`
- `MeteredSessionStore` is requested to `remove a session` and `store an aggregated value for a session`
- `MeteredWindowedKeyValueIterator` is requested to `windowedKey`
- `SessionKeySchema` is requested to...FIXME
- `WindowKeySchema` is requested to...FIXME
- `WindowStoreIteratorWrapper` is requested to...FIXME

`Windowed` takes the following when created:

- Key of `k` type
- `Window`

TimeWindowedSerializer

`TimeWindowedSerializer` is a [WindowedSerializer](#) for the keys of `WindowedSerdes.TimeWindowedSerde` and `FullTimewindowedSerde` windowed serdes.

`TimeWindowedSerializer` takes a Kafka [Serializer](#) when created.

When requested to configure, `TimeWindowedSerializer` uses `DEFAULT_WINDOWED_KEY_SERDE_INNER_CLASS` and `DEFAULT_WINDOWED_VALUE_SERDE_INNER_CLASS` configuration properties for the serde of keys and values, respectively, unless the [Serializer](#) is defined already.

Scala API for Kafka Streams

Scala API for Kafka Streams is a separate Kafka Streams module (a Scala library) that acts as a wrapper over the existing Java API for Kafka Streams.

The Scala API is available in `org.apache.kafka.streams.scala` package.

As a separate Scala library you have to define the dependency in `build.sbt`.

```
// Note two percent signs (%%) to encode Scala version
libraryDependencies += "org.apache.kafka" %% "kafka-streams-scala" % "2.3.0"
```

The Scala API for Kafka Streams defines Scala-friendly types that wrap the corresponding Kafka Streams types and simply delegate all method calls to the underlying Java object with the purpose of making it much more expressive, with less boilerplate and more succinct.

- [StreamsBuilder](#)
- [KGroupedStream](#)
- [KGroupedTable](#)
- [KStream](#)
- [KTable](#)
- [SessionWindowedKStream](#)
- [TimeWindowedKStream](#)

Beside the Scala-friendly types, the Scala API for Kafka Streams defines implicit conversions, i.e. [Serdes](#), and [ImplicitConversions](#).

```
import org.apache.kafka.streams.scala._
import ImplicitConversions._
import Serdes._
```

The Scala API for Kafka Streams comes with [Consumed](#) Scala object that allows for creating [Consumed](#) instances with key and value `Serdes` objects available in implicit scope.

You could also use `Serdes` when defining `StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG` and `StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG`.

```
import org.apache.kafka.streams.scala.Serdes._  
Serdes.String.getClass.getName
```

Note

Scala API for Kafka Streams was added in Kafka Streams 2.0.0 as [KAFKA-6670 Implement a Scala wrapper library for Kafka Streams.](#)

ImplicitConversions

`ImplicitConversions` is a Scala object that contains the [implicit conversions](#) between the Scala wrapper objects and the underlying Java objects.

You should import `ImplicitConversions` to use the implicit conversions in a Scala application that uses the Kafka Streams library.

```
import org.apache.kafka.streams.scala.ImplicitConversions._
```

Table 1. Implicit Conversions

Name	Description
<code>consumedFromSerde</code>	<pre>consumedFromSerde[K, V](implicit keySerde: Serde[K], valueSerde: Serde[V]): Consumed[K, V]</pre> <p>Creates a Consumed instance</p>
<code>groupedFromSerde</code>	<pre>groupedFromSerde[K, V](implicit keySerde: Serde[K], valueSerde: Serde[V]): Grouped[K, V]</pre> <p>Creates a Grouped instance</p>
<code>joinedFromKeyValueOtherSerde</code>	<pre>joinedFromKeyValueOtherSerde[K, V, VO](implicit keySerde: Serde[K], valueSerde: Serde[V], otherValueSerde: Serde[VO]): Joined[K, V, VO]</pre> <p>Creates a Joined instance</p>
<code>materializedFromSerde</code>	<pre>materializedFromSerde[K, V, S <: StateStore](implicit keySerde: Serde[K], valueSerde: Serde[V]): Materialized[K, V, S]</pre> <p>Creates a Materialized instance</p>
<code>producedFromSerde</code>	<pre>producedFromSerde[K, V](implicit keySerde: Serde[K], valueSerde: Serde[V]): Produced[K, V]</pre> <p>Creates a Produced instance</p>

<code>tuple2ToKeyValue</code>	<code>tuple2ToKeyValue[K, V](tuple: (K, V)): KeyValue[K, V]</code> Creates a <code>KeyValue</code> instance
<code>wrapKGroupedStream</code>	<code>wrapKGroupedStream[K, V](inner: KGroupedStreamJ[K, V])</code> Extends (<i>wraps</i>) a Java-based <code>KGroupedStream</code> with Scala API
<code>wrapKGroupedTable</code>	<code>wrapKGroupedTable[K, V](inner: KGroupedTableJ[K, V]): KGroupedTable[K, V]</code> Extends (<i>wraps</i>) a Java-based <code>KGroupedTable</code> with Scala API
<code>wrapKStream</code>	<code>wrapKStream[K, V](inner: KStreamJ[K, V]): KStream[K, V]</code> Extends (<i>wraps</i>) a Java-based <code>KStream</code> with Scala API
<code>wrapKTable</code>	<code>wrapKTable[K, V](inner: KTableJ[K, V]): KTable[K, V]</code> Extends (<i>wraps</i>) a Java-based <code>KTable</code> with Scala API
<code>wrapSessionWindowedKStream</code>	<code>wrapSessionWindowedKStream[K, V](inner: SessionWindowedKStreamJ[K, V]): SessionWindowedKStream[K, V]</code> Extends (<i>wraps</i>) a Java-based <code>SessionWindowedKStream</code> with Scala API
<code>wrapTimeWindowedKStream</code>	<code>wrapTimeWindowedKStream[K, V](inner: TimeWindowedKStreamJ[K, V]): TimeWindowedKStream[K, V]</code> Extends (<i>wraps</i>) a Java-based <code>TimeWindowedKStream</code> with Scala API

Serd़es

Serd़es is a Scala object that contains the implicit conversions with serializers and deserializers for known Scala and Java types and fromFn to create a new Serde[T] (from T or (String, T)).

You should import serdes to use the implicit conversions in a Scala application that uses the Kafka Streams library.

```
import org.apache.kafka.streams.scala.Serde._
```

You should also use Serdes when defining StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG and StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG .

```
import org.apache.kafka.streams.scala.Serde._  
Serdes.String.getClass.getName
```

Table 1. Implicit Conversions

Name	Serde
ByteArray	Serde[Array[Byte]]
Bytes	Serde[org.apache.kafka.common.utils.Bytes]
Double	Serde[Double]
Float	Serde[Float]
Integer	Serde[Int]
JavaDouble	Serde[java.lang.Double]
JavaFloat	Serde[java.lang.Float]
JavaInteger	Serde[java.lang.Integer]
JavaLong	Serde[java.lang.Long]
Long	Serde[Long]
String	Serde[String]
sessionWindowedSerde[T]	WindowedSerdes.SessionWindowedSerde[T]
timeWindowedSerde[T]	WindowedSerdes.TimeWindowedSerde[T]

fromFn Method

```
fromFn[T >: Null](
  serializer: T => Array[Byte],
  deserializer: Array[Byte] => Option[T]): Serde[T]
fromFn[T >: Null](
  serializer: (String, T) => Array[Byte],
  deserializer: (String, Array[Byte]) => Option[T]): Serde[T]
```

fromFn ...FIXME

Note	fromFn is used when...FIXME
------	-----------------------------

Consumed

`Consumed` Scala object is part of [Scala API for Kafka Streams](#) that defines `with` factory methods for creating `Consumed` instances with key and value `Serde` objects available in implicit scope.

Consumed's "with" Factory Methods

```
// Note the backticks to use "with" reserved keyword
// ConsumedJ is simply an import alias for the Java-aware Consumed

`with`[K, V](
  timestampExtractor: TimestampExtractor,
  resetPolicy: Topology.AutoOffsetReset
)(implicit keySerde: Serde[K], valueSerde: Serde[V]): ConsumedJ[K, V]

`with`[K, V](
  implicit keySerde: Serde[K], valueSerde: Serde[V]): ConsumedJ[K, V]

`with`[K, V](timestampExtractor: TimestampExtractor)(
  implicit keySerde: Serde[K], valueSerde: Serde[V]): ConsumedJ[K, V]

`with`[K, V](resetPolicy: Topology.AutoOffsetReset)(
  implicit keySerde: Serde[K], valueSerde: Serde[V]): ConsumedJ[K, V]
```

Example: Creating Consumed Instance using Scala API for Kafka Streams

```
import org.apache.kafka.streams.scala._
import ImplicitConversions._
import Serdes._

import org.apache.kafka.streams.scala.kstream.Consumed
val consumed = Consumed.`with`[Long, String]

scala> :type consumed
org.apache.kafka.streams.kstream.Consumed[Long, String]
```

Tip	Read up on Implicit Parameters .
-----	--

Produced

Produced Scala object is part of [Scala API for Kafka Streams](#) that defines `with` factory methods for creating [Produced](#) instances with key and value `Serde` objects available in implicit scope.

Produced's "with" Factory Methods

```
// Note the backticks to use "with" reserved keyword
// ProducedJ is simply an import alias for the Java-aware Produced

`with`[K, V](
  implicit keySerde: Serde[K], valueSerde: Serde[V]): ProducedJ[K, V]

`with`[K, V](partitioner: StreamPartitioner[K, V])(
  implicit keySerde: Serde[K], valueSerde: Serde[V]): ProducedJ[K, V]
```

Example: Creating Produced Instance using Scala API for Kafka Streams

```
import org.apache.kafka.streams.scala._
import ImplicitConversions._
import Serdes._

import org.apache.kafka.streams.scala.kstream.Produced
val produced = Produced.`with`[Long, String]

scala> :type produced
org.apache.kafka.streams.kstream.Produced[Long, String]
```

Tip	Read up on Implicit Parameters .
-----	--

Grouped

`Grouped` Scala object is part of [Scala API for Kafka Streams](#) that defines `with` factory methods for creating `Grouped` instances with key and value `serdes` objects available in implicit scope.

Grouped's "with" Factory Methods

```
// Note the backticks to use "with" reserved keyword
// GroupedJ is simply an import alias for the Java-aware Grouped

`with`[K, V](
  implicit keySerde: Serde[K], valueSerde: Serde[V]): GroupedJ[K, V]

`with`[K, V](name: String)(
  implicit keySerde: Serde[K], valueSerde: Serde[V]): GroupedJ[K, V]
```

Example: Creating Grouped Instance using Scala API for Kafka Streams

```
import org.apache.kafka.streams.scala._
import ImplicitConversions._
import Serdes._

import org.apache.kafka.streams.scala.kstream.Grouped
val grouped = Grouped.`with`[Long, String]

scala> :type grouped
org.apache.kafka.streams.kstream.Grouped[Long, String]
```

Tip	Read up on Implicit Parameters .
-----	--

Materialized

`Materialized` Scala object is part of [Scala API for Kafka Streams](#) that defines `with` and `as` factory methods for creating `Materialized` instances with key and value `Serde` objects available in implicit scope.

Materialized's Factory Method

```
// Note the backticks to use "with" reserved keyword
// MaterializedJ is simply an import alias for the Java-aware Materialized
// ByteArray*Store types are type aliases for *Store[Bytes, Array[Byte]]
// e.g. ByteArrayWindowStore = WindowStore[Bytes, Array[Byte]]

`with`[K, V, S <: StateStore](
  implicit keySerde: Serde[K], valueSerde: Serde[V]): MaterializedJ[K, V, S]

as[K, V, S <: StateStore](storeName: String)(
  implicit keySerde: Serde[K], valueSerde: Serde[V]): MaterializedJ[K, V, S]

as[K, V](supplier: KeyValueBytesStoreSupplier)(
  implicit keySerde: Serde[K], valueSerde: Serde[V]): MaterializedJ[K, V, ByteArrayKeyValueStore]

as[K, V](supplier: SessionBytesStoreSupplier)(
  implicit keySerde: Serde[K], valueSerde: Serde[V]): MaterializedJ[K, V, ByteArraySessionStore]

as[K, V](supplier: WindowBytesStoreSupplier)(
  implicit keySerde: Serde[K], valueSerde: Serde[V]): MaterializedJ[K, V, ByteArrayWindowStore]
```

Example: Creating Materialized Instance using Scala API for Kafka Streams

```
import org.apache.kafka.streams.scala._
import ImplicitConversions._
import Serdes._

import org.apache.kafka.streams.scala.kstream.Materialized
import org.apache.kafka.streams.state.WindowStore
val materialized = Materialized.`with`[Long, String, ByteArrayWindowStore]

scala> :type materialized
org.apache.kafka.streams.kstream.Materialized[Long, String, org.apache.kafka.streams.scala.ByteArrayWindowStore]
```

Tip	Read up on Implicit Parameters .
-----	--

Low-Level Processor API

Kafka Streams offers **Processor API** with the following low-level data processing abstractions:

- [Processor](#) for stream processing nodes
- [ProcessorContext](#) to access the topology and record metadata
- [ProcessorSupplier](#) for creating `Processor` instances
- [Punctuator](#)

Processor Contract — Stream Processing Node

`Processor` is the main **abstraction** of the [Low-Level Processor API](#) for **record stream processors** (aka *stream processing nodes*) that can process one record at a time.

`Processor` can be added to a [Topology](#) using `Topology.addProcessor` operator (via [ProcessorSupplier](#)).

Note	<p>Streams DSL — High-Level Stream Processing DSL comes with the <code>KStream.process</code> operator to add a custom <code>Processor</code> to a topology (via ProcessorSupplier).</p>
------	--

The lifecycle of a `Processor` is fully controlled by a corresponding [ProcessorNode](#).

Table 1. Processor Contract

Method	Description
<code>close</code>	<pre style="margin: 0;"><code>void close()</code></pre> <p>Closes the processor</p> <p>Used exclusively when <code>ProcessorNode</code> is requested to close.</p>
<code>init</code>	<pre style="margin: 0;"><code>void init(ProcessorContext context)</code></pre> <p>Initializes the processor with a ProcessorContext (that can be used to decide whether a record should be forwarded downstream to child processors if there are any)</p> <p>Used exclusively when <code>ProcessorNode</code> is requested to init</p>
<code>process</code>	<pre style="margin: 0;"><code>void process(K key, V value)</code></pre> <p>Processes a single record (as a pair of a <code>k</code> key and a <code>v</code> value)</p> <p>Used exclusively when <code>ProcessorNode</code> is requested to process a single record (as a key and value pair).</p>

Tip	Use AbstractProcessor when you want to develop a custom <code>Processor</code> .
-----	--

Table 2. Processors (Direct Implementations)

Processor	Description
AbstractProcessor	Abstract processor that manages a ProcessorContext instance and provides a no-op <code>close</code> implementation
KStreamTransformValuesProcessor	
KTableSuppressProcessor	

AbstractProcessor—Base for Stream Processors

`AbstractProcessor` is the base implementation of the [Processor contract](#) for [stream processors](#) that manages a [ProcessorContext](#) and provides a no-op implementation of [close](#).

`AbstractProcessor` manages a [ProcessorContext](#) that is assigned when an `AbstractProcessor` is requested to [initialize](#).

Table 1. AbstractProcessors

AbstractProcessor	Description
KStreamAggregateProcessor	
KStreamBranchProcessor	Forwards a record to exactly one of the child processors
KStreamFilterProcessor	
KStreamFlatMapProcessor	
KStreamFlatMapValuesProcessor	
KStreamJoinWindowProcessor	
KStreamKStreamJoinProcessor	
KStreamKTableJoinProcessor	
KStreamMapProcessor	
KStreamMapProcessor	
KStreamPassThroughProcessor	
KStreamPeekProcessor	Executes an action with records (<i>peeks at them</i>)
KStreamPrintProcessor	
KStreamReduceProcessor	

KStreamSessionWindowAggregateProcessor	
KStreamTransformProcessor	
KStreamWindowAggregateProcessor	
KTableAggregateProcessor	
KTableFilterProcessor	
KTableKTableJoinMergeProcessor	
KTableKTableJoinProcessor	
KTableKTableLeftJoinProcessor	
KTableKTableOuterJoinProcessor	
KTableKTableRightJoinProcessor	
KTableMapProcessor	
KTableMapValuesProcessor	
KTableReduceProcessor	
KTableSourceProcessor	
KTableTransformValuesProcessor	

ProcessorContext Contract

`ProcessorContext` is the [contract](#) of [processor contexts](#) that allow to access to contextual information when a [processor](#) is executed.

`ProcessorContext` is used when:

- `Processor`, `StateStore`, `Transformer`, `ValueTransformer`, `ValueTransformerWithKey` and `KTableValueGetter` are requested to initialize
- `Task` is requested for the `ProcessorContext`
- `StoreChangeLogger` is created

Tip	Use AbstractProcessor to manage the <code>ProcessorContext</code> of a custom Processor.
-----	--

Table 1. ProcessorContext Contract

Method	Description
<code>appConfigs</code>	<code>Map<String, Object> appConfigs()</code>
<code>appConfigsWithPrefix</code>	<code>Map<String, Object> appConfigsWithPrefix(String prefix)</code>
<code>applicationId</code>	<code>String applicationId()</code>
<code>commit</code>	<code>void commit()</code>
<code>forward</code>	<code>void forward(K key, V value)</code> <code>void forward(K key, V value, To to)</code> <p>Fowards a record downstream (to child processors if there are any or <code>To</code> processor)</p>
<code>getStateStore</code>	<code>StateStore getStateStore(String name)</code> <p>Retrieves the <code>state store</code> by name</p> <p>Used when...FIXME</p>

headers	<code>Headers headers()</code>
keySerde	<code>Serde<?> keySerde()</code>
metrics	<code>StreamsMetrics metrics()</code>
	<code>StreamsMetrics</code>
offset	<code>long offset()</code>
partition	<code>int partition()</code>
register	<pre>void register(StateStore store, StateRestoreCallback stateRestoreCallback)</pre> <p>Registers a state store (and a StateRestoreCallback)</p> <p>Used when:</p> <ul style="list-style-type: none"> • State stores (AbstractRocksDBSegmentedBytesStore, InMemoryKeyValueStore, InMemorySessionStore, InMemoryTimeOrderedKeyValueBuffer, InMemoryWindowStore, MemoryLRUCache, and RocksDBStore) are requested to initialize • ForwardingDisabledProcessorContext is requested to register a state store (and a StateRestoreCallback)
schedule	<pre>Cancellable schedule(Duration interval, PunctuationType type, Punctuator callback)</pre> <p>Schedules a Punctuator (a periodic action) to be executed at stream (<code>STREAM_TIME</code>) or system (<code>WALL_CLOCK_TIME</code>) time and that can be cancelled</p> <ul style="list-style-type: none"> • <code>STREAM_TIME</code> advances by the processing of messages in accordance with the timestamp as extracted by the TimestampExtractor • <code>WALL_CLOCK_TIME</code> uses the system time which advances at the polling interval

<code>stateDir</code>	<code>File stateDir()</code>
<code>taskId</code>	<code>TaskId taskId()</code>
<code>timestamp</code>	<code>long timestamp()</code>
<code>topic</code>	<code>String topic()</code>
<code>valueSerde</code>	<code>Serde<?> valueSerde()</code>

Table 2. ProcessorContexts (Direct Implementations and Extensions)

ProcessorContext	Description
ForwardingDisabledProcessorContext	
InternalProcessorContext	Extension of <code>ProcessorContext</code>

TaskId

`TaskId` takes the following to be created:

- Topic group ID
- Assigned partition ID

`TaskId` is [created](#) when:

- `DefaultPartitionGrouper` is requested to [partitionGroups](#)
- `GlobalProcessorContextImpl` is [created](#)
- `TaskId` is requested to [parse](#) and [readFrom](#)

When requested for a textual representation (`toString`), `TaskId` simply returns the [topic group ID](#) and the [assigned partition ID](#) separated by `_` (underscore), e.g. `0_0`.

```
import org.apache.kafka.streams.processor.TaskId
val topicGroupId = 0
val partition = 0
val tid = new TaskId(topicGroupId, partition)
scala> println(tid)
0_0
```

Creating TaskId from Textual Representation — `parse` Factory Method

```
TaskId parse(final String taskIdStr)
```

`parse` [creates](#) a `TaskId` from the given textual representation.

`parse` [throws](#) a `TaskIdFormatException` if the given `taskIdStr` is of incorrect format.

```
import org.apache.kafka.streams.processor.TaskId
val incorrectTaskId = "hello_world"
scala> TaskId.parse(incorrectTaskId)
org.apache.kafka.streams.errors.TaskIdFormatException: Task id cannot be parsed correctly from hello_world
  at org.apache.kafka.streams.processor.TaskId.parse(TaskId.java:58)
  ... 36 elided
```

Note

- `parse` is used when:
- `StateDirectory` is requested to `cleanRemovedTasks`
 - `TaskManager` is requested to `cachedTaskIds`

Serializing TaskId — `writeTo` Method

```
void writeTo(final ByteBuffer buf)
void writeTo(final DataOutputStream out)
```

`writeTo` simply writes the `topicGroupId` and the `partition` out to the given output (either `buf` or `out`).

Note

- `writeTo` is used when:
- `SubscriptionInfo` is requested to `encodeTasks`
 - `AssignmentInfo` is requested to `encodeActiveAndStandbyTaskAssignment`

Creating TaskId from Encoded Representation (Deserialization) — `readFrom` Factory Method

```
TaskId readFrom(final ByteBuffer buf)
TaskId readFrom(final DataInputStream in)
```

`readFrom` simply creates a `TaskId` from (the numbers) from the given input (either `buf` or `in`).

Note

- `readFrom` is used when:
- `SubscriptionInfo` is requested to `decodeTasks`
 - `AssignmentInfo` is requested to `decodeActiveTasks` and `decodeStandbyTasks`

To

To

To is...FIXME

Punctuator Contract — Scheduled Periodic Actions

`Punctuator` is the [abstraction](#) that Kafka Streams developers use to [execute scheduled periodic actions](#) (aka **punctuate**) on a record stream.

```
void punctuate(long timestamp)
```

`punctuate` is used exclusively when `ProcessorNode` is requested to [execute a Punctuator](#).

`Punctuator` is scheduled (registered) using [ProcessorContext.schedule](#) method.

Cancellable

Cancellable is...FIXME

ProcessorSupplier Contract

`ProcessorSupplier` is the [abstraction](#) of `processor factories` that can [create](#) (aka *supply*) a [record processor](#).

`ProcessorSupplier` defines a single `get` method that can [supply a record processor](#).

```
Processor<K, V> get()
```

`get` is used when `ProcessorNodeFactory` is requested to [build a ProcessorNode](#) (and also for a description of `ProcessorParameters`).

Table 1. ProcessorSuppliers (Direct Implementations)

ProcessorSupplier	Description
KStreamAggProcessorSupplier	
KStreamBranch	Represents <code>KStream.branch</code> operator
KStreamFilter	
KStreamFlatMap	
KStreamFlatMapValues	
KStreamGlobalKTableJoin	
KStreamJoinWindow	
KStreamKStreamJoin	
KStreamKTableJoin	
KStreamMap	
KStreamMapValues	
KStreamPassThrough	
KStreamPeek	Represents <code>KStream.foreach</code> and <code>KStream.peek</code> operators
KStreamPrint	
KStreamTransform	
KStreamTransformValues	
KTableProcessorSupplier	
KTableSource	

StreamPartitioner

StreamPartitioner is...FIXME

TopicNameExtractor — Dynamic Routing of Output Records

`TopicNameExtractor` is the abstraction of topic name extractors that Kafka Streams developers use for **dynamic routing of records** by extracting the name of the topic to publish a record to.

Note

`TopicNameExtractor` can be specified using the high-level `KStream.to` or the low-level `Topology.addSink` operators.

Table 1. TopicNameExtractor Contract

Method	Description
<code>extract</code>	<pre>String extract(final K key, final V value, final RecordContext recordContext)</pre> <p>Extracts the name of the topic to publish a record to Used exclusively when <code>SinkNode</code> is requested to process a record</p>

Note

`StaticTopicNameExtractor` is an internal implementation of the [TopicNameExtractor Contract](#) in Kafka Streams.

Tip

Read up on the feature in [KIP-303: Add Dynamic Routing in Streams Sink](#).

RecordContext — Record Metadata

`RecordContext` is the [abstraction](#) of [record contexts](#) that Kafka Streams developers use for **dynamic routing of output records** (using [TopicNameExtractor](#)) based on the metadata of a record, i.e. [headers](#), [offset](#), [partition](#), [timestamp](#), and [topic](#).

Note	ProcessorRecordContext is the default implementation of the RecordContext Contract in Kafka Streams.
------	--

Table 1. RecordContext Contract

Method	Description
<code>headers</code>	<code>Headers headers()</code> The headers of the record
<code>offset</code>	<code>long offset()</code> The position of the record in the corresponding Kafka partition
<code>partition</code>	<code>int partition()</code> The partition the record has been received from
<code>timestamp</code>	<code>long timestamp()</code> The timestamp of the record
<code>topic</code>	<code>String topic()</code> The topic the record has been received from

TimestampExtractor Contract

`TimestampExtractor` is a [contract](#) of timestamp extractors that extract a timestamp from a `record` for **event time** (aka *stream time*) semantics.

```
package org.apache.kafka.streams.processor;

interface TimestampExtractor {
    long extract(ConsumerRecord<Object, Object> record, long previousTimestamp);
}
```

The extracted timestamp is in milliseconds and can never be negative (or [will be dropped](#)).

You can define a custom timestamp extractor for reading a topic as a [KStream](#) or a [KTable](#) in an [Consumed](#) object (using [with](#) or [withTimestampExtractor](#)).

```
val timestampExtractor = new TimestampExtractor {
    def extract(record: ConsumerRecord[Object, Object], previousTimestamp: Long) = ???
}
val consumed = Consumed.`with`(timestampExtractor)

val builder = new StreamsBuilder
builder.stream(topic = "t1", consumed)
builder.table(topic = "t1", consumed)
```

When not defined using a [Consumed](#), Kafka Streams uses the default extractor as configured using [default.timestamp.extractor](#) configuration property.

```
val props = new java.util.Properties
import org.apache.kafka.streams.StreamsConfig
props.setProperty(StreamsConfig.DEFAULT_TIMESTAMP_EXTRACTOR_CLASS_CONFIG, ...)
```

`TimestampExtractor` can be used to schedule a periodic operation for processors (using [ProcessorContext.schedule](#) with `PunctuationType.STREAM_TIME`).

Table 1. TimestampExtractor Contract

Method	Description
<code>extract</code>	Used exclusively when <code>RecordQueue</code> is requested to add Kafka ConsumerRecords (as StampedRecords) .

Table 2. TimestampExtractors

TimestampExtractor	Description
WallclockTimestampExtractor	
ExtractRecordMetadataTimestamp	
Note	<code>TimestampExtractor</code> is an <code>Evolving</code> contract which means that compatibility may be broken at a minor release.

`TimestampExtractor` is used to create a [SourceNodeFactory](#), [RecordQueue](#), [SourceNodeFactory](#).

WallclockTimestampExtractor

`WallclockTimestampExtractor` is a [TimestampExtractor](#) that [uses the current wall clock time as timestamp](#) when requested to extract a timestamp from a record.

extract Method

```
long extract(final ConsumerRecord<Object, Object> record, final long previousTimestamp)
```

Note

`extract` is part of [TimestampExtractor Contract](#) to extract a timestamp from a record.

`extract` simply requests the Java [System](#) for the [current time in milliseconds](#).

ExtractRecordMetadataTimestamp

`ExtractRecordMetadataTimestamp` is the base of [TimestampExtractors](#) that use an [onInvalidTimestamp error handler](#) when a record has a negative (invalid) timestamp value (while extracting embedded metadata timestamps from Kafka messages).

```
package org.apache.kafka.streams.processor;

abstract class ExtractRecordMetadataTimestamp implements TimestampExtractor {
    // only required methods that have no implementation
    // the others follow
    abstract long onInvalidTimestamp(
        final ConsumerRecord<Object, Object> record,
        final long recordTimestamp,
        final long previousTimestamp);
}
```

Note	<code>ExtractRecordMetadataTimestamp</code> is an Evolving contract which means that compatibility may be broken at a minor release.
------	--

Table 1. `ExtractRecordMetadataTimestamp` Contract

Method	Description
onInvalidTimestamp	Used when...FIXME

Table 2. `ExtractRecordMetadataTimestamps`

<code>ExtractRecordMetadataTimestamp</code>	Description
FailOnInvalidTimestamp	
UsePreviousTimeOnInvalidTimestamp	
LogAndSkipOnInvalidTimestamp	

extract Method

```
long extract(final ConsumerRecord<Object, Object> record, final long previousTimestamp)
```

Note	<code>extract</code> is part of TimestampExtractor Contract to extract a timestamp from a record.
------	---

`extract` ...FIXME

FailOnInvalidTimestamp Timestamp Extractor

`FailOnInvalidTimestamp` is used as the [default TimestampExtractor](#).

PartitionGrouper Contract

PartitionGrouper is the abstraction of partition groupers that can partitionGroups.

Table 1. PartitionGrouper Contract

Method	Description
partitionGroups	<pre>Map<TaskId, Set<TopicPartition>> partitionGroups(Map<Integer, Set<String>> topicGroups, Cluster metadata)</pre> <p>Used exclusively when StreamsPartitionAssignor is requested to assign tasks to consumer clients</p>
Note	<p>DefaultPartitionGrouper is the one and only known implementation of the PartitionGrouper Contract.</p>

DefaultPartitionGrouper

`DefaultPartitionGrouper` is a concrete [PartitionGrouper](#) that...FIXME

partitionGroups Method

```
Map<TaskId, Set<TopicPartition>> partitionGroups(  
    final Map<Integer, Set<String>> topicGroups,  
    final Cluster metadata)
```

Note

`partitionGroups` is part of the [PartitionGrouper Contract](#) to...FIXME.

`partitionGroups` ...FIXME

Finding Maximum Number Of Partitions For Topics — maxNumPartitions Method

```
int maxNumPartitions(  
    final Cluster metadata,  
    final Set<String> topics)
```

`maxNumPartitions` ...FIXME

Note

`maxNumPartitions` is used exclusively when `DefaultPartitionGrouper` is requested to [partitionGroups](#).

StateRestoreCallback Contract

`StateRestoreCallback` is the abstraction of objects that can restore.

Table 1. StateRestoreCallback Contract

Method	Description
<code>restore</code>	<pre>void restore(byte[] key, byte[] value)</pre> <p>Used exclusively when <code>StateRestoreCallbackAdapter</code> factory object is requested for a <code>RecordBatchingStateRestoreCallback</code> for a given <code>StateRestoreCallback</code> (that is neither a <code>RecordBatchingStateRestoreCallback</code> nor a <code>BatchingStateRestoreCallback</code>)</p>

Functional Interface and StateRestoreCallback's Implementations

`StateRestoreCallback` is a **functional interface** in Java.

Functional interface is an interface that has just one abstract method (aside from the methods of `java.lang.Object`), and thus represents a single function contract.

In addition to the usual process of creating an interface instance by declaring and instantiating a class, instances of functional interfaces can be created with method reference expressions and lambda expressions.

Because of this simplification of the Java language, `StateRestoreCallback` implementations are *usually* anonymous classes and are defined and used in the following **state stores**:

- [InMemoryKeyValueStore](#)
- [InMemorySessionStore](#)
- [InMemoryTimeOrderedKeyValueBuffer](#)
- [InMemoryWindowStore](#)
- [MemoryLRUCache](#)

The following are the regular named implementations.

Table 2. StateRestoreCallbacks (Direct Implementations and Extensions Only)

StateRestoreCallback	Description
AbstractNotifyingRestoreCallback	
BatchingStateRestoreCallback	

BatchingStateRestoreCallback

BatchingStateRestoreCallback is...FIXME

StateRestoreListener Contract

`StateRestoreListener` is the contract of [objects](#) that want to be notified about the store-related events, i.e. [onBatchRestored](#), [onRestoreEnd](#) and [onRestoreStart](#).

Note

A Kafka Streams developer uses [KafkaStreams.setGlobalStateRestoreListener](#) method to register a `StateRestoreListener` in a Kafka Streams application.

Table 1. StateRestoreListener Contract

Method	Description
<code>onBatchRestored</code>	<pre>void onBatchRestored(TopicPartition topicPartition, String storeName, long batchEndOffset, long numRestored)</pre> <p>Used when:</p> <ul style="list-style-type: none"> • <code>GlobalStateManagerImpl</code> is requested to restoreState (when requested to register a state store) • <code>StateRestorer</code> is requested to restoreBatchCompleted
<code>onRestoreEnd</code>	<pre>void onRestoreEnd(TopicPartition topicPartition, String storeName, long totalRestored)</pre> <p>Used when:</p> <ul style="list-style-type: none"> • <code>GlobalStateManagerImpl</code> is requested to restoreState (when requested to register a state store) • <code>StateRestorer</code> is requested to restoreDone
<code>onRestoreStart</code>	<pre>void onRestoreStart(TopicPartition topicPartition, String storeName, long startingOffset, long endingOffset)</pre> <p>Used when:</p> <ul style="list-style-type: none"> • <code>GlobalStateManagerImpl</code> is requested to restoreState (when requested to register a state store) • <code>StateRestorer</code> is requested to restoreStarted

Table 2. StateRestoreListeners (Direct Implementations)

StateRestoreListener	Description
AbstractNotifyingBatchingRestoreCallback	
AbstractNotifyingRestoreCallback	
CompositeRestoreListener	
DelegatingStateRestoreListener	

AbstractNotifyingBatchingRestoreCallback

AbstractNotifyingBatchingRestoreCallback is...FIXME

AbstractNotifyingRestoreCallback

AbstractNotifyingRestoreCallback is...FIXME

StateListener — KafkaStreams State Listener

StateListener is...FIXME

CacheFlushListener

CacheFlushListener is...FIXME

StreamsMetrics

`StreamsMetrics` is a contract of streams metrics that [FIXME](#).

```
package org.apache.kafka.streams;

interface StreamsMetrics {
    Map<MetricName, ? extends Metric> metrics();
    Sensor addLatencyAndThroughputSensor(
        final String scopeName,
        final String entityName,
        final String operationName,
        final Sensor.RecordingLevel recordingLevel,
        final String... tags);
    void recordLatency(
        final Sensor sensor,
        final long startNs,
        final long endNs);
    Sensor addThroughputSensor(
        final String scopeName,
        final String entityName,
        final String operationName,
        final Sensor.RecordingLevel recordingLevel,
        final String... tags);
    void recordThroughput(
        final Sensor sensor,
        final long value);
    Sensor addSensor(
        final String name,
        final Sensor.RecordingLevel recordingLevel);
    Sensor addSensor(
        final String name,
        final Sensor.RecordingLevel recordingLevel,
        final Sensor... parents);
    void removeSensor(final Sensor sensor);
}
```

Note	<code>StreamsMetrics</code> is an Evolving contract which means that compatibility may be broken at a minor release.
------	--

Table 1. StreamsMetrics Contract

Method	Description
metrics	Used when...FIXME
recordLatency	Recording a latency with a Kafka Sensor Used when...FIXME
Note	<p>StreamsMetricsImpl is the one and only known direct implementation of StreamsMetrics Contract in Kafka Streams 2.3.0.</p>

StreamsMetricsImpl

`StreamsMetricsImpl` is a concrete `StreamsMetrics` that...FIXME

`StreamsMetricsImpl` is created when:

- `GlobalStreamThread` is created
- `StreamsMetricsThreadImpl` is created

When requested for the metrics, `StreamsMetricsImpl` simply requests the Kafka Metrics for them.

Note

`StreamsMetricsThreadImpl` is a custom `StreamsMetricsImpl` that is used...
FIXME...for efficiency.

measureLatencyNs Method

```
void measureLatencyNs(final Time time, final Runnable action, final Sensor sensor)
```

`measureLatencyNs` ...FIXME

Note

`measureLatencyNs` is used when...FIXME

Recording Latency with Sensor— recordLatency Method

```
void recordLatency(Sensor sensor, long startNs, long endNs)
```

Note

`recordLatency` is part of `StreamsMetrics Contract` to record a latency with a Kafka Sensor .

`recordLatency` simply requests the input Kafka Sensor to record the latency (i.e. the difference between the input `endNs` and `startNs` timestamps).

Creating StreamsMetricsImpl Instance

`StreamsMetricsImpl` takes the following when created:

- Kafka Metrics

- Thread name

`StreamsMetricsImpl` initializes the [internal registries and counters](#).

StreamsMetricsThreadImpl

`StreamsMetricsThreadImpl` is a concrete [StreamsMetricsImpl](#).

`StreamsMetricsThreadImpl` is created for a [StreamThread](#) (when `KafkaStreams` is created).

Table 1. StreamsMetricsThreadImpl's Sensors

Name	Description
commitTimeSensor	
pollTimeSensor	
processTimeSensor	
punctuateTimeSensor	
taskCreatedSensor	
tasksClosedSensor	
skippedRecordsSensor	

Creating StreamsMetricsThreadImpl Instance

`StreamsMetricsThreadImpl` takes the following when created:

- `Metrics`
- `groupName`
- `prefix`
- Tags (as `Map<String, String>`)

`StreamsMetricsThreadImpl` initializes the [sensors](#).

createMeter Internal Method

```
Meter createMeter(Metrics metrics, SampledStat stat, String baseName, String descriptiveName)
```

`createMeter ...FIXME`

Note	createMeter is used when...FIXME
------	----------------------------------

removeAllSensors Method

```
void removeAllSensors()
```

removeAllSensors ...FIXME

Note	removeAllSensors is used when...FIXME
------	---------------------------------------

TaskMetrics

TaskMetrics is...FIXME

StoreChangeLogger

`StoreChangeLogger` is created when the following state stores are requested to initialize:

- `ChangeLoggingKeyValueBytesStore`
- `ChangeLoggingSessionBytesStore`
- `ChangeLoggingWindowBytesStore`

Creating StoreChangeLogger Instance

`StoreChangeLogger` takes the following to be created:

- Name of a state store
- `ProcessorContext`
- Partition ID
- `StateSerdes<K, V>`

`StoreChangeLogger` initializes the internal properties.

logChange Method

```
void logChange(K key, V value) (1)
void logChange(K key, V value, long timestamp)
```

1. Uses the ProcessorContext for the timestamp

`logChange` simply requests the `RecordCollector` to send the key and the value to the topic and the partition (using the `keySerializer` and the `valueSerializer`).

Note	<p><code>logChange</code> is used when:</p> <ul style="list-style-type: none"> • <code>ChangeLoggingKeyValueBytesStore</code> is requested to log • <code>ChangeLoggingSessionBytesStore</code> is requested to remove and put • <code>ChangeLoggingWindowBytesStore</code> is requested to log • <code>ChangeLoggingTimestampedKeyValueBytesStore</code> and <code>ChangeLoggingTimestampedWindowBytesStore</code> are requested to log
------	--

Internal Properties

Name	Description
collector	<p><code>RecordCollector</code></p> <p>Used exclusively when <code>StoreChangeLogger</code> is requested to <code>logChange</code>.</p>
topic	<p>The name of the changelog topic (given the application ID and the store name)</p> <p>Used when...FIXME</p>

DelegatingStateRestoreListener

`DelegatingStateRestoreListener` is a concrete `StateRestoreListener` that simply intercepts the state-related events (e.g. `onRestoreStart`, `onBatchRestored` and `onRestoreEnd`) and notifies the [user-defined global StateRestoreListener](#).

Note

`DelegatingStateRestoreListener` is a final class of `KafkaStreams` class and so has access to the internals of a `KafkaStreams` instance.

`DelegatingStateRestoreListener` is [created](#) for a `KafkaStreams` (that uses it to create a `GlobalStreamThread` and `StreamThreads`).

`DelegatingStateRestoreListener` takes no arguments when created.

onBatchRestored Method

```
void onBatchRestored(  
    final TopicPartition topicPartition,  
    final String storeName,  
    final long batchEndOffset,  
    final long numRestored)
```

Note

`onBatchRestored` is part of the [StateRestoreListener Contract](#) to...FIXME.

`onBatchRestored` ...FIXME

onRestoreEnd Method

```
void onRestoreEnd(  
    final TopicPartition topicPartition,  
    final String storeName,  
    final long batchEndOffset,  
    final long numRestored)
```

Note

`onRestoreEnd` is part of the [StateRestoreListener Contract](#) to...FIXME.

`onRestoreEnd` ...FIXME

onRestoreStart Method

```
void onRestoreStart(  
    final TopicPartition topicPartition,  
    final String storeName,  
    final long batchEndOffset,  
    final long numRestored)
```

Note

`onRestoreStart` is part of the [StateRestoreListener Contract](#) to...FIXME.

`onRestoreStart` ...FIXME

TopologyTestDriver

TopologyTestDriver is...FIXME

getAllStateStores Method

```
Map<String, StateStore> getAllStateStores()
```

getAllStateStores ...FIXME

Note	getAllStateStores is used when...FIXME
------	--

pipeInput Method

```
void pipeInput(final ConsumerRecord<byte[], byte[]> consumerRecord)
```

pipeInput ...FIXME

Note	pipeInput is used when...FIXME
------	--------------------------------

InternalTopologyBuilder

`InternalTopologyBuilder` is used to build a topology of processor nodes (for `StreamThread.StandbyTaskCreator` and `StreamThread.TaskCreator` task factories and hence stream processor tasks themselves).

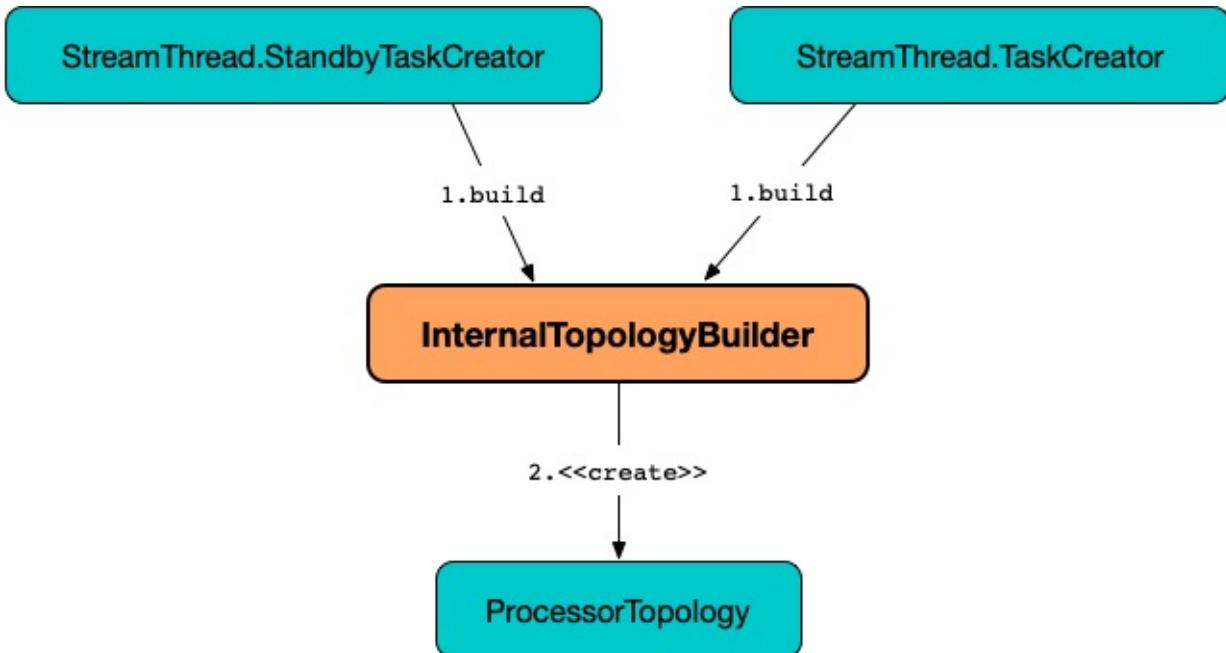


Figure 1. InternalTopologyBuilder and ProcessorTopology

`InternalTopologyBuilder` is created exclusively for a `Topology` (which is simply the frontend to the internals of topology development).

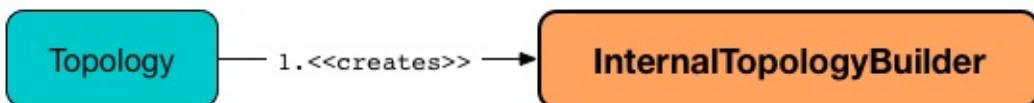


Figure 2. InternalTopologyBuilder and Topology

`InternalTopologyBuilder` takes no arguments when created.

```

import org.apache.kafka.streams.processor.internals.InternalTopologyBuilder
val builder = new InternalTopologyBuilder
  
```

`InternalTopologyBuilder` uses node factories to build a topology. `InternalTopologyBuilder` supports `ProcessorNodeFactory`, `SourceNodeFactory`, and `SinkNodeFactory` factory types only (and throws a `TopologyException` for unknown factories).

`InternalTopologyBuilder` can have one or more sources that are added (*registered*) using `addSource` method (which simply registers a `SourceNodeFactory` under a name).

```
scala> :type builder
org.apache.kafka.streams.processor.internals.InternalTopologyBuilder

val sourceNodeName = "sourceNode"

import org.apache.kafka.streams.processor.WallclockTimestampExtractor
val timestampExtractor = new WallclockTimestampExtractor
import org.apache.kafka.common.serialization.StringDeserializer
val keyDeserializer = new StringDeserializer
val valDeserializer = new StringDeserializer
val topics = Seq("input")
import org.apache.kafka.streams.Topology.AutoOffsetReset
builder.addSource(
  AutoOffsetReset.LATEST,
  sourceNodeName,
  timestampExtractor,
  keyDeserializer,
  valDeserializer,
  topics: _*)
```

`InternalTopologyBuilder` can have zero or more processor nodes that are added (registered) using `addProcessor` method (which simply registers a `ProcessorNodeFactory` under a name with a `ProcessorSupplier` and one or more predecessors). Since processors require that the predecessors are already added to the topology, a topology has to have at least one `source processor already added`.

```
scala> :type builder
org.apache.kafka.streams.processor.internals.InternalTopologyBuilder

assert(!builder.getSourceTopicNames.isEmpty)

val sourceTopicName = "input"

import collection.JavaConverters._
assert(builder.getSourceTopicNames.asScala.contains(sourceTopicName))

import org.apache.kafka.streams.processor.{Processor, ProcessorContext, ProcessorSupplier}
val supplier = new ProcessorSupplier[String, String] {
  def get = new Processor[String, String] {
    def init(context: ProcessorContext): Unit = {}
    def process(key: String, value: String): Unit = {}
    def close(): Unit = {}
  }
}
val processorNodeName = "processorNode"
val sourceNodeName = "sourceNode"
val predecessorNames = Seq(sourceNodeName)
builder.addProcessor(processorNodeName, supplier, predecessorNames: _*)
```

`InternalTopologyBuilder` can have zero or more sink nodes that are added (registered) using `addSink` method (which simply registers a `SinkNodeFactory` under a name).

```
scala> :type builder
org.apache.kafka.streams.processor.internals.InternalTopologyBuilder

// FIXME: Finish the demo
```

`InternalTopologyBuilder` requires an **application ID**. The application ID is used as a namespace for the internal topics (so they do not clash with the topics of other Kafka Streams applications). The application ID is set using `setApplicationId` (and is the `APPLICATION_ID_CONFIG` configuration property that `KafkaStreams` requires when created).

```
InternalTopologyBuilder setApplicationId(final String applicationId)
```

`InternalTopologyBuilder` uses the **application id** when:

- `buildProcessorNode` and `topicGroups` for the names of changelog topics (when a `StateStoreFactory` has `logging enabled`)
- `decorateTopic` to prefix (*decorate*) internal topic names

`InternalTopologyBuilder` has to be optimized using `rewriteTopology` (that sets the required **application ID**).

```
scala> :type builder
org.apache.kafka.streams.processor.internals.InternalTopologyBuilder

import org.apache.kafka.streams.StreamsConfig
import java.util.Properties
val props = new Properties
props.put(StreamsConfig.APPLICATION_ID_CONFIG, "app")
props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, ":9092")
val config = new StreamsConfig(props)
builder.rewriteTopology(config)
```

`InternalTopologyBuilder` can be **described** (using a `TopologyDescription`).

```

scala> :type builder
org.apache.kafka.streams.processor.internals.InternalTopologyBuilder

// add nodes and stores

val description = builder.describe
scala> println(description)
Topologies:
Sub-topology: 0
  Source: sourceNode (topics: [input])
    --> processorNode
  Processor: processorNode (stores: [])
    --> none
    <- sourceNode

```

Global Source Node is a node from a [SourceNodeFactory](#) with exactly one [topic](#) and registered in [globalTopics](#). You can use [isGlobalSource](#) predicate to check if a name is of a global source node.

Table 1. InternalTopologyBuilder's Internal Properties (e.g. Registries, Counters and Flags)

Name	Description
copartitionSourceGroups	
earliestResetPatterns	
earliestResetTopics	Global state stores by name <ul style="list-style-type: none"> A new global StateStore is added exclusively when <code>InternalTopologyBuilder</code> is requested to add a global state store to a topology <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <p>Note There are two types of StateStores, i.e. global and regular. Use <code>allStateStoreName</code> to access them all.</p> </div>
globalStateStores	<div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 10px;"> <p>Note <code>InternalTopologyBuilder</code> comes with <code>globalStateStores</code> method to access <code>globalStateStores</code> registry as an unmodifiable collection. It is used when:</p> <ul style="list-style-type: none"> <code>KafkaStreams</code> is created (and creates a GlobalStateStoreProvider for the QueryableStoreProvider) <code>StreamsMetadataState</code> is created </div> Names of global topics

<code>globalTopics</code>	<ul style="list-style-type: none"> A new name is added when <code>InternalTopologyBuilder</code> is requested to add a global state store to a topology Used when <code>InternalTopologyBuilder</code> is requested to check if a node name is of a global source node
<code>internalTopicNames</code>	<p>Names of the internal topics that were auto-created when <code>InternalTopologyBuilder</code> was requested to addInternalTopic</p> <p>A new topic name is added when <code>InternalTopologyBuilder</code> is requested to addInternalTopic</p>
<code>latestResetPatterns</code>	
<code>latestResetTopics</code>	
<code>nodeFactories</code>	<p>NodeFactories by node name</p> <ul style="list-style-type: none"> A new <code>NodeFactory</code> is added when <code>InternalTopologyBuilder</code> is requested to add a global state store to a topology, addProcessor, addSink and addSource
<code>nodeGrouper</code>	<p>QuickUnion of the names of node groups</p> <p>Used when...FIXME</p>
<code>nodeGroups</code>	<p>Node groups by ID, i.e. groups of node names that can be looked up by a group ID (<code>Map<Integer, Set<String>></code>)</p> <p>Initialized when <code>InternalTopologyBuilder</code> is requested for node groups (the very first time since it never changes once initialized)</p> <ul style="list-style-type: none"> Reset (<i>nullified</i>) when registering a state store (as a StoreBuilder) <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p>Note nodeGroups accessor is used to access <code>nodeGroups</code> registry.</p> </div>
<code>nodeToSinkTopic</code>	
<code>nodeToSourcePatterns</code>	
	Topic names by the source processor node name (without the application ID prefix for internal topics)

<code>nodeToSourceTopics</code>	<p>New entries are added when <code>InternalTopologyBuilder</code> is requested for the following:</p> <ul style="list-style-type: none"> • <code>addSource</code> and <code>addGlobalStore</code> • <code>setRegexMatchedTopicsToSourceNodes</code>
<code>sourceTopicNames</code>	<p>Collection of <code>registered</code> topic names Used when...FIXME</p>
<code>stateFactories</code>	<p><code>StateStoreFactories</code> by name (of the <code>StoreBuilder</code>) (<code>Map<String, StateStoreFactory></code>)</p> <ul style="list-style-type: none"> • A new <code>StateStoreFactory</code> added when registering a state store (as <code>StoreBuilder</code>) <p>Used when connecting a state store with a processor <code>node</code>, <code>buildProcessorNode</code>, <code>allStateStoreName</code>, <code>topicGroups</code></p>
<code>stateStoreNameToSourceRegex</code>	
<code>stateStoreNameToSourceTopics</code>	
<code>storeToChangelogTopic</code>	<p>Names of the <code>state stores</code> and the names of the corresponding changelog topics (<code>Map<String, string></code>)</p> <p><code>storeToChangelogTopic</code> manages <code>state stores</code> with the <code>StateStoreFactory</code> with change-logging enabled</p> <p>A new pair is added when <code>InternalTopologyBuilder</code> is requested to <code>buildProcessorNode</code> and associate the names of a state store and a topic</p>
<code>subscriptionUpdates</code>	
<code>topicPattern</code>	<p>Source topics pattern (to subscribe to)</p> <ul style="list-style-type: none"> • Initialized the first time when <code>InternalTopologyBuilder</code> is requested for the source topics pattern
<code>topicToPatterns</code>	

Enable `DEBUG` logging level for `org.apache.kafka.streams.processor.internals.InternalTopologyBuilder` logger to see what happens inside.

Add the following line to `log4j.properties`:

Tip

```
log4j.logger.org.apache.kafka.streams.processor.internals.InternalTopologyBuild
```

Refer to [Application Logging Using log4j](#).

Adding Application ID to Topic (As Prefix) — `decorateTopic` Internal Method

```
String decorateTopic(final String topic)
```

`decorateTopic` ...FIXME

Note

`decorateTopic` is used when:

- `InternalTopologyBuilder` `buildSinkNode`, `buildSourceNode`, `maybeDecorateInternalSourceTopics` and `topicGroups`
- `SinkNodeFactory` is requested to `build` a sink node

buildSinkNode Internal Method

```
void buildSinkNode(
    Map<String, ProcessorNode> processorMap,
    Map<String, SinkNode> topicSinkMap,
    Set<String> repartitionTopics,
    SinkNodeFactory sinkNodeFactory,
    SinkNode node)
```

`buildSinkNode` ...FIXME

Note

`buildSinkNode` is used exclusively when `InternalTopologyBuilder` is requested to `build` a topology of processor nodes.

maybeDecorateInternalSourceTopics Internal Method

```
List<String> maybeDecorateInternalSourceTopics(final Collection<String> sourceTopics)
```

maybeDecorateInternalSourceTopics ...FIXME

Note

`maybeDecorateInternalSourceTopics` is used when:

- `InternalTopologyBuilder` is requested to `copartitionGroups`, `resetTopicsPattern`, `sourceTopicPattern` and `stateStoreNameToSourceTopics`
- `SourceNodeFactory` is requested to `build` a source node

resetTopicsPattern Internal Method

```
Pattern resetTopicsPattern(
    final Set<String> resetTopics,
    final Set<Pattern> resetPatterns,
    final Set<String> otherResetTopics,
    final Set<Pattern> otherResetPatterns)
```

resetTopicsPattern ...FIXME

Note

`resetTopicsPattern` is used when...FIXME

copartitionGroups Method

```
Collection<Set<String>> copartitionGroups()
```

copartitionGroups ...FIXME

Note

`copartitionGroups` is used exclusively when `StreamsPartitionAssignor` is requested to `perform group assignment (assign tasks to consumer clients)`.

copartitionSources Method

```
void copartitionSources(
    Collection<String> sourceNodes)
```

copartitionSources ...FIXME

Note

`copartitionSources` is used exclusively when `AbstractStream` is requested to `ensureJoinableWith`.

Adding Processor to Topology — `addProcessor` Method

```
void addProcessor(
    String name,
    ProcessorSupplier supplier,
    String... predecessorNames)
```

`addProcessor` simply registers a new `ProcessorNodeFactory` by the given name in the `nodeFactories` internal registry.

`addProcessor` also adds the name to the `nodeGrouper` and unites the processor name with the predecessors.

In the end, `addProcessor` resets the `nodeGroups` collection (i.e. `null`).

Note	A processor has a unique name, a <code>ProcessorSupplier</code> and at least one predecessor (that cannot be itself)
------	--

`addProcessor` requires that the given name, the `ProcessorSupplier` and the predecessor names are all defined (i.e. not `null`) or throws a `NullPointerException`.

`addProcessor` requires that the given name is unique across all the registered `nodeFactories` or throws a `TopologyException`.

`addProcessor` requires that there is at least one predecessor name given or throws a `TopologyException`.

Note	<p><code>addProcessor</code> is used when:</p> <ul style="list-style-type: none"> • <code>Topology</code> is requested to <code>add</code> a processor • <code>StreamsGraphNodes</code> (i.e. <code>KTableKTableJoinNode</code>, <code>OptimizableRepartitionNode</code>, <code>ProcessorGraphNode</code>, <code>StatefulProcessorNode</code>, <code>StreamStreamJoinNode</code>, <code>StreamTableJoinNode</code>, <code>TableProcessorNode</code>, and <code>TableSourceNode</code>) are requested to write to a <code>topology</code> (when <code>InternalStreamsBuilder</code> is requested to <code>build a topology</code> when <code>StreamsBuilder</code> is requested to <code>build a topology</code>)
------	--

`buildProcessorNode` Internal Method

```
void buildProcessorNode(
    Map<String, ProcessorNode> processorMap,
    Map<String, StateStore> stateStoreMap,
    ProcessorNodeFactory factory,
    ProcessorNode node)
```

```
buildProcessorNode ...FIXME
```

Note

`buildProcessorNode` is used exclusively when `InternalTopologyBuilder` is requested to build a topology of processor nodes.

buildSourceNode Internal Method

```
void buildSourceNode(
    Map<String, SourceNode> topicSourceMap,
    Set<String> repartitionTopics,
    SourceNodeFactory sourceNodeFactory,
    SourceNode node)
```

```
buildSourceNode ...FIXME
```

Note

`buildSourceNode` is used exclusively when `InternalTopologyBuilder` is requested to build a topology of processor nodes.

Adding Source Node to Topology — addSource Method

```
void addSource(
    Topology.AutoOffsetReset offsetReset,
    String name,
    TimestampExtractor timestampExtractor,
    Deserializer keyDeserializer,
    Deserializer valDeserializer,
    Pattern topicPattern)
void addSource(
    Topology.AutoOffsetReset offsetReset,
    String name,
    TimestampExtractor timestampExtractor,
    Deserializer keyDeserializer,
    Deserializer valDeserializer,
    String... topics)
```

`addSource` simply registers a new `SourceNodeFactory` by the given name in the `nodeFactories` internal registry.

`addSource` `maybeAddToResetList` every topic in the given topics.

`addSource` adds few inputs to the following internal registries:

- Topics to `sourceTopicNames`
- Name with the topics to `nodeToSourceTopics`

- Name to [nodeGrouper](#)

In the end, `addSource` resets the [nodeGroups](#) collection (i.e. `null`).

Note	A source processor has a unique name, a Topology.AutoOffsetReset , a TimestampExtractor , key and value deserializers, a ProcessorSupplier and at least one topic.
------	--

`addSource` requires that:

- There is at least one topic or throws a [TopologyException](#)
- Name is specified (not `null`) and unique across all the registered [nodeFactories](#) or throws a [TopologyException](#)
- No topic [has been registered earlier](#)

Note	<code>addSource</code> is used when: <ul style="list-style-type: none"> • <code>Topology</code> is requested to add a source node • StreamsGraphNode (i.e. StreamSourceNode) is requested to writeToTopology
------	--

maybeAddToResetList Internal Method

```
void maybeAddToResetList(
  final Collection<T> earliestResets,
  final Collection<T> latestResets,
  final Topology.AutoOffsetReset offsetReset,
  final T item)
```

`maybeAddToResetList` ...FIXME

Note	<code>maybeAddToResetList</code> is used when...FIXME
------	---

validateTopicNotAlreadyRegistered Internal Method

```
void validateTopicNotAlreadyRegistered(final String topic)
```

`validateTopicNotAlreadyRegistered` ...FIXME

Note	<code>validateTopicNotAlreadyRegistered</code> is used when...FIXME
------	---

Connecting Processor with State Stores

— `connectProcessorAndStateStores` Method

```
void connectProcessorAndStateStores(
    String processorName,
    String... stateStoreNames)
```

`connectProcessorAndStateStores` simply [connectProcessorAndStateStore](#) with `processorName` and every state store name in `stateStoreNames`.

`connectProcessorAndStateStores` reports a `NullPointerException` when `processorName`, `stateStoreNames` or any state store name are `nulls`.

`connectProcessorAndStateStores` reports a `TopologyException` when `stateStoreNames` is an empty collection.

Note `connectProcessorAndStateStores` (plural) is a public method that uses the internal [connectProcessorAndStateStore](#) (singular) for a "bulk connect".

Note `connectProcessorAndStateStores` is used when:

- `Topology` is requested to [connect](#) a processor with state stores
- `StreamsGraphNode` (i.e. `KTableKTableJoinNode`, `StatefulProcessorNode`, `StreamTableJoinNode`, and `TableProcessorNode`) are requested to [write](#) to a topology

Adding Global State Store to Topology

— `addGlobalStore` Method

```
void addGlobalStore(
    StoreBuilder storeBuilder,
    String sourceName,
    TimestampExtractor timestampExtractor,
    Deserializer keyDeserializer,
    Deserializer valueDeserializer,
    String topic,
    String processorName,
    ProcessorSupplier stateUpdateSupplier)
```

`addGlobalStore` first [validateGlobalStoreArguments](#) followed by [validateTopicNotAlreadyRegistered](#).

`addGlobalStore` creates a `ProcessorNodeFactory` with the given `processorName`, `sourceName` (as [predecessors](#)) and `stateUpdateSupplier` (as [supplier](#)).

`addGlobalStore` then does the following housekeeping tasks:

1. Adds the given `topic` to `globalTopics` internal registry
2. Creates a `SourceNodeFactory` and registers it in `nodeFactories` internal registry as `sourceName`
3. Associates the `sourceName` with `topic` to `nodeToSourceTopics`
4. Requests `QuickUnion` of the names of node groups to add the `sourceName`
5. Requests `ProcessorNodeFactory` to add a state store as `name`
6. Associates the `processorName` with `nodeFactory` in `nodeFactories`
7. Requests `QuickUnion` of the names of node groups to add the `processorName`
8. Requests `QuickUnion` of the names of node groups to unite the `processorName` and `predecessors`
9. Associates the `name` with the `store` in `globalStateStores`

In the end, `addGlobalStore` associates the names of the state store and the topic (with the name and topic).

Note	<p><code>addGlobalStore</code> is used when:</p> <ul style="list-style-type: none"> • <code>Topology</code> is requested to <code>addGlobalStore</code> • <code>GlobalStoreNode</code> and <code>TableSourceNode</code> are requested to <code>writeToTopology</code>
------	---

Validating Arguments for Creating Global State Store — `validateGlobalStoreArguments` Internal Method

```
void validateGlobalStoreArguments(
    final String sourceName,
    final String topic,
    final String processorName,
    final ProcessorSupplier stateUpdateSupplier,
    final String storeName,
    final boolean loggingEnabled)
```

`validateGlobalStoreArguments` validates the input parameters (before adding a global state store to a topology).

`validateGlobalStoreArguments` throws a `NullPointerException` when `sourceName`, `topic`, `stateUpdateSupplier` or `processorName` are null .

`validateGlobalStoreArguments` throws a `TopologyException` when:

- `nodeFactories` contains `sourceName` or `processorName`
- `storeName` is already registered in `stateFactories` or `globalStateStores`
- `loggingEnabled` is enabled (i.e. `true`)
- `sourceName` and `processorName` are equal

Note

`validateGlobalStoreArguments` is used exclusively when `InternalTopologyBuilder` is requested to [add a global state store to a topology](#).

Registering State Store with Topic (Associating Names)

— `connectSourceStoreAndTopic` Method

```
void connectSourceStoreAndTopic(
    String sourceStoreName,
    String topic)
```

`connectSourceStoreAndTopic` adds the given `sourceStoreName` with the `topic` to [storeToChangelogTopic](#) internal registry.

`connectSourceStoreAndTopic` reports a `TopologyException` when `storeToChangelogTopic` has `sourceStoreName` already registered.

Source store [sourceStoreName] is already added.

Note

`connectSourceStoreAndTopic` is used when:

- `InternalTopologyBuilder` is requested to [add a global state store to a topology](#)
- `TableSourceNode` is requested to [write to a topology](#)

Connecting State Store with Processor Node

— `connectProcessorAndStateStore` Internal Method

```
void connectProcessorAndStateStore(
    String processorName,
    String stateStoreName)
```

Note

`connectProcessorAndStateStore` (singular) is an internal method that is used by the public `connectProcessorAndStateStores` (plural).

`connectProcessorAndStateStore` gets the `StateStoreFactory` for the given `stateStoreName` (in `stateFactories`).

`connectProcessorAndStateStore` then unites all `users` of the `StateStoreFactory` with the given `processorName`. `connectProcessorAndStateStore` adds the `processorName` to the `users`.

`connectProcessorAndStateStore` gets the `NodeFactory` for the given `processorName` (in `nodeFactories`). Only when the `NodeFactory` is a `ProcessorNodeFactory`, `connectProcessorAndStateStore` registers the `stateStoreName` with the `ProcessorNodeFactory`.

In the end, `connectProcessorAndStateStore` `connectStateStoreNameToSourceTopicsOrPattern` (with the input `stateStoreName` and the `ProcessorNodeFactory`).

`connectProcessorAndStateStore` reports a `TopologyException` when the input `stateStoreName` or `processorName` have not been registered yet or the `processorName` is the name of a source or sink node.

Note

`connectProcessorAndStateStore` is used when `InternalTopologyBuilder` is requested to add a state store and connect a processor with state stores.

connectStateStoreNameToSourceTopicsOrPattern Internal Method

```
void connectStateStoreNameToSourceTopicsOrPattern(
    String stateStoreName,
    ProcessorNodeFactory processorNodeFactory)
```

`connectStateStoreNameToSourceTopicsOrPattern ...FIXME`

Note

`connectStateStoreNameToSourceTopicsOrPattern` is used exclusively when `InternalTopologyBuilder` is requested to connect a processor with a state store

Adding State Store to Topology — `addStateStore` Method

```
void addStateStore(
    StoreBuilder<?> storeBuilder,
    String... processorNames) (1)
void addStateStore(
    StoreBuilder<?> storeBuilder,
    boolean allowOverride,
    String... processorNames)
```

1. Uses `false` for the `allowOverride` flag

`addStateStore` creates a [StateStoreFactory](#) (with the [StoreBuilder](#)) and registers it (in the [stateFactories](#) internal registry).

`addStateStore` then connects the state store with processors (if they are given).

In the end, `addStateStore` resets (*nullify*) the [nodeGroups](#) internal registry.

`addStateStore` throws a [TopologyException](#) when the given [StoreBuilder](#) has already been registered (in the [stateFactories](#) internal registry) and the `allowOverride` flag is off (`false`):

```
StateStore [name] is already added.
```

Note	<p><code>addStateStore</code> is used (with the <code>allowOverride</code> flag disabled) when:</p> <ul style="list-style-type: none"> • <code>Topology</code> is requested to add a state store and associate it with processors • <code>KTableKTableJoinNode</code> is requested to writeToTopology • <code>StatefulProcessorNode</code> is requested to writeToTopology • <code>StateStoreNode</code> is requested to writeToTopology • <code>StreamStreamJoinNode</code> is requested to writeToTopology • <code>TableProcessorNode</code> is requested to writeToTopology • <code>TableSourceNode</code> is requested to writeToTopology
------	--

Topic Groups (TopicsInfos By IDs) — `topicGroups` Method

```
Map<Integer, TopicsInfo> topicGroups()
```

`topicGroups` ...FIXME

Note

`topicGroups` is used exclusively when `StreamsPartitionAssignor` is requested to [assign](#).

Getting Node Groups by ID — `nodeGroups` Accessor Method

```
synchronized Map<Integer, Set<String>> nodeGroups()
```

`nodeGroups` gives [node groups by id](#).

If [node groups by id](#) registry has not been initialized yet, `nodeGroups` creates the [node groups](#) that are the [node groups](#) from now on.

Note

`nodeGroups` is used when `InternalTopologyBuilder` is requested to build a [topology for a topic group ID](#), `globalNodeGroups` and `topicGroups`

Building Global Processor Task Topology — `buildGlobalStateTopology` Method

```
ProcessorTopology buildGlobalStateTopology()
```

`buildGlobalStateTopology` `globalNodeGroups` and builds a [processor task topology](#) with the global node groups.

`buildGlobalStateTopology` returns `null` if `globalNodeGroups` is empty.

Note

`buildGlobalStateTopology` is used exclusively when `KafkaStreams` is [created](#).

`describeGlobalStore` Internal Method

```
void describeGlobalStore(final TopologyDescription description, final Set<String> nodes, int id)
```

`describeGlobalStore` ...FIXME

Note

`describeGlobalStore` is used exclusively when `InternalTopologyBuilder` is requested to [describe](#).

`nodeGroupContainsGlobalSourceNode` Internal Method

```
void nodeGroupContainsGlobalSourceNode(final TopologyDescription description, final Set<String> nodes, int id)
```

nodeGroupContainsGlobalSourceNode ...FIXME

Note

`nodeGroupContainsGlobalSourceNode` is used exclusively when `InternalTopologyBuilder` is requested to [describe](#).

Checking If Node Name Is Of Global Source Node — `isGlobalSource` Internal Method

```
boolean isGlobalSource(final String nodeName)
```

`isGlobalSource` looks up a [NodeFactory](#) by the input node name (in the `nodeFactories` internal registry).

`isGlobalSource` is positive (i.e. `true`) when the following all hold:

- `nodeName` is the name of a [SourceNodeFactory](#) with exactly one [topic](#)
- The single topic is among `globalTopics`

Otherwise, `isGlobalSource` is negative (i.e. `false`).

Note

`isGlobalSource` is used when `InternalTopologyBuilder` is requested to [describeGlobalStore](#), [globalNodeGroups](#) and [nodeGroupContainsGlobalSourceNode](#).

Global Node Groups — `globalNodeGroups` Internal Method

```
Set<String> globalNodeGroups()
```

`globalNodeGroups` gives [node groups](#) with at least one [global source node](#).

Note

`globalNodeGroups` is used when `InternalTopologyBuilder` is requested to build a [processor task topology](#) and [global processor task topology](#).

Building Node Groups — `makeNodeGroups` Internal Method

```
Map<Integer, Set<String>> makeNodeGroups()
```

`makeNodeGroups` starts with no node groups and the local counter of node group IDs as `0`.

Note

`makeNodeGroups` uses Java's [java.util.LinkedHashMap](#) that is *Hash table and linked list implementation of the Map interface, with predictable iteration order*.

`makeNodeGroups` takes the names of registered source nodes (from the [nodeToSourceTopics](#) and [nodeToSourcePatterns](#) internal registries).

`makeNodeGroups` sorts the names of the source nodes in ascending order (per the natural ordering) and [putNodeGroupName](#) for every source node name.

Note

While [putNodeGroupName](#), `makeNodeGroups` may end up with a new node group ID. After processing all source node names, the node group ID is the last group ID assigned.

`makeNodeGroups` takes the non-source node names (from the [nodeFactories](#) internal registry that are not in the [nodeToSourceTopics](#) internal registry).

`makeNodeGroups` does the same group ID assignment as for the source node names, i.e. sorts the names in ascending order and [putNodeGroupName](#) for every node name.

In the end, `makeNodeGroups` returns the node (names) groups by ID.

Note

`makeNodeGroups` is used when [InternalTopologyBuilder](#) is requested to describe a topology, and [get node groups](#).

putNodeGroupName Internal Method

```
int putNodeGroupName(
    final String nodeName,
    final int nodeGroupId,
    final Map<Integer, Set<String>> nodeGroups,
    final Map<String, Set<String>> rootToNodeGroup)
```

`putNodeGroupName` takes the name of a node, the current node group ID, the current node groups and the `rootToNodeGroup`.

`putNodeGroupName` requests [QuickUnion](#) of the names of node groups for the root node of the input `nodeName`.

`putNodeGroupName` gets the node group for the root node from the input `rootToNodeGroup` and adds the input `nodeName` to it.

If the root node was not found in the input `rootToNodeGroup`, `putNodeGroupName` registers the root node with an empty node group in `rootToNodeGroup`. `putNodeGroupName` then registers the empty node group with an incremented node group ID in `nodeGroups`.

In the end, `putNodeGroupName` gives the input `nodeGroupId` or a new node group ID if the root node was not found in the input `rootToNodeGroup`.

Note

`putNodeGroupName` is used exclusively when `InternalTopologyBuilder` is requested to [create the node groups](#).

Describing Topology (as `TopologyDescription`) — `describe` Method

```
TopologyDescription describe()
```

```
describe ...FIXME
```

```

import org.apache.kafka.streams.processor.internals.InternalTopologyBuilder
val itb = new InternalTopologyBuilder()

// Create a state store builder
import org.apache.kafka.streams.state.Stores
val lruMapSupplier = Stores.lruMap("input-stream", 5)
import org.apache.kafka.common.serialization.Serdes
import org.apache.kafka.streams.state.{KeyValueStore, StoreBuilder}
val storeBuilder = Stores.keyValueStoreBuilder(
    lruMapSupplier,
    Serdes.Long(),
    Serdes.Long()).
    withLoggingDisabled

// Add the state store as a global state store
import org.apache.kafka.streams.processor.TimestampExtractor
val timestampExtractor: TimestampExtractor = null
import org.apache.kafka.common.serialization.LongDeserializer
val keyDeserializer = new LongDeserializer
val valueDeserializer = new LongDeserializer
import org.apache.kafka.streams.kstream.internals.KTableSource
import org.apache.kafka.streams.processor.ProcessorSupplier
import java.lang.{Long => JLong}
val stateUpdateSupplier: ProcessorSupplier[JLong, JLong] = new KTableSource("global-store")
itb.addGlobalStore(
    // Required to make the code compile
    storeBuilder.asInstanceOf[StoreBuilder[KeyValueStore[_, _]]],
    "sourceName",
    timestampExtractor,
    keyDeserializer,
    valueDeserializer,
    "global-store-topic",
    "processorName",
    stateUpdateSupplier)

import org.apache.kafka.streams.TopologyDescription
val td: TopologyDescription = itb.describe
scala> println(td)
Topologies:
Sub-topology: 0 for global store (will not generate tasks)
  Source: sourceName (topics: global-store-topic)
    --> processorName
  Processor: processorName (stores: [input-stream])
    --> none
    <-- sourceName

```

Note	<code>describe</code> is used exclusively when <code>Topology</code> is requested to describe .
------	---

describeSubtopology Internal Method

```
void describeSubtopology(
    TopologyDescription description,
    Integer subtopologyId,
    Set<String> nodeNames)
```

describeSubtopology ...FIXME

Note

`describeSubtopology` is used exclusively when `InternalTopologyBuilder` is requested to [describe itself](#).

describeGlobalStore Internal Method

```
void describeGlobalStore(
    final TopologyDescription description,
    final Set<String> nodes, int id)
```

describeGlobalStore ...FIXME

Note

`describeGlobalStore` is used exclusively when `InternalTopologyBuilder` is requested to [describe itself](#).

Adding Sink Node to Topology — addSink Method

```
void addSink(
    String name,
    String topic,
    Serializer<K> keySerializer,
    Serializer<V> valSerializer,
    StreamPartitioner<? super K, ? super V> partitioner,
    String... predecessorNames) (1)
void addSink(
    String name,
    TopicNameExtractor<K, V> topicExtractor,
    Serializer<K> keySerializer,
    Serializer<V> valSerializer,
    StreamPartitioner<? super K, ? super V> partitioner,
    String... predecessorNames)
```

1. Uses StaticTopicNameExtractor

`addSink` creates a `SinkNodeFactory` (passing all the inputs along) and registers (`adds`) it in the `nodeFactories` internal registry (under the input `name`).

`addSink` registers the input `topic` with the input `name` in the `nodeToSinkTopic` internal registry.

`addSink` adds the input `name` to the `nodeGrouper` internal registry and requests it to `unite` the input `name` with the input `predecessorNames`.

Note

- `addSink` is used when:
- `GroupedTableOperationRepartitionNode` is requested to `writeToTopology`
 - `OptimizableRepartitionNode` is requested to `writeToTopology`
 - `Topology` is requested to `add a sink`

Adding Internal Topic Name — `addInternalTopic` Method

```
void addInternalTopic(  
    String topicName)
```

`addInternalTopic` simply adds the input `topicName` to the `internalTopicNames` internal registry.

Note

- `addInternalTopic` is used when:
- `KStreamImpl` is requested to `createRepartitionedSource`
 - `KGroupedTableImpl` is requested to `buildAggregate` (for `reduce`, `count` and `aggregate` operators)

Building Topology of Processor Nodes — `build` Method

```
ProcessorTopology build() (1)  
ProcessorTopology build(  
    Integer topicGroupId) (2)  
  
// PRIVATE  
private ProcessorTopology build(  
    Set<String> nodeGroup)
```

1. Uses `build` with an undefined `topicGroupId` (i.e. `null`)
2. Uses `build` with `nodeGroup` being the node names for a given `topicGroupId`

The private `build` takes the `NodeFactories` (from the `nodeFactories` internal registry).

For every `NodeFactory` the private `build` checks if the node (by its `name`) is included in the input `nodeGroup` (with the assumption that it is when the `nodeGroup` is `null` which can happen when a group ID could not be found in the `nodeGroups` internal registry) and, if it is, does the following:

1. Requests the `NodeFactory` to [build a processor node](#) (and adds it to a local `processorMap` of processors by their names)
2. For [ProcessorNodeFactories](#), `build` [buildProcessorNode](#)
3. For [SourceNodeFactories](#), `build` [buildSourceNode](#)
4. For [SinkNodeFactories](#), `build` [buildSinkNode](#)

In the end, `build` creates a [ProcessorTopology](#).

`build` throws a `TopologyException` for unknown `NodeFactories`.

Unknown definition class: [className]

Note	<code>nodeGroup</code> can be either global node groups (aka <i>global state topology</i>), a single or all node groups .
Note	The private <code>build</code> is used when <code>InternalTopologyBuilder</code> is requested to build a processor task topology (for a group ID) and build a global processor task topology .
Note	The no-argument <code>build</code> is used exclusively when <code>KafkaStreams</code> is created (as a sanity check to fail-fast in case a <code>ProcessorTopology</code> could not be built due to some exception).

Building Processor Task Topology For Group ID — `build` Factory Method

```
ProcessorTopology build(
    Integer topicGroupId)
```

This variant of `build` takes either a group ID or `null` (see the parameter-less [build\(\)](#)).

For the input `topicGroupId` specified (i.e. non-`null`), `build` looks up the group ID in the `nodeGroups` internal registry and [builds the topology](#) (for the node names in the node group).

When the input `topicGroupId` is undefined (i.e. `null`), `build` takes the node names (from the `nodeGroups` internal registry) and removes `globalNodeGroups`. In the end, `build` builds the topology (for the node names).

Note	<p><code>build</code> (with a topic group ID) is used when:</p> <ul style="list-style-type: none"> • <code>InternalTopologyBuilder</code> is requested to build a processor task topology (with no group ID specified) • <code>StandbyTaskCreator</code> is requested to create a standby task for a given task ID • <code>TaskCreator</code> is requested to create a stream task for a given task ID
------	---

allStateStoreName Method

```
Set<String> allStateStoreName()
```

`allStateStoreName` simply returns the state store names (the keys) from the `stateFactories` and `globalStateStores` internal registries.

Note	<p><code>allStateStoreName</code> is used exclusively when <code>TopologyTestDriver</code> is requested to <code>getAllStateStores</code>.</p>
------	--

Creating InternalTopicConfig (Given Name and StateStoreFactory) — `createChangelogTopicConfig` Internal Method

```
InternalTopicConfig createChangelogTopicConfig(
    StateStoreFactory factory,
    String name)
```

`createChangelogTopicConfig` creates a `UnwindowedChangelogTopicConfig` or a `WindowedChangelogTopicConfig` per the `isWindowStore` flag of the input `StateStoreFactory`.

Internally, `createChangelogTopicConfig` requests the input `StateStoreFactory` for `isWindowStore` flag.

Note	<p><code>isWindowStore</code> flag is enabled when a <code>StateStoreFactory</code> is created for a <code>WindowStoreBuilder</code>.</p>
------	---

If `isWindowStore` flag is enabled (`true`), `createChangelogTopicConfig` does the following:

1. Requests the input `StateStoreFactory` for `logConfig` and uses it to create a `WindowedChangelogTopicConfig` (for the input `name`)
2. Requests the input `StateStoreFactory` for `retentionPeriod` and uses it to requests the `WindowedChangelogTopicConfig` to `setRetentionMs`

If `isWindowStore` flag is disabled (`false`), `createChangelogTopicConfig` requests the input `StateStoreFactory` for `logConfig` and uses it to create a `UnwindowedChangelogTopicConfig` (for the input `name`).

Note

`createChangelogTopicConfig` is used exclusively when `InternalTopologyBuilder` is requested for `topic groups`.

Source Topics — `sourceTopicPattern` Method

```
Pattern sourceTopicPattern()
```

`sourceTopicPattern` returns the cached `source topics pattern` if available.

If not, `sourceTopicPattern` takes the subscribed topics from the `nodeToSourceTopics` internal registry and sorts them into ascending order (using natural ordering).

Before returning the `source topics pattern`, `sourceTopicPattern` `buildPatternForOffsetResetTopics` and saves the result in the `topicPattern` internal registry.

Note

`sourceTopicPattern` is used when:

- `StreamThread` is requested to run the main record processing loop and `enforceRebalance`
- `TaskManager` is requested to `updateSubscriptionsFromAssignment` and `updateSubscriptionsFromMetadata`

buildPatternForOffsetResetTopics Internal Method

```
Pattern buildPatternForOffsetResetTopics(
    Collection<String> sourceTopics,
    Collection<Pattern> sourcePatterns)
```

`buildPatternForOffsetResetTopics` ...FIXME

Note

`buildPatternForOffsetResetTopics` is used when `InternalTopologyBuilder` is requested to `resetTopicsPattern` and `sourceTopicPattern`.

setRegexMatchedTopicsToSourceNodes Internal Method

```
void setRegexMatchedTopicsToSourceNodes()
```

setRegexMatchedTopicsToSourceNodes ...FIXME

Note

`setRegexMatchedTopicsToSourceNodes` is used exclusively when `InternalTopologyBuilder` is requested to [updateSubscriptions](#).

updateSubscriptions Method

```
void updateSubscriptions(
    final SubscriptionUpdates subscriptionUpdates,
    final String logPrefix)
```

updateSubscriptions ...FIXME

Note

`updateSubscriptions` is used exclusively when `InternalTopologyBuilder` is requested to [updateSubscribedTopics](#).

updateSubscribedTopics Method

```
void updateSubscribedTopics(final Set<String> topics, final String logPrefix)
```

updateSubscribedTopics ...FIXME

Note

`updateSubscribedTopics` is used when `TaskManager` is requested to [updateSubscriptionsFromAssignment](#) and [updateSubscriptionsFromMetadata](#).

rewriteTopology Method

```
InternalTopologyBuilder rewriteTopology(final StreamsConfig config)
```

rewriteTopology ...FIXME

Note

`rewriteTopology` is used exclusively when `KafkaStreams` is [created](#).

adjust Internal Method

```
void adjust(final StreamsConfig config)
```

adjust ...FIXME

Note

adjust is used exclusively when InternalTopologyBuilder is requested to rewriteTopology.

InternalTopologyBuilder.AbstractNode

`InternalTopologyBuilder.AbstractNode` is the base implementation of the `TopologyDescription.Node` contract for topology nodes with the size of the (sub)topology rooted at this node (including the node itself).

`InternalTopologyBuilder.AbstractNode` takes a single `name` to be created.

`InternalTopologyBuilder.AbstractNode` has a `size` which is the size of the (sub)topology rooted at this node (including the node itself).

`InternalTopologyBuilder.AbstractNode` has **predecessors** and **successors** that are `TopologyDescription.Nodes` before and after the node in a topology (a node graph).

Table 1. InternalTopologyBuilder.AbstractNodes

AbstractNode	Description
Source	
Processor	
Sink	

Adding Predecessor — `addPredecessor` Method

```
void addPredecessor(TopologyDescription.Node predecessor)
```

`addPredecessor` simply adds a new predecessor node to the `predecessors` internal registry.

Note	<code>addPredecessor</code> is used exclusively when <code>InternalTopologyBuilder</code> is requested to <code>describe a sub-topology</code> (when requested to <code>describe itself</code>).
------	---

Adding Successor — `addSuccessor` Method

```
void addSuccessor(TopologyDescription.Node successor)
```

`addSuccessor` simply adds a new successor node to the `successors` internal registry.

Note	<code>addSuccessor</code> is used exclusively when <code>InternalTopologyBuilder</code> is requested to <code>describe a sub-topology</code> (when requested to <code>describe itself</code>).
------	---

InternalTopologyBuilder.Processor

`InternalTopologyBuilder.Processor` is...FIXME

InternalTopologyBuilder.Sink

`InternalTopologyBuilder.Sink` is...FIXME

InternalTopologyBuilder.Source

```
InternalTopologyBuilder.Source is...FIXME
```

NodeFactory Contract

`NodeFactory` is the abstraction of processor node factories that can build and describe a processor node.

`NodeFactory` takes the following to be created:

- Name of a processor to be built
- Node predecessors (by their names) that a processor will be a child of

Note	<code>NodeFactory</code> is a Java private static abstract class of <code>InternalTopologyBuilder</code> and cannot be created directly. It is created indirectly for the concrete processor node factories and only for and by <code>InternalTopologyBuilder</code> .
------	--

Table 1. NodeFactory Contract

Method	Description
<code>build</code>	<pre>ProcessorNode build()</pre> <p>Builds a processor node</p> <p>Used exclusively when <code>InternalTopologyBuilder</code> is requested to build a topology of processor nodes (aka <i>processor topology</i>)</p>
<code>describe</code>	<pre>AbstractNode describe()</pre> <p>Describes a processor node (using <code>AbstractNode</code>)</p> <p>Used exclusively when <code>InternalTopologyBuilder</code> is requested to <code>describeSubtopology</code> (when requested to <code>describe a topology</code>)</p>

Table 2. NodeFactories

NodeFactory	Description
<code>ProcessorNodeFactory</code>	
<code>SinkNodeFactory</code>	Factory of <code>SinkNodes</code>
<code>SourceNodeFactory</code>	

ProcessorNodeFactory

`ProcessorNodeFactory` is a concrete `NodeFactory` that can `build` a processor node for a `ProcessorSupplier` and optional `state stores`.

Note	<code>ProcessorNodeFactory</code> is a private class.
------	---

`ProcessorNodeFactory` is `created` exclusively when `InternalTopologyBuilder` is requested to `register a global state store` or `addProcessor`.

`ProcessorNodeFactory` manages **names of the state stores** that a `Processor` can be associated with (as part of a `ProcessorNode`).

`ProcessorNodeFactory` can have state stores registered (by the name) (which happens when `InternalTopologyBuilder` is requested to `register a global state store` or `connectProcessorAndStateStore`).

build Factory Method

<code>ProcessorNode build()</code>

`build` requests `ProcessorSupplier` to get a `Processor` and then creates a `ProcessorNode` with the `name`, the `processor` and the `stateStoreNames`.

Creating ProcessorNodeFactory Instance

`ProcessorNodeFactory` takes the following when created:

- Processor name
- Predecessor nodes (by name)
- `ProcessorSupplier`

SinkNodeFactory

`SinkNodeFactory` is a custom [NodeFactory](#) that can [build](#) and [describe](#) a [SinkNode](#).

`SinkNodeFactory` is [created](#) exclusively when `InternalTopologyBuilder` is requested to register a sink.

Building SinkNode — `build` Method

```
ProcessorNode build()
```

Note	<code>build</code> is part of NodeFactory Contract to build a processor node.
------	---

`build` simply creates a [SinkNode](#) (with the [name](#), the [key serializer](#), the [value serializer](#) and the [StreamPartitioner](#)).

Note	The name of the topic of the <code>SinkNode</code> is the topic with application id for an internal topic.
------	--

Creating SinkNodeFactory Instance

`SinkNodeFactory` takes the following when created:

- Node name
- Names of the predecessors
- Topic name
- Key `Serializer`
- Value `Serializer`
- [StreamPartitioner](#)

Describing SinkNode — `describe` Method

```
Sink describe()
```

Note	<code>describe</code> is part of NodeFactory Contract to describe a processor node.
------	---

`describe` ...FIXME

SourceNodeFactory — NodeFactory With No Predecessors

`SourceNodeFactory` is a [NodeFactory](#) that has no [predecessors](#).

`SourceNodeFactory` is [created](#) when `InternalTopologyBuilder` is requested to register a [global state store](#) or [source topic](#).

Note	<code>SourceNodeFactory</code> is an internal <code>private</code> class of <code>InternalTopologyBuilder</code> .
------	--

Creating SourceNodeFactory Instance

`SourceNodeFactory` takes the following when created:

- Node name
- Topic names
- Topic pattern
- [TimestampExtractor](#)
- Key deserializer
- Value deserializer

Building ProcessorNode — `build` Method

ProcessorNode	<code>build()</code>
---------------	----------------------

Note	<code>build</code> is part of NodeFactory Contract to build a processor node.
------	---

`build` creates a [SourceNode](#) with the [name](#) and the source topic names (as recorded in the [nodeToSourceTopics](#) internal registry).

Internally, `build` looks up the [node name](#) in the [nodeToSourceTopics](#) internal registry to find the topic names (the node is attached to).

`build` creates a [SourceNode](#) with the [name](#), the source topic names (possibly [decorated](#) if internal), and the other properties, i.e. the [timestampExtractor](#), the [keyDeserializer](#) and the [valDeserializer](#)

It is (hardly) possible that no topic names could be found, but when it happens, `build` creates a [SourceNode](#) with the [pattern](#) (and the other properties, i.e. the [name](#), the [timestampExtractor](#), the [keyDeserializer](#) and the [valDeserializer](#)).

InternalTopologyBuilder.TopologyDescription

`InternalTopologyBuilder.TopologyDescription` is...FIXME

Note	Don't get confused with TopologyDescription that is the contract of <code>InternalTopologyBuilder.TopologyDescription</code> .
------	--

GlobalStore

GlobalStore is...FIXME

StateStoreFactory

`StateStoreFactory` is the internal class of `InternalTopologyBuilder` for...FIXME

`StateStoreFactory` is [created](#) exclusively when `InternalTopologyBuilder` is requested to register a state store (as a `StoreBuilder`).

`StateStoreFactory` takes a single `StoreBuilder` to be created.

InternalStreamsBuilder

`InternalStreamsBuilder` is an internal entity that `StreamsBuilder` (of `Streams DSL — High-Level Stream Processing DSL`) uses to **build a topology** by adding nodes to the `stream graph` using the high-level stream operators.

`InternalStreamsBuilder` is **created** exclusively for `StreamsBuilder`.

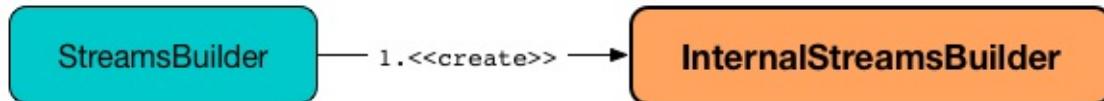


Figure 1. Creating InternalStreamsBuilder

`InternalStreamsBuilder` takes a `InternalTopologyBuilder` to be created.

```

import org.apache.kafka.streams.processor.internals.InternalTopologyBuilder
val intTopologyBuilder = new InternalTopologyBuilder

import org.apache.kafka.streams.kstream.internals.InternalStreamsBuilder
val intStreamsBuilder = new InternalStreamsBuilder(intTopologyBuilder)
  
```

`InternalStreamsBuilder` manages a `StreamsGraphNode` as the **root node** (of the `topology to build`). Every time `InternalStreamsBuilder` is requested to add a `stream`, a `table`, a `global table`, a `state store` or a `global state store` the root node **gets a corresponding child node added**. Once the topology is of a proper structure, `InternalStreamsBuilder` can simply be requested to **build it (with optional optimizations applied)**.

```

import org.apache.kafka.streams.kstream.internals.InternalStreamsBuilder
assert(intStreamsBuilder.isInstanceOf[InternalStreamsBuilder])

val topics = Seq("input")
import org.apache.kafka.streams.kstream.internals.ConsumedInternal
val intConsumed = new ConsumedInternal[String, String]()
import scala.collection.JavaConversions.-
// Add a new graph node (stream) to the root node
intStreamsBuilder.stream(topics, intConsumed)

intStreamsBuilder.buildAndOptimizeTopology

val root = intStreamsBuilder.root
import org.apache.kafka.streams.kstream.internals.graph.StreamsGraphNode
assert(root.isInstanceOf[StreamsGraphNode])
assert(root.children.size == 1)

// Let's explore what we built

val streamSourceNode = root.children.head
println(streamSourceNode)
/**
StreamSourceNode{topicNames=[input], topicPattern=null, consumedInternal=org.apache.kafka.streams.kstream.internals.ConsumedInternal@e1781} StreamsGraphNode{nodeName='KSTREAM-SOURCE-0000000000', buildPriority=0, hasWrittenToTopology=true, keyChangingOperation=false, valueChangingOperation=false, mergeNode=false, parentNodes=[root]}
*/

```

```

println(intTopologyBuilder.describe)
/**
Topologies:
  Sub-topology: 0
    Source: KSTREAM-SOURCE-0000000000 (topics: [input])
      --> none
*/

```

`InternalStreamsBuilder` is an [InternalNameProvider](#) and can be requested for a name for new processors and stores.

`InternalStreamsBuilder` uses `index` counter that starts at `0` and is incremented every time a new name for a [processor](#) or a [store](#) is requested. That makes all names unique and sources and stores uniquely identified by name.

Enable `ALL` logging level for `org.apache.kafka.streams.kstream.internals.InternalStreamsBuilder` logger to see what happens inside.

Add the following line to `log4j.properties`:

Tip

```
log4j.logger.org.apache.kafka.streams.kstream.internals.InternalStreamsBuilder=A
```

Refer to [Application Logging Using log4j](#).

Adding State Store to Topology — `addStateStore` Method

```
void addStateStore(  
    StoreBuilder builder)
```

`addStateStore` creates a `StateStoreNode` (with the given `StoreBuilder`) and adds it to the root node.

Note

`addStateStore` is used exclusively when `streamsBuilder` is requested to [add a state store to a topology](#).

Adding Stream to Topology (and Creating KStream) — `stream` Method

```
KStream<K, V> stream(  
    Collection<String> topics,  
    ConsumedInternal<K, V> consumed)  
KStream<K, V> stream(  
    Pattern topicPattern,  
    ConsumedInternal<K, V> consumed)
```

`stream` creates a new processor name with `KSTREAM-SOURCE` prefix.

`stream` creates a new `StreamSourceNode` and adds it to the root node.

In the end, `stream` returns a new instance of `KStreamImpl` (with the new processor name that is also used as the name of the single source node and the `repartitionRequired` flag off).

```

import org.apache.kafka.streams.processor.internals.InternalTopologyBuilder
val internalTopologyBuilder = new InternalTopologyBuilder()

import org.apache.kafka.streams.kstream.internals.InternalStreamsBuilder
val internalStreamsBuilder = new InternalStreamsBuilder(internalTopologyBuilder)

val topics = Seq("input")
import org.apache.kafka.streams.kstream.internals.ConsumedInternal
val consumed = new ConsumedInternal[String, String]()
import collection.JavaConverters.-
val kstream = internalStreamsBuilder.stream(topics.asJava, consumed)
scala> :type kstream
org.apache.kafka.streams.kstream.KStream[String, String]

import org.apache.kafka.streams.kstream.internals.KStreamImpl
assert(kstream.isInstanceOf[KStreamImpl[String, String]])

```

Note`stream` is used when:

- `StreamsBuilder` is requested to `stream`
- `KStreamImpl` is requested to `through`

Adding Table to Topology (and Creating KTable) — `table` Method

```

KTable<K, V> table(
    String topic,
    ConsumedInternal<K, V> consumed,
    MaterializedInternal<K, V, KeyValueStore<Bytes, byte[]>> materialized)

```

`table` ...FIXME**Note**`table` is used exclusively when `streamsBuilder` is requested to add a table to a topology.

Adding Global State Store to Topology — `addGlobalStore` Method

```

void addGlobalStore(
    StoreBuilder<KeyValueStore> storeBuilder,
    String topic,
    ConsumedInternal consumed,
    ProcessorSupplier stateUpdateSupplier)
void addGlobalStore(
    StoreBuilder<KeyValueStore> storeBuilder,
    String sourceName,
    String topic,
    ConsumedInternal consumed,
    String processorName,
    ProcessorSupplier stateUpdateSupplier)

```

`addGlobalStore` ...FIXME

Note

`addGlobalStore` is used exclusively when `StreamsBuilder` is requested to [addGlobalStore](#).

Adding Global Table to Topology (and Creating GlobalKTable) — `globalTable` Method

```

GlobalKTable<K, V> globalTable(
    String topic,
    ConsumedInternal<K, V> consumed,
    MaterializedInternal<K, V, KeyValueStore<Bytes, byte[]>> materialized)

```

`globalTable` creates a new [GlobalKTableImpl](#).

Internally, `globalTable` requests the `MaterializedInternal` to disable logging (regardless whether it was enabled initially or not).

Note

`GlobalKTables` use state stores with logging disabled.

`globalTable` then creates a [KeyValueStoreMaterializer](#) (with the input `MaterializedInternal` with logging disabled) and requests it to [materialize](#) (and create a `StoreBuilder`).

`globalTable` creates a `TableSourceNode` (with the `storeBuilder`, the `source processor name` with `KSTREAM-SOURCE-` prefix, and `isGlobalKTable` flag on)

`globalTable` adds the `TableSourceNode` to the `root` node.

In the end, `globalTable` creates a [GlobalKTableImpl](#) (with a new `KTableSourceValueGetterSupplier` and the `queryable` flag of the `MaterializedInternal`).

Note

`globalTable` is used exclusively when `StreamsBuilder` is requested to add a global table to a topology.

New Unique Processor Name — `newProcessorName` Method

```
String newProcessorName(  
    String prefix)
```

Note

`newProcessorName` is part of [InternalNameProvider Contract](#) to create a new unique name for a [processor](#).

`newProcessorName` simply takes the input `prefix` followed by the [index](#).

Note

The [index](#) counter is what makes it bound to a `InternalStreamsBuilder`.

```
import org.apache.kafka.streams.kstream.internals.InternalStreamsBuilder  
import org.apache.kafka.streams.processor.internals.InternalTopologyBuilder  
val newBuilder = new InternalStreamsBuilder(new InternalTopologyBuilder)  
  
val name = newBuilder.newProcessorName("PREFIX")  
scala> println(name)  
PREFIX0000000001
```

New Unique Store Name — `newStoreName` Method

```
String newStoreName(  
    String prefix)
```

Note

`newStoreName` is part of [InternalNameProvider Contract](#) to create a new unique name for a [state store](#).

`newStoreName` simply concatenates the input `prefix`, `STATE-STORE-` and the [index](#).

Note

The [index](#) counter is what makes it bound to a `InternalStreamsBuilder`.

```

import org.apache.kafka.streams.kstream.internals.InternalStreamsBuilder
import org.apache.kafka.streams.processor.internals.InternalTopologyBuilder
val newBuilder = new InternalStreamsBuilder(new InternalTopologyBuilder)

val name = newBuilder.newStoreName("PREFIX")
scala> println(name)
PREFIXSTATE-STORE-0000000001

```

Adding Child Node — `addGraphNode` Method

```

void addGraphNode(
    StreamsGraphNode parent,
    StreamsGraphNode child)
void addGraphNode(
    Collection<StreamsGraphNode> parents,
    StreamsGraphNode child)

```

`addGraphNode` simply requests the input `StreamsGraphNode` to add the given child node.

In the end, `addGraphNode` `maybeAddNodeForOptimizationMetadata`.

Note	<code>addGraphNode</code> is used in <i>many places</i> in <code>GroupedStreamAggregateBuilder</code> , <code>InternalStreamsBuilder</code> , <code>KGroupedTableImpl</code> , <code>KStreamImpl</code> , <code>KStreamImplJoin</code> , and <code>KTableImpl</code> .
------	--

Building Topology (with Optional Optimizations) — `buildAndOptimizeTopology` Method

```

void buildAndOptimizeTopology() (1)
void buildAndOptimizeTopology(
    Properties props)

```

1. For testing only; Uses `null` for the `Properties`

`buildAndOptimizeTopology` does `maybePerformOptimizations` (with the given `Properties`).

`buildAndOptimizeTopology ...FIXME`

Note	<code>buildAndOptimizeTopology</code> is used exclusively when <code>streamsBuilder</code> is requested to <code>build a topology</code> .
------	--

`maybeAddNodeForOptimizationMetadata` Internal Method

```
void maybeAddNodeForOptimizationMetadata(
    StreamsGraphNode node)
```

maybeAddNodeForOptimizationMetadata ...FIXME

Note

maybeAddNodeForOptimizationMetadata is used exclusively when InternalStreamsBuilder is requested to [adding a child node](#).

maybePerformOptimizations Internal Method

```
void maybePerformOptimizations(
    Properties props)
```

maybePerformOptimizations ...FIXME

Note

maybePerformOptimizations is used exclusively when InternalStreamsBuilder is requested to [build a topology \(with optional optimizations\)](#).

getKeyChangingParentNode Internal Method

```
StreamsGraphNode getKeyChangingParentNode(
    StreamsGraphNode repartitionNode)
```

getKeyChangingParentNode ...FIXME

Note

getKeyChangingParentNode is used exclusively when InternalStreamsBuilder is requested to [maybeAddNodeForOptimizationMetadata](#).

maybeOptimizeRepartitionOperations Internal Method

```
void maybeOptimizeRepartitionOperations()
```

maybeOptimizeRepartitionOperations ...FIXME

Note

maybeOptimizeRepartitionOperations is used exclusively when InternalStreamsBuilder is requested to [maybePerformOptimizations](#).

createRepartitionNode Internal Method

```
OptimizableRepartitionNode createRepartitionNode(
    String repartitionTopicName,
    Serde keySerde,
    Serde valueSerde)
```

createRepartitionNode ...FIXME

Note

`createRepartitionNode` is used when...FIXME

findParentNodeMatching Internal Method

```
StreamsGraphNode findParentNodeMatching(
    StreamsGraphNode startSeekingNode,
    Predicate<StreamsGraphNode> parentNodePredicate)
```

findParentNodeMatching ...FIXME

Note

`findParentNodeMatching` is used when...FIXME

getFirstRepartitionTopicName Internal Method

```
String getFirstRepartitionTopicName(
    Collection<OptimizableRepartitionNode> repartitionNodes)
```

getFirstRepartitionTopicName ...FIXME

Note

`getFirstRepartitionTopicName` is used when...FIXME

getRepartitionSerdes Internal Method

```
GroupedInternal getRepartitionSerdes(
    Collection<OptimizableRepartitionNode> repartitionNodes)
```

getRepartitionSerdes ...FIXME

Note

`getRepartitionSerdes` is used when...FIXME

maybeUpdateKeyChangingRepartitionNodeMap Internal Method

```
void maybeUpdateKeyChangingRepartitionNodeMap()
```

maybeUpdateKeyChangingRepartitionNodeMap ...FIXME

Note

maybeUpdateKeyChangingRepartitionNodeMap is used when...FIXME

optimizeKTableSourceTopics Internal Method

```
void optimizeKTableSourceTopics()
```

optimizeKTableSourceTopics ...FIXME

Note

optimizeKTableSourceTopics is used when...FIXME

StreamsGraphNode Contract — Graph Nodes that Write to Topology

`StreamsGraphNode` is the [abstraction](#) of [graph nodes](#) that can [write to a topology](#).

Table 1. StreamsGraphNode Contract (Abstract Methods Only)

Method	Description
<code>writeToTopology</code>	<pre>void writeToTopology(InternalTopologyBuilder topologyBuilder)</pre> <p>Used exclusively when <code>InternalStreamsBuilder</code> is requested to build a topology (when <code>StreamsBuilder</code> is requested to build a topology)</p>

`StreamsGraphNode` is used in [InternalStreamsBuilder](#) to [build a topology](#).

`InternalStreamsBuilder` uses the [root node](#) internally that all child `StreamsGraphNodes` can be [added to](#).

```
import org.apache.kafka.streams.processor.internals.InternalTopologyBuilder
val intTopologyBuilder = new InternalTopologyBuilder

import org.apache.kafka.streams.kstream.internals.InternalStreamsBuilder
val intStreamsBuilder = new InternalStreamsBuilder(intTopologyBuilder)

import org.apache.kafka.streams.state.Stores
val storeName = "input-stream"
val lruMapSupplier = Stores.lruMap(storeName, 5)
import org.apache.kafka.common.serialization.Serdes
import org.apache.kafka.streams.state.{KeyValueStore, StoreBuilder}
val storeBuilder = Stores.keyValueStoreBuilder(
    lruMapSupplier,
    Serdes.Long(),
    Serdes.Long())

intStreamsBuilder.addStateStore(storeBuilder)

val root = intStreamsBuilder.root
import org.apache.kafka.streams.kstream.internals.graph.StreamsGraphNode
assert(root.isInstanceOf[StreamsGraphNode])
scala> println(root.toString)
StreamsGraphNode{nodeName='root', buildPriority=null, hasWrittenToTopology=true, keyChangingOperation=false, valueChangingOperation=false, mergeNode=false, parentNodes=[]}

val children = root.children
assert(children.isInstanceOf[java.util.Collection[StreamsGraphNode]])

import scala.collection.JavaConverters.-
val stateStoreNode = children.asScala.head
import org.apache.kafka.streams.kstream.internals.graph.StateStoreNode
assert(stateStoreNode.isInstanceOf[StateStoreNode])
scala> println(stateStoreNode)
StateStoreNode{ name='input-stream', logConfig={}, loggingEnabled='true'}

// Triggers StreamsGraphNode.writeToTopology of all nodes
intStreamsBuilder.buildAndOptimizeTopology

intTopologyBuilder.setApplicationId("required to complete optimization")

assert(intTopologyBuilder.allStateStoreName.asScala.head == storeName)
```

Table 2. StreamsGraphNodes (Direct Implementations and Extensions Only)

StreamsGraphNode	Description
BaseJoinProcessorNode	Base of KTableKTableJoinNode and StreamStreamJoinNode
BaseRepartitionNode	Base of GroupedTableOperationRepartitionNode and OptimizableRepartitionNode
ProcessorGraphNode	Represents stateless operators in KStreamImpl and KTableImpl Base of StatefulProcessorNode
StateStoreNode	Represents StreamsBuilder.addStateStore operator Base of GlobalStoreNode
StreamSinkNode	Represents KStreamImpl.to operator
StreamSourceNode	Represents StreamsBuilder.stream and KStreamImpl.through operators Base of TableSourceNode
StreamTableJoinNode	Represents KStreamImpl.join and KStreamImpl.leftJoin operators
TableProcessorNode	Represents KTableImpl.filter , KTableImpl.filterNot , KTableImpl.mapValues , and KTableImpl.transformValues operators

`StreamsGraphNode` takes a single **node name** to be created.

Note

`StreamsGraphNode` is a Java abstract class and cannot be [created](#) directly. It is created indirectly for the [concrete StreamsGraphNodes](#) and as the [root node of InternalStreamsBuilder](#).

isValueChangingOperation Method

```
boolean isValueChangingOperation()
```

`isValueChangingOperation` simply returns the `valueChangingOperation` internal flag.

Note

`isValueChangingOperation` is used exclusively when `InternalStreamsBuilder` is requested to [find the key-changing parent node](#).

setValueChangingOperation Method

```
void setValueChangingOperation(  
    boolean valueChangingOperation)
```

`setValueChangingOperation` simply sets the `valueChangingOperation` internal flag to the given `valueChangingOperation` value.

Note	<code>setValueChangingOperation</code> is used when <code>KStreamImpl</code> is requested to <code>mapValues</code> , <code>flatMapValues</code> , <code>doTransformValues</code> , and <code>doFlatTransformValues</code>
------	--

Adding Child Node — addChild Method

```
void addChild(  
    StreamsGraphNode childNode)
```

`addChild` ...FIXME

Note	<code>addChild</code> is used when <code>InternalStreamsBuilder</code> is requested to add a child node and <code>maybeOptimizeRepartitionOperations</code> .
------	---

Describing Itself (Textual Representation) — toString Method

```
String toString()
```

Note	<code>toString</code> is part of the <code>java.lang.Object</code> for a string representation of the object.
------	---

`toString` ...FIXME

Internal Properties

Name	Description
valueChangingOperation	<p>Represents whether the <code>StreamsGraphNode</code> is value-changing and changes record values (<code>true</code>) or not (<code>false</code>)</p> <ul style="list-style-type: none">• Set via setValueChangingOperation• Available as isValueChangingOperation <p>Used when <code>streamsGraphNode</code> is requested for the textual representation</p>

BaseJoinProcessorNode Contract

BaseJoinProcessorNode is...FIXME

BaseRepartitionNode Contract

BaseRepartitionNode is the base of [StreamsGraphNodes](#) that...FIXME

ProcessorParameters

`ProcessorParameters` represents a [ProcessorSupplier](#) and a processor name to use to add a new processor (to [InternalTopologyBuilder](#)).

`ProcessorParameters` abstracts away different number of processors of [StreamsGraphNodes](#) (i.e. [KTableKTableJoinNode](#), [OptimizableRepartitionNode](#), [ProcessorGraphNode](#), [StatefulProcessorNode](#), [StreamStreamJoinNode](#), [StreamTableJoinNode](#), [TableProcessorNode](#), and [TableSourceNode](#)) when requested to write to a topology (when `InternalStreamsBuilder` is requested to build a topology when `StreamsBuilder` is requested to build a topology).

GlobalStoreNode

GlobalStoreNode is...FIXME

writeToTopology Method

```
void writeToTopology(final InternalTopologyBuilder topologyBuilder)
```

Note `writeToTopology` is part of the [StreamsGraphNode Contract](#) to...FIXME.

`writeToTopology` requests the [StoreBuilder](#) to disable change-logging on state stores.

`writeToTopology` then requests the given `InternalTopologyBuilder` to addGlobalStore.

GroupedTableOperationRepartitionNode

`GroupedTableOperationRepartitionNode` is a [StreamsGraphNode](#) (indirectly as a [BaseRepartitionNode](#)) that is used by `KGroupedTableImpl` for all supported streaming operators, i.e. `reduce`, `count` and `aggregate`.

```
// Building a topology using Streams DSL
// Scala API for Kafka Streams
import org.apache.kafka.streams.scala._
import ImplicitConversions._
import Serdes._

val builder = new StreamsBuilder
builder
    .table[String, String]("input")
    .groupBy[String, String]((k,v) => (k,v)) // creates a KGroupedTableImpl
    .count // creates a GroupedTableOperationRepartitionNode
val topology = builder.build
// Note the "-repartition" topic
scala> println(topology.describe)
Topologies:
Sub-topology: 0
  Source: KSTREAM-SOURCE-0000000001 (topics: [input])
    --> KTABLE-SOURCE-0000000002
  Processor: KTABLE-SOURCE-0000000002 (stores: [input-STATE-STORE-0000000000])
    --> KTABLE-SELECT-0000000003
    <-- KSTREAM-SOURCE-0000000001
  Processor: KTABLE-SELECT-0000000003 (stores: [])
    --> KSTREAM-SINK-0000000005
    <-- KTABLE-SOURCE-0000000002
  Sink: KSTREAM-SINK-0000000005 (topic: KTABLE-AGGREGATE-STATE-STORE-0000000004-repartition)
    <-- KTABLE-SELECT-0000000003

Sub-topology: 1
  Source: KSTREAM-SOURCE-0000000006 (topics: [KTABLE-AGGREGATE-STATE-STORE-0000000004-repartition])
    --> KTABLE-AGGREGATE-0000000007
  Processor: KTABLE-AGGREGATE-0000000007 (stores: [KTABLE-AGGREGATE-STATE-STORE-0000000004])
    --> KTABLE-MAPVALUES-0000000008
    <-- KSTREAM-SOURCE-0000000006
  Processor: KTABLE-MAPVALUES-0000000008 (stores: [])
    --> none
    <-- KTABLE-AGGREGATE-0000000007
```

`GroupedTableOperationRepartitionNode` is [created](#) exclusively when `GroupedTableOperationRepartitionNodeBuilder` is requested to [build one](#).

`GroupedTableOperationRepartitionNode` takes the following when created:

- **Name** of the node
- **Key Serde** (Apache Kafka's [Serde](#) for record keys)
- **Value Serde** (Apache Kafka's [Serde](#) for record values)
- **Sink Name**
- **Source Name**
- **Name of the repartition topic**
- `ProcessorParameters`

`GroupedTableOperationRepartitionNode` allows creating `GroupedTableOperationRepartitionNodeBuilders` using `groupedTableOperationNodeBuilder` factory method.

```
GroupedTableOperationRepartitionNodeBuilder<K1, V1> groupedTableOperationNodeBuilder()
```

`groupedTableOperationNodeBuilder` is used exclusively when `KGroupedTableImpl` is requested to [createRepartitionNode](#) (for all supported streaming operators, i.e. [reduce](#), [count](#) and [aggregate](#)).

writeToTopology Method

```
void writeToTopology(final InternalTopologyBuilder topologyBuilder)
```

Note	<code>writeToTopology</code> is part of the StreamsGraphNode Contract to...FIXME.
------	---

`writeToTopology` ...FIXME

GroupedTableOperationRepartitionNodeBuilder

`GroupedTableOperationRepartitionNodeBuilder` is...FIXME

`GroupedTableOperationRepartitionNodeBuilder` is [created](#) when...FIXME

`GroupedTableOperationRepartitionNodeBuilder` takes no arguments when created.

Note

`GroupedTableOperationRepartitionNodeBuilder` is a Java static inner class of [GroupedTableOperationRepartitionNode](#) and can only be created using the [GroupedTableOperationRepartitionNode.groupedTableOperationNodeBuilder](#) factory method.

Creating GroupedTableOperationRepartitionNode Instance — `build` Method

`GroupedTableOperationRepartitionNode<K, V> build()`

`build` simply creates a [GroupedTableOperationRepartitionNode](#).

Note

`build` is used exclusively when `KGroupedTableImpl` is requested to [createRepartitionNode](#) (for all supported streaming operators, i.e. [reduce](#), [count](#) and [aggregate](#)).

KTableKTableJoinNode

`KTableKTableJoinNode` is a concrete `StreamsGraphNode` (as a `BaseJoinProcessorNode`) that is created (using `KTableKTableJoinNodeBuilder.build` method) for `KTableImpl.join`, `KTableImpl.leftJoin` and `KTableImpl.outerJoin` operators.

In other words, `KTableKTableJoinNode` represents `KTableImpl.join`, `KTableImpl.leftJoin` and `KTableImpl.outerJoin` operators.

Creating KTableKTableJoinNode Instance

`KTableKTableJoinNode` takes the following to be created:

- Node name
- `ProcessorParameters<K, Change<V1>>` of this join side
- `ProcessorParameters<K, Change<V2>>` of the other join side
- `ProcessorParameters<K, Change<VR>>` of the join merger
- Name of this join side
- Name of the other join side
- Key `Serde` (`Serde<K>`)
- Value `Serde` (`Serde<VR>`)
- Names of the state stores of this join side
- Names of the state stores of the other join side
- `StoreBuilder` of `TimestampedKeyValueStore` (`StoreBuilder<TimestampedKeyValueStore<K, VR>>`)

writeToTopology Method

```
void writeToTopology(
    InternalTopologyBuilder topologyBuilder)
```

Note	<code>writeToTopology</code> is part of the <code>StreamsGraphNode Contract</code> to...FIXME.
------	--

`writeToTopology` ...FIXME

Creating KTableKTableJoinNode — build Method

```
KTableKTableJoinNode<K, V1, V2, VR> build()
```

build ...FIXME

Note

`build` is used exclusively when `KTableImpl` is requested to `doJoin` (for `KTableImpl.join`, `KTableImpl.leftJoin` and `KTableImpl.outerJoin` operators).

OptimizableRepartitionNode

OptimizableRepartitionNode is...FIXME

writeToTopology Method

```
void writeToTopology(final InternalTopologyBuilder topologyBuilder)
```

Note	writeToTopology is part of the StreamsGraphNode Contract to...FIXME.
------	--

writeToTopology ...FIXME

ProcessorGraphNode

`ProcessorGraphNode` is a [graph node](#) for stateless operators in [KStreamImpl](#) and [KTableImpl](#).

When requested to [writeToTopology](#), `ProcessorGraphNode` simply requests the given [InternalTopologyBuilder](#) to add a processor (with the name and the [ProcessorSupplier](#) as defined by the given [ProcessorParameters](#)).

`ProcessorGraphNode` is [created](#) when:

- `KStreamImpl` is requested to [filter](#), [filterNot](#), [internalSelectKey](#) (for [selectKey](#) and [groupBy](#) operators), [map](#), [mapValues](#), [print](#), [flatMap](#), [flatMapValues](#), [branch](#), [merge](#), [foreach](#), [peek](#)
- `KTableImpl` is requested to [toStream](#) and [groupBy](#)
- `KStreamImplJoin` is requested to [join](#) (when `KStreamImpl` is requested to [doJoin](#) for [join](#), [outerJoin](#) and [leftJoin](#) join operators)

`ProcessorGraphNode` takes the following to be created:

- Node name
- `ProcessorParameters<K, V>`
- `repartitionRequired` flag (default: `false`)

writeToTopology Method

```
void writeToTopology(final InternalTopologyBuilder topologyBuilder)
```

Note	<code>writeToTopology</code> is part of the StreamsGraphNode Contract to...FIXME.
------	---

`writeToTopology` requests the given `InternalTopologyBuilder` to add a processor with the name and the [ProcessorSupplier](#) as defined by the given [ProcessorParameters](#).

StatefulProcessorNode

`StatefulProcessorNode` is a concrete `StreamsGraphNode` (as a `ProcessorGraphNode`) that represents stateful operators of the following:

- `KGroupedStreamImpl`
- `KGroupedTableImpl`
- `KStreamImpl`
- `KTableImpl`

`StatefulProcessorNode` is `created` when:

- `KStreamImpl` is requested to `KStreamImpl.transform` and `KStreamImpl.flatTransform`, `KStreamImpl.transformValues`, `KStreamImpl.flatTransformValues`, `KStreamImpl.process`
- `GroupedStreamAggregateBuilder` is requested to `build` (for `KStream.groupBy` and `KStream.groupByKey` streaming operators, incl. `windowedBy` operator with a `Windows` or a `SessionWindows`)
- `KGroupedTableImpl` is requested to `doAggregate` (for `KGroupedTableImpl.aggregate`, `KGroupedTableImpl.count`, and `KGroupedTableImpl.reduce` operators)
- `KTableImpl` is requested to `suppress`

`StatefulProcessorNode` extends the parent `ProcessorGraphNode` with the two state-related parameters - the `state store names` and a `StoreBuilder`.

Creating StatefulProcessorNode Instance

`StatefulProcessorNode` takes the following to be created:

- Node name
- `ProcessorParameters<K, V>`
- Names of the state stores or a `StoreBuilder`

writeToTopology Method

```
void writeToTopology(
    InternalTopologyBuilder topologyBuilder)
```

Note

`writeToTopology` is part of the [StreamsGraphNode Contract](#) to...FIXME.

`writeToTopology` requests the given [InternalTopologyBuilder](#) to add a processor (with the name and the [ProcessorSupplier](#) as defined by the [ProcessorParameters](#)).

With [state store names](#) given, `writeToTopology` requests the given [InternalTopologyBuilder](#) to connect them with the processor.

With a [StoreBuilder](#) given, `writeToTopology` requests the given [InternalTopologyBuilder](#) to add a state store (with the [storeBuilder](#) and the processor).

StateStoreNode

`StateStoreNode` is a concrete `StreamsGraphNode` that is `created` exclusively when `InternalStreamsBuilder` is requested to `add a state store to a topology`.

In other words, `StateStoreNode` represents `StreamsBuilder.addStateStore` operator.

`StateStoreNode` takes a single `StoreBuilder` to be created.

`writeToTopology` Method

```
void writeToTopology(  
    InternalTopologyBuilder topologyBuilder)
```

Note

`writeToTopology` is part of the `StreamsGraphNode Contract` to...FIXME.

`writeToTopology` simply requests the given `InternalTopologyBuilder` to `add a state store` (as the `StoreBuilder`).

StreamSinkNode

`StreamSinkNode` is a `StreamsGraphNode` that is `created` exclusively when `KStreamImpl` is requested to add a `StreamSinkNode` to a node graph.

In other words, `StreamSinkNode` represents `KStreamImpl.to` operator.

`StreamSinkNode` takes the following when created:

- **Name** of the node
- `TopicNameExtractor`
- `ProducedInternal`

writeToTopology Method

```
void writeToTopology(final InternalTopologyBuilder topologyBuilder)
```

Note `writeToTopology` is part of the `StreamsGraphNode Contract` to...FIXME.

`writeToTopology` simply requests the given `InternalTopologyBuilder` to `addSink` (per `ProducedInternal`).

`writeToTopology` creates a new `WindowedStreamPartitioner` as the `StreamPartitioner` when no `streamPartitioner` is specified and the `key serializer` is a `WindowedSerializer`.

StreamSourceNode

`StreamSourceNode` is a `StreamsGraphNode` that is `created` when `InternalStreamsBuilder` is requested to add a `KStream` to a topology.

In other words, `StreamSourceNode` represents `StreamsBuilder.stream` and `KStreamImpl.through` operators.

`StreamSourceNode` takes the following when created:

- **Name** of the node
- **Topic names** (`Collection<String>`) or **topic pattern** (Java's `java.util.regex.Pattern`)
- `ConsumedInternal`

writeToTopology Method

```
void writeToTopology(final InternalTopologyBuilder topologyBuilder)
```

Note	<code>writeToTopology</code> is part of the <code>StreamsGraphNode Contract</code> to...FIXME.
------	--

`writeToTopology` simply requests the given `InternalTopologyBuilder` to `addSource` (per `ConsumedInternal` and `topicPattern` or `topicNames`).

StreamStreamJoinNode

`StreamStreamJoinNode` is a concrete `StreamsGraphNode` (as a `BaseJoinProcessorNode`) that is created (using `StreamStreamJoinNode.build` method) for `KStreamImpl.join`, `KStreamImpl.leftJoin`, and `KStreamImpl.outerJoin` operators.

In other words, `StreamStreamJoinNode` represents `KStreamImpl.join`, `KStreamImpl.leftJoin`, and `KStreamImpl.outerJoin` operators.

Creating StreamStreamJoinNode Instance

`StreamStreamJoinNode` takes the following to be created:

- Node name
- `ValueJoiner<? super V1, ? super V2, ? extends VR>`
- `ProcessorParameters<K, V1>` of this join side
- `ProcessorParameters<K, V2>` of the other join side
- `ProcessorParameters<K, VR>` of the join merger
- `ProcessorParameters<K, V1>`
- `ProcessorParameters<K, V2>`
- `StoreBuilder` of `WindowStore` (`StoreBuilder<WindowStore<K, V1>>`)
- `StoreBuilder` of `WindowStore` (`StoreBuilder<WindowStore<K, V2>>`)
- `Joined<K, V1, V2>`

writeToTopology Method

```
void writeToTopology(
    InternalTopologyBuilder topologyBuilder)
```

Note	<code>writeToTopology</code> is part of the <code>StreamsGraphNode Contract</code> to...FIXME.
------	--

`writeToTopology` ...FIXME

Creating StreamStreamJoinNode — build Method

```
StreamStreamJoinNode<K, V1, V2, VR> build()
```

`build` simply creates a new `StreamStreamJoinNode`.

Note

`build` is used exclusively when `KTableImpl` is requested to `doJoin` (for `KTableImpl.join`, `KTableImpl.leftJoin` and `KTableImpl.outerJoin` operators).

StreamTableJoinNode

`StreamTableJoinNode` is a concrete `StreamsGraphNode` that represents (and is created for) `KStreamImpl.join` and `KStreamImpl.leftJoin` operators.

Creating StreamTableJoinNode Instance

`StreamTableJoinNode` takes the following to be created:

- Node name
- `ProcessorParameters<K, V>`
- Names of the state stores
- Node name of the other join side

`writeToTopology` Method

```
void writeToTopology(  
    InternalTopologyBuilder topologyBuilder)
```

Note

`writeToTopology` is part of the `StreamsGraphNode Contract` to...FIXME.

`writeToTopology` ...FIXME

TableProcessorNode

`TableProcessorNode` is a concrete `StreamsGraphNode` that is created when `KTableImpl` is requested to `doFilter`, `doMapValues`, and `doTransformValues`.

In other words, `TableProcessorNode` represents `KTableImpl.filter`, `KTableImpl.filterNot`, `KTableImpl.mapValues`, and `KTableImpl.transformValues` operators.

Creating TableProcessorNode Instance

`TableProcessorNode` takes the following to be created:

- Node name
- `ProcessorParameters<K, V>`
- `StoreBuilder` of `TimestampedKeyValueStore` (`StoreBuilder<TimestampedKeyValueStore<K, V>>`)
- Names of state stores

Note	<code>Names of state stores</code> can only be specified when <code>TableProcessorNode</code> is created when <code>KTableImpl</code> is requested to <code>doTransformValues</code> (when requested to <code>transformValues</code>).
------	---

writeToTopology Method

```
void writeToTopology(
    InternalTopologyBuilder topologyBuilder)
```

Note	<code>writeToTopology</code> is part of the <code>StreamsGraphNode Contract</code> to...FIXME.
------	--

`writeToTopology` requests the given `InternalTopologyBuilder` to add a processor (with the `processorName` and `processorSupplier` of the given `ProcessorParameters`).

`writeToTopology` requests the given `InternalTopologyBuilder` to connect the processor with the state stores if there are any `store names` specified.

`writeToTopology` requests the given `InternalTopologyBuilder` to add a state store if a `StoreBuilder` is specified.

TableSourceNode

`TableSourceNode` is a concrete `StreamsGraphNode` (as a `StreamSourceNode`) that is created (using `StreamStreamJoinNode.build` method) when `InternalStreamsBuilder` is requested to add a `KTable` and a `GlobalKTable` to a topology.

In other words, `TableSourceNode` represents `StreamsBuilder.table` and `StreamsBuilder.globalTable` operators.

Creating TableSourceNode Instance

`TableSourceNode` takes the following to be created:

- Name of the node
- Name of the source
- Name of the topic
- `ConsumedInternal`
- `MaterializedInternal`
- `ProcessorParameters<K, V>`
- `isGlobalKTable` flag

writeToTopology Method

```
void writeToTopology(  
    InternalTopologyBuilder topologyBuilder)
```

Note	<code>writeToTopology</code> is part of the <code>StreamsGraphNode Contract</code> to...FIXME.
------	--

`writeToTopology` creates a `KeyValueStoreMaterializer` and requests it to `materialize` (to a `StoreBuilder` of `KeyValueStore`).

`writeToTopology` branches off per `isGlobalKTable` flag.

When the `isGlobalKTable` flag is enabled (`true`), `writeToTopology` requests the given `InternalTopologyBuilder` to add a global key-value state store (to a topology).

When the `isGlobalKTable` flag is disabled (`false`), `writeToTopology` ...FIXME

Creating TableSourceNode — `build` Method

```
TableSourceNode<K, V> build()
```

`build` simply creates a new `TableSourceNode`.

Note

`build` is used exclusively when `InternalStreamsBuilder` is requested to `table` and `globalTable`.

KStreamAggProcessorSupplier Contract

`KStreamAggProcessorSupplier` is the extension contract of `ProcessorSuppliers` that allows implementations for `views` and to turn `sendingOldValues` flag on.

```
package org.apache.kafka.streams.kstream.internals;

interface KStreamAggProcessorSupplier<K, RK, V, T> extends ProcessorSupplier<K, V> {
    void enableSendingOldValues();
    KTableValueGetterSupplier<RK, T> view();
}
```

Table 1. KStreamAggProcessorSupplier Contract

Method	Description
<code>enableSendingOldValues</code>	Used when...FIXME
<code>view</code>	Used when...FIXME

Table 2. KStreamAggProcessorSuppliers

KStreamAggProcessorSupplier	Description
<code>KStreamWindowReduce</code>	
<code>KStreamAggregate</code>	
<code>KStreamSessionWindowAggregate</code>	
<code>KStreamReduce</code>	
<code>KStreamWindowAggregate</code>	

KStreamBranch — ProcessorSupplier of KStreamBranchProcessors

`KStreamBranch` is a custom `ProcessorSupplier` of `KStreamBranchProcessors` for `KStream.branch` operator.

```
// Scala API for Kafka Streams
import org.apache.kafka.streams.scala._
import ImplicitConversions._
import Serdes._

val builder = new StreamsBuilder
def alwaysTrue(k: String, v: String) = true
builder
  .stream[String, String]("input")
  .branch(alwaysTrue)
val topology = builder.build
scala> println(topology.describe)
Topologies:
Sub-topology: 0
  Source: KSTREAM-SOURCE-0000000000 (topics: [input])
    --> KSTREAM-BRANCH-0000000001
  Processor: KSTREAM-BRANCH-0000000001 (stores: [])
    --> KSTREAM-BRANCHCHILD-0000000002
    <-- KSTREAM-SOURCE-0000000000
  Processor: KSTREAM-BRANCHCHILD-0000000002 (stores: [])
    --> none
    <-- KSTREAM-BRANCH-0000000001
```

`KStreamBranch` takes the following to be created:

- `Predicate<K, V>[]`
- Child nodes

`KStreamBranch` is `created` exclusively when `KStreamImpl` is requested to `branch`.

Note	<code>KStreamImpl</code> is the default <code>KStream</code> .
------	--

When requested for a Processor, `KStreamBranch` gives a new `KStreamBranchProcessor`.

KStreamBranchProcessor

`KStreamBranchProcessor` is a custom record processor (indirectly as `AbstractProcessor`) that allows for forwarding a record to exactly one of the child processors (*branching on them*).

When requested to [process a record](#), `KStreamBranchProcessor` walks over the [predicates](#) and requests each and every predicate to test the record. When `true`, `process` requests the `ProcessorContext` to [forward the record](#) to a corresponding [child processor](#).

Note

`process` requests the predicates until positive is found or finishes without forwarding a record.

KStreamFilterProcessor

KStreamFilterProcessor is...FIXME

KStreamFilter — ProcessorSupplier of KStreamFilterProcessors for filter and filterNot Streaming Operators

`KStreamFilter` is a [ProcessorSupplier](#) of [KStreamFilterProcessors](#).

`KStreamFilter` is [created](#) when `KStreamImpl` is requested to execute the following streaming operators:

- `KStreamImpl.filter`
- `KStreamImpl.filterNot`
- `KStreamImpl.createRepartitionedSource`

`KStreamFilter` takes the following when created:

- `Predicate` (of `k` keys and `v` values)
- `filterNot` flag

`KStreamFilter` creates a new [KStreamFilterProcessor](#) whenever requested to supply one.

`KStreamFilterProcessor` is an [AbstractProcessor](#) that requests the [ProcessorContext](#) to forward a record (to downstream processors) only when the [Predicate](#) is met.

KStreamJoinWindowProcessor

`KStreamJoinWindowProcessor` is a [processor](#) that is used for

??? `KStream.transform` stateful operator.

`KStreamJoinWindowProcessor` is [created](#) when...FIXME

`KStreamJoinWindowProcessor` takes no arguments when created.

Initializing Processor— `init` Method

```
void init(ProcessorContext context)
```

Note

`init` is part of the [Processor Contract](#) to initialize the [processor](#).

`init` ...FIXME

Processing Record— `process` Method

```
void process(K key, V value)
```

Note

`process` is part of the [Processor Contract](#) to process a single record.

`process` ...FIXME

KStreamPeek — ProcessorSupplier of KStreamPeekProcessors

`KStreamPeek` is a custom [ProcessorSupplier](#) of [KStreamPeekProcessors](#) for [KStream.foreach](#) and [KStream.peek](#) operators.

```
// Scala API for Kafka Streams
import org.apache.kafka.streams.scala._
import ImplicitConversions._
import Serdes._

val builder = new StreamsBuilder
builder
    .stream[String, String]("input")
    .peek { (k,v) => println(s"($k, $v)") }
    .foreach { (k,v) => println(s"($k, $v)") }
val topology = builder.build
scala> println(topology.describe)
Topologies:
Sub-topology: 0
  Source: KSTREAM-SOURCE-0000000000 (topics: [input])
    --> KSTREAM-PEEK-0000000001
  Processor: KSTREAM-PEEK-0000000001 (stores: [])
    --> KSTREAM-FOREACH-0000000002
    <-- KSTREAM-SOURCE-0000000000
  Processor: KSTREAM-FOREACH-0000000002 (stores: [])
    --> none
    <-- KSTREAM-PEEK-0000000001
```

`KStreamPeek` takes the following to be created:

- `ForeachAction`
- `forwardDownStream` flag

`KStreamPeek` is [created](#) when `KStreamImpl` is requested to [foreach](#) (the `forwardDownStream` flag is disabled) and [peek](#) (the `forwardDownStream` flag is enabled).

Note	<code>KStreamImpl</code> is the default <code>KStream</code> .
------	--

When [requested for a Processor](#), `KStreamPeek` gives a new `KStreamPeekProcessor`.

KStreamPeekProcessor

`KStreamPeekProcessor` is a custom record processor (indirectly as `AbstractProcessor`) that allows for executing an action with records (*peeks at them*).

When requested to process a record, `KStreamPeekProcessor` executes the `ForeachAction` with the record. If the `forwardDownStream` flag is enabled, `KStreamPeekProcessor` requests the `ProcessorContext` to forward the record downstreams.

KStreamPassThrough

KStreamPassThrough is...FIXME

KStreamSessionWindowAggregateProcessor

`KStreamSessionWindowAggregateProcessor` is a concrete [stream processor](#) that...FIXME

`KStreamSessionWindowAggregateProcessor` is created exclusively when

`KStreamSessionWindowAggregate` is requested to [supply a stream processor](#).

Note

`KStreamSessionWindowAggregateProcessor` is a private class of [KStreamSessionWindowAggregate](#).

Table 1. KStreamSessionWindowAggregateProcessor's Internal Properties (e.g. Registries, Counters and Flags)

Name	Description
<code>store</code>	SessionStore Used when...FIXME
<code>tupleForwarder</code>	TupleForwarder of windowed keys Used when...FIXME

Processing Single Record — `process` Method

```
void process(final K key, final V value)
```

Note

`process` is part of [Processor Contract](#) to...FIXME.

`process` ...FIXME

Initializing Stream Processor — `init` Method

```
void init(ProcessorContext context)
```

Note

`init` is part of [Processor Contract](#) to initialize a stream processor.

`init` requests the input `ProcessorContext` for the [SessionStore](#) (by the `storeName` of the owning `KStreamSessionWindowAggregate`) and sets it as the current [SessionStore](#).

`init` creates a `ForwardingCacheFlushListener` (with the `sendOldValues` flag of the owning `KStreamSessionWindowAggregate`).

In the end, `init` creates a `TupleForwarder` (with the `SessionStore`, the `ForwardingCacheFlushListener` and the `sendOldValues` flag of the owning `KStreamSessionWindowAggregate`).

KStreamSessionWindowAggregate — ProcessorSupplier of KStreamSessionWindowAggregateProcessors

`KStreamSessionWindowAggregate` is a [KStreamAggProcessorSupplier](#) that...FIXME

`KStreamSessionWindowAggregate` is [created](#) when:

- `KGroupedStreamImpl` is requested to [aggregate](#)
- `SessionWindowedKStreamImpl` is requested to [aggregate](#) and [doAggregate](#)

`KStreamSessionWindowAggregate` uses `sendOldValues` flag that is disabled (`false`) by default and can be [enabled](#).

get Method

```
Processor<K, V> get()
```

Note	<code>get</code> is part of ProcessorSupplier Contract to supply a stream processor.
------	--

`get` ...FIXME

Creating KStreamSessionWindowAggregate Instance

`KStreamSessionWindowAggregate` takes the following when created:

- `SessionWindows`
- Store name
- `Initializer<T>`
- `Aggregator<? super K, ? super V, T>`
- `Merger<? super K, T>`

enableSendingOldValues Method

```
void enableSendingOldValues()
```

Note

`enableSendingOldValues` is part of [KStreamAggProcessorSupplier Contract](#) to...
FIXME.

`enableSendingOldValues` simply turns the `sendOldValues` flag on.

mergeSessionWindow Internal Method

```
SessionWindow mergeSessionWindow(final SessionWindow one, final SessionWindow two)
```

`mergeSessionWindow` ...FIXME

Note

`mergeSessionWindow` is used when...FIXME

KStreamTransformProcessor

`KStreamTransformProcessor` is a [processor](#) that is used for `KStream.transform` stateful operator.

`KStreamTransformProcessor` is [created](#) when `KStreamTransform` is requested for a [processor](#).

`KStreamTransformProcessor` takes a single [Transformer](#) when created.

Closing Processor— `close` Method

```
void close()
```

Note `close` is part of the [Processor Contract](#) to close the [processor](#).

`close` ...FIXME

Initializing Processor— `init` Method

```
void init(ProcessorContext context)
```

Note `init` is part of the [Processor Contract](#) to initialize the [processor](#).

`init` ...FIXME

Processing Record— `process` Method

```
void process(K1 key, V1 value)
```

Note `process` is part of the [Processor Contract](#) to process a single record.

`process` ...FIXME

KStreamTransform — Supplier of KStreamTransformProcessors

`KStreamTransform` is a [ProcessorSupplier](#) of [KStreamTransformProcessors](#).

`KStreamTransform` is created exclusively when `KStreamImpl` is requested to [transform](#).

`KStreamTransform` takes a single [TransformerSupplier](#) when created.

When [requested for a processor](#), `KStreamTransform` creates a new [KStreamTransformProcessor](#) with a new [Transformer](#) (after [requesting one](#) from the [TransformerSupplier](#)).

KStreamTransformValuesProcessor

`KStreamTransformValuesProcessor` is a [stream processor](#) that...FIXME

`KStreamTransformValuesProcessor` is [created](#) exclusively when `KStreamTransformValues` is requested to [supply a stream processor](#).

`KStreamTransformValuesProcessor` takes a `InternalValueTransformerWithKey` when created.

Note

`KStreamTransformValuesProcessor` is `public static` class of a `KStreamTransformValues` processor supplier.

Initializing Processor Node (Given ProcessorContext)

— `init` Method

```
void init(final ProcessorContext context)
```

Note

`init` is part of [Processor Contract](#) to initialize a stream processor.

`init` ...FIXME

KStreamTransformValues — Supplier of KStreamTransformValuesProcessors

`KStreamTransformValues` is a [ProcessorSupplier](#) that supplies [KStreamTransformValuesProcessors](#).

`KStreamTransformValues` is [created](#) exclusively when `KStreamImpl` is requested to [transform values with optional state](#) (aka *stateful record-by-record value transformation*).

`KStreamTransformValues` takes a `InternalValueTransformerWithKeySupplier` when created.

get Method

```
Processor<K, V> get()
```

Note	<code>get</code> is part of ProcessorSupplier Contract to supply a stream processor.
------	--

`get ...FIXME`

KStreamWindowAggregateProcessor

KStreamWindowAggregateProcessor is a [processor](#) that is used for...FIXME

KStreamWindowAggregateProcessor is [created](#) when...FIXME

KStreamWindowAggregateProcessor takes no arguments when created.

Initializing Processor— init Method

```
void init(final ProcessorContext context)
```

Note	init is part of the Processor Contract to initialize the processor .
------	--

init ...FIXME

Processing Record— process Method

```
void process(final K key, final V value)
```

Note	process is part of the Processor Contract to process a single record.
------	---

process ...FIXME

KStreamWindowAggregate

KStreamWindowAggregate is...FIXME

KTableSourceProcessor

`KTableSourceProcessor` is a stream processor that is created when `KTableSource` is requested to supply one.

`KTableSourceProcessor` takes no arguments when created.

When requested to initialize, `KTableSourceProcessor` ...FIXME

When requested to process a record, `KTableSourceProcessor` ...FIXME

KTableSource — ProcessorSupplier of KTableSourceProcessors

`KTableSource` is a [ProcessorSupplier](#) of [KTableSourceProcessors](#).

`KTableSource` is [created](#) when `InternalStreamsBuilder` is requested to add a [KTable](#) and a [GlobalKTable](#) to a topology (when `StreamsBuilder` is requested to create a [KTable](#) and [GlobalKTable](#) for topics, respectively).

`KTableSource` takes a store name when created.

`KTableSource` creates a new [KTableSourceProcessor](#) whenever requested to supply one.

KTableSuppressProcessor

KTableSuppressProcessor is...FIXME

KTableValueGetter

```
KTableValueGetter is...FIXME
```

KTableValueGetterSupplier

`KTableValueGetterSupplier` is the [contract](#) of `KTableValueGetterSuppliers` that can [get](#) a `KTableValueGetter` for `storeNames`.

```
package org.apache.kafka.streams.kstream.internals;

interface KTableValueGetterSupplier<K, V> {
    KTableValueGetter<K, V> get();
    String[] storeNames();
}
```

Table 1. KTableValueGetterSupplier Contract

Method	Description
<code>get</code>	Used when...FIXME
<code>storeNames</code>	Used when...FIXME

Table 2. KTableValueGetterSuppliers

KTableValueGetterSupplier	Description
KTableMaterializedValueGetterSupplier	
KTableSourceValueGetterSupplier	
KTableKTableAbstractJoinValueGetterSupplier	

KTableMaterializedValueGetterSupplier

KTableMaterializedValueGetterSupplier is...FIXME

KTableSourceValueGetterSupplier

`KTableSourceValueGetterSupplier` is a [KTableValueGetterSupplier](#) that...FIXME

`KTableSourceValueGetterSupplier` is [created](#) when...FIXME

`KTableSourceValueGetterSupplier` takes the store name when created.

KTableKTableAbstractJoinValueGetterSupplier

KTableKTableAbstractJoinValueGetterSupplier is...FIXME

WrappedStateStore

`WrappedStateStore` is the base implementation of the [StateStore](#) and [CachedStateStore](#) contracts for [state stores](#) that simply delegate all operations to the [wrapped state store](#).

`WrappedStateStore` allows for composing state stores and building more sophisticated state stores (e.g. [MeteredKeyValueStore](#) with [CachingKeyValueStore](#) and [ChangeLoggingKeyValueBytesStore](#)).

Note	Kafka Streams developers use Stores utility to use the more sophisticated state stores.
------	---

Table 1. WrappedStateStores

WrappedStateStore	Description
CachingKeyValueStore	
CachingSessionStore	
CachingWindowStore	
ChangeLoggingKeyValueBytesStore	
ChangeLoggingSessionBytesStore	
ChangeLoggingWindowBytesStore	
MeteredKeyValueStore	
MeteredSessionStore	
MeteredWindowStore	
RocksDBSessionStore	
RocksDBWindowStore	

`WrappedStateStore` takes a single [StateStore](#) (*wrapped state store*) to be created.

Note	<code>WrappedStateStore</code> is a Java abstract class and cannot be created directly. It is created indirectly for the concrete WrappedStateStores .
------	--

isTimestamped Static Method

```
boolean isTimestamped(StateStore stateStore)
```

`isTimestamped` is positive (`true`) when either of the following holds:

- The given `StateStore` is a `TimestampedBytesStore`
- The `wrapped StateStore` is timestamped (recursively)

Otherwise, `isTimestamped` is negative (`false`).

Note	<code>isTimestamped</code> is used exclusively when <code>AbstractStateManager</code> is requested for a <code>RecordConverter</code> (for a <code>StateStore</code>).
------	---

`isTimestamped` is used exclusively when `AbstractStateManager` is requested for a `RecordConverter` (for a `StateStore`).

CachingKeyValueStore

`CachingKeyValueStore` is a concrete [KeyValueStore](#) of `Bytes` keys and `byte[]` values (i.e. `KeyValueStore<Bytes, byte[]>`).

`CachingKeyValueStore` is also a concrete [WrappedStateStore](#) of [KeyValueStore](#) of `Bytes` keys and `byte[]` values (i.e. `WrappedStateStore<KeyValueStore<Bytes, byte[]>, byte[]>`, `byte[]>`).

`CachingKeyValueStore` is a concrete [CachedStateStore](#).

`CachingKeyValueStore` is [created](#) when:

- `KeyValueStoreBuilder` is requested to [build a KeyValueStore](#) with caching enabled
- `TimestampedKeyValueStoreBuilder` is requested to build a [TimestampedKeyValueStore](#) with [caching enabled](#)

`CachingKeyValueStore` takes a single [KeyValueStore](#) (*underlying state store*) to be created.

Initializing State Store — `init` Method

```
void init(
    ProcessorContext context,
    StateStore root)
```

Note	<code>init</code> is part of the StateStore Contract to initialize the state store.
------	---

`init` [initInternal](#) with the [ProcessorContext](#).

`init` then requests the [underlying KeyValueStore](#) to initialize.

In the end, `init` saves the current thread as the [streamThread](#) internal registry.

`initInternal` Internal Method

```
void initInternal(ProcessorContext context)
```

`initInternal` saves the [ProcessorContext](#) as the [context](#) internal registry.

`initInternal` requests the [InternalProcessorContext](#) for a [ThreadCache](#) that is saved as the [cache](#) internal registry.

`initInternal` creates the name of the cache for the task ID and the store name and that is saved as the `cacheName` internal registry.

In the end, `initInternal` requests the `ThreadCache` to `addDirtyEntryFlushListener` with the cache name and a `dirtyEntryFlushListener` that simply `putAndMaybeForward` the `DirtyEntries`.

Note	<code>initInternal</code> is used exclusively when <code>CachingKeyValueStore</code> is requested to initialize.
------	--

putAndMaybeForward Internal Method

```
void putAndMaybeForward(  
    ThreadCache.DirtyEntry entry,  
    InternalProcessorContext context)
```

`putAndMaybeForward` ...FIXME

Note	<code>putAndMaybeForward</code> is used exclusively when <code>CachingKeyValueStore</code> is requested to initialize (via <code>initInternal</code>).
------	---

remove Method

```
void remove(final Windowed<Bytes> sessionKey)
```

Note	<code>remove</code> is part of SessionStore Contract to...FIXME.
------	--

`remove` ...FIXME

Closing State Store — close Method

```
void close()
```

Note	<code>close</code> is part of...FIXME
------	---------------------------------------

`close` ...FIXME

Internal Properties

Name	Description
cache	ThreadCache Used when...FIXME
cacheName	
context	InternalProcessorContext Used when...FIXME
streamThread	Thread Used when...FIXME

CachingSessionStore

CachingSessionStore is...FIXME

Initialize State Store — `init` Method

```
void init(final ProcessorContext context, final StateStore root)
```

Note

`init` is part of...FIXME

`init` ...FIXME

`remove` Method

```
void remove(final Windowed<Bytes> sessionKey)
```

Note

`remove` is part of [SessionStore Contract](#) to...FIXME.

`remove` ...FIXME

`putAndMaybeForward` Internal Method

```
void putAndMaybeForward(final ThreadCache.DirtyEntry entry, final InternalProcessorContext context)
```

`putAndMaybeForward` ...FIXME

Note

`putAndMaybeForward` is used when...FIXME

Closing State Store — `close` Method

```
void close()
```

Note

`close` is part of...FIXME

`close` ...FIXME

CachingWindowStore

CachingWindowStore is...FIXME

Initialize State Store — `init` Method

```
void init(final ProcessorContext context, final StateStore root)
```

Note

`init` is part of...FIXME

`init` ...FIXME

`remove` Method

```
void remove(final Windowed<Bytes> sessionKey)
```

Note

`remove` is part of [SessionStore Contract](#) to...FIXME.

`remove` ...FIXME

`putAndMaybeForward` Internal Method

```
void putAndMaybeForward(final ThreadCache.DirtyEntry entry, final InternalProcessorContext context)
```

`putAndMaybeForward` ...FIXME

Note

`putAndMaybeForward` is used when...FIXME

Closing State Store — `close` Method

```
void close()
```

Note

`close` is part of...FIXME

`close` ...FIXME

ChangeLoggingKeyValueBytesStore

`ChangeLoggingKeyValueBytesStore` is a concrete `KeyValueStore` of `Bytes` keys and `byte[]` values (i.e. `KeyValueStore<Bytes, byte[]>`) that uses a `StoreChangeLogger` to log records.

`ChangeLoggingKeyValueBytesStore` is also a concrete `WrappedStateStore` of `KeyValueStore` of `Bytes` keys and `byte[]` values (i.e. `wrappedStateStore<KeyValueStore<Bytes, byte[]>, byte[], byte[]>`).

`ChangeLoggingKeyValueBytesStore` is created exclusively when `KeyValueStoreBuilder` is requested to build a `KeyValueStore` with logging enabled.

`ChangeLoggingKeyValueBytesStore` takes a single `KeyValueStore` (*inner state store*) to be created.

Initializing State Store — `init` Method

```
void init(
    ProcessorContext context,
    StateStore root)
```

Note	<code>init</code> is part of the StateStore Contract to initialize the state store.
------	---

`init` requests the inner `KeyValueStore` to initialize (with the given `ProcessorContext` and the root `StateStore`).

`init` then creates a new `StateSerdes` and `StoreChangeLogger` (for the name of the changelog topic per application ID and store name) that is saved as the `changeLogger` internal registry.

In the end, `init` registers an eviction listener if the inner state store is a `MemoryLRUCache` for `setWhenEldestRemoved` to log removed record (with the key evicted to be with `null` value to indicate removal).

`put` Method

```
void put(
    Bytes key,
    byte[] value)
```

Note	<code>put</code> is part of the StateStore Contract to...FIXME.
------	---

```
put ...FIXME
```

putIfAbsent Method

```
byte[] putIfAbsent(  
    Bytes key,  
    byte[] value)
```

Note

`putIfAbsent` is part of the [StateStore Contract](#) to...FIXME.

```
putIfAbsent ...FIXME
```

putAll Method

```
void putAll(List<KeyValue<Bytes, byte[]>> entries)
```

Note

`putAll` is part of the [StateStore Contract](#) to...FIXME.

```
putAll ...FIXME
```

delete Method

```
byte[] delete(Bytes key)
```

Note

`delete` is part of the [StateStore Contract](#) to...FIXME.

```
delete ...FIXME
```

Logging Records — log Method

```
void log(  
    Bytes key,  
    byte[] value)
```

`log` simply requests the [StoreChangeLogger](#) to [logChange](#) with the key and the value.

Note

`log` is used when `ChangeLoggingKeyValueBytesStore` is requested to [initialize](#), [put](#), [putIfAbsent](#), [putAll](#), and [delete](#).

ChangeLoggingSessionBytesStore

`ChangeLoggingSessionBytesStore` is a [SessionStore](#) that uses Kafka `Bytes` (immutable byte arrays) for keys.

`ChangeLoggingSessionBytesStore` is [created](#) exclusively when `SessionStoreBuilder` is requested to [build a MeteredSessionStore](#) (with [logging enabled](#)).

`ChangeLoggingSessionBytesStore` takes a [SessionStore](#) when created.

Table 1. ChangeLoggingSessionBytesStore's Internal Properties (e.g. Registries, Counters and Flags)

Name	Description
<code>changeLogger</code>	StoreChangeLogger Used when...FIXME
<code>innerStateSerde</code>	<code>StateSerdes</code> Used when...FIXME

Initialize State Store — `init` Method

```
void init(final ProcessorContext context, final StateStore root)
```

Note	<code>init</code> is part of...FIXME
------	--------------------------------------

`init` ...FIXME

Storing Aggregated Value for Session Key — `put` Method

```
void put(final Windowed<Bytes> sessionKey, final byte[] aggregate)
```

Note	<code>put</code> is part of SessionStore Contract to store an aggregated value for a session key.
------	---

`put` requests the [SessionStore](#) to store the aggregated value for the given session key.

In the end, `put` ...FIXME

ChangeLoggingWindowBytesStore

ChangeLoggingWindowBytesStore is...FIXME

Initialize State Store — init Method

```
void init(final ProcessorContext context, final StateStore root)
```

Note

init is part of...FIXME

init ...FIXME

MeteredKeyValueStore

MeteredKeyValueStore is...FIXME

MeteredSessionStore

`MeteredSessionStore` is a [SessionStore](#) that...FIXME

Initialize State Store — `init` Method

```
void init(final ProcessorContext context, final StateStore root)
```

Note

`init` is part of...FIXME

`init` ...FIXME

Storing Aggregated Value for Session — `put` Method

```
void put(final Windowed<K> sessionKey, final V aggregate)
```

Note

`put` is part of [SessionStore Contract](#) to store an aggregated value for a session.

`put` ...FIXME

Removing Aggregated Value for Session — `remove` Method

```
void remove(final Windowed<K> sessionKey)
```

Note

`remove` is part of [SessionStore Contract](#) to remove an aggregated value for a session.

`remove` ...FIXME

MeteredWindowStore — State Store of WindowStoreBuilder

`MeteredWindowStore` is a concrete `WindowStore` that registers the [metric sensors](#) (when [initialized](#)).

`MeteredWindowStore` is [created](#) exclusively when `WindowStoreBuilder` is requested for a [state store](#).

Table 1. MeteredWindowStore's (Latency and Throughput) Metric Sensors

Metric Sensor	Name	RecordingLevel	Description
<code>putTime</code>	<code>put</code>	DEBUG	Latency of put
<code>fetchTime</code>	<code>fetch</code>	DEBUG	Latency of fetch , all and fetchAll
<code>flushTime</code>	<code>flush</code>	DEBUG	Latency of flush
<code>restoreTime</code>	<code>restore</code>	DEBUG	Latency of init

Note

All the [metric sensors](#) are at `DEBUG` recording level so you have to turn them on using `metrics.recording.level` streams property.

Creating MeteredWindowStore Instance

`MeteredWindowStore` takes the following when created:

- [WindowStore](#)
- Scope name for the [metrics](#) (aka *metric scope*)
- `Time`
- `Serde<K>` for keys
- `Serde<V>` for values

`MeteredWindowStore` initializes the [internal registries and counters](#).

Initializing State Store — `init` Method

```
void init(final ProcessorContext context, final StateStore root)
```

Note	init is part of the StateStore Contract to initializes the state store.
------	---

init ...FIXME

init requests the ProcessorContext for the StreamsMetrics and registers the metric sensors.

init ...FIXME

RocksDBSessionStore

RocksDBSessionStore is...FIXME

Initialize State Store — `init` Method

```
void init(final ProcessorContext context, final StateStore root)
```

Note

`init` is part of...FIXME

`init` ...FIXME

RocksDBWindowStore

`RocksDBWindowStore` is a custom [WindowStore](#) that uses the [SegmentedBytesStore](#) (as the underlying [state store](#)).

`RocksDBWindowStore` is [created](#) exclusively when `RocksDbWindowBytesStoreSupplier` is requested for a [state store](#) (and gives a `RocksDBWindowStore` with a [RocksDBSegmentedBytesStore](#)).

Creating RocksDBWindowStore Instance

`RocksDBWindowStore` takes the following when created:

- [SegmentedBytesStore](#)
- `Serde<K>` for keys
- `Serde<V>` for values
- `retainDuplicates` flag
- `windowSize`

`RocksDBWindowStore` initializes the [internal registries and counters](#).

init Method

```
void init(final ProcessorContext context, final StateStore root)
```

Note	<code>init</code> is part of the StateStore Contract to initialize the state store.
------	---

`init ...FIXME`

InMemoryKeyValueStore

InMemoryKeyValueStore is...FIXME

Initialize State Store — init Method

```
void init(final ProcessorContext context, final StateStore root)
```

Note

init is part of [StateStore Contract](#) to initialize the state store.

init ...FIXME

InMemorySessionStore

InMemorySessionStore is...FIXME

Initialize State Store — init Method

```
void init(  
    ProcessorContext context,  
    StateStore root)
```

Note

init is part of...FIXME

init ...FIXME

InMemoryWindowStore

InMemoryWindowStore is...FIXME

MemoryLRUCache

MemoryLRUCache is...FIXME

Initialize State Store — `init` Method

```
void init(final ProcessorContext context, final StateStore root)
```

Note

`init` is part of [StateStore Contract](#) to initialize the [state store](#).

`init` ...FIXME

MeteredKeyValueBytesStore

MeteredKeyValueBytesStore is a concrete [KeyValueStore](#) that...FIXME

init Method

FIXME

Note	init is part of HERE Contract to...FIXME.
------	---

init ...FIXME

RocksDBStore

RocksDBStore is a concrete [KeyValueStore](#) that...FIXME

RocksDBStore is always [persistent](#).

Initialize State Store — `init` Method

```
void init(final ProcessorContext context, final StateStore root)
```

Note

`init` is part of [StateStore Contract](#) to initialize the [state store](#).

`init` ...FIXME

RocksDBTimestampedStore

RocksDBTimestampedStore is...FIXME

AbstractStoreBuilder — Fluent API for State Store Builders

`AbstractStoreBuilder` is the base implementation of the `StoreBuilder` contract for `state store builders` that offer a fluent API to `build state stores`.

Note	<code>AbstractStoreBuilder</code> is a Java abstract class and cannot be <code>created</code> directly. It is created indirectly for the <code>concrete AbstractStoreBuilders</code> .
------	--

Table 1. AbstractStoreBuilders

AbstractStoreBuilder	Description
<code>KeyValueStoreBuilder</code>	Builds <code>KeyValueStores</code>
<code>SessionStoreBuilder</code>	Builds <code>SessionStores</code>
<code>TimestampedKeyValueStoreBuilder</code>	Builds...FIXME
<code>TimestampedWindowStoreBuilder</code>	Builds...FIXME
<code>WindowStoreBuilder</code>	Builds <code>WindowStores</code>

Creating AbstractStoreBuilder Instance

`AbstractStoreBuilder` takes the following to be created:

- Name of the state stores to build
- `Serde<K>` for keys
- `Serde<V>` for values
- `Time`

`AbstractStoreBuilder` initializes the `internal properties`.

Note	<code>AbstractStoreBuilder</code> is a Java abstract class and cannot be <code>created</code> directly. It is created indirectly for the <code>concrete AbstractStoreBuilders</code> .
------	--

withLoggingEnabled Method

```
StoreBuilder<T> withLoggingEnabled(Map<String, String> config)
```

Note

`withLoggingEnabled` is part of the [StoreBuilder Contract](#) to enable change-logging on [state stores](#).

`withLoggingEnabled` simply turns the `enableLogging` flag on (`true`) and the `logConfig` to the given config (`Map<String, String>`).

withLoggingDisabled Method

```
StoreBuilder<T> withLoggingDisabled()
```

Note

`withLoggingDisabled` is part of the [StoreBuilder Contract](#) to disable change-logging on [state stores](#).

`withLoggingDisabled` simply turns the `enableLogging` flag off (`false`) and clears the `logConfig`.

withCachingEnabled Method

```
StoreBuilder<T> withCachingEnabled()
```

Note

`withCachingEnabled` is part of the [StoreBuilder Contract](#) to enable caching on [state stores](#).

`withCachingEnabled` simply turns the `enableCaching` flag on (`true`).

withCachingDisabled Method

```
StoreBuilder<T> withCachingDisabled()
```

Note

`withCachingDisabled` is part of the [StoreBuilder Contract](#) to disable caching on [state stores](#).

`withCachingDisabled` simply turns the `enableCaching` flag off (`false`).

Internal Properties

Name	Description
enableCaching	Flag to control whether to enable caching (<code>true</code>) or not (<code>false</code>)
enableLogging	Flag to control whether to enable logging (<code>true</code>) or not (<code>false</code>) Default: <code>true</code>
logConfig	Change-logging configuration (<code>Map<String, String></code>)

KeyValueStoreBuilder

`KeyValueStoreBuilder` is a [StoreBuilder](#) (indirectly as a [AbstractStoreBuilder](#)) that allows for building [KeyValueStores](#).

`KeyValueStoreBuilder` can be created using [Stores.keyValueStoreBuilder](#).

```
import org.apache.kafka.streams.state.Stores
// Using Scala API for Kafka Streams
import org.apache.kafka.streams.scala.Serdes
val storeBuilder = Stores.keyValueStoreBuilder(
  Stores.inMemoryKeyValueStore("store-name"),
  Serdes.Integer,
  Serdes.Long)
scala> :type storeBuilder
org.apache.kafka.streams.state.StoreBuilder[org.apache.kafka.streams.state.KeyValueStore[Int, Long]]

import org.apache.kafka.streams.state.internals.KeyValueStoreBuilder
assert(storeBuilder.isInstanceOf[KeyValueStoreBuilder[_, _]])
```

When requested for a state store, `KeyValueStoreBuilder` creates a new [MeteredKeyValueBytesStore](#) (with the state store from the [KeyValueBytesStoreSupplier](#)).

With caching enabled, `build` creates a [CachingKeyValueStore](#).

With logging enabled, `build` creates a [ChangeLoggingKeyValueBytesStore](#).

Note	Logging is enabled by default.
------	--------------------------------

Creating KeyValueStoreBuilder Instance

`KeyValueStoreBuilder` takes the following when created:

- [KeyValueBytesStoreSupplier](#)
- `Serde<K>` for keys
- `Serde<V>` for values
- `Time`

Building KeyValueStore — `build` Method

```
KeyValueStore<K, V> build()
```

Note

`build` is part of the [StoreBuilder Contract](#) to build a [state store](#).

`build` first requests the [KeyValueBytesStoreSupplier](#) for a [StateStore](#).

`build` then creates a [MeteredKeyValueStore](#) (possibly with a [CachingKeyValueStore](#) with [caching enabled](#) and [ChangeLoggingKeyValueBytesStore](#) with [logging enabled](#)).

maybeWrapLogging Internal Method

```
KeyValueStore<Bytes, byte[]> maybeWrapLogging(
    KeyValueStore<Bytes, byte[]> inner)
```

`maybeWrapLogging` simply returns the given inner [KeyValueStore](#) with [logging disabled](#).

Otherwise, with [logging enabled](#), `maybeWrapLogging` creates a [ChangeLoggingKeyValueBytesStore](#).

Note

`maybeWrapLogging` is used exclusively when `KeyValueStoreBuilder` is requested to build a [KeyValueStore](#).

maybeWrapCaching Internal Method

```
KeyValueStore<Bytes, byte[]> maybeWrapCaching(
    KeyValueStore<Bytes, byte[]> inner)
```

`maybeWrapCaching` simply returns the given inner [KeyValueStore](#) with [caching disabled](#).

Otherwise, with [caching enabled](#), `maybeWrapCaching` creates a [CachingKeyValueStore](#).

Note

`maybeWrapCaching` is used exclusively when `KeyValueStoreBuilder` is requested to build a [KeyValueStore](#).

SessionStoreBuilder

`SessionStoreBuilder` is an [AbstractStoreBuilder](#) that builds a [MeteredSessionStore](#) (possibly with [caching](#) and [logging](#)).

`SessionStoreBuilder` is [created](#) exclusively when `stores` is requested to [create one](#) (with `Time.SYSTEM time`).

Creating SessionStoreBuilder Instance

`SessionStoreBuilder` takes the following when created:

- [SessionBytesStoreSupplier](#)
- `Serde` for keys
- `Serde` for values
- `Time`

Building MeteredSessionStore (with Optional Caching and Logging) — `build` Method

```
SessionStore<K, V> build()
```

Note	<code>build</code> is part of StoreBuilder Contract to build a StateStore .
------	---

`build` ...FIXME

maybeWrapCaching Internal Method

```
SessionStore<Bytes, byte[]> maybeWrapCaching(final SessionStore<Bytes, byte[]> inner)
```

`maybeWrapCaching` ...FIXME

Note	<code>maybeWrapCaching</code> is used when...FIXME
------	--

maybeWrapLogging Internal Method

```
SessionStore<Bytes, byte[]> maybeWrapLogging(final SessionStore<Bytes, byte[]> inner)
```

maybeWrapLogging ...FIXME

Note

maybeWrapLogging is used when...FIXME

TimestampedKeyValueStoreBuilder

`TimestampedKeyValueStoreBuilder` is...FIXME

TimestampedWindowStoreBuilder

`TimestampedWindowStoreBuilder` is...FIXME

WindowStoreBuilder

`WindowStoreBuilder` is a `StoreBuilder` (indirectly as an `AbstractStoreBuilder`) that is used to build `WindowStores` (using a given `WindowBytesStoreSupplier`).

`WindowStoreBuilder` takes the following when created:

- `WindowBytesStoreSupplier`
- `Serde` for keys
- `Serde` for values
- `Time`

`WindowStoreBuilder` can however be created using `Stores.windowStoreBuilder`.

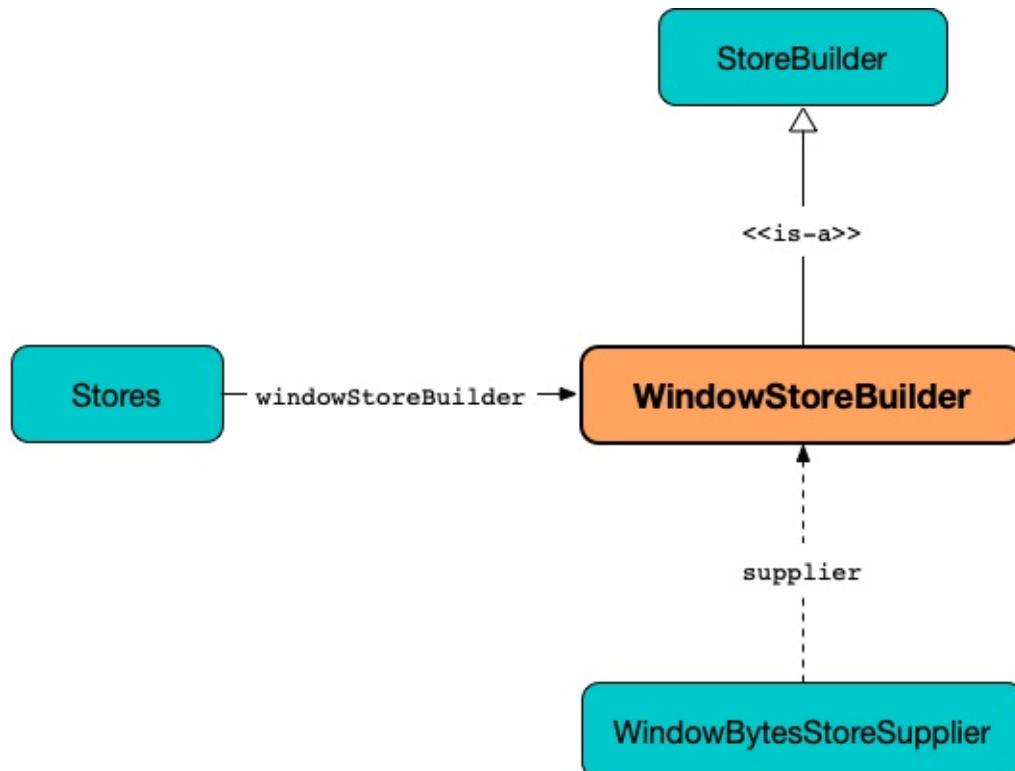


Figure 1. Creating `WindowStoreBuilder` using `Stores.windowStoreBuilder`

```

import org.apache.kafka.streams.state.Stores
// Using Scala API for Kafka Streams
import org.apache.kafka.streams.scala.Serdes
import java.time.Duration
val storeBuilder = Stores.windowStoreBuilder(
    Stores.persistentWindowStore(
        "queryable-store-name",
        Duration.ofMinutes(1),
        Duration.ofSeconds(30),
        false),
    Serdes.Integer,
    Serdes.Long)
scala> :type storeBuilder
org.apache.kafka.streams.state.StoreBuilder[org.apache.kafka.streams.state.WindowStore[
Int, Long]]

import org.apache.kafka.streams.state.internals.WindowStoreBuilder
assert(storeBuilder.isInstanceOf[WindowStoreBuilder[_, _]])

```

When requested to [build a state store](#), `WindowStoreBuilder` creates a new [MeteredWindowStore](#) (with the [state store](#) and the [metricsScope](#) from the [WindowBytesStoreSupplier](#)).

```

scala> :type storeBuilder
org.apache.kafka.streams.state.StoreBuilder[org.apache.kafka.streams.state.WindowStore[
Int, Long]]

val windowStore = storeBuilder.build
scala> :type windowStore
org.apache.kafka.streams.state.WindowStore[Int, Long]

```

With [caching enabled](#), `build` creates a [MeteredWindowStore](#) (with a [CachingWindowStore](#)).

```

scala> :type storeBuilder
org.apache.kafka.streams.state.StoreBuilder[org.apache.kafka.streams.state.WindowStore[
Int, Long]]

val withCachingWindowStoreBuilder = storeBuilder.withCachingEnabled

val windowStore = withCachingWindowStoreBuilder.build
import org.apache.kafka.streams.state.internals.MeteredWindowStore
assert(windowStore.isInstanceOf[MeteredWindowStore[_, _]])

```

With logging enabled, `build` creates a [MeteredWindowStore](#) (with a [ChangeLoggingWindowBytesStore](#)).

maybeWrapLogging Internal Method

```
WindowStore<Bytes, byte[]> maybeWrapLogging(  
    final WindowStore<Bytes, byte[]> inner)
```

maybeWrapLogging ...FIXME

Note	maybeWrapLogging is used when...FIXME
------	---------------------------------------

maybeWrapCaching Internal Method

```
WindowStore<Bytes, byte[]> maybeWrapCaching(  
    final WindowStore<Bytes, byte[]> inner)
```

maybeWrapCaching ...FIXME

Note	maybeWrapCaching is used when...FIXME
------	---------------------------------------

retentionPeriod Method

```
long retentionPeriod()
```

retentionPeriod ...FIXME

Note	retentionPeriod is used when...FIXME
------	--------------------------------------

InMemorySessionBytesStoreSupplier

`InMemorySessionBytesStoreSupplier` is a concrete [SessionBytesStoreSupplier](#) of [InMemorySessionStores](#) (`SessionStore<Bytes, byte[]>`).

`InMemorySessionBytesStoreSupplier` is [created](#) when `stores` factory is requested to [inMemorySessionStore](#).

Creating State Store — `get` Method

```
SessionStore<Bytes, byte[]> get()
```

Note

`get` is part of the [StoreSupplier Contract](#) to get a [state store](#).

`get` simply creates a new [InMemorySessionStore](#).

RocksDbKeyValueBytesStoreSupplier

`RocksDbKeyValueBytesStoreSupplier` is a [KeyValueBytesStoreSupplier](#).

`RocksDbKeyValueBytesStoreSupplier` is [created](#) when `Stores` utility is used to create persistent [KeyValueStores](#):

- `Stores.persistentKeyValueStore` (with the `returnTimestampedStore` off)
- `Stores.persistentTimestampedKeyValueStore` (with the `returnTimestampedStore` on)

`RocksDbKeyValueBytesStoreSupplier` uses **rocksdb-state** metric scope for the [metrics](#) recorded by [Metered stores](#).

Creating RocksDbKeyValueBytesStoreSupplier Instance

`RocksDbKeyValueBytesStoreSupplier` takes the following to be created:

- Name
- `returnTimestampedStore` flag

Supplying KeyValueStore — `get` Method

```
KeyValueStore<Bytes, byte[]> get()
```

Note `get` is part of the [StoreSupplier Contract](#) to supply ([get](#)) a state store.

`get` gives the following [KeyValueStores](#):

- `RocksDBTimestampedStore` when `returnTimestampedStore` is on (`true`)
- `RocksDBStore` when `returnTimestampedStore` is off (`false`)

RocksDbSessionBytesStoreSupplier

`RocksDbSessionBytesStoreSupplier` is a concrete [SessionBytesStoreSupplier](#) of [RocksDBSessionStores](#) (`SessionStore<Bytes, byte[]>`).

`RocksDbSessionBytesStoreSupplier` is created when `stores` factory is requested to [persistentSessionStore](#).

Creating State Store — `get` Method

```
SessionStore<Bytes, byte[]> get()
```

Note

`get` is part of the [StoreSupplier Contract](#) to get a [state store](#).

`get` simply creates a new [RocksDBSessionStore](#).

RocksDbWindowBytesStoreSupplier

`RocksDbWindowBytesStoreSupplier` is a concrete [WindowBytesStoreSupplier](#) that supplies [RocksDBWindowStores](#) over [RocksDBSegmentedBytesStores](#).

In fact, `RocksDbWindowBytesStoreSupplier` is the one and only known [WindowBytesStoreSupplier](#).

`RocksDbWindowBytesStoreSupplier` uses **rocksdb-window-state** as the metrics scope.

`RocksDbWindowBytesStoreSupplier` is created when `stores` factory is requested to [persistentWindowStore](#).

```
import org.apache.kafka.streams.state.Stores
val windowStore = Stores.persistentWindowStore("window-store-name", 1000, 2, 1000, true
)

scala> :type windowStore
org.apache.kafka.streams.state.WindowBytesStoreSupplier

import org.apache.kafka.streams.state.internals.RocksDbWindowBytesStoreSupplier
assert(windowStore.isInstanceOf[RocksDbWindowBytesStoreSupplier])
```

Creating State Store — get Method

```
WindowStore<Bytes, byte[]> get()
```

Note	<code>get</code> is part of the StoreSupplier Contract to get a state store .
------	---

`get` creates a [RocksDBSegmentedBytesStore](#) (with the `name`, the `retentionPeriod` and the `segments`).

`get` then returns a new [RocksDBWindowStore](#) (with the `RocksDBSegmentedBytesStore` and the `retainDuplicates` and the `windowSize`).

Creating RocksDbWindowBytesStoreSupplier Instance

`RocksDbWindowBytesStoreSupplier` takes the following when created:

- Name
- `retentionPeriod`

- Number of segments (must be `2` or higher)
- `windowSize`
- `retainDuplicates` flag

KeyValueToTimestampedKeyValueByteStoreAdapter

`KeyValueToTimestampedKeyValueByteStoreAdapter` is a concrete `KeyValueStore` that...FIXME

`KeyValueToTimestampedKeyValueByteStoreAdapter` is always `persistent`.

WindowToTimestampedWindowByteStoreAdapter

`WindowToTimestampedWindowByteStoreAdapter` is a concrete [WindowStore](#) that...FIXME

`WindowToTimestampedWindowByteStoreAdapter` is always persistent.

SegmentedBytesStore Contract

`SegmentedBytesStore` is the extension of the [StateStore contract](#) for segmented-bytes state stores that [FIXME](#).

Table 1. SegmentedBytesStore Contract

Method	Description
<code>all</code>	<code>KeyValueIterator<Bytes, byte[]> all()</code> Used when...FIXME
<code>fetch</code>	<code>KeyValueIterator<Bytes, byte[]> fetch(Bytes keyFrom, Bytes keyTo, long from, long to)</code> <code>KeyValueIterator<Bytes, byte[]> fetch(Bytes key, long from, long to)</code> Used when...FIXME
<code>fetchAll</code>	<code>KeyValueIterator<Bytes, byte[]> fetchAll(long from, long to)</code> Used when...FIXME
<code>get</code>	<code>byte[] get(Bytes key)</code> Used when...FIXME
<code>put</code>	<code>void put(Bytes key, byte[] value)</code> Used when...FIXME
<code>remove</code>	<code>void remove(Bytes key)</code> Used when...FIXME

Note

[AbstractRocksDBSegmentedBytesStore](#) is the only known direct implementation of the [SegmentedBytesStore Contract](#) in Kafka Streams.

AbstractRocksDBSegmentedBytesStore

AbstractRocksDBSegmentedBytesStore is a concrete [SegmentedBytesStore](#) that...FIXME

AbstractRocksDBSegmentedBytesStore is always [persistent](#).

RocksDBSegmentedBytesStore

`RocksDBSegmentedBytesStore` is a concrete [StateStore](#) (as a [AbstractRocksDBSegmentedBytesStore](#) of `KeyValueSegments`) that...FIXME

`RocksDBSegmentedBytesStore` is [created](#) when:

- `RocksDbSessionBytesStoreSupplier` is requested for a [SessionStore](#)
- `RocksDbWindowBytesStoreSupplier` is requested for a [WindowStore](#)

Creating RocksDBSegmentedBytesStore Instance

`RocksDBSegmentedBytesStore` takes the following to be created:

- Name
- Metric scope
- `retention`
- `segmentInterval`
- `KeySchema`

TimeOrderedKeyValueBuffer

`TimeOrderedKeyValueBuffer` is the [extension](#) of the [StateStore](#) contract for [time-ordered key-value state stores](#) that [FIXME](#).

Table 1. TimeOrderedKeyValueBuffer Contract

Method	Description
bufferSize	<pre>long bufferSize()</pre> <p>Used when...FIXME</p>
evictWhile	<pre>void evictWhile(Supplier<Boolean> predicate, Consumer<Eviction<K, V>> callback)</pre> <p>Used when...FIXME</p>
minTimestamp	<pre>long minTimestamp()</pre> <p>Used when...FIXME</p>
numRecords	<pre>int numRecords()</pre> <p>Used when...FIXME</p>
priorValueForBuffered	<pre>Maybe<ValueAndTimestamp<V>> priorValueForBuffered(K key)</pre> <p>Used when...FIXME</p>
put	<pre>void put(long time, K key, Change<V> value, ProcessorRecordContext recordContext)</pre> <p>Used when...FIXME</p>
setSerdesIfNull	<pre>void setSerdesIfNull(Serde<K> keySerde, Serde<V> valueSerde)</pre> <p>Used when...FIXME</p>
Note	<p>InMemoryTimeOrderedKeyValueBuffer is the default and only known implementation of the TimeOrderedKeyValueBuffer Contract in Kafka Streams.</p>

InMemoryTimeOrderedKeyValueBuffer

`InMemoryTimeOrderedKeyValueBuffer` is a concrete `TimeOrderedKeyValueBuffer` that is [created](#) exclusively when `KTableImpl` is requested to [suppress](#).

Creating InMemoryTimeOrderedKeyValueBuffer Instance

`InMemoryTimeOrderedKeyValueBuffer` takes the following to be created:

- Name of a state store
- `loggingEnabled` flag
- `Serde<K>` for keys
- `Serde<V>` for values

`InMemoryTimeOrderedKeyValueBuffer` initializes the [internal properties](#).

build Method

```
InMemoryTimeOrderedKeyValueBuffer<K, V> build()
```

Note `build` is part of the [StoreBuilder Contract](#) to create (*build*) a [state store](#).

`build` simply creates a new `InMemoryTimeOrderedKeyValueBuffer`.

CachedStateStore

CachedStateStore is...FIXME

ProcessorNode

`ProcessorNode` is a processing node in a processor topology that is identified by a `name` and have zero or more `child processor nodes`.

Note	<code>SourceNode</code> and <code>SinkNode</code> are specialized <code>ProcessorNodes</code> .
------	---

`ProcessorNode` is `created` exclusively when `ProcessorNodeFactory` is requested to `build a processor node`.

`ProcessorNode` can be `initialized` (in a given `ProcessorContext`) and `closed`.

`ProcessorNode` can be `punctuated`.

`ProcessorNode` has a `human-friendly / textual representation` that is particularly helpful for debugging.

Initializing ProcessorNode — `init` Method

```
void init(ProcessorContext context)
```

`init ...FIXME`

Note	<code>init</code> is used when: <ul style="list-style-type: none"> • <code>GlobalStateUpdateTask</code> is requested to <code>initTopology</code> • <code>StreamTask</code> is requested to <code>initialize the ProcessorNodes in a ProcessorTopology</code>
------	---

`init` is used when:

- `GlobalStateUpdateTask` is requested to `initTopology`
- `StreamTask` is requested to `initialize the ProcessorNodes in a ProcessorTopology`

Processing Single Record (As Key and Value Pair) — `process` Method

```
void process(final K key, final V value)
```

`process` requests the `Processor` to process the given key and value pair.

In the end, `process` requests the `NodeMetrics` for the `nodeProcessTimeSensor` to record the processing time.

Note

- `process` is used when:
- `GlobalProcessorContextImpl` and `ProcessorContextImpl` are requested to forward
 - `GlobalStateUpdateTask` is requested to update
 - `StreamTask` is requested to process a single record

Creating ProcessorNode Instance

`ProcessorNode` takes the following to be created:

- Name
- `Processor` (`Processor<K, V>` of `K` keys and `V` values)
- Names of the associated state stores

`ProcessorNode` initializes the internal properties.

Adding Child Processor Node — `addChild` Method

```
void addChild(ProcessorNode<?, ?> child)
```

`addChild` ...FIXME

Note

`addChild` is used when `InternalTopologyBuilder` is requested to build a `SinkNode` and `ProcessorNode` (when requested to build a topology of processor nodes).

close Method

```
void close()
```

`close` ...FIXME

Note

`close` is used exclusively when `StreamTask` is requested to `closeTopology` (when `StreamTask` is requested to `suspend`).

Describing Itself (Textual Representation) — `toString` Method

```
String toString() (1)
String toString(string indent)
```

1. Uses an empty `indent`

Note

`toString` is part of Java's [java.lang.Object Contract](#) to return a human-friendly / textual representation of an object.

`toString` ...FIXME

Executing Punctuator (Scheduled Periodic Action) — `punctuate` Method

```
void punctuate(
    long timestamp,
    Punctuator punctuator)
```

`punctuate` requests the given [Punctuator](#) to [punctuate](#) with the given `timestamp`.

In the end, `punctuate` requests the [NodeMetrics](#) for the [nodePunctuateTimeSensor](#) to record the time for node punctuation.

Note

`punctuate` is used exclusively when `StreamTask` is requested to [punctuate a processor](#).

Internal Properties

Name	Description
<code>children</code>	Child <code>ProcessorNodes</code> Used when...FIXME
<code>nodeMetrics</code>	NodeMetrics Used when...FIXME
<code>time</code>	<code>Time</code> Used when...FIXME

SourceNode

SourceNode is a [ProcessorNode](#) that...FIXME

SourceNode is [created](#) exclusively when [SourceNodeFactory](#) is requested to [build a processor node](#) (when...FIXME)

SourceNode gets a [ProcessorContext](#) when [initialized](#). It is later used when SourceNode processes a record (and simply forwards it downstream).

init Method

```
void init(ProcessorContext context)
```

Note init is part of [ProcessorNode Contract](#) to...FIXME.

init ...FIXME

Processing Record — process Method

```
void process(  
    K key,  
    V value)
```

Note process is part of [ProcessorNode Contract](#) to...FIXME.

process simply requests [ProcessorContext](#) to [forward](#) followed by informing the [sourceNodeForwardSensor](#) that a record was processed.

Creating SourceNode Instance

SourceNode takes the following when created:

- Node name
- List of topics
- [TimestampExtractor](#)
- Key deserializer
- Value deserializer

SourceNode initializes the [internal registries and counters](#).

SinkNode — ProcessorNode with no Child Nodes

`SinkNode` is a custom [ProcessorNode](#) with no child processor nodes.

`SinkNode` is created exclusively when `SinkNodeFactory` is requested to build a processor node (when `InternalTopologyBuilder` is requested to build a topology of processor nodes).

`SinkNode` uses a [TopicNameExtractor](#) for **dynamic routing** when processing a record.

Processing Record — `process` Method

```
void process(final K key, final V value)
```

Note	<code>process</code> is part of ProcessorNode Contract to process a record.
------	---

`process` ...FIXME

Creating SinkNode Instance

`SinkNode` takes the following when created:

- Name of the sink
- [TopicNameExtractor](#)
- Kafka `Serializer` for keys
- Kafka `Serializer` for values
- [StreamPartitioner](#)

Adding Child Processor Node — `addChild` Method

```
void addChild(final ProcessorNode<?, ?> child)
```

Note	<code>addChild</code> is part of ProcessorNode Contract to add a child processor node.
------	--

`addChild` simply throws an `UnsupportedOperationException` with the following message:

```
sink node does not allow addChild
```

Initializing Processor Node — `init` Method

```
void init(final ProcessorContext context)
```

Note

`init` is part of [ProcessorNode Contract](#) to...FIXME.

`init` ...FIXME

Describing Itself (Textual Representation) — `toString` Method

```
void toString(final ProcessorContext context)
```

Note

`toString` is part of [ProcessorNode Contract](#) for the human-friendly / textual representation of a `ProcessorNode`.

`toString` ...FIXME

NodeMetrics

`NodeMetrics` is...FIXME

`NodeMetrics` is [created](#) exclusively when `ProcessorNode` is requested to [init](#).

Table 1. NodeMetrics's Internal Properties (e.g. Registries, Counters and Flags)

Name	Description
<code>nodeProcessTimeSensor</code>	<code>Kafka Sensor</code> Used when...FIXME
<code>nodePunctuateTimeSensor</code>	<code>Kafka Sensor</code> Used when...FIXME
<code>sourceNodeForwardSensor</code>	<code>Kafka Sensor</code> Used when...FIXME
<code>sourceNodeSkippedDueToDeserializationError</code>	<code>Kafka Sensor</code> Used when...FIXME
<code>nodeCreationSensor</code>	<code>Kafka Sensor</code> Used when...FIXME
<code>nodeDestructionSensor</code>	<code>Kafka Sensor</code> Used when...FIXME

Creating NodeMetrics Instance

`NodeMetrics` takes the following when created:

- [StreamsMetrics](#)
- [Name](#) of a `ProcessorNode`
- [ProcessorContext](#)

`NodeMetrics` initializes the [internal registries and counters](#).

InternalTopicConfig

InternalTopicConfig is...FIXME

WindowedChangelogTopicConfig

`WindowedChangelogTopicConfig` is an [InternalTopicConfig](#) that...FIXME

`WindowedChangelogTopicConfig` is [created](#) exclusively when `InternalTopologyBuilder` is requested for [create an InternalTopicConfig \(for a given name and StateStoreFactory\)](#).

When [created](#), `WindowedChangelogTopicConfig` creates a `WINDOWED_STORE_CHANGELOG_TOPIC_DEFAULT_OVERRIDES` configuration properties that...FIXME

Creating WindowedChangelogTopicConfig Instance

`WindowedChangelogTopicConfig` takes the following when created:

- Name
- Topic configs (as `Map<String, String>`)

`WindowedChangelogTopicConfig` initializes the [internal registries and counters](#).

setRetentionMs Method

```
void setRetentionMs(final long retentionMs)
```

`setRetentionMs` ...FIXME

Note	<code>setRetentionMs</code> is used when...FIXME
------	--

UnwindowedChangelogTopicConfig

`UnwindowedChangelogTopicConfig` is an [InternalTopicConfig](#) that...FIXME

`UnwindowedChangelogTopicConfig` is [created](#) exclusively when `InternalTopologyBuilder` is requested for [create an InternalTopicConfig \(for a given name and StateStoreFactory\)](#).

When [created](#), `UnwindowedChangelogTopicConfig` creates a `UNWINDOWED_STORE_CHANGELOG_TOPIC_DEFAULT_OVERRIDES` configuration properties that...FIXME

Creating UnwindowedChangelogTopicConfig Instance

`UnwindowedChangelogTopicConfig` takes the following when created:

- Name
- Topic configs (as `Map<String, String>`)

`UnwindowedChangelogTopicConfig` initializes the [internal registries and counters](#).

WindowedStreamPartitioner — Default StreamPartitioner of Windowed Keys

`WindowedStreamPartitioner` is a [StreamPartitioner](#) of [Windowed](#) keys (and `v` values).

`WindowedStreamPartitioner` is used as the default `StreamPartitioner` when [WindowedSerializer](#) is used for record keys.

`WindowedStreamPartitioner` is [created](#) exclusively when `streamSinkNode` is requested to [writeToTopology](#).

`WindowedStreamPartitioner` takes a [WindowedSerializer](#) when created.

When requested to [determine the partition for a record](#), `WindowedStreamPartitioner` requests the [WindowedSerializer](#) to [serializeBaseKey](#) (for the given topic and `windowedKey`). It then generates 32-bit murmur2 hash from the byte array and chooses the partition.

WindowedSerializer Contract

`WindowedSerializer` is the [extension](#) of the `Serializer` contract in Apache Kafka for [serializers](#) that can [serialize Windowed keys](#).

`WindowedSerializer` defines the single `serializeBaseKey` method to convert ([serialize](#)) objects of [Windowed](#) type to bytes.

```
byte[] serializeBaseKey(String topic, Windowed<T> data)
```

`serializeBaseKey` is used exclusively when `WindowedStreamPartitioner` is requested to [determine the partition for a record](#).

Table 1. WindowedSerializers

WindowedSerializer	Description
SessionWindowedSerializer	
TimeWindowedSerializer	

DefaultKafkaClientSupplier

`DefaultKafkaClientSupplier` is the default [KafkaClientSupplier](#) that...FIXME

SessionWindow

`SessionWindow` is a concrete `Window` with a custom `overlap`.

`SessionWindow` is `created` when:

- `KStreamSessionWindowAggregate` is requested to `mergeSessionWindow`
- `KStreamSessionWindowAggregateProcessor` is requested to `process a single record`
- `SessionKeySchema` is requested to `upperRangeFixedSize` and `lowerRangeFixedSize`

Creating SessionWindow Instance

`SessionWindow` takes the following when created:

- The start timestamp of the window (in milliseconds)
- The end timestamp of the window (in milliseconds)

Checking If Two Windows Overlap — `overlap` Method

```
boolean overlap(final Window other) throws IllegalArgumentException
```

Note	<code>overlap</code> is part of <code>Window Contract</code> to check whether a window overlaps with another.
------	---

`overlap` compares the start and end timestamps and is positive (`true`) when `end` timestamp of either window is exactly or greater than the `start` timestamp of the other window.

`overlap` throws a `IllegalArgumentException` when the `other` window is not a `SessionWindow`.

```
Cannot compare windows of different type. Other window has type [className].
```

TimeWindow

TimeWindow is a concrete [window](#).

UnlimitedWindow

UnlimitedWindow is...FIXME

AbstractStream

`AbstractStream` is the base abstraction of the [streams](#).

Table 1. AbstractStreams

AbstractStream	Description
<code>KGroupedStreamImpl</code>	
<code>KGroupedTableImpl</code>	
<code>KStreamImpl</code>	
<code>KTableImpl</code>	
<code>SessionWindowedKStreamImpl</code>	
<code>TimeWindowedKStreamImpl</code>	

Every `AbstractStream` has the following properties:

- Name
- Key Serde (`Serde<K>`)
- Value Serde (`Serde<V>`)
- Names of the source nodes (`Set<String>`)
- [StreamsGraphNode](#)
- [InternalStreamsBuilder](#)

ensureJoinableWith Method

```
Set<String> ensureJoinableWith(
    AbstractStream<K, ?> other)
```

`ensureJoinableWith ...FIXME`

Note

- `ensureJoinableWith` is used when:
- `KStreamImpl` is requested to `join`, `outerJoin` and `leftJoin`
 - `KTableImpl` is requested to `join`, `leftJoin` and `outerJoin`

toValueTransformerWithKeySupplier Static Method

```
ValueTransformerWithKeySupplier<K, V, VR> toValueTransformerWithKeySupplier(
    ValueTransformerSupplier<V, VR> valueTransformerSupplier)
```

`toValueTransformerWithKeySupplier` ...FIXME

Note

`toValueTransformerWithKeySupplier` is used when `KStreamImpl` is requested to `transformValues` and `flatTransformValues`.

reverseJoiner Static Method

```
ValueJoiner<T2, T1, R> reverseJoiner(
    ValueJoiner<T1, T2, R> joiner)
```

`reverseJoiner` ...FIXME

Note

- `reverseJoiner` is used when:
- `KStreamImplJoin` is requested to `join` (for `KStreamImpl.join`, `KStreamImpl.outerJoin` and `KStreamImpl.leftJoin` operators)
 - `KTableImpl` is requested to `doJoin` (for `KTableImpl.join`, `KTableImpl.leftJoin` and `KTableImpl.outerJoin` operators)

withKey Static Method

```
ValueMapperWithKey<K, V, VR> withKey(
    ValueMapper<V, VR> valueMapper)
```

`withKey` ...FIXME

Note

- `withKey` is used when:
- `KStreamImpl` is requested to `mapValues` and `flatMapValues`
 - `KTableImpl` is requested to `mapValues`

GlobalKTableImpl

`GlobalKTableImpl` is a [GlobalKTable](#) (of `k` primary keys and `v` value changes) that uses a [KTableValueGetterSupplier](#) for the `queryableStoreName` when `queryable`.

`GlobalKTableImpl` is created exclusively when `InternalStreamsBuilder` is requested to add a global table to a topology (exclusively when `StreamsBuilder` is requested to create a `GlobalKTable`).

```
import org.apache.kafka.streams.StreamsBuilder
val builder = new StreamsBuilder

val globalTable = builder.globalTable("topic")

import org.apache.kafka.streams.kstream.internals.GlobalKTableImpl
val impl = globalTable.asInstanceOf[GlobalKTableImpl[_, _]]

scala> println(impl.queryableStoreName)
null
```

Creating GlobalKTableImpl Instance

`GlobalKTableImpl` takes the following when created:

- [KTableValueGetterSupplier](#) (of `k` primary keys and `v` value changes)
- `queryable` flag

queryableStoreName Method

```
String queryableStoreName()
```

Note

`queryableStoreName` is part of [GlobalKTable Contract](#) to...FIXME.

Only when `queryable` is enabled, `queryableStoreName` requests the [KTableValueGetterSupplier](#) for `storeNames` and takes the very first one.

Otherwise, `queryableStoreName` returns `null`.

KGroupedStreamImpl

`KGroupedStreamImpl` is a concrete `stream` of grouped records that is created for `KStream.groupBy` and `KStream.groupByKey` streaming operators.

`KGroupedStreamImpl` takes the following to be created:

- Name
- Names of the source nodes
- `GroupedInternal<K, V>`
- `repartitionRequired` flag
- `StreamsGraphNode`
- `InternalStreamsBuilder`

`KGroupedStreamImpl` uses a `GroupedStreamAggregateBuilder` for...FIXME

aggregate Method

```
KTable<K, VR> aggregate(
    final Initializer<VR> initializer,
    final Aggregator<? super K, ? super V, VR> aggregator)
KTable<K, VR> aggregate(
    final Initializer<VR> initializer,
    final Aggregator<? super K, ? super V, VR> aggregator,
    final Materialized<K, VR, KeyValueStore<Bytes, byte[]>> materialized)
```

Note	<code>aggregate</code> is part of <code>KGroupedStream Contract</code> to...FIXME.
------	--

`aggregate` ...FIXME

reduce Method

```
KTable<K, V> reduce(final Reducer<V> reducer)
KTable<K, V> reduce(
    final Reducer<V> reducer,
    final Materialized<K, V, KeyValueStore<Bytes, byte[]>> materialized)
```

Note	<code>reduce</code> is part of <code>KGroupedStream Contract</code> to...FIXME.
------	---

```
reduce ...FIXME
```

Creating SessionWindowedKStream Instance — windowedBy Method

```
SessionWindowedKStream<K, V> windowedBy(final SessionWindows windows)
```

Note

windowedBy is part of KGroupedStream Contract to...FIXME.

```
windowedBy ...FIXME
```

Creating TimeWindowedKStream Instance — windowedBy Method

```
TimeWindowedKStream<K, V> windowedBy(final Windows<W> windows)
```

Note

windowedBy is part of KGroupedStream Contract to...FIXME.

```
windowedBy ...FIXME
```

Building KTable — doAggregate Internal Method

```
KTable<K, T> doAggregate(
    final KStreamAggProcessorSupplier<K, ?, V, T> aggregateSupplier,
    final String functionName,
    final MaterializedInternal<K, T, KeyValueStore<Bytes, byte[]>> materializedInternal)
```

```
doAggregate ...FIXME
```

Note

doAggregate is used when KGroupedStreamImpl is requested to reduce, aggregate and doCount (when count).

doCount Internal Method

```
KTable<K, Long> doCount(final Materialized<K, Long, KeyValueStore<Bytes, byte[]>> mate-
rialized)
```

`doCount` creates a `MaterializedInternal` for the given `Materialized` and requests it to `generateStoreNameIfNeeded` (with the `InternalStreamsBuilder` and **KSTREAM-AGGREGATE-** store name prefix).

Unless specified (and not `null`), `doCount` defines key and value serdes.

In the end, `doCount doAggregate` with a new `KStreamAggregate`, **KSTREAM-AGGREGATE-** function name prefix, and the `MaterializedInternal`.

Note	<code>doCount</code> is used exclusively when <code>KGroupedStreamImpl</code> is requested to <code>count</code> .
------	--

count Method

```
KTable<K, Long> count()
KTable<K, Long> count(final Materialized<K, Long, KeyValueStore<Bytes, byte[]>> materialized)
```

Note	<code>count</code> is part of the <code>KGroupedStream Contract</code> to...FIXME.
------	--

`count` simply `doCount` with the given `Materialized` or one with no state store name, the key serde and `Long` serde for values.

KGroupedTableImpl

`KGroupedTableImpl` is...FIXME

buildAggregate Internal Method

```
void buildAggregate(
    final ProcessorSupplier<K, Change<V>> aggregateSupplier,
    final String topic,
    final String funcName,
    final String sourceName,
    final String sinkName)
```

`buildAggregate` ...FIXME

Note	<code>buildAggregate</code> is used exclusively when <code>KGroupedTableImpl</code> is requested to doAggregate .
------	---

doAggregate Internal Method

```
<T> KTable<K, T> doAggregate(
    final ProcessorSupplier<K, Change<V>> aggregateSupplier,
    final String functionName,
    final MaterializedInternal<K, T, KeyValueStore<Bytes, byte[]>> materialized)
```

`doAggregate` ...FIXME

Note	<code>doAggregate</code> is used when <code>KGroupedTableImpl</code> is requested to reduce , count and aggregate .
------	---

reduce Method

```
KTable<K, V> reduce(
    final Reducer<V> adder,
    final Reducer<V> subtractor,
    final Materialized<K, V, KeyValueStore<Bytes, byte[]>> materialized)
```

Note	<code>reduce</code> is part of the KGroupedTable Contract to...FIXME.
------	---

`reduce` ...FIXME

count Method

```
KTable<K, Long> count(final Materialized<K, Long, KeyValueStore<Bytes, byte[]>> materialized)
```

Note

`count` is part of the [KGroupedTable Contract](#) to...FIXME.

`count` ...FIXME

aggregate Method

```
KTable<K, VR> aggregate(
    final Initializer<VR> initializer,
    final Aggregator<? super K, ? super V, VR> adder,
    final Aggregator<? super K, ? super V, VR> subtractor,
    final Materialized<K, VR, KeyValueStore<Bytes, byte[]>> materialized)
```

Note

`aggregate` is part of the [KGroupedTable Contract](#) to...FIXME.

`aggregate` ...FIXME

createRepartitionNode Internal Method

```
GroupedTableOperationRepartitionNode<K, V> createRepartitionNode(
    final String sinkName,
    final String sourceName,
    final String topic)
```

`createRepartitionNode` ...FIXME

Note

`createRepartitionNode` is used exclusively when `KGroupedTableImpl` is requested to `doAggregate` (when requested to `reduce`, `count` and `aggregate`).

KStreamAggregate

`KStreamAggregate` is...FIXME

KStreamImpl

`KStreamImpl` is a concrete `stream` and the default `KStream`.

`KStreamImpl` is `created` every time a transformation is executed and when...FIXME

`KStreamImpl` uses **KSTREAM-SOURCE-** as the prefix for...FIXME

`KStreamImpl` uses **KSTREAM-SINK-** as the prefix for...FIXME

`KStreamImpl` uses **KSTREAM-TRANSFORMVALUES-** as the prefix for...FIXME

Table 1. KStreamImpl's Internal Properties (e.g. Registries, Counters and Flags)

Name	Description
<code>defaultKeyValueMapper</code>	<code>KeyValueMapper</code> that simply concatenates the key and the value of a record when <code>executed</code> (i.e. applied to a record stream)

doStreamTableJoin Internal Method

```
KStream<K, VR> doStreamTableJoin(
    KTable<K, V0> other,
    ValueJoiner<? super V, ? super V0, ? extends VR> joiner,
    Joined<K, V, V0> joined,
    boolean leftJoin)
```

`doStreamTableJoin` ...FIXME

Note	<code>doStreamTableJoin</code> is used when <code>KStreamImpl</code> is requested to <code>join</code> and <code>leftJoin</code> .
------	--

doJoin Internal Method

```
KStream<K, VR> doJoin(
    KStream<K, V0> other,
    ValueJoiner<? super V, ? super V0, ? extends VR> joiner,
    JoinWindows windows,
    Joined<K, V, V0> joined,
    KStreamImplJoin join)
```

`doJoin` ...FIXME

Note	<code>doJoin</code> is used when <code>KStreamImpl</code> is requested to <code>join</code> , <code>outerJoin</code> and <code>leftJoin</code> .
------	--

Transforming Values with Optional State — `transformValues` Method

```
KStream<K, V1> transformValues(
    final ValueTransformerSupplier<? super V, ? extends V1> valueTransformerSupplier,
    final String... stateStoreNames)
KStream<K, VR> transformValues(
    final ValueTransformerWithKeySupplier<? super K, ? super V, ? extends VR> valueTransformerSupplier,
    final String... stateStoreNames)
```

Note	<code>transformValues</code> is part of KStream Contract for a stateful record-by-record value transformation.
------	--

`transformValues` ...FIXME

`transformValues` reports a `NullPointerException` when either `ValueTransformerSupplier` or `ValueTransformerWithKeySupplier` is null .

process Method

```
void process(
    ProcessorSupplier<? super K, ? super V> processorSupplier,
    String... stateStoreNames)
```

Note	<code>process</code> is part of KStream Contract for...FIXME
------	--

`process` ...FIXME

Transforming Record Into Zero or More Output Records with State (Stateful Transformation) — `transform` Operator

```
KStream<K1, V1> transform(
    TransformerSupplier<
        ? super K,
        ? super V,
        KeyValue<K1, V1>> transformerSupplier,
    String... stateStoreNames)
```

Note	<code>transform</code> is part of KStream API to transform a record into zero or more output records with state.
------	--

`transform` requests `InternalStreamsBuilder` for a new processor name with **KSTREAM-TRANSFORM** prefix.

`transform` then creates a new `KStreamTransform` (for the input `transformerSupplier`) and requests `InternalTopologyBuilder` to register a new processor supplier under the name.

Note

`transform` uses `InternalStreamsBuilder` to access the current `InternalTopologyBuilder`.

`transform` requests `InternalTopologyBuilder` to connect the processor supplier with state stores if specified.

In the end, `transform` creates a new `KStreamImpl` for the processor name and `repartitionRequired` flag turned on.

`transform` reports a `NullPointerException` when `transformerSupplier` is `null`.

Registering Processor Node with KStreamMapValues Processor Supplier and KSTREAM-MAPVALUES Prefix — `mapValues` Operator

```
<V1> KStream<K, V1> mapValues(final ValueMapper<? super V, ? extends V1> mapper)
```

Note

`mapValues` is part of `KStream Contract` to...FIXME.

`mapValues` creates a new `KStreamImpl` for a new processor (with the current `InternalStreamsBuilder`, source nodes and `repartitionRequired` flag).

Internally, `mapValues` requests `InternalStreamsBuilder` for a new processor name with **KSTREAM-MAPVALUES** prefix and registers a `KStreamMapValues` processor supplier under the name.

Note

`mapValues` uses `InternalStreamsBuilder` to access the current `InternalTopologyBuilder`.

Passing Stream Through Topic — `through` Method

```
KStream<K, V> through(
    String topic) (1)
KStream<K, V> through(
    String topic,
    Produced<K, V> produced)
```

1. Uses `Produced.with` factory method to create a `Produced`

Note `through` is part of [KStream Contract](#) to materialize a stream to a topic (*pass it through*) and create a new `kstream` from the topic (using the `Produced` for configuration).

`through` creates a new `ProducedInternal` for the given `Produced` and sets the key and value serdes if both `null` with the `keySerde` and `valSerde`, respectively.

`through` writes the stream to the topic (with the `ProducedInternal`).

In the end, `through` creates a `ConsumedInternal` (with `FailOnInvalidTimestamp` timestamp extractor) and requests the `InternalStreamsBuilder` to create a `KStream` to read from the topic.

Registering Processor Node with KStreamPrint Processor Supplier and KSTREAM-PRINTER Prefix — `print` Operator

```
void print(final Printed<K, V> printed)
```

Note `print` is part of [KStream Contract](#) to...FIXME.

`print` creates a `PrintedInternal` for the input `Printed`.

`print` requests `InternalStreamsBuilder` for a new processor name with **KSTREAM-PRINTER** prefix and registers a `KstreamPrint` (with `PrintForeachAction`) processor supplier under the name.

Note `print` uses `InternalStreamsBuilder` to access the current `InternalTopologyBuilder`.

Adding StreamSinkNode to Node Graph — `to` Operator

```
void to(
    String topic) (1)
void to(
    String topic,
    Produced<K, V> produced)
void to(
    TopicNameExtractor<K, V> topicExtractor)
void to(
    TopicNameExtractor<K, V> topicExtractor,
    Produced<K, V> produced)
```

1. Calls the other `to` with `Produced` of `nulls`

Note	<code>to</code> is part of KStream Contract to...FIXME.
------	---

`to` merely passes the call on to the internal `to` with a new `ProducedInternal` for the input `Produced`.

to Internal Method

```
void to(
    final TopicNameExtractor<K, V> topicExtractor,
    final ProducedInternal<K, V> produced)
```

`to` requests the [InternalStreamsBuilder](#) for a [new processor name](#) with [KSTREAM-SINK](#) prefix.

`to` creates a new [StreamSinkNode](#) and requests the [InternalStreamsBuilder](#) to [add it](#) to the parent [StreamsGraphNode](#).

Note	<code>to</code> is used in <code>to</code> operators.
------	---

repartitionForJoin Internal Method

```
KstreamImpl<K, V> repartitionForJoin(
    final Serde<K> keySerde,
    final Serde<V> valSerde)
```

`repartitionForJoin` ...FIXME

Note	<code>repartitionForJoin</code> is used when...FIXME
------	--

Creating KStreamImpl Instance

`KStreamImpl` takes the following when created:

- **Name**
- `Serde` for keys
- `Serde` for values
- **Names of the source nodes**
- **repartitionRequired flag**

- Parent StreamsGraphNode
- InternalStreamsBuilder

`KStreamImpl` initializes the internal registries and counters.

Transforming Values with State — `transformValues` Internal Method

```
private <VR> KStream<K, VR> transformValues(
    final InternalValueTransformerWithKeySupplier<? super K, ? super VR, ? extends VR> internalValueTransformerWithKeySupplier,
    final String... stateStoreNames)
```

`transformValues` requests `InternalStreamsBuilder` for a new processor name with **KSTREAM-TRANSFORMVALUES** prefix.

`transformValues` then creates a new `KStreamTransformValues` (for the input `internalValueTransformerWithKeySupplier`) and requests `InternalTopologyBuilder` to register a new processor supplier under the name.

Note	<code>transformValues</code> uses <code>InternalStreamsBuilder</code> to access the current <code>InternalTopologyBuilder</code> .
------	--

`transformValues` requests `InternalTopologyBuilder` to connect the processor supplier with state stores if specified.

In the end, `transformValues` creates a new `KStreamImpl` for the processor name.

Note	<code>transformValues</code> is used exclusively when <code>KStreamImpl</code> is requested to <code>transformValues</code> .
------	---

createRepartitionedSource Static Method

```
String createRepartitionedSource(
    final InternalStreamsBuilder builder,
    final Serde<K1> keySerde,
    final Serde<V1> valSerde,
    final String topicNamePrefix,
    final String name)
```

`createRepartitionedSource` requests the input `InternalStreamsBuilder` for the `InternalTopologyBuilder` and does the following:

- Requests the `InternalTopologyBuilder` to [addInternalTopic](#) with the topic name as the input `topicNamePrefix` (if defined) or the input `name` and `-repartition` suffix
- Requests the `InternalStreamsBuilder` for a new processor name with `KSTREAM-FILTER-` prefix and requests the `InternalTopologyBuilder` to [addProcessor](#) with the new processor name and a new `KStreamFilter` (that filters out `null` keys) and the `name` predecessor
- Requests the `InternalStreamsBuilder` for a new processor name with `KSTREAM-SINK-` prefix and requests the `InternalTopologyBuilder` to [add a sink node](#) with the new processor name, the repartition topic and the new `KStreamFilter` as a predecessor
- Requests the `InternalStreamsBuilder` for a new processor name with `KSTREAM-SOURCE-` prefix (aka `sourceName`) and requests the `InternalTopologyBuilder` to [add a source node](#) with the new processor name, a `FailOnInvalidTimestamp` and the repartition topic

In the end, `createRepartitionedSource` returns the source name.

```
// CAUTION: FIXME Example
```

- | | |
|------|--|
| Note | <code>createRepartitionedSource</code> is used when: <ul style="list-style-type: none"> <code>GroupedStreamAggregateBuilder</code> is requested to repartitionIfRequired <code>KStreamImpl</code> is requested to repartitionForJoin |
|------|--|

createWindowedStateStore Internal Static Method

```
<K, V> StoreBuilder<WindowStore<K, V>> createWindowedStateStore(
    final JoinWindows windows,
    final Serde<K> keySerde,
    final Serde<V> valueSerde,
    final String storeName)
```

`createWindowedStateStore` ...FIXME

- | | |
|------|--|
| Note | <code>createWindowedStateStore</code> is used exclusively when <code>KStreamImplJoin</code> is requested to join . |
|------|--|

groupBy Method

```
KGroupedStream<KR, V> groupBy(
    final KeyValueMapper<? super K, ? super V, KR> selector)
KGroupedStream<KR, V> groupBy(
    final KeyValueMapper<? super K, ? super V, KR> selector,
    final Grouped<KR, V> grouped)
```

Note`groupBy` is part of the [KStream Contract](#) to...FIXME.`groupBy` ...FIXME

groupByKey Method

```
KGroupedStream<K, V> groupByKey()
KGroupedStream<K, V> groupByKey(final Grouped<K, V> grouped)
```

Note`groupByKey` is part of the [KStream Contract](#) to...FIXME.`groupByKey` ...FIXME

filter Method

```
KStream<K, V> filter(final Predicate<? super K, ? super V> predicate)
```

Note`filter` is part of the [KStream Contract](#) to...FIXME.`filter` ...FIXME

filterNot Method

```
KStream<K, V> filterNot(final Predicate<? super K, ? super V> predicate)
```

Note`filterNot` is part of the [KStream Contract](#) to...FIXME.`filterNot` ...FIXME

flatTransform Method

```
KStream<K1, V1> flatTransform(
    TransformerSupplier<
        ? super K,
        ? super V,
        Iterable<KeyValue<K1, V1>>> transformerSupplier,
    String... stateStoreNames)
```

Note`flatTransform` is part of the [KStream Contract](#) to...FIXME.`flatTransform` ...FIXME

flatTransformValues Method

```
KStream<K, VR> flatTransformValues(
    ValueTransformerSupplier<
        ? super V,
        Iterable<VR>> valueTransformerSupplier,
    String... stateStoreNames)
KStream<K, VR> flatTransformValues(
    ValueTransformerWithKeySupplier<
        ? super K,
        ? super V,
        Iterable<VR>> valueTransformerSupplier,
    String... stateStoreNames)
```

Note`flatTransformValues` is part of the [KStream Contract](#) to...FIXME.`flatTransformValues` ...FIXME

doTransformValues Internal Method

```
KStream<K, VR> doTransformValues(
    ValueTransformerWithKeySupplier<
        ? super K,
        ? super V,
        ? extends VR> valueTransformerWithKeySupplier,
    String... stateStoreNames)
```

`doTransformValues` requests the [InternalStreamsBuilder](#) for a new processor name with [KSTREAM-TRANSFORMVALUES](#) prefix.

`doTransformValues` creates a new [StatefulProcessorNode](#) with the new processor name, the given `stateStoreNames` and the `repartitionRequired` flag.

`doTransformValues` requests the `StatefulProcessorNode` to [setValueChangingOperation](#).

`doTransformValues` requests the `InternalStreamsBuilder` to add the `statefulProcessorNode` (with the `StreamsGraphNode` as the parent).

In the end, `doTransformValues` creates a new `KStreamImpl` (with the new processor name, the `sourceNodes`, the `repartitionRequired` flag, the `statefulProcessorNode` itself and the `InternalStreamsBuilder`).

Note	<code>doTransformValues</code> is used exclusively when <code>KStreamImpl</code> is requested to <code>transformValues</code> .
------	---

internalSelectKey Internal Method

```
ProcessorGraphNode<K, V> internalSelectKey(
    final KeyValueMapper<? super K, ? super V, ? extends KR> mapper)
```

`internalSelectKey` ...FIXME

Note	<code>internalSelectKey</code> is used when <code>KStreamImpl</code> is requested to <code>selectKey</code> and <code>groupBy</code> .
------	--

doFlatTransform Internal Method

```
KStream<K1, V1> doFlatTransform(
    TransformerSupplier<
        ? super K,
        ? super V,
        Iterable<KeyValue<K1, V1>>> transformerSupplier,
        String... stateStoreNames)
```

`doFlatTransform` ...FIXME

Note	<code>doFlatTransform</code> is used when <code>KStreamImpl</code> is requested to <code>transform</code> and <code>flatTransform</code> .
------	--

doFlatTransformValues Internal Method

```
KStream<K, VR> doFlatTransformValues(
    ValueTransformerWithKeySupplier<
        ? super K,
        ? super V,
        Iterable<VR>> valueTransformerWithKeySupplier,
        String... stateStoreNames)
```

```
doFlatTransformValues ...FIXME
```

Note

`doFlatTransformValues` is used exclusively when `kStreamImpl` is requested to [flatTransformValues](#).

globalTableJoin Internal Method

```
KStream<K, VR> globalTableJoin(  
    GlobalKTable<KG, VG> globalTable,  
    KeyValueMapper<  
        ? super K,  
        ? super V,  
        ? extends KG> keyMapper,  
    ValueJoiner<  
        ? super V,  
        ? super VG,  
        ? extends VR> joiner,  
    boolean leftJoin)
```

```
globalTableJoin ...FIXME
```

Note

`globalTableJoin` is used when `kStreamImpl` is requested to [join](#) and [leftJoin](#).

KTableImpl

`KTableImpl` is...FIXME

mapValues Method

```
KTable<K, VR> mapValues(
    final ValueMapper<? super V, ? extends VR> mapper)
KTable<K, VR> mapValues(
    final ValueMapper<? super V, ? extends VR> mapper,
    final Materialized<K, VR, KeyValueStore<Bytes, byte[]>> materialized)
KTable<K, VR> mapValues(
    final ValueMapperWithKey<? super K, ? super V, ? extends VR> mapper)
KTable<K, VR> mapValues(
    final ValueMapperWithKey<? super K, ? super V, ? extends VR> mapper,
    final Materialized<K, VR, KeyValueStore<Bytes, byte[]>> materialized)
```

Note	<code>mapValues</code> is part of KTable Contract to...FIXME.
------	---

`mapValues` ...FIXME

Note	<code>mapValues</code> is used when...FIXME
------	---

join Method

```
<V1, R> KTable<K, R> join(
    final KTable<K, V1> other,
    final ValueJoiner<? super V, ? super V1, ? extends R> joiner)
<V0, VR> KTable<K, VR> join(
    final KTable<K, V0> other,
    final ValueJoiner<? super V, ? super V0, ? extends VR> joiner,
    final Materialized<K, VR, KeyValueStore<Bytes, byte[]>> materialized)
```

Note	<code>join</code> is part of KTable Contract to...FIXME.
------	--

`join` ...FIXME

Note	<code>join</code> is used when...FIXME
------	--

leftJoin Method

```
<V1, R> KTable<K, R> leftJoin(
    final KTable<K, V1> other,
    final ValueJoiner<? super V, ? super V1, ? extends R> joiner)
<V0, VR> KTable<K, VR> leftJoin(
    final KTable<K, V0> other,
    final ValueJoiner<? super V, ? super V0, ? extends VR> joiner,
    final Materialized<K, VR, KeyValueStore<Bytes, byte[]>> materialized)
```

Note	<code>leftJoin</code> is part of KTable Contract to...FIXME.
------	--

`leftJoin` ...FIXME

Note	<code>leftJoin</code> is used when...FIXME
------	--

outerJoin Method

```
<V1, R> KTable<K, R> outerJoin(
    final KTable<K, V1> other,
    final ValueJoiner<? super V, ? super V1, ? extends R> joiner)
<V0, VR> KTable<K, VR> outerJoin(
    final KTable<K, V0> other,
    final ValueJoiner<? super V, ? super V0, ? extends VR> joiner,
    final Materialized<K, VR, KeyValueStore<Bytes, byte[]>> materialized)
```

Note	<code>outerJoin</code> is part of KTable Contract to...FIXME.
------	---

`outerJoin` ...FIXME

Note	<code>outerJoin</code> is used when...FIXME
------	---

doFilter Internal Method

```
KTable<K, V> doFilter(
    Predicate<? super K, ? super V> predicate,
    MaterializedInternal<K, V, KeyValueStore<Bytes, byte[]>> materialized,
    boolean filterNot)
```

`doFilter` ...FIXME

Note	<code>doFilter</code> is used when <code>KTableImpl</code> is requested to filter and filterNot .
------	---

doMapValues Internal Method

```
KTable<K, VR> doMapValues(
    ValueMapperWithKey<
        ? super K,
        ? super V,
        ? extends VR> mapper,
    MaterializedInternal<K, VR, KeyValueStore<Bytes, byte[]>> materializedInternal)
```

doMapValues ...FIXME

Note	doMapValues is used exclusively when <code>KTableImpl</code> is requested to mapValues .
------	--

doTransformValues Internal Method

```
KTable<K, VR> doTransformValues(
    ValueTransformerWithKeySupplier<
        ? super K,
        ? super V,
        ? extends VR> transformerSupplier,
    MaterializedInternal<K, VR, KeyValueStore<Bytes, byte[]>> materialized,
    String... stateStoreNames)
```

doTransformValues ...FIXME

Note	doTransformValues is used exclusively when <code>KTableImpl</code> is requested to transformValues .
------	--

buildJoin Internal Method

```
<V1, R> KTable<K, R> buildJoin(
    final AbstractStream<K> other,
    final ValueJoiner<? super V, ? super V1, ? extends R> joiner,
    final boolean leftOuter,
    final boolean rightOuter,
    final String joinMergeName,
    final String internalQueryableName)
```

`buildJoin` [ensureJoinableWith](#) the `other` stream.

buildJoin ...FIXME

Note	buildJoin is used exclusively when <code>KTableImpl</code> is requested to doJoin .
------	---

doJoin Internal Method

```
KTable<K, VR> doJoin(
    KTable<K, V0> other,
    ValueJoiner<? super V, ? super V0, ? extends VR> joiner,
    MaterializedInternal<K, VR, KeyValueStore<Bytes, byte[]>> materializedInternal,
    boolean leftOuter,
    boolean rightOuter)
```

doJoin ...FIXME

Note	doJoin is used when KTableImpl is requested to join, leftJoin and outerJoin.
------	--

valueGetterSupplier Method

```
KTableValueGetterSupplier<K, V> valueGetterSupplier()
```

valueGetterSupplier ...FIXME

Note	valueGetterSupplier is used when...FIXME
------	--

enableSendingOldValues Method

```
void enableSendingOldValues()
```

enableSendingOldValues ...FIXME

Note	enableSendingOldValues is used when...FIXME
------	---

filter Method

```
KTable<K, V> filter(
    final Predicate<? super K, ? super V> predicate)
KTable<K, V> filter(
    final Predicate<? super K, ? super V> predicate,
    final Materialized<K, V, KeyValueStore<Bytes, byte[]>> materialized)
```

Note	filter is part of the KTable Contract to...FIXME.
------	---

filter ...FIXME

filterNot Method

```
KTable<K, V> filterNot(
    final Predicate<? super K, ? super V> predicate)
KTable<K, V> filterNot(
    final Predicate<? super K, ? super V> predicate,
    final Materialized<K, V, KeyValueStore<Bytes, byte[]>> materialized)
```

Note`filterNot` is part of the [KTable Contract](#) to...FIXME.`filterNot` ...FIXME

transformValues Method

```
KTable<K, VR> transformValues(
    final ValueTransformerWithKeySupplier<? super K, ? super V, ? extends VR> transformerSupplier,
    final Materialized<K, VR, KeyValueStore<Bytes, byte[]>> materialized,
    final String... stateStoreNames)
KTable<K, VR> transformValues(
    final ValueTransformerWithKeySupplier<? super K, ? super V, ? extends VR> transformerSupplier,
    final String... stateStoreNames)
```

Note`transformValues` is part of the [KTable Contract](#) to...FIXME.`transformValues` ...FIXME

suppress Method

```
KTable<K, V> suppress(
    Suppressed<? super K> suppressed)
```

Note`suppress` is part of the [KTable Contract](#) to...FIXME.`suppress` ...FIXME

SessionWindowedKStreamImpl — Default SessionWindowedKStream

`SessionWindowedKStreamImpl` is a concrete `AbstractStream` and a `SessionWindowedKStream` (with the default implementation of `aggregate`, `count` and `reduce` operators).

`SessionWindowedKStreamImpl` is `created` exclusively when `KGroupedStreamImpl` is requested to `windowedBy (with a SessionWindows)`.

Table 1. `SessionWindowedKStreamImpl`'s Internal Properties (e.g. Registries, Counters and Flags)

Name	Description
<code>countMerger</code>	<code>Merger<? super K, ? super Long></code> that simply adds two values together. Used as the merger in <code>count</code>
<code>reduceInitializer</code>	<code>Initializer<? super V></code> that simply returns <code>null</code> when executed. Used as the initializer in <code>reduce</code>

aggregate Method

```
KTable<Windowed<K>, T> aggregate(
    final Initializer<T> initializer,
    final Aggregator<? super K, ? super V, T> aggregator,
    final Merger<? super K, T> sessionMerger)
KTable<Windowed<K>, VR> aggregate(
    final Initializer<VR> initializer,
    final Aggregator<? super K, ? super V, VR> aggregator,
    final Merger<? super K, VR> sessionMerger,
    final Materialized<K, VR, SessionStore<Bytes, byte[]>> materialized)
```

Note	<code>aggregate</code> is part of SessionWindowedKStream Contract to...FIXME.
------	---

`aggregate` ...FIXME

count Method

```
KTable<Windowed<K>, Long> count()
KTable<Windowed<K>, Long> count(final Materialized<K, Long, SessionStore<Bytes, byte[]>> materialized)
```

Note

`count` is part of [SessionWindowedKStream Contract](#) to...FIXME.

`count` ...FIXME

reduce Method

```
KTable<Windowed<K>, V> reduce(final Reducer<V> reducer)
KTable<Windowed<K>, V> reduce(
    final Reducer<V> reducer,
    final Materialized<K, V, SessionStore<Bytes, byte[]>> materialized)
```

Note

`reduce` is part of [SessionWindowedKStream Contract](#) to...FIXME.

`reduce` ...FIXME

Creating SessionWindowedKStreamImpl Instance

`SessionWindowedKStreamImpl` takes the following when created:

- `SessionWindows`
- [InternalStreamsBuilder](#)
- Names of the source nodes
- Name
- Key Serde (`Serde<K>`)
- Value Serde (`Serde<V>`)
- [GroupedStreamAggregateBuilder<K, V>](#)

`SessionWindowedKStreamImpl` initializes the [internal registries and counters](#).

doAggregate Internal Method

```
KTable<Windowed<K>, VR> doAggregate(  
    final Initializer<VR> initializer,  
    final Aggregator<? super K, ? super V, VR> aggregator,  
    final Merger<? super K, VR> merger,  
    final Serde<VR> serde)
```

doAggregate ...FIXME

Note

doAggregate is used when SessionWindowedKStreamImpl is requested to count, aggregate and reduce.

Materializing StoreBuilder (of SessionStores) — materialize Internal Method

```
StoreBuilder<SessionStore<K, VR>> materialize(  
    final MaterializedInternal<K, VR, SessionStore<Bytes, byte[]>> materialized)
```

materialize ...FIXME

Note

materialize is used exclusively when SessionWindowedKStreamImpl is requested to aggregate.

storeBuilder Internal Method

```
StoreBuilder<SessionStore<K, VR>> storeBuilder(  
    final String storeName,  
    final Serde<VR> aggValueSerde)
```

storeBuilder ...FIXME

Note

storeBuilder is used exclusively when SessionWindowedKStreamImpl is requested to doAggregate (when requested to count, aggregate and reduce).

aggregatorForReducer Internal Method

```
Aggregator<K, V, V> aggregatorForReducer(final Reducer<V> reducer)
```

aggregatorForReducer ...FIXME

Note

aggregatorForReducer is used when...FIXME

mergerForAggregator Internal Method

```
Merger<K, V> mergerForAggregator(final Aggregator<K, V, V> aggregator)
```

mergerForAggregator ...FIXME

Note

mergerForAggregator is used when...FIXME

doCount Internal Method

```
KTable<Windowed<K>, Long> doCount(final Materialized<K, Long, SessionStore<Bytes, byte[]>> materialized)
```

doCount ...FIXME

Note

doCount is used exclusively when SessionWindowedKStreamImpl is requested to count.

TimeWindowedKStreamImpl

`TimeWindowedKStreamImpl` is a stream for [time-windowed streaming aggregations](#).

`TimeWindowedKStreamImpl` is [created](#) exclusively when `KGroupedStreamImpl` is requested to [windowedBy](#) (with a Windows).

`TimeWindowedKStreamImpl` uses a [GroupedStreamAggregateBuilder](#) for...FIXME

Creating StoreBuilder (of WindowStores) — `materialize` Internal Method

```
StoreBuilder<WindowStore<K, VR>> materialize(
    final MaterializedInternal<K, VR, WindowStore<Bytes, byte[]>> materialized)
```

`materialize` creates a `StoreBuilder` of `WindowStores` per [MaterializedInternal](#).

Internally, `materialize` requests the given `MaterializedInternal` for [WindowBytesStoreSupplier](#).

Unless a [WindowBytesStoreSupplier](#) is specified, `materialize` ...FIXME

`materialize` uses the given `MaterializedInternal` for [retention](#).

`materialize` throws a `IllegalArgumentException` when...FIXME

```
The retention period of the window store [name] must be no smaller than its window size plus the grace period. Got size=[[size]], grace=[[gracePeriodMs]], retention=[[retentionPeriod]]
```

`materialize` uses the `Stores` factory object to [create a WindowBytesStoreSupplier](#) (with the store name and the `retentionPeriod` of the `MaterializedInternal` and the window size).

With `WindowBytesStoreSupplier` specified, `materialize` uses the `Stores` factory object to [create a StoreBuilder of WindowStores](#) (with the `WindowBytesStoreSupplier` and the key and value serdes from the given `MaterializedInternal`).

If the given `MaterializedInternal` has [loggingEnabled](#) flag enabled, `materialize` requests the `StoreBuilder` to [withLoggingEnabled](#) (with the `logConfig` of the given `MaterializedInternal`) or [withLoggingDisabled](#) otherwise.

If the given `MaterializedInternal` has `cachingEnabled` flag enabled, `materialize` requests the `StoreBuilder` to `withCachingEnabled`.

In the end, `materialize` returns the `StoreBuilder` of `WindowStores`.

Note	<code>materialize</code> is used when <code>TimeWindowedKStreamImpl</code> is requested to <code>doCount</code> , <code>aggregate</code> and <code>reduce</code> (when the internal <code>GroupedStreamAggregateBuilder</code> is requested to <code>build</code>).
------	--

aggregate Method

```
KTable<Windowed<K>, VR> aggregate(
    final Initializer<VR> initializer,
    final Aggregator<? super K, ? super V, VR> aggregator)
KTable<Windowed<K>, VR> aggregate(
    final Initializer<VR> initializer,
    final Aggregator<? super K, ? super V, VR> aggregator,
    final Materialized<K, VR, WindowStore<Bytes, byte[]>> materialized)
```

Note	<code>aggregate</code> is part of the <code>TimeWindowedKStream Contract</code> to...FIXME.
------	---

`aggregate` ...FIXME

reduce Method

```
KTable<Windowed<K>, V> reduce(final Reducer<V> reducer) (1)
KTable<Windowed<K>, V> reduce(
    final Reducer<V> reducer,
    final Materialized<K, V, WindowStore<Bytes, byte[]>> materialized)
```

1. Uses `Materialized` with the serdes for `keys` and `values`

Note	<code>reduce</code> is part of the <code>TimeWindowedKStream Contract</code> to combine record stream (by a grouped key).
------	---

`reduce` creates a `MaterializedInternal` for the given `Materialized` and requests it to `generateStoreNameIfNeeded` (with `KSTREAM-REDUCE-` store prefix).

In the end, `reduce` requests the `GroupedStreamAggregateBuilder` to `build` (with `KSTREAM-REDUCE-` store prefix and a new `kstreamWindowReduce`).

count Method

```
KTable<Windowed<K>, Long> count()
KTable<Windowed<K>, Long> count(final Materialized<K, Long, WindowStore<Bytes, byte[]> materialized)
```

Note`count` is part of the [TimeWindowedKStream Contract](#) to...FIXME.`count` ...FIXME

doCount Internal Method

```
KTable<Windowed<K>, Long> doCount(final Materialized<K, Long, WindowStore<Bytes, byte[]>> materialized)
```

`doCount` ...FIXME**Note**`doCount` is used exclusively when `TimeWindowedKStreamImpl` is requested to `count`.

Creating TimeWindowedKStreamImpl Instance

`TimeWindowedKStreamImpl` takes the following when created:

- [Windows](#)
- [InternalStreamsBuilder](#)
- Names of the source topics
- Name
- `Serde<K>` for keys
- `Serde<V>` for values
- [GroupedStreamAggregateBuilder](#)
- [StreamsGraphNode](#)

`TimeWindowedKStreamImpl` initializes the [internal registries and counters](#).

MaterializedInternal

`MaterializedInternal` is a [Materialized](#) that...FIXME

`MaterializedInternal` can be **queriable** which is...FIXME

`MaterializedInternal` uses the `loggingEnabled` flag for...FIXME

`MaterializedInternal` uses the `cachingEnabled` flag for...FIXME

`MaterializedInternal` uses the `topicConfig` registry with mappings for...FIXME

`MaterializedInternal` allows accessing the optional [StoreSupplier](#).

`MaterializedInternal` uses the [store name](#) for...FIXME

generateStoreNameIfNeeded Method

```
void generateStoreNameIfNeeded(
    final InternalNameProvider nameProvider,
    final String generatedStorePrefix)
```

`generateStoreNameIfNeeded` ...FIXME

Note	<code>generateStoreNameIfNeeded</code> is used when...FIXME
------	---

withLoggingDisabled Method

```
Materialized<K, V, S> withLoggingDisabled()
```

`withLoggingDisabled` turns the `loggingEnabled` flag off and removes all of the mappings from the [topicConfig](#).

In the end, `withLoggingDisabled` returns the current `Materialized` instance.

Note	<code>withLoggingDisabled</code> is used exclusively when <code>InternalStreamsBuilder</code> is requested to add a GlobalKTable to the topology .
------	--

Accessing Optional StoreSupplier— storeSupplier Method

```
StoreSupplier<S> storeSupplier()
```

`storeSupplier` simply returns the [StoreSupplier](#).

Note

`storeSupplier` is used when [KeyValueStoreMaterializer](#), [SessionWindowedKStreamImpl](#), and [TimeWindowedKStreamImpl](#) are requested to materialize a [StoreBuilder](#).

KeyValueStoreMaterializer

`KeyValueStoreMaterializer` can materialize a `StoreBuilder` (of `KeyValueStores`) (based on the given `MaterializedInternal`).

`KeyValueStoreMaterializer` is created (and immediately requested to materialize a `StoreBuilder`) when:

- `InternalStreamsBuilder` is requested to add a `KTable` or `GlobalKTable` to the topology
- `KGroupedStreamImpl` is requested to `doAggregate` (for `aggregate`, `count`, and `reduce` aggregation operators)
- `KGroupedTableImpl` is requested to `doAggregate` (for `aggregate`, `count`, and `reduce` aggregation operators)
- `KTableKTableJoinNode` and `TableProcessorNode` are requested to `writeToTopology`

`KeyValueStoreMaterializer` takes a single `MaterializedInternal` (with a `KeyValueStore<Bytes, byte[]>` state store) when created.

Materializing `StoreBuilder` (of `KeyValueStores`) — `materialize` Method

```
StoreBuilder<KeyValueStore<K, V>> materialize()
```

`materialize` creates ("materializes") a `StoreBuilder` (of `KeyValueStores`) based on the `MaterializedInternal`.

Internally, `materialize` requests the `MaterializedInternal` for the optional `StoreSupplier` (which is expected a `KeyValueBytesStoreSupplier`).

If not defined, `materialize` requests the `MaterializedInternal` for the `store name` and uses the `Stores` factory object for a new `persistent KeyValueBytesStoreSupplier`.

`materialize` uses the `Stores` factory object for a new `StoreBuilder<KeyValueStore<K, V>>` (for the `KeyValueBytesStoreSupplier`, the `keySerde` and the `valueSerde` of the `MaterializedInternal`).

`materialize` enables or disables the logging (of the `StoreBuilder`) per the `loggingEnabled` flag of the `MaterializedInternal`.

`materialize` enables caching (of the `StoreBuilder`) if caching is enabled for the [MaterializedInternal](#).

Note	<p><code>materialize</code> is used when:</p> <ul style="list-style-type: none">• <code>InternalStreamsBuilder</code> is requested to add a KTable or GlobalKTable to the topology• <code>KGroupedStreamImpl</code> is requested to doAggregate (for aggregate, count, and reduce aggregation operators)• <code>KGroupedTableImpl</code> is requested to doAggregate (for aggregate, count, and reduce aggregation operators)• <code>KTableKTableJoinNode</code> and <code>TableProcessorNode</code> are requested to writeToTopology
------	--

InternalNameProvider Contract

`InternalNameProvider` is the abstraction of [internal name providers](#) that give a new name for a [processor](#) or a [store](#).

Table 1. InternalNameProvider Contract

Method	Description
<code>newProcessorName</code>	<pre>String newProcessorName(String prefix)</pre> <p>Returns a name of a processor with the given prefix</p> <p>Used when:</p> <ul style="list-style-type: none"> • <code>GroupedStreamAggregateBuilder</code> is requested to build • <code>InternalStreamsBuilder</code> is requested to stream, table, globalTable and addGlobalStore • <code>KGroupedTableImpl</code> is requested to doAggregate • <code>KStreamImpl</code> is requested to filter, etc. • <code>KStreamImplJoin</code> is requested to join • <code>KTableImpl</code> is requested to doFilter, etc.
<code>newStoreName</code>	<pre>String newStoreName(String prefix)</pre> <p>Returns a name of a state store with the given prefix</p> <p>Used when:</p> <ul style="list-style-type: none"> • <code>KGroupedStreamImpl</code>, <code>SessionWindowedKStreamImpl</code>, and <code>TimeWindowedKStreamImpl</code> are requested to count • <code>KTableImpl</code> is requested to suppress • <code>MaterializedInternal</code> is requested to generateStoreNameIfNeeded
Note	<code>InternalStreamsBuilder</code> is the one and only known implementation of InternalNameProvider Contract in Kafka Streams.

GroupedStreamAggregateBuilder

`GroupedStreamAggregateBuilder` is created exclusively for a `KGroupedStreamImpl` (for `KStream.groupBy` and `KStream.groupByKey` streaming operators).

`GroupedStreamAggregateBuilder` takes the following to be created:

- `InternalStreamsBuilder`
- `GroupedInternal<K, V>`
- `repartitionRequired` flag
- Names of the source nodes
- Name
- `StreamsGraphNode`

Creating KTableImpl (For KStreamAggProcessorSupplier And New StatefulProcessorNode) — build Method

```
<T> KTable<K, T> build(
    KStreamAggProcessorSupplier<K, ?, V, T> aggregateSupplier,
    String functionName,
    StoreBuilder storeBuilder,
    boolean isQueryable)
```

`build` ...FIXME

	<p><code>build</code> is used when:</p> <ul style="list-style-type: none"> • <code>KGroupedStreamImpl</code> is requested to <code>doAggregate</code> (for <code>reduce</code>, <code>aggregate</code> and <code>count</code> operators after <code>KStream.groupBy</code> and <code>KStream.groupByKey</code>)
Note	<ul style="list-style-type: none"> • <code>SessionWindowedKStreamImpl</code> is requested to <code>doCount</code> (for <code>count</code> operator), <code>reduce</code> and <code>aggregate</code> • <code>TimeWindowedKStreamImpl</code> is requested to <code>doCount</code> (for <code>count</code> operator), <code>reduce</code> and <code>aggregate</code>

repartitionIfRequired Internal Method

```
String repartitionIfRequired(final String queryableStoreName)
```

`repartitionIfRequired ...FIXME`

Note

`repartitionIfRequired` is used exclusively when
`GroupedStreamAggregateBuilder` is requested to `build`.

KStreamImplJoin

`KStreamImplJoin` is...FIXME

join Method

```
KStream<K1, R> join(  
    KStream<K1, V1> lhs,  
    KStream<K1, V2> other,  
    ValueJoiner<? super V1, ? super V2, ? extends R> joiner,  
    JoinWindows windows,  
    Joined<K1, V1, V2> joined)
```

`join` ...FIXME

Note

`join` is used exclusively when `kstreamImpl` is requested to `doJoin` (for `KStreamImpl.join`, `KStreamImpl.outerJoin` and `KStreamImpl.leftJoin` operators).

StaticTopicNameExtractor

StaticTopicNameExtractor is...FIXME

ConsumedInternal — Internal Accessors to Consumed Metadata

`ConsumedInternal` is a [Consumed](#) that is used internally as an accessor object to the following properties of the managed [Consumed](#) instance:

- `Deserializer` of keys (only when [Serde](#) for keys is defined)
- `Deserializer` of values (only when [Serde](#) for values is defined)
- **Offset reset policy** ([Topology.AutoOffsetReset](#))
- [TimestampExtractor](#)

`ConsumedInternal` takes a single [Consumed](#), its parts or none at all (that are used to create a `Consumed`) when created:

- `Key Serde`
- `Value Serde`
- [TimestampExtractor](#)
- **Offset reset policy** ([Topology.AutoOffsetReset](#))

`ConsumedInternal` is [created](#) when:

- `StreamsBuilder` is requested to [stream](#), [table](#), [globalTable](#), and [addGlobalStore](#)
- `KStreamImpl` is requested to [through](#)

ProducedInternal — Internal Accessors to Produced Metadata

ProducedInternal is...FIXME

QuickUnion

QuickUnion (of elements of type `T`) is...FIXME

root Method

```
T root(T id)
```

root ...FIXME

Note	<code>root</code> is used when...FIXME
------	--

add Method

```
void add(T id)
```

add ...FIXME

Note	<code>add</code> is used when...FIXME
------	---------------------------------------

unite Method

```
void unite(T id1, T... idList)
```

unite ...FIXME

Note	<code>unite</code> is used when...FIXME
------	---

TopicsInfo

`TopicsInfo` is a simple "container" with the following:

- Names of the sink topics
- Names of the source topics
- `Map<String, InternalTopicConfig>`
- `Map<String, InternalTopicConfig>`

`TopicsInfo` is [created](#) exclusively when `InternalTopologyBuilder` is requested for the [topic groups](#) (when a node group has at least one source topic, incl. repartition or state changelog topics).

ProcessorTopology — Physical Processor Task Topology

`ProcessorTopology` is a **physical processor task topology** (of [logical processor node topology](#)).

A `ProcessorTopology` is associated with every [task](#).

A `ProcessorTopology` is used to create [AbstractTask](#), [GlobalStateManagerImpl](#), [GlobalStateUpdateTask](#), [GlobalStreamThread](#), [StandbyTask](#), and [StreamTask](#).

`ProcessorTopology` is [created](#) when `InternalTopologyBuilder` is requested to build [processor task](#) and [global state](#) topologies.

Note	Once created , all internal registries in <code>ProcessorTopology</code> will never change.
------	---

`ProcessorTopology` can also be created using the "[with](#)" methods.

Table 1. ProcessorTopology's with Methods

Method	Description
with	<pre>ProcessorTopology with(List<ProcessorNode> processorNodes, Map<String, SourceNode> sourcesByTopic, List<StateStore> stateStoresByName, Map<String, String> storeToChangelogTopic)</pre>
withGlobalStores	<pre>ProcessorTopology withGlobalStores(List<StateStore> stateStores, Map<String, String> storeToChangelogTopic)</pre>
withLocalStores	<pre>ProcessorTopology withLocalStores(List<StateStore> stateStores, Map<String, String> storeToChangelogTopic)</pre>
withRepartitionTopics	<pre>ProcessorTopology withRepartitionTopics(List<ProcessorNode> processorNodes, Map<String, SourceNode> sourcesByTopic, Set<String> repartitionTopics)</pre>
withSources	<pre>ProcessorTopology withSources(List<ProcessorNode> processorNodes, Map<String, SourceNode> sourcesByTopic)</pre>

```
// Using Scala with the Java-centric API of Kafka Streams
import collection.JavaConverters._

import org.apache.kafka.streams.processor.internals.ProcessorNode
val processorNodes = Seq.empty[ProcessorNode[_, _]].asJava
import org.apache.kafka.streams.processor.internals.SourceNode
val sourcesByTopic = Map.empty[String, SourceNode[_, _]].asJava
import org.apache.kafka.streams.processor.StateStore
val stateStoresByName = Seq.empty[StateStore].asJava
val storeToChangelogTopic = Map.empty[String, String].asJava

import org.apache.kafka.streams.processor.internals.ProcessorTopology
val topology = ProcessorTopology.`with`(processorNodes, sourcesByTopic, stateStoresByName, storeToChangelogTopic)

// The topology is empty (no children) so nothing is printed out except the header "ProcessorTopology:"
scala> println(topology)
ProcessorTopology:
```

Creating ProcessorTopology Instance

`ProcessorTopology` takes the following when created:

- `ProcessorNodes`
- `SourceNodes` by name
- `SinkNodes` by name
- Local `state stores`
- Global `state stores`
- Names of the `state stores` and the names of the corresponding changelog topics (from the `InternalTopologyBuilder`)
- Names of repartition topics

hasPersistentLocalStore Method

```
boolean hasPersistentLocalStore()
```

`hasPersistentLocalStore` is positive (`true`) when one of the `local StateStores` (in the `stateStores` internal registry) is `persistent`.

Note	<code>hasPersistentLocalStore</code> is used exclusively when <code>KafkaStreams</code> is <code>created</code> (to create a <code>StateDirectory</code>).
------	---

hasPersistentGlobalStore Method

```
boolean hasPersistentGlobalStore()
```

`hasPersistentGlobalStore` is positive (`true`) when one of the `global StateStores` (in the `globalStateStores` internal registry) is `persistent`.

Note	<code>hasPersistentGlobalStore</code> is used exclusively when <code>KafkaStreams</code> is <code>created</code> (to create a <code>StateDirectory</code>).
------	--

Task Contract—Stream Processor Tasks

`Task` is the abstraction of stream processor tasks.

Table 1. Task Contract

Method	Description
<code>applicationId</code>	<pre>String applicationId()</pre> <p>Application ID of the Kafka Streams application the task belongs to</p>
<code>changelogPartitions</code>	<pre>Collection<TopicPartition> changelogPartitions()</pre> <p>Changelog partitions associated with the task</p> <p>Used when:</p> <ul style="list-style-type: none"> <code>AssignedStreamsTasks</code> is requested to updateRestored and addToRestoring <code>AssignedTasks</code> is requested to mark a task as ready for execution and processing records <code>streamTask</code> is requested to initialize state stores
<code>close</code>	<pre>void close(boolean clean, boolean isZombie)</pre> <p>Closes the task</p> <p>Used when:</p> <ul style="list-style-type: none"> <code>AssignedTasks</code> is requested to closeNonRunningTasks, suspendTasks, closeZombieTask, close, and closeUnclean <code>StandbyTask</code> is requested to closeSuspended
<code>closeSuspended</code>	<pre>void closeSuspended(boolean clean, boolean isZombie, RuntimeException e)</pre> <p>Used when:</p>

	<ul style="list-style-type: none"> AssignedTasks is requested to <code>closeNonAssignedSuspendedTasks</code> StreamTask is requested to <code>close</code>
<code>commit</code>	<pre>void commit()</pre> <p>Commits the task</p> <p>Used when:</p> <ul style="list-style-type: none"> AssignedStreamsTasks is requested to <code>maybeCommitPerUserRequested</code> AssignedTasks is requested to <code>commit</code> StandbyTask is requested to <code>close</code>
<code>commitNeeded</code>	<pre>boolean commitNeeded()</pre> <p>Used when:</p> <ul style="list-style-type: none"> AssignedStreamsTasks is requested to <code>maybeCommitPerUserRequested</code> AssignedTasks is requested to <code>commit</code>
<code>context</code>	<pre>ProcessorContext context()</pre> <p>ProcessorContext of the task</p>
<code>getStore</code>	<pre>StateStore getStore(String name)</pre> <p>Looks up the state store by name</p> <p>Used exclusively when <code>StreamThreadStateStoreProvider</code> is requested for the stores</p>
<code>hasStateStores</code>	<pre>boolean hasStateStores()</pre>
<code>id</code>	<pre>TaskId id()</pre> <p>TaskId of the task</p>
	<pre>boolean initializeStateStores()</pre>

	<p>Initializes state stores (of the ProcessorTopology)</p> <p>Enabled (true) if the task has no state stores that need restoring.</p> <ul style="list-style-type: none"> StandbyTask always returns true <p>Used exclusively when AssignedTasks is requested to initialize new tasks</p>
initializeTopology	<pre>void initializeTopology()</pre> <p>Initializes the topology of processor nodes</p> <p>Used exclusively when AssignedTasks is requested to mark the specified task as ready for execution and processing records</p>
partitions	<pre>Set<TopicPartition> partitions()</pre> <p>Used when:</p> <ul style="list-style-type: none"> AssignedStreamsTasks is requested to addToRestoring AssignedTasks is requested to maybeResumeSuspendedTask, transitionToRunning, and closeNonAssignedSuspendedTasks StreamThread is requested to updateThreadMetadata
resume	<pre>void resume()</pre> <p>Resumes the task</p> <p>Used exclusively when AssignedTasks is requested to attempt to resume suspended task (when the task has actually been suspended)</p>
suspend	<pre>void suspend()</pre> <p>Suspends the task</p> <p>Used exclusively when AssignedTasks is requested to suspend tasks.</p>
topology	<pre>ProcessorTopology topology()</pre> <p>ProcessorTopology of the task</p>

	<pre>String toString(String indent)</pre>
	Describes the task (textual representation)

Note	<p>AbstractTask is the base implementation of the Task Contract in Kafka Streams (with StandbyTask and StreamTask as the only concrete implementations).</p>
------	--

AbstractTask — Base Processor Task

`AbstractTask` is the base implementation of the [Task contract](#) for [tasks](#) that [FIXME](#).

Note

`AbstractTask` is a Java abstract class and cannot be [created](#) directly. It is created indirectly for the [concrete AbstractTasks](#).

`AbstractTask` has `taskInitialized` flag that is on (i.e. `true`) to mark when:

- `StandbyTask` has finished [initializeStateStores](#) successfully
- `StreamTask` has finished [initializeTopology](#)

The `taskInitialized` flag is used by the [concrete AbstractTasks](#) as an optimization when:

- `StandbyTask` is requested to [close](#)
- `StreamTask` is requested to [closeTopology](#)

`AbstractTask` has `taskClosed` flag that is enabled (i.e. `true`) to mark when [StandbyTask](#) and [StreamTask](#) have already been requested to close.

Table 1. AbstractTasks

AbstractTask	Description
StandbyTask	
StreamTask	

Creating AbstractTask Instance

`AbstractTask` takes the following to be created:

- `TaskId`
- Assigned Kafka partitions
- [ProcessorTopology](#)
- Kafka [Consumer](#) (`Consumer<byte[], byte[]>`)
- [ChangelogReader](#)
- `isStandby` flag
- [StateDirectory](#)

- StreamsConfig

AbstractTask initializes the internal registries and counters.

Note

AbstractTask is a Java abstract class and cannot be created directly. It is created indirectly for the concrete AbstractTasks.

Closing State Manager — closeStateManager Method

```
void closeStateManager(final boolean writeCheckpoint) throws ProcessorStateException
```

closeStateManager ...FIXME

Note

closeStateManager is used when...FIXME

Checking If Topology Uses State Stores — hasStateStores Method

```
boolean hasStateStores()
```

Note

hasStateStores is part of Task Contract to...FIXME.

hasStateStores is positive (and returns true) when ProcessorTopology has at least one local state store.

Flushing All State Stores — flushState Method

```
void flushState()
```

flushState simply requests the ProcessorStateManager to flush.

Note

flushState is used exclusively when StreamTask is requested to commit (using the custom flushState).

recordCollectorOffsets Method

```
Map<TopicPartition, Long> recordCollectorOffsets()
```

recordCollectorOffsets ...FIXME

Note	<code>recordCollectorOffsets</code> is used when...FIXME
------	--

Registering and Initializing State Stores — `registerStateStores` Method

```
void registerStateStores()
```

`registerStateStores` simply exits when the [ProcessorTopology](#) has no state stores.

`registerStateStores` requests the [StateDirectory](#) for the [lock](#) for the state directory of the task (by [TaskId](#)).

`registerStateStores` prints out the following TRACE message to the logs:

```
Initializing state stores
```

`registerStateStores` [updateOffsetLimits](#).

In the end, for every [StateStore](#) in the [ProcessorTopology](#), `registerStateStores` prints out the following TRACE message to the logs:

```
Initializing store [name]
```

`registerStateStores` requests the [InternalProcessorContext](#) to [uninitialize](#) and then requests the [StateStore](#) to [initialize](#) (with the [InternalProcessorContext](#) and itself).

`registerStateStores` throws a [LockException](#) when requesting the [StateDirectory](#) for the lock for the state directory of the task failed:

```
[logPrefix]Failed to lock the state directory for task [id]
```

`registerStateStores` throws a [StreamsException](#) when requesting the [StateDirectory](#) for the lock for the state directory of the task failed with an [IOException](#):

```
[logPrefix]Fatal error while trying to lock the state directory for task [id]
```

Note	<code>registerStateStores</code> is used when the concrete Tasks (StandbyTask and StreamTask) are requested to initialize state stores .
------	--

changelogPartitions Method

```
Collection<TopicPartition> changelogPartitions()
```

Note `changelogPartitions` is part of [Task Contract](#) to get the changelog partitions of a task.

`changelogPartitions` simply requests the [ProcessorStateManager](#) for the [changelog partitions](#).

Accessing State Store by Name — `getStore` Method

```
StateStore getStore(final String name)
```

Note `getStore` is part of the [Task Contract](#) to access the [state store](#) by name.

`getStore` simply requests the [ProcessorStateManager](#) for the [StateStore](#) by name.

updateOffsetLimits Method

```
void updateOffsetLimits()
```

`updateOffsetLimits ...FIXME`

Note

`updateOffsetLimits` is used when:

- `AbstractTask` is requested to [registerStateStores](#)
- `StandbyTask` is requested to [resume](#)
- `StandbyTask` is requested to [commit](#)

reinitializeStateStoresForPartitions Method

```
void reinitializeStateStoresForPartitions(  
    Collection<TopicPartition> partitions)
```

`reinitializeStateStoresForPartitions` simply requests the [ProcessorStateManager](#) to [reinitializeStateStores](#) for the input `partitions` and the [InternalProcessorContext](#).

Note

- `reinitializeStateStoresForPartitions` is used when:
- `StoreChangelogReader` is requested to `restore`
 - `StreamThread` is requested to `maybeUpdateStandbyTasks`

Checkpointable Offsets

— `activeTaskCheckpointableOffsets` Method

```
Map<TopicPartition, Long> activeTaskCheckpointableOffsets()
```

`activeTaskCheckpointableOffsets` simply returns an empty collection (of checkpointable offsets).

Internal Properties

Name	Description
applicationId	
commitNeeded	
eosEnabled	<p>Flag that controls whether Exactly-Once Support is enabled (<code>true</code>) or not (<code>false</code>)</p> <ul style="list-style-type: none"> Enabled (<code>true</code>) when <code>StreamsConfig.PROCESSING_GUARANTEE_CONFIG</code> configuration property is <code>StreamsConfig.EXACTLY_ONCE</code> <p>Used when (directly or indirectly as <code>isEosEnabled</code> public method):</p> <ul style="list-style-type: none"> <code>ProcessorStateManager</code> is created <code>StoreChangelogReader</code> is requested to <code>restore state stores from changelog topics</code> <code>StreamTask</code> is created (and <code>initializeTransactions</code>), is requested to <code>initializeTopology</code>, <code>resume</code>, <code>commit</code>, <code>suspend</code>, and <code>maybeAbortTransactionAndCloseRecordCollector</code>
processorContext	<p><code>InternalProcessorContext</code>, i.e.</p> <ul style="list-style-type: none"> <code>ProcessorContextImpl</code> for <code>StreamTask</code> <code>StandbyContextImpl</code> for <code>StandbyTask</code>
stateMgr	<p><code>ProcessorStateManager</code></p> <p>Created when <code>AbstractTask</code> is created</p> <p>Used when:</p> <ul style="list-style-type: none"> <code>AbstractTask</code> is requested to <code>getStore</code>, <code>updateOffsetLimits</code>, <code>flushState</code>, <code>reinitializeStateStoresForPartitions</code>, <code>closeStateManager</code> and <code>changelogPartitions</code> <code>StandbyTask</code> is created (and creates the <code>InternalProcessorContext</code>), and is requested to <code>initializeStateStores</code>, <code>flushAndCheckpointState</code> and <code>update</code> <code>StreamTask</code> is created (and creates the <code>InternalProcessorContext</code>), and is requested to <code>commit</code> and <code>commitOffsets</code>

StandbyTask

`StandbyTask` is a concrete [processor task](#) that...FIXME

`StandbyTask` is [created](#) when...FIXME

`StandbyTask` is a concrete [AbstractTask](#) with `isStandby` flag on.

Tip Enable ALL logging level for
`org.apache.kafka.streams.processor.internals.StandbyTask` logger to see what happens inside.

Add the following line to `log4j.properties` :

```
log4j.logger.org.apache.kafka.streams.processor.internals.StandbyTask=ALL
```

Refer to [Application Logging Using log4j](#).

closeSuspended Method

```
void closeSuspended(  
    final boolean clean,  
    final boolean isZombie,  
    final RuntimeException e)
```

Note `closeSuspended` is part of [Task Contract](#) to...FIXME.

`closeSuspended` ...FIXME

Initializing State Stores — initializeStateStores Method

```
boolean initializeStateStores()
```

Note `initializeStateStores` is part of [Task Contract](#) to initialize state stores.

`initializeStateStores` ...FIXME

Closing Task — close Method

```
void close(
    final boolean clean,
    final boolean isZombie)
```

Note

`close` is part of [Task Contract](#) to close the task.

`close` prints out the following DEBUG message to the logs:

Closing

With the input `clean` flag on, `close commit` followed by `closeStateManager` (with `writeCheckpoint` flag on when `clean` was on and `commit` was successful).

`close` marks the `standbyTask` as closed (i.e. sets `taskClosed` to `true`).

`close` returns immediately if the `StandbyTask` has not been [initialized](#) yet, i.e. when `StandbyTask` has not been requested to [initializeStateStores](#) yet.

Note

The input `isZombie` flag is ignored (and therefore never used) as `StandbyTask` can never be a zombie.

initializeTopology Method

```
void initializeTopology()
```

Note

`initializeTopology` is part of [Task Contract](#) to...FIXME.

`initializeTopology` does nothing (and is simply a *noop*).

Resuming Task — resume Method

```
void resume()
```

Note

`resume` is part of the [Task Contract](#) to...FIXME.

`resume` ...FIXME

Creating StandbyTask Instance

`StandbyTask` takes the following to be created:

- `TaskId`
- Assigned Kafka `TopicPartitions`
- `ProcessorTopology`
- Kafka `Consumer` (`Consumer<byte[], byte[]>`)
- `ChangelogReader`
- `StreamsConfig`
- `StreamsMetricsImpl`
- `StateDirectory`

`StandbyTask` initializes the [internal properties](#).

flushAndCheckpointState Internal Method

```
void flushAndCheckpointState()
```

`flushAndCheckpointState` ...FIXME

Note	<code>flushAndCheckpointState</code> is used when <code>StandbyTask</code> is requested to commit and suspend .
------	---

commit Method

```
void commit()
```

Note	<code>commit</code> is part of Task Contract to...FIXME
------	---

`commit` ...FIXME

suspend Method

```
void suspend()
```

Note	<code>suspend</code> is part of Task Contract to...FIXME
------	--

`suspend` ...FIXME

Updating Standby Replicas Of State Store (From Single Change Log Partition) — update Method

```
List<ConsumerRecord<byte[], byte[]>> update(
    final TopicPartition partition,
    final List<ConsumerRecord<byte[], byte[]>> records)
```

`update` prints out the following TRACE message to the logs:

```
Updating standby replicas of its state store for partition [partition]
```

`update` then simply requests the [ProcessorStateManager](#) to [updateStandbyStates](#) and returns the result.

Note

`update` is used exclusively when `StreamThread` is requested to [maybeUpdateStandbyTasks](#).

Internal Properties

Name	Description
<code>processorContext</code>	InternalProcessorContext Used when...FIXME

StreamTask

`StreamTask` is a concrete [stream processor task](#) that uses a [PartitionGroup](#) (with the [partitions assigned](#)) to determine which record should be [processed](#) (as ordered by partition timestamp).

When requested to [process a single record](#), `StreamTask` requests the [PartitionGroup](#) for the [next stamped record \(record with timestamp\)](#) and the [RecordQueue](#). `StreamTask` uses a [RecordInfo](#) to hold the [RecordQueue](#) (with the source processor node and the partition) of the currently-processed stamped record. Eventually, `StreamTask` requests the source processor node (of the [RecordQueue](#) and the partition) to [process the record](#).

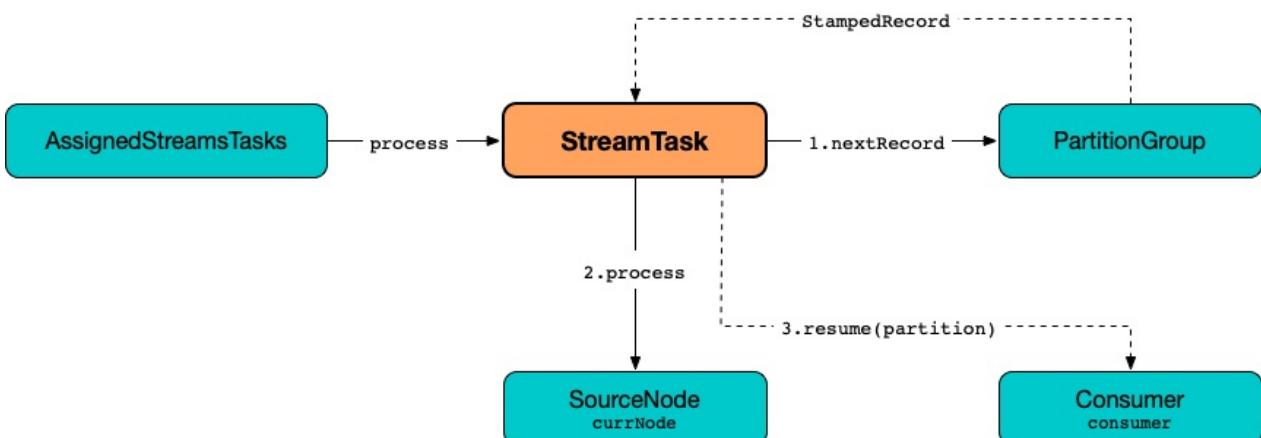


Figure 1. StreamTask and Processing Single Record

Note

It is at the discretion of a processor node (incl. a source processor node) to [forward the record downstream](#) (to child processors if there are any).

`StreamTask` is [created](#) exclusively when `TaskCreator` is requested to [create one](#).

`StreamTask` is a concrete [ProcessorNodePunctuator](#) that can punctuate processors ([execute scheduled periodic actions](#)).

When requested to [initialize a processor topology](#) (as a `task`), `StreamTask` ...FIXME

`StreamTask` uses `commitOffsetNeeded` flag to...FIXME

`StreamTask` uses `buffered.records.per.partition` configuration property (default: `1000`) to control whether to pause or resume a partition (when [processing a single record](#) or [bufferring new records](#), respectively).

`StreamTask` uses `max.task.idle.ms` configuration property (default: `0L`) to...FIXME

Tip	<p>Enable <code>ALL</code> logging level for <code>org.apache.kafka.streams.processor.internals.StreamTask</code> logger to see what happens inside.</p> <p>Add the following line to <code>log4j.properties</code> :</p> <pre>log4j.logger.org.apache.kafka.streams.processor.internals.StreamTask=ALL</pre> <p>Refer to Application Logging Using log4j.</p>
-----	--

Creating StreamTask Instance

`StreamTask` takes the following to be created:

- `TaskId`
- Assigned partitions (Kafka [TopicPartition](#))
- [ProcessorTopology](#)
- Kafka [Consumer](#) (`Consumer<byte[], byte[]>`)
- [ChangelogReader](#)
- [StreamsConfig](#)
- [StreamsMetricsImpl](#)
- [StateDirectory](#)
- [ThreadCache](#)
- `Time`
- [ProducerSupplier](#)
- [RecordCollector](#)
- `closeSensor Kafka Sensor`

`StreamTask` initializes the [internal properties](#).

StreamTask and PartitionGroup (of RecordCollectors per Assigned Partition)

`StreamTask` creates a [PartitionGroup](#) (with [RecordQueues](#) per every partition assigned) when...FIXME

`StreamTask` uses the `PartitionGroup` for the following:

- Buffer new records (from a partition)
- `isProcessable`
- Process a single record
- `closeTopology`
- `closeSuspended`
- `numBuffered`
- `maybePunctuateStreamTime`

StreamTask and RecordCollector

`StreamTask` creates a new `RecordCollector` (or is given one for testing) when `created`.

The `RecordCollector` is requested to `initialize` (with the `Kafka Producer`) when `StreamTask` is `created` and `resumed`.

`StreamTask` uses the `RecordCollector` for the following:

- Creating a `InternalProcessorContext` (while `being created`)
- Getting offsets when `activeTaskCheckpointableOffsets`
- Flushing the internal Kafka producer when `flushState`
- `suspend` and `maybeAbortTransactionAndCloseRecordCollector`

The `RecordCollector` is requested to `close` when `StreamTask` is requested to `suspend` and `maybeAbortTransactionAndCloseRecordCollector`.

Suspending Task — `suspend` Method

```
void suspend() (1)

// PRIVATE API
suspend(boolean clean, boolean isZombie)
```

1. Uses `clean` enabled (`true`) and `isZombie` disabled (`false`)

Note	<code>suspend</code> is part of <code>Task Contract</code> to suspend the task.
------	---

`suspend` prints out the following DEBUG message to the logs:

Suspending

`suspend` closes the processor topology.

With `clean` flag enabled (`true`), `suspend` commits the task (with `startNewTransaction` flag disabled) first. With exactly-once support enabled, `suspend` requests the ProcessorStateManager to checkpoint (with the checkpointable offsets) and the RecordCollector to close.

With `clean` flag disabled (`false`), `suspend` maybeAbortTransactionAndCloseRecordCollector.

Note	The private <code>suspend</code> is used when <code>StreamTask</code> is requested to close.
------	--

maybeAbortTransactionAndCloseRecordCollector Internal Method

```
void maybeAbortTransactionAndCloseRecordCollector(
    boolean isZombie)
```

`maybeAbortTransactionAndCloseRecordCollector` ...FIXME

Note	<code>maybeAbortTransactionAndCloseRecordCollector</code> is used exclusively when <code>suspend</code> (with <code>clean</code> flag disabled).
------	--

Closing Processor Topology — closeTopology Internal Method

```
void closeTopology()
```

`closeTopology` prints out the following TRACE message to the logs:

```
Closing processor topology
```

`closeTopology` requests the PartitionGroup to clear.

With the task initialized, `closeTopology` requests every ProcessorNode (in the ProcessorTopology) to close.

In case of `RuntimeException` while closing ProcessorNodes, `closeTopology` re-throws it.

Note	<code>closeTopology</code> is used exclusively when <code>streamTask</code> is requested to suspend .
------	---

Closing Task — `close` Method

```
void close(
    boolean clean,
    boolean isZombie)
```

Note	<code>close</code> is part of Task Contract to close the task.
------	--

`close` prints out the following DEBUG message to the logs:

```
Closing
```

`close` suspends the task followed by `closeSuspended`. In the end, `close` sets the `taskClosed` flag off (`false`).

Initializing Topology (of Processor Nodes) — `initializeTopology` Method

```
void initializeTopology()
```

Note	<code>initializeTopology</code> is part of Task Contract to initialize a topology of processor nodes .
------	--

`initializeTopology` initializes the `ProcessorNodes` in the `ProcessorTopology`.

With [exactly-once support enabled](#), `initializeTopology` requests the `Kafka Producer` to start a new transaction (using `Producer.beginTransaction`) and turns the `transactionInFlight` flag on.

`initializeTopology` then requests the `InternalProcessorContext` to initialize.

In the end, `initializeTopology` turns the `taskInitialized` flag on (`true`) and the `idleStartTime` to `UNKNOWN`.

Initializing ProcessorNodes (in ProcessorTopology) — `initTopology` Internal Method

```
void initTopology()
```

`initTopology` prints out the following TRACE message to the logs:

```
Initializing processor nodes of the topology
```

`initTopology` then walks over all the [processor nodes](#) in the [topology](#) and requests them to [initialize](#) (one by one). While doing this node initialization, `initTopology` requests the [InternalProcessorContext](#) to [set the current node](#) to the processor node that is currently initialized and, after initialization, [resets the current node](#) (to `null`).

Note

`initTopology` is used exclusively when `streamTask` is requested to [initialize the topology](#).

Processing Single Record — `process` Method

```
boolean process()
```

`process` processes a single record and returns `true` when processed successfully, and `false` when there were no records to process.

Internally, `process` requests the [PartitionGroup](#) for the [next stamped record](#) (record with timestamp) and the [RecordQueue](#) (with the [RecordInfo](#)).

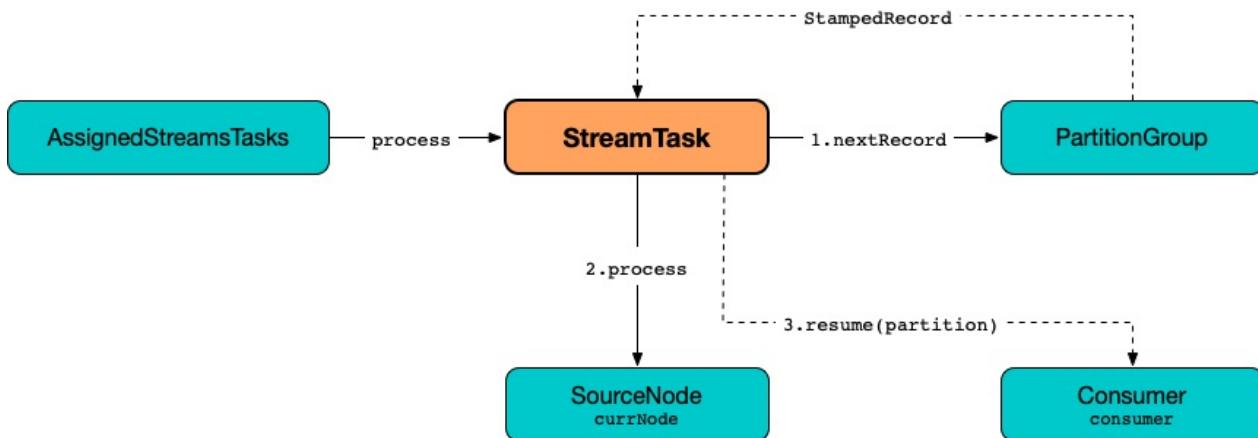


Figure 2. StreamTask and Processing Single Record

`process` prints out the following TRACE message to the logs:

```
Start processing one record [record]
```

`process` requests the [RecordInfo](#) for the [source processor node](#).

`process` updates the [ProcessorContext](#) to hold the current record and the source processor node.

`process` requests the source processor node to [process](#) the record.

`process` prints out the following TRACE message to the logs:

```
Completed processing one record [record]
```

`process` requests the [RecordInfo](#) for the [partition](#) and stores it and the record's [offset](#) in the [consumedOffsets](#) internal registry.

`process` turns the [commitNeeded](#) flag on (`true`).

(only if the size of the [queue](#) of the [RecordInfo](#) is exactly [buffered.records.per.partition](#) configuration property (default: `1000`)) `process` requests the [Kafka consumer](#) to resume the partition (and consume records from the partition again).

`process` resets the current node (and requests the [InternalProcessorContext](#) to [set the current node to be null](#)).

In case of a [ProducerFencedException](#), `process` throws a [TaskMigratedException](#).

In case of a [KafkaException](#), `process` throws a [StreamsException](#).

Note	<code>process</code> is used exclusively when AssignedStreamsTasks is requested to request the running stream tasks to process records (one record per task) .
------	--

closeSuspended Method

```
void closeSuspended(
    boolean clean,
    boolean isZombie,
    RuntimeException firstException)
```

Note	<code>closeSuspended</code> is part of Task Contract to...FIXME.
------	--

`closeSuspended` ...FIXME

Buffering New Records (From Partition) — addRecords Method

```
void addRecords(
    TopicPartition partition,
    Iterable<ConsumerRecord<byte[], byte[]>> records)
```

`addRecords` simply requests the [PartitionGroup](#) to add the new records to the [RecordQueue](#) for the specified partition.

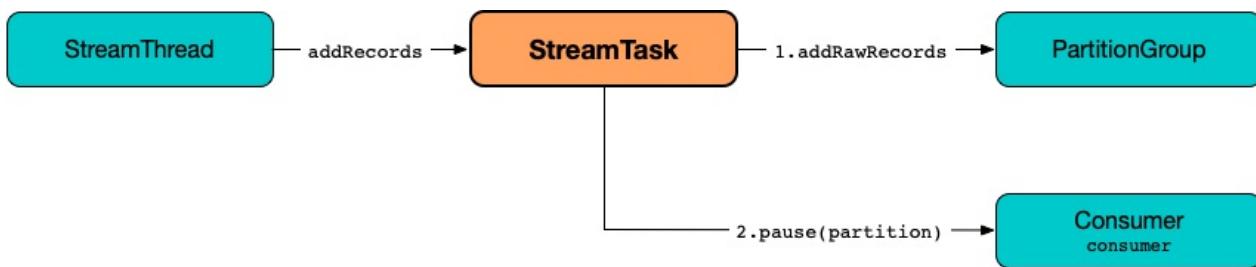


Figure 3. StreamTask and Buffering New Records

`addRecords` prints out the following TRACE message to the logs:

```
Added records into the buffered queue of partition [partition], new queue size is [new QueueSize]
```

When the size of the buffered record queue exceeds [buffered.records.per.partition](#) configuration property, `addRecords` requests the [Kafka Consumer](#) to pause the partition.

Note	<code>addRecords</code> uses Consumer.pause method to "pause the partition", i.e. to suspend fetching from the requested partitions. Future calls to KafkaConsumer.poll will not return any records from these partitions until they have been resumed using KafkaConsumer.resume .
------	---

Note	<code>addRecords</code> is used when: <ul style="list-style-type: none"> • <code>StreamThread</code> is requested to add records to active stream tasks (and report skipped records) • <code>TopologyTestDriver</code> is requested to pipeInput
------	--

Punctuating Processor (Executing Scheduled Periodic Action) — `punctuate` Method

```
void punctuate(
  ProcessorNode node,
  long timestamp,
  PunctuationType type,
  Punctuator punctuator)
```

Note	<code>punctuate</code> is part of ProcessorNodePunctuator Contract to punctuate a processor.
------	--

`punctuate` [updateProcessorContext](#) with a "dummy" stamped record and the given [ProcessorNode](#).

`punctuate` prints out the following TRACE message to the logs:

```
Punctuating processor [name] with timestamp [timestamp] and punctuation type [type]
```

In the end, `punctuate` requests the given [ProcessorNode](#) to `punctuate`.

In case of a `ProducerFencedException`, `punctuate` throws a `TaskMigratedException`.

In case of a `KafkaException`, `punctuate` throws a `StreamsException`:

```
[logPrefix]Exception caught while punctuating processor '[name]'
```

Attempting to Punctuate by Stream Time

— `maybePunctuateStreamTime` Method

```
boolean maybePunctuateStreamTime()
```

`maybePunctuateStreamTime` requests the [PartitionGroup](#) for the minimum partition timestamp across all partitions.

`maybePunctuateStreamTime` requests the stream-time PunctuationQueue to `mayPunctuate` with the minimum timestamp.

In the end, `maybePunctuateStreamTime` returns whatever the stream-time PunctuationQueue returned.

If the minimum timestamp is [UNKNOWN](#), `maybePunctuateStreamTime` returns `false`.

Note	<code>maybePunctuateStreamTime</code> is used exclusively when <code>AssignedStreamsTasks</code> is requested to punctuate running stream tasks .
------	---

Attempting to Punctuate by System Time

— `maybePunctuateSystemTime` Method

```
boolean maybePunctuateSystemTime()
```

```
maybePunctuateSystemTime ...FIXME
```

Note	<code>maybePunctuateSystemTime</code> is used exclusively when <code>AssignedStreamsTasks</code> is requested to punctuate running stream tasks .
------	---

Scheduling Cancellable Periodic Action (Punctuator)

— schedule Method

```
// PUBLIC API
Cancellable schedule(
    long interval,
    PunctuationType type,
    Punctuator punctuator)
// PACKAGE PROTECTED
Cancellable schedule(
    long startTime,
    long interval,
    PunctuationType type,
    Punctuator punctuator)
```

`schedule` chooses the `PunctuationQueue` and the `startTime` per the specified `PunctuationType` that can either be `STREAM_TIME` or `WALL_CLOCK_TIME`.

For `STREAM_TIME`, `schedule` always uses `0L` as the `startTime` and the `stream-time PunctuationQueue`.

For `WALL_CLOCK_TIME`, `schedule` uses the current time and the specified `interval` as the `startTime` and the `system-time PunctuationQueue`.

`schedule` then creates a new `PunctuationSchedule` (with the current processor of the `InternalProcessorContext`) and requests the appropriate `PunctuationQueue` to `schedule` it.

Note	<code>schedule</code> is used exclusively when <code>ProcessorContextImpl</code> is requested to <code>schedule a cancellable periodic action</code> .
------	--

Initializing State Stores — `initializeStateStores` Method

```
boolean initializeStateStores()
```

Note	<code>initializeStateStores</code> is part of <code>Task Contract</code> to initialize state stores.
------	--

`initializeStateStores` prints out the following TRACE message to the logs:

```
Initializing state stores
```

`initializeStateStores` `registerStateStores`.

In the end, `initializeStateStores` returns `true` if the task has any changelog partitions. Otherwise, `initializeStateStores` returns `false`.

Committing Task — `commit` Method

```
void commit() (1)
void commit(
    boolean startNewTransaction)
```

1. Uses `startNewTransaction` flag enabled (`true`)

Note	<code>commit</code> is part of Task Contract to commit the task.
------	--

`commit` simply [commits](#) with the `startNewTransaction` flag enabled (`true`).

`commit` Internal Method

```
void commit(
    boolean startNewTransaction)
```

`commit` prints out the following DEBUG message to the logs:

```
Committing
```

`commit flushState`.

(only when [exactly-once support](#) is disabled) `commit` requests the [ProcessorStateManager](#) to [checkpoint](#) with the [checkpointable offsets](#).

For every partition and offset (in the [consumedOffsets](#) internal registry), `commit` requests the [ProcessorStateManager](#) to [putOffsetLimit](#) with the partition and the offset incremented.

(only when [exactly-once support](#) is disabled), `commit` requests the [Consumer](#) to synchronously commit (`Consumer.commitSync`) the partitions and offsets from the [consumedOffsets](#) internal registry).

Note	<code>Consumer.commitSync</code> commits the specified offsets for a given list of partitions. This commits offsets to Kafka. The offsets committed using this API will be used on the first fetch after every rebalance and also on startup. The committed offset should be the next message your application will consume (and that's why the offsets are incremented).
------	---

With [exactly-once support](#) enabled, `commit` requests the [Producer](#) to `sendOffsetsToTransaction` for the partitions and offsets in the `consumedOffsets` internal registry and the `applicationId`. `commit` requests the [Producer](#) to `commitTransaction` and sets the `transactionInFlight` off (`false`). With the given `transactionInFlight` enabled (`true`), `commit` requests the [Producer](#) to `beginTransaction` and sets the `transactionInFlight` on (`true`).

In the end, `commit` sets the `commitNeeded` and `commitRequested` flags off (`false`), and requests the [TaskMetrics](#) for the `taskCommitTimeSensor` to record the duration (i.e. the time since `commit` was executed).

Note

`commit` is used when `streamTask` is requested to `commit` (with the `startNewTransaction` flag enabled) and `suspend` (with the `startNewTransaction` flag disabled).

activeTaskCheckpointableOffsets Method

```
Map<TopicPartition, Long> activeTaskCheckpointableOffsets()
```

Note

`activeTaskCheckpointableOffsets` is part of the [AbstractTask Contract](#) to return the checkpointable offsets.

`activeTaskCheckpointableOffsets` ...FIXME

Flushing State Stores And Producer (RecordCollector) — flushState Method

```
void flushState()
```

Note

`flushState` is part of [AbstractTask Contract](#) to flush all `state stores` registered with the task.

`flushState` prints out the following TRACE message to the logs:

```
Flushing state and producer
```

`flushState` [flushes state stores](#).

`flushState` requests the [RecordCollector](#) to flush (the internal Kafka producer).

isProcessable Method

```
boolean isProcessable(
    long now)
```

`isProcessable` returns `true` when one of the following is met:

- All `RecordQueues` have at least one record buffered of the `PartitionGroup`
- The task is enforced to be processable, i.e. the time between `now` and the `idleStartTime` is at least or larger than the `max.task.idle.ms` configuration property (default: `0L`)

Otherwise, `isProcessable` returns `false`.

Note	(FIXME) <code>isProcessable</code> does some minor accounting.
------	--

Note	<code>isProcessable</code> is used exclusively when <code>AssignedStreamsTasks</code> is requested to request the running stream tasks to process records (one record per task).
------	--

Resuming Task — `resume` Method

```
void resume()
```

Note	<code>resume</code> is part of the Task Contract to resume the task.
------	--

`resume` prints out the following DEBUG message to the logs:

```
Resuming
```

`resume` then does further processing only when [Exactly-Once Support](#) is enabled.

`resume` ...FIXME

`numBuffered` Method

```
int numBuffered()
```

`numBuffered` simply requests the `PartitionGroup` for the `numBuffered`.

Note	<code>numBuffered</code> seems to be used for tests only.
------	---

Requesting Commit — `requestCommit` Method

```
void requestCommit()
```

`requestCommit` simply turns the `commitRequested` internal flag on (`true`).

Note

`requestCommit` is used exclusively when `ProcessorContextImpl` is requested to `commit`.

Purgable Offsets of Repartition Topics (of Topology) — `purgableOffsets` Method

```
Map<TopicPartition, Long> purgableOffsets()
```

In essence, `purgableOffsets` returns the partition-offset pairs for the `consumedOffsets` of the repartition topics (i.e. the `ProcessorTopology` uses as `repartition topics`).

`purgableOffsets` ...FIXME

Note

`purgableOffsets` is used exclusively when `AssignedStreamsTasks` is requested for the `purgable offsets of the repartition topics (of a topology)`.

initializeTransactions Internal Method

```
void initializeTransactions()
```

`initializeTransactions` simply requests the `Producer` to `initTransactions`.

In case of `TimeoutException`, `initializeTransactions` prints out the following ERROR message to the logs:

```
Timeout exception caught when initializing transactions for task [id]. This might happen if the broker is slow to respond, if the network connection to the broker was interrupted, or if similar circumstances arise. You can increase producer parameter `max.block.ms` to increase this timeout.
```

In the end, `initializeTransactions` throws a `StreamsException`.

```
[logPrefix]Failed to initialize task [id] due to timeout.
```

Note

`initializeTransactions` is used when `streamTask` is `created` and requested to `resume` (both with `exactly-once support enabled`).

Updating InternalProcessorContext

— updateProcessorContext Internal Method

```
void updateProcessorContext(
    StampedRecord record,
    ProcessorNode currNode)
```

`updateProcessorContext` requests the `InternalProcessorContext` to set the current `ProcessorRecordContext` to a new `ProcessorRecordContext` (per the input `StampedRecord`).

`updateProcessorContext` then requests the `InternalProcessorContext` to set the current `ProcessorNode` to the input `ProcessorNode`.

Note

`updateProcessorContext` is used when `streamTask` is requested to process a single record and execute a scheduled periodic action (aka punctuate).

Internal Properties

Name	Description
<code>commitRequested</code>	<p>Flag that indicates whether a <code>commit</code> was requested (<code>true</code>)</p> <p>Default: <code>false</code></p> <p>Disabled (<code>false</code>) after <code>commit</code></p> <p>Used exclusively when <code>AssignedStreamsTasks</code> is requested to <code>maybeCommitPerUserRequested</code>.</p>
<code>consumedOffsets</code>	<p>Consumer offsets by <code>TopicPartitions</code> (<code>Map<TopicPartition, Long></code>) that <code>streamTask</code> has processed successfully</p> <p>Used when requested for the purgable offsets of the repartition topics of a topology, to <code>activeTaskCheckpointableOffsets</code>, and to <code>commit</code></p>
<code>idleStartTime</code>	
<code>producer</code>	<p>Kafka <code>Producer</code> (<code>Producer<byte[], byte[]></code>)</p> <p>Created when <code>streamTask</code> is <code>created</code> and <code>resumed</code> by requesting the <code>ProducerSupplier</code> to supply a Producer</p> <p>Cleared (<i>nullified</i>) when <code>streamTask</code> is requested to <code>suspend</code> and <code>maybeAbortTransactionAndCloseRecordCollector</code></p> <p>Used for the following:</p>

	<ul style="list-style-type: none"> Requesting the RecordCollector to initialize (when StreamTask is created and resumed) initializeTopology, initializeTransactions, maybeAbortTransactionAndCloseRecordCollector, and commit for exactly-once support producerMetrics
recordInfo	<p>RecordInfo (that holds a RecordQueue with the source processor node and the partition the currently-processed stamped record came from)</p> <p>Created empty alongside the StreamTask and "fill up" with the RecordQueue when requested to process a single record</p>
streamTimePunctuationQueue	PunctuationQueue
systemTimePunctuationQueue	PunctuationQueue
taskMetrics	<p>TaskMetrics for the TaskId and the StreamsMetricsImpl</p> <p>Used when StreamTask is requested for the following:</p> <ul style="list-style-type: none"> isProcessable (to record an occurrence of taskEnforcedProcessSensor sensor) commit (to record a value of taskCommitTimeSensor sensor) closeSuspended (to remove all task sensors)
transactionInFlight	<p>Flag that controls whether the producer should abort transaction (with exactly-once support enabled)</p> <p>Default: false</p> <ul style="list-style-type: none"> Enabled (<code>true</code>) in initializeTopology and commit Disabled (<code>false</code>) in maybeAbortTransactionAndCloseRecordCollector and commit

ProcessorContextImpl — ProcessorContext for StreamTask

`ProcessorContextImpl` is a concrete `ProcessorContext` (as a `AbstractProcessorContext`) that is `created` exclusively for a `StreamTask`.

`ProcessorContextImpl` is a concrete `RecordCollector.Supplier` that supplies the given `RecordCollector` when `requested`.

Accessing State Store by Name — `getStateStore` Method

```
StateStore getStateStore(String name)
```

Note	<code>getStateStore</code> is part of <code>ProcessorContext Contract</code> to access the <code>state store</code> by name.
------	--

`getStateStore` ...FIXME

Scheduling Cancellable Periodic Action (Punctuator) — `schedule` Method

```
Cancellable schedule(  
    Duration interval,  
    PunctuationType type,  
    Punctuator callback)
```

Note	<code>schedule</code> is part of <code>ProcessorContext Contract</code> to schedule a <code>Punctuator</code> (a periodic action) that can be <code>cancelled</code> .
------	--

`schedule` simply requests the `StreamTask` to `schedule` the specified `Punctuator`.

Creating ProcessorContextImpl Instance

`ProcessorContextImpl` takes the following to be created (most for `AbstractProcessorContext`):

- `TaskId`
- `StreamTask`

- StreamsConfig
- RecordCollector
- ProcessorStateManager
- StreamsMetricsImpl
- ThreadCache

`ProcessorContextImpl` initializes the internal properties.

forward Method

```
void forward(final K key, final V value)
void forward(final K key, final V value, final int childIndex)
void forward(final K key, final V value, final String childName)
void forward(final K key, final V value, final To to)
void forward(
    final ProcessorNode child,
    final K key,
    final V value) (1)
```

1. Private API

Note	<code>forward</code> is part of the ProcessorContext Contract to...FIXME.
------	---

`forward` ...FIXME

Committing StreamTask — commit Method

```
void commit()
```

Note	<code>commit</code> is part of the ProcessorContext Contract to request a commit.
------	---

`commit` simply requests the StreamTask to commit.

ProducerSupplier

ProducerSupplier is...FIXME

AssignedTasks

`AssignedTasks` is the [abstraction](#) of [collections of tasks](#) to manage status of multiple tasks as a whole.

`AssignedTasks` uses the following [internal registries](#) to determine the status of a task:

- [running](#) for tasks that are considered **running** (that `AssignedStreamsTasks` uses when requested to [process](#))
- [created](#) for tasks that are considered **new**
- [suspended](#) for tasks that are considered **suspended**
- [restoring](#) for tasks that are considered **restoring**

`AssignedTasks` gives the [toString](#) method to list the tasks by their status.

Note	<code>AssignedTasks</code> is a Java abstract class and cannot be created directly. It is created indirectly when the concrete AssignedTasks are.
------	---

Table 1. AssignedTasks

Task	Description
<code>AssignedStandbyTasks</code>	Manages StandbyTasks
<code>AssignedStreamsTasks</code>	Manages StreamTasks

close Method

```
void close(final boolean clean)
```

`close` ...FIXME

Note	<code>close</code> is used when...FIXME
------	---

Closing Zombie Task — `closeZombieTask` Method

```
RuntimeException closeZombieTask(final T task)
```

`closeZombieTask` ...FIXME

Note

`closeZombieTask` is used when...FIXME

Removing All Task References — `clear` Method

`void clear()`

`clear` simply removes all entries from the internal registries: `runningByPartition`, `restoringByPartition`, `running`, `created`, `suspended` and `restoredPartitions`.

Note

`clear` is used exclusively when `AssignedTasks` is requested to `close`.

Suspending Tasks — `suspendTasks` Internal Method

`RuntimeException suspendTasks(final Collection<T> tasks)``suspendTasks` ...FIXME

Note

`suspendTasks` is used exclusively when `AssignedTasks` is requested to `suspend all active tasks`.

Checking If There Is At Least One Running Task — `hasRunningTasks` Method

`boolean hasRunningTasks()`

`hasRunningTasks` simply checks whether there is at least one task registered in `running` internal registry or not.

Note

`hasRunningTasks` is used exclusively when `TaskManager` is requested to check if there are any `running` or `standby` tasks registered.

Marking Task As Ready For Execution and Processing Records — `transitionToRunning` Internal Method

`void transitionToRunning(final T task)`

`transitionToRunning` simply records the given task as available for processing records (*transitions the task to the running state*).

Internally, `transitionToRunning` prints out the following DEBUG message to the logs:

```
Transiting [taskTypeName] [taskId] to running
```

`transitionToRunning` marks the task as running (by adding it to the `running` internal registry).

Note

Registering a task in the `running` internal registry is the only way to mark the task as running and ready for [processing records](#).

`transitionToRunning` requests the task to [initialize the topology of processor nodes](#).

`transitionToRunning` registers the `partitions` and the `changelog partitions` of the task (in the `runningByPartition` internal registry).

Note

`transitionToRunning` is used when:

- `AssignedStreamsTasks` is requested to [updateRestored](#)
- `AssignedTasks` is requested to [initialize new tasks](#) and attempt to resume a [suspended task](#)

Adding New Task — `addNewTask` Method

```
void addNewTask(final T task)
```

`addNewTask` just adds the input `task` in `created` internal registry.

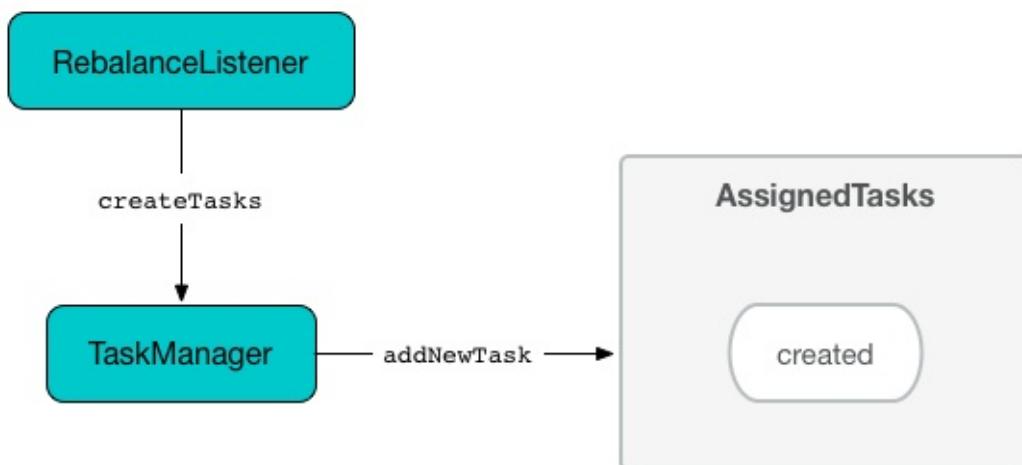


Figure 1. AssignedTasks and Adding New Task

Note

`addNewTask` is used exclusively when `TaskManager` is requested to [create processor tasks for assigned topic partitions](#) (that in turn triggers `addStandbyTasks` and `addStreamTasks`).

Initializing New Tasks — `initializeNewTasks` Method

```
void initializeNewTasks()
```

`initializeNewTasks` initializes new tasks (that are in the `created` state), i.e. moves tasks from the `created` internal registry to either `restoring` or `running` registries per whether a task has state stores that may need restoring or not, respectively.

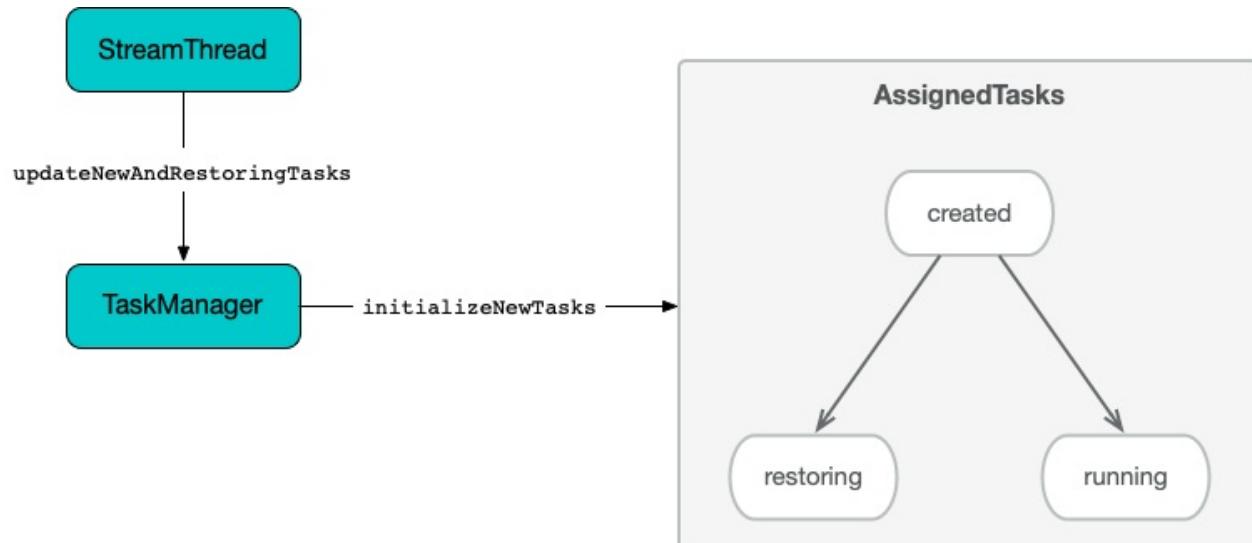


Figure 2. `AssignedTasks.initializeNewTasks`

Note

`initializeNewTasks` does nothing unless there is at least one task in the `created` internal registry.

`initializeNewTasks` prints out the following DEBUG message to the logs:

```
Initializing [taskTypeName]s [created]
```

`initializeNewTasks` walks over all tasks in the `created` internal registry.

`initializeNewTasks` requests a task to `initialize state stores (of the topology)`.

If the above yielded `true`, `initializeNewTasks` marks the task as ready for execution and processing records.

Otherwise, when the above `state store initialization` yielded `false`, `initializeNewTasks` prints out the following DEBUG message to the logs and the concrete `AssignedStreamTasks` is requested to register the StreamTask as restoring.

```
Transitioning [taskTypeName] [taskId] to restoring
```

In the end, `initializeNewTasks` removes the task (after being processed successfully) from the `created` internal registry.

In case of `LockException`, `initializeNewTasks` prints out the following TRACE message to the logs:

```
Could not create [taskTypeName] [taskId] due to [message]; will retry
```

Note

`initializeNewTasks` is used exclusively when `TaskManager` is requested to `updateNewAndRestoringTasks`.

Closing Non-Assigned Suspended Tasks — `closeNonAssignedSuspendedTasks` Method

```
void closeNonAssignedSuspendedTasks(final Map<TaskId, Set<TopicPartition>> newAssignment)
```

`closeNonAssignedSuspendedTasks` closes non-assigned tasks that were suspended, but are no longer assigned to the Kafka Streams instance or the partitions of the task and the assignment do not match.

Internally, `closeNonAssignedSuspendedTasks` takes the `suspended` tasks and for every task checks if either condition holds:

1. `newAssignment` does not contain the id of the suspended task
2. The `partitions` of the suspended task are not equal the partitions in `newAssignment` for the task id

If either condition holds, `closeNonAssignedSuspendedTasks` prints out the following DEBUG message to the logs, requests the task to `closeSuspended` (with the `clean` flag enabled) and in the end removes the task from `suspended` tasks.

```
Closing suspended and not re-assigned [taskType] [id]
```

In case of a `Exception`, `closeNonAssignedSuspendedTasks` prints out the following ERROR message to the logs followed by the exception message.

```
Failed to remove suspended [taskType] [id] due to the following error
```

Note

`closeNonAssignedSuspendedTasks` is used exclusively when `TaskManager` is requested to `create processor tasks for assigned topic partitions`.

Attempting to Resume Suspended Task

— `maybeResumeSuspendedTask` Method

```
boolean maybeResumeSuspendedTask(
    final TaskId taskId,
    final Set<TopicPartition> partitions)
```

`maybeResumeSuspendedTask` returns `true` after successful transitioning a task (by its `taskId`) from suspended to running state when the partitions of the suspended task and the input ones are equal. Otherwise, `maybeResumeSuspendedTask` reports an exception (`RuntimeException` or `TaskMigratedException`) or returns `false`.

Internally, `maybeResumeSuspendedTask` branches off per whether the task (for the given `TaskId`) is **suspended** or not.

If not, `maybeResumeSuspendedTask` returns `false`.

If the task is **suspended**, `maybeResumeSuspendedTask` prints out the following TRACE message to the logs:

```
found suspended [taskTypeName] [taskId]
```

`maybeResumeSuspendedTask` checks whether the **partitions** of the task are exactly the input `partitions`.

If the partitions do not match, `maybeResumeSuspendedTask` prints out the following WARN message to the logs:

```
couldn't resume task [taskId] assigned partitions [partitions], task partitions [partitions]
```

If however the partitions are equal, `maybeResumeSuspendedTask` removes the task (by the input `taskId`) from **suspended** registry and requests the task to **resume**.

`maybeResumeSuspendedTask` schedules the task for execution and prints out the following TRACE message to the logs:

```
resuming suspended [taskTypeName] [taskId]
```

`maybeResumeSuspendedTask` returns `true`.

In case of `TaskMigratedException`, `maybeResumeSuspendedTask` [closeZombieTask](#). If it gives a `RuntimeException`, `maybeResumeSuspendedTask` re-throws it. Otherwise, `maybeResumeSuspendedTask` removes the task (by the input `taskId`) from [suspended](#) registry (*again?!*) and re-throws the initial `TaskMigratedException`.

Note

`maybeResumeSuspendedTask` is used when `TaskManager` is requested to [create processor tasks for assigned topic partitions](#) (and register new [standby](#) and [stream](#) tasks).

Describing Itself (Textual Representation) — `toString` Method

```
String toString(String indent)
```

`toString` gives a text representation and [describes](#) the following:

- [running](#) tasks with "Running:" header
- [suspended](#) tasks with "Suspended:" header
- [restoring](#) tasks with "Restoring:" header
- [created](#) tasks with "New:" header

`FIXME toString in action`

Note

`toString` is used exclusively when `TaskManager` is requested to [describe itself](#).

`describe` Internal Method

```
void describe(
    StringBuilder builder,
    Collection<T> tasks,
    String indent,
    String name)
```

`describe` simply requests every task in the input `tasks` to [describe itself](#) and uses the `indent` and `name` to create a text representation.

`FIXME toString in action`

Note

`describe` is used exclusively when `AssignedTasks` is requested for a [text representation](#).

Getting Partitions of New Tasks with State Store — `uninitializedPartitions` Method

```
Set<TopicPartition> uninitializedPartitions()
```

`uninitializedPartitions` gives the [partitions](#) of the new tasks (from [created](#) registry) that [have state store](#).

Note

`uninitializedPartitions` gives an empty set of partitions if [created](#) is empty, i.e. has no tasks registered.

Note

`uninitializedPartitions` is used exclusively when `TaskManager` is requested to [create processor tasks for assigned topic partitions](#).

Suspending All Active Tasks — `suspend` Method

```
RuntimeException suspend()
```

`suspend` prints out the following TRACE message to the logs and [suspendTasks](#) (from [running](#)).

```
Suspending running [taskTypeName] [runningTaskIds]
```

`suspend` prints out the following TRACE message to the logs and [closeNonRunningTasks](#) (from [restoring](#)).

```
Close restoring [taskTypeName] [restoring]
```

`suspend` prints out the following TRACE message to the logs and [closeNonRunningTasks](#) (from [created](#)).

```
Close created [taskTypeName] [created]
```

`suspend` removes all task ids from [previousActiveTasks](#) and adds the task ids from [running](#).

In the end, `suspend` removes all entries from `running`, `restoring`, `created`, `runningByPartition` and `restoringByPartition`.

Note

`suspend` is used exclusively when `TaskManager` is requested to `suspend all active and standby stream tasks and state`.

closeNonRunningTasks Internal Method

```
RuntimeException closeNonRunningTasks(final Collection<T> tasks)
```

`closeNonRunningTasks` `closes` every task in the given `tasks` one by one (with `clean` and `isZombie` flags off).

In case of a `RuntimeException`, `closeNonRunningTasks` prints out the following ERROR to the logs followed by the exception.

```
Failed to close [taskTypeName], [id]"
```

Note

`closeNonRunningTasks` is used exclusively when `AssignedTasks` is requested to `suspend all active tasks` (and the input tasks are `restoring` and `created`).

Executing Task Action with Every Running Task — applyToRunningTasks Method

```
void applyToRunningTasks(final TaskAction<T> action)
```

`applyToRunningTasks` applies the input `action` to every `running` task.

`applyToRunningTasks` throws the first `RuntimeException` if thrown.

Note

`applyToRunningTasks` is used when:

- `AssignedStreamsTasks` is requested to `maybeCommit`
- `AssignedTasks` is requested to `commit`

applyToRunningTasks and TaskMigratedException

In case of a `TaskMigratedException`, `applyToRunningTasks` prints out the following INFO message to the logs:

```
Failed to commit [taskTypeName] [taskId] since it got migrated to another thread already. Closing it as zombie before triggering a new rebalance.
```

`applyToRunningTasks` [closeZombieTask](#). If it gives a `RuntimeException`, `applyToRunningTasks` re-throws it. Otherwise, `applyToRunningTasks` removes the task (from the iterator but what about [running](#)?) and re-throws the initial `TaskMigratedException`.

applyToRunningTasks and RuntimeException

In case of a `RuntimeException`, `applyToRunningTasks` prints out the following ERROR message to the logs followed by the exception.

```
Failed to [actionName] [taskTypeName] [taskId] due to the following error:
```

`applyToRunningTasks` records the `RuntimeException` for a later re-throwing.

Creating AssignedTasks Instance

`AssignedTasks` takes the following when created:

- `LogContext`
- `taskTypeName`

`AssignedTasks` initializes the [internal properties](#).

Registering Task for (State Store) Restoring — `addToRestoring` Internal Method

```
void addToRestoring(final T task)
```

`addToRestoring` records the input `task` in the `restoring` internal registry.

`addToRestoring` records the task's `partitions` and `changelogPartitions` in the `restoringByPartition` internal registry.

Note

`addToRestoring` is used exclusively when `AssignedTasks` is requested to [initialize new tasks](#) (and the task is a `StreamTask` and [has state stores that need restoring](#)).

Committing Running Tasks — `commit` Method

```
int commit()
```

`commit` ...FIXME

Note	<code>commit</code> is used exclusively when <code>TaskManager</code> is requested to <code>commitAll</code> .
------	--

Checking Whether All Tasks Are Running — `allTasksRunning` Method

```
boolean allTasksRunning()
```

`allTasksRunning` is positive (`true`) when there are no `created` and `suspended` tasks. Otherwise, `allTasksRunning` is negative (`false`).

Note	<code>allTasksRunning</code> is used exclusively when <code>TaskManager</code> is requested to <code>updateNewAndRestoringTasks</code> .
------	--

`runningTaskFor` Method

```
T runningTaskFor(TopicPartition partition)
```

`runningTaskFor` simply looks up the task (`T`) for the given `partition` (in the `runningByPartition` internal registry).

Note	<code>runningTaskFor</code> is used when <code>TaskManager</code> is requested to look up the <code>StreamTask</code> and <code>StandbyTask</code> for a given partition.
------	---

`allTasks` Method

```
List<T> allTasks()
```

`allTasks` ...FIXME

Note	<code>allTasks</code> is used when...FIXME
------	--

`allAssignedTaskIds` Method

```
Set<TaskId> allAssignedTaskIds()
```

`allAssignedTaskIds ...FIXME`

Note	<code>allAssignedTaskIds</code> is used when...FIXME
------	--

Internal Properties

Name	Description
<code>commitAction</code>	<code>TaskAction</code> that requests running tasks to commit at commit
<code>created</code>	New tasks by their ids <ul style="list-style-type: none"> • Tasks are added when <code>AssignedTasks</code> is requested to <code>addNewTask</code> • Tasks are removed when <code>AssignedTasks</code> is requested to <code>initializeNewTasks</code>, <code>suspend</code> or <code>clear</code>
<code>previousActiveTasks</code>	Previously active tasks
<code>restoring</code>	
<code>log</code>	
<code>running</code>	Running tasks by TaskId (<code>Map<TaskId, T></code>) <code>running</code> is a <code>java.util.concurrent.ConcurrentHashMap</code> , i.e. <code>ConcurrentHashMap<TaskId, Task></code> . Used when...FIXME Tasks IDs are added or removed as follows: <ul style="list-style-type: none"> • Added when <code>AssignedTasks</code> is requested to <code>transition a task to a running state</code> • Removed when <code>AssignedTasks</code> is requested to <code>suspend</code> or <code>clear</code>
<code>runningByPartition</code>	Kafka partitions per task (that processes records)
<code>suspended</code>	

AssignedStandbyTasks — AssignedTasks For StandbyTasks

`AssignedStandbyTasks` is a concrete `AssignedTasks` for `StandbyTasks` that is created exclusively for the `TaskManager` for a `StreamThread` (when `KafkaStreams` is created).

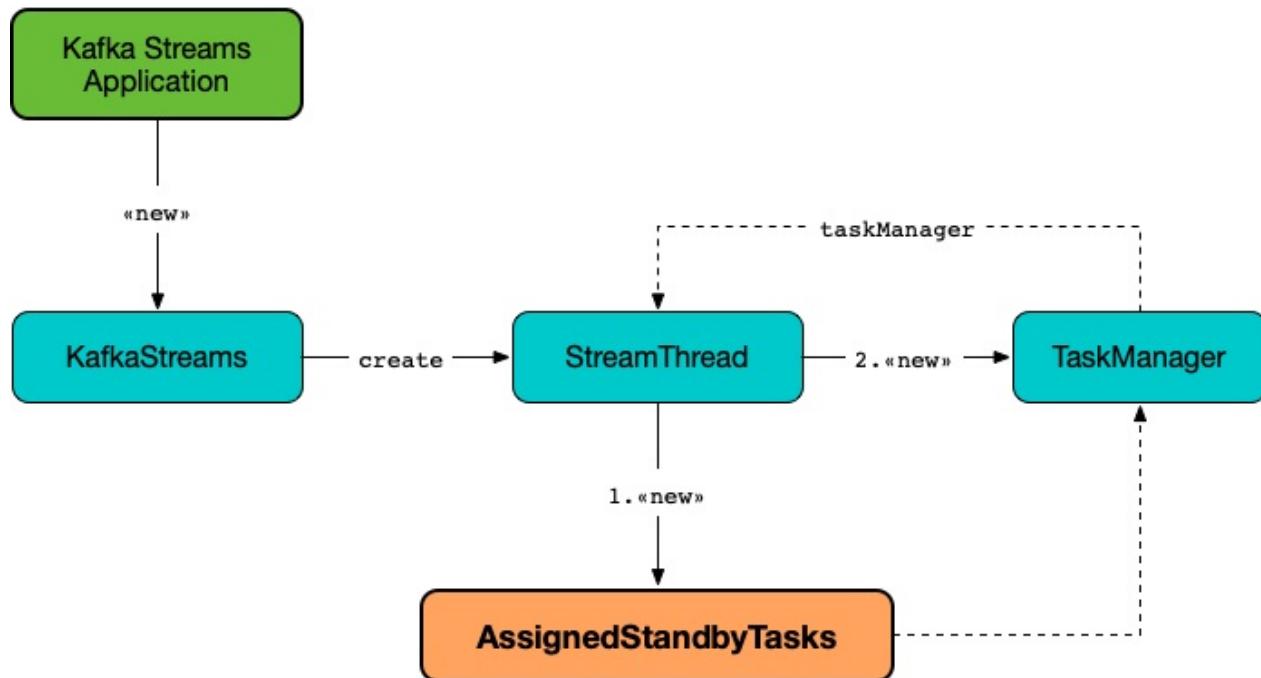


Figure 1. `AssignedStandbyTasks`, `TaskManager` and `StreamThread`

`AssignedStandbyTasks` takes a `LogContext` when created.

`AssignedStandbyTasks` uses **standby task** for `taskTypeName`.

Tip	<p>Enable <code>ALL</code> logging level for <code>org.apache.kafka.streams.processor.internals.AssignedStandbyTasks</code> logger to see what happens inside.</p> <p>Add the following line to <code>log4j.properties</code> :</p> <pre>log4j.logger.org.apache.kafka.streams.processor.internals.AssignedStandbyTasks=A</pre> <p>Refer to Application Logging Using log4j.</p>
------------	--

AssignedStreamsTasks — AssignedTasks For StreamTasks

`AssignedStreamsTasks` is a concrete `AssignedTasks` of `StreamTasks` that...FIXME

`AssignedStreamsTasks` is `created` for a `StreamThread` (when `KafkaStreams` is `created`).

`AssignedStreamsTasks` is a `RestoringTasks` that...FIXME

It *appears* that `AssignedStreamsTasks` simply operates on the running tasks (i.e. the tasks that are in `running` internal registry). When requested to `process` or `punctuate` `AssignedStreamsTasks` simply walks over the `running` internal registry and triggers execution of every task.

`AssignedStreamsTasks` uses the `maybeCommit` task action (`TaskAction<StreamTask>`) that is used in `maybeCommit`. The task action takes a `stream task` and checks if the task `needs a commit`. If so, the action does the following:

1. Increments the `committed` internal counter
2. Requests the stream task to `commit`
3. Prints out the following DEBUG message to the logs:

```
Committed active task [id] per user request in
```

`AssignedStreamsTasks` takes a `LogContext` when created.

`AssignedStreamsTasks` uses **stream task** for `taskTypeName`.

Tip	<p>Enable <code>ALL</code> logging level for <code>org.apache.kafka.streams.processor.internals.AssignedStreamsTasks</code> logger to see what happens inside.</p> <p>Add the following line to <code>log4j.properties</code> :</p> <pre>log4j.logger.org.apache.kafka.streams.processor.internals.AssignedStreamsTasks=A</pre> <p>Refer to Application Logging Using log4j.</p>
-----	--

Processing Records Using Running Stream Tasks (One Record per Task) — `process` Method

```
int process()
```

`process` requests every `running stream task` (that `is processable`) to process a single `record`.

In the end, `process` returns how many `stream tasks` processed a single record.

Note

`process` is used exclusively when `TaskManager` is requested to `process records by the running stream tasks (one record per task)`.

process and TaskMigratedException

In case of a `TaskMigratedException`, `process` prints out the following INFO message to the logs:

```
Failed to process stream task [id] since it got migrated to another thread already. Closing it as zombie before triggering a new rebalance.
```

`process` then `closes the task` (considering the task a zombie). If this reports a `RuntimeException`, `process` re-throws it.

`process` removes the task from `running` and throws the `TaskMigratedException`.

process and RuntimeException

In case of a `RuntimeException`, `process` prints out the following ERROR message to the logs followed by the exception.

```
Failed to process stream task [id] due to the following error:
```

`process` re-throws the `RuntimeException`.

Committing Running Stream Tasks that Requested It — maybeCommit Method

```
int maybeCommit()
```

`maybeCommit` resets the `committed` internal counter (to `0`) and `executes` the `maybeCommitAction` task action to every `running task` (that modifies `committed`).

In the end, `maybeCommit` gives the number of running stream tasks that `needed a commit`.

Note

`maybeCommit` is used exclusively when `TaskManager` is requested to `maybeCommitActiveTasks`.

Punctuating Running Stream Tasks (by Stream and System Time) — `punctuate` Method

```
int punctuate()
```

`punctuate` walks over the `running stream tasks` and requests them to attempt to punctuate by `stream` and `system` time.

For every successfully executed punctuation, `punctuate` increments an internal `punctuated` counter.

In the end, `punctuate` returns the internal `punctuated` counter.

In case of a `TaskMigratedException`, `punctuate` prints out the following INFO message to the logs, `closes the zombie task`, and possibly removes the task from the `running stream tasks`.

```
Failed to punctuate stream task [taskId] since it got migrated to another thread already. Closing it as zombie before triggering a new rebalance.
```

In case of a `KafkaException`, `punctuate` prints out the following ERROR message to the logs and re-throws the exception.

```
Failed to punctuate stream task [taskId] due to the following error:
```

Note

`punctuate` is used exclusively when `TaskManager` is requested to `punctuate stream tasks`.

updateRestored Method

```
void updateRestored(Collection<TopicPartition> restored)
```

`updateRestored` prints out the following TRACE message to the logs:

```
Stream task changelog partitions that have completed restoring so far: [restored]
```

`updateRestored` adds all of the `restored` partitions to the `restoredPartitions` internal registry unless they are already present.

For every pair (of `TaskId` and `StreamTask`) in the `restoring` internal registry, `updateRestored` requests the `changelog partitions` of the `StreamTask` and checks whether they are all in the `restoredPartitions` internal registry or not.

If the `changelog partitions` are all in the `restoredPartitions` registry, `updateRestored` `transitionToRunning` the task, removes the task from the `restoring` registry and then prints out the following TRACE message to the logs:

```
Stream task [id] completed restoration as all its changelog partitions [changelogPartitions] have been applied to restore state
```

If the `changelog partitions` are not all in the `restoredPartitions` registry, `updateRestored` simply prints out the following TRACE message to the logs:

```
Stream task [id] cannot resume processing yet since some of its changelog partitions have not completed restoring: [outstandingPartitions]
```

In the end, `updateRestored` removes all of the elements from the `restoredPartitions` internal registry when `allTasksRunning`.

Note

`updateRestored` is used exclusively when `TaskManager` is requested to `updateNewAndRestoringTasks`.

addToRestoring Method

```
void addToRestoring(StreamTask task)
```

`addToRestoring` adds the given `StreamTask` to the `restoring` internal registry.

`addToRestoring` adds the `partitions` of the given `StreamTask` to the `restoringByPartition` internal registry.

`addToRestoring` adds the `changelog partitions` of the given `StreamTask` to the `restoringByPartition` internal registry.

Note

`addToRestoring` is used exclusively when `AssignedTasks` is requested to `initialize new tasks` (when `TaskManager` is requested to `updateNewAndRestoringTasks`).

Checking Whether All StreamTasks Are Running

— `allTasksRunning` Method

```
boolean allTasksRunning()
```

Note

`allTasksRunning` is part of the [AssignedTasks Contract](#) to check whether all tasks are running or not.

`allTasksRunning` is positive (`true`) when [all StreamTasks are running](#) of the parent `AssignedTasks` and there are no [restoring](#) tasks. Otherwise, `allTasksRunning` is negative (`false`).

maybeCommitPerUserRequested Method

```
int maybeCommitPerUserRequested()
```

`maybeCommitPerUserRequested` returns how many [running stream tasks](#) have been requested to [commit](#).

Internally, `maybeCommitPerUserRequested` walks over the [running stream tasks](#) and requests them to [commit](#) when the commit was [requested](#) or [needed](#).

For every commit, `maybeCommitPerUserRequested` increments an internal `committed` counter and prints out the following DEBUG message to the logs:

```
Committed active task [taskId] per user request in
```

In the end, `maybeCommitPerUserRequested` returns the internal `committed` counter.

In case of a `TaskMigratedException`, `maybeCommitPerUserRequested` prints out the following INFO message to the logs, [closes the zombie task](#), and possibly removes the task from the [running stream tasks](#).

```
Failed to commit [taskId] since it got migrated to another thread already. Closing it
as zombie before triggering a new rebalance.
```

In case of a `RuntimeException`, `maybeCommitPerUserRequested` prints out the following ERROR message to the logs and re-throws the exception.

```
Failed to commit StreamTask [taskId] due to the following error:
```

Note

`maybeCommitPerUserRequested` is used exclusively when `TaskManager` is requested to `maybeCommitActiveTasksPerUserRequested`.

Purgable Offsets of Repartition Topics (of Topology) — `recordsToDelete` Method

```
Map<TopicPartition, Long> recordsToDelete()
```

`recordsToDelete` simply requests all the running StreamTasks for the purgable offsets of the repartition topics (of a topology).

Note

`recordsToDelete` is used exclusively when `TaskManager` is requested to attempt to purge (delete) committed records.

`closeAllRestoringTasks` Method

```
RuntimeException closeAllRestoringTasks()
```

`closeAllRestoringTasks` ...FIXME

Note

`closeAllRestoringTasks` is used exclusively when `TaskManager` is requested to suspend all (active and standby) stream tasks and state.

`clear` Method

```
void clear()
```

`clear` requests the parent `AssignedTasks` to `clear`.

In the end, `clear` clears up (removes all elements from) the `restoring`, `restoringByPartition`, and `restoredPartitions` internal registries.

Note

`clear` is used exclusively when `AssignedTasks` is requested to `close`.

Describing Itself (Textual Representation) — `toString` Method

```
String toString(String indent)
```

Note

`toString` is part of the [AssignedTasks Contract](#) to describe itself.

`toString` requests the parent `AssignedTasks` to [describe itself](#) and then [describe](#) (with the `StreamTasks` from the [restoring](#) registry and `Restoring: name`).

`FIXME toString in action`

allTasks Method

`List<StreamTask> allTasks()`

Note

`allTasks` is part of the [AssignedTasks Contract](#) to get all [stream processor tasks](#).

`allTasks` requests the parent `AssignedTasks` for the [all tasks](#) and then adds the [restoring](#) tasks.

allAssignedTaskIds Method

`Set<TaskId> allAssignedTaskIds()`

Note

`allAssignedTaskIds` is part of the [AssignedTasks Contract](#) to get all [assigned TaskIds](#).

`allAssignedTaskIds` requests the parent `AssignedTasks` for the [assigned task IDs](#) and then adds the [restoring](#) task IDs.

Internal Properties

Name	Description
committed	Number of...FIXME
log	
restoredPartitions	<p>Restored partitions (<code>Set<TopicPartition></code>)</p> <ul style="list-style-type: none"> • New partitions added in <code>updateRestored</code> • Partitions are removed in <code>updateRestored</code> (when... FIXME) • Cleared up (all partitions removed) in <code>closeAllRestoringTasks</code>, <code>clear</code> and in <code>updateRestored</code> (when <code>allTasksRunning</code>)
restoring	<p>Lookup table of <code>StreamTasks</code> by <code>TaskId</code> (<code>Map<TaskId, StreamTask></code>)</p> <ul style="list-style-type: none"> • Entries added in <code>addToRestoring</code> • Entries removed in <code>updateRestored</code> • Cleared up (all mappings removed) in <code>closeAllRestoringTasks</code>, <code>clear</code> • Printed out in <code>toString</code> <p>Used in <code>allTasks</code>, <code>allTasksRunning</code>, <code>closeAllRestoringTasks</code>, <code>allAssignedTaskIds</code></p>
restoringByPartition	

ProcessorNodePunctuator Contract

`ProcessorNodePunctuator` is the [abstraction](#) of [processor node punctuators](#) that can [punctuate a processor](#), i.e. execute scheduled periodic actions.

```
void punctuate(  
    ProcessorNode node,  
    long streamTime,  
    PunctuationType type,  
    Punctuator punctuator)
```

`punctuate` is used exclusively when `PunctuationQueue` is requested to [attempt to punctuate](#) (for `StreamTask` that attempts to punctuate by [stream](#) and [system](#) time).

Note	StreamTask is the default and only known implementation of the ProcessorNodePunctuator Contract in Kafka Streams.
------	---

[StreamTask](#) is the default and only known implementation of the [ProcessorNodePunctuator Contract](#) in Kafka Streams.

TaskManager

TaskManager manages active and standby tasks of a StreamThread.

TaskManager is created alongside a StreamThread (with empty AssignedStreamsTasks and AssignedStandbyTasks).

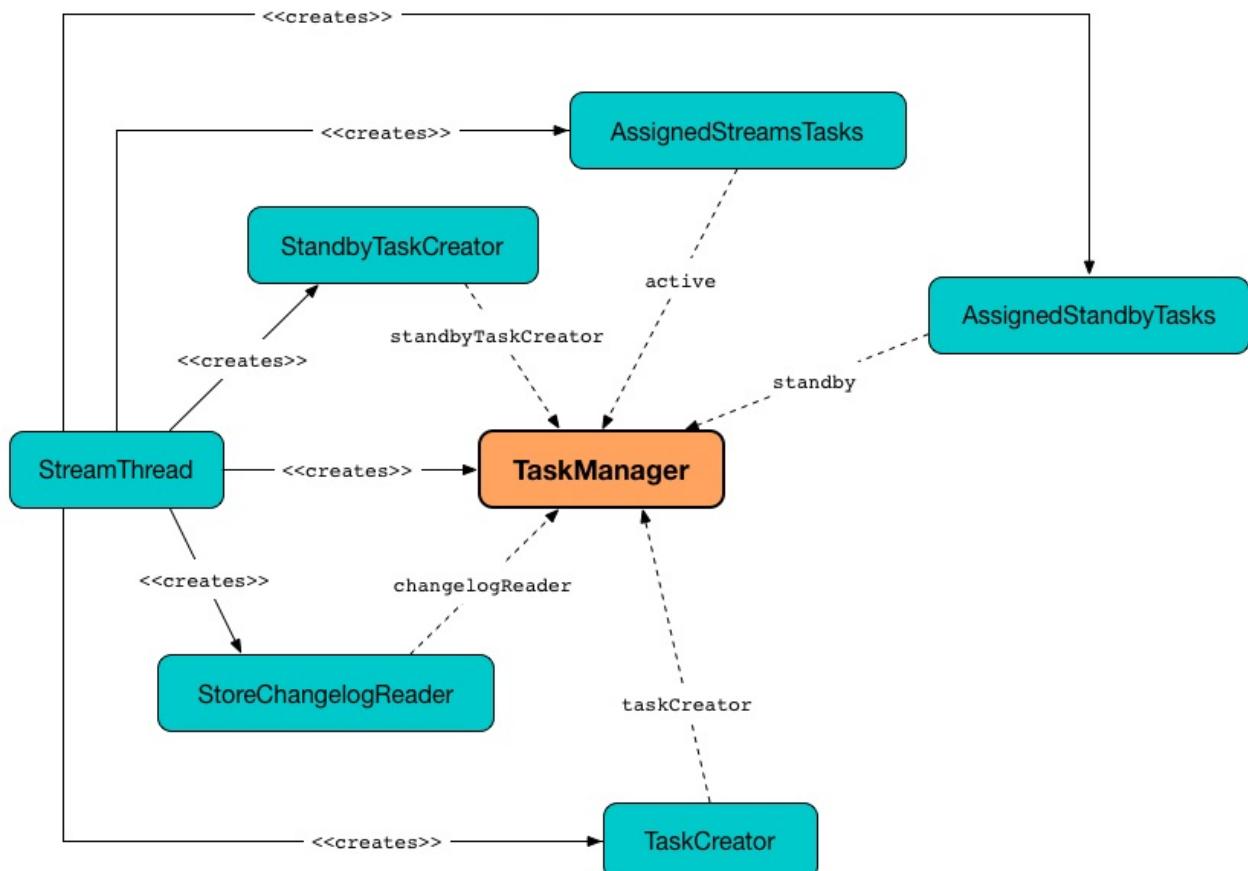


Figure 1. Creating TaskManager

TaskManager creates new tasks when RebalanceListener handles onPartitionsAssigned (and requests to create processor tasks for assigned topic partitions that in turn adds new stream tasks).

TaskManager uses the StandbyTaskCreator to create standby tasks when requested to addStandbyTasks.

TaskManager is given a AssignedStandbyTasks when created for the following:

- createTasks (and addStandbyTasks in particular)
- suspendTasksAndState
- Looking up a StandbyTask per partition
- updateNewAndRestoringTasks

- [assignStandbyPartitions](#)
- [commitAll](#)
- [shutdown](#)

When [created](#), `TaskManager` is given a `StreamsMetadataState` that is used exclusively to [get notifications about the changes in cluster metadata](#). The `StreamsMetadataState` can then be displayed when `TaskManager` is requested for [text representation](#).

Displaying StreamsMetadataState (In Textual Representation)

```
// FIXME: TaskManager.toString
```

Tip	<p>Enable <code>ALL</code> logging level for <code>org.apache.kafka.streams.processor.internals.TaskManager</code> logger to see what happens inside.</p> <p>Add the following line to <code>log4j.properties</code> :</p> <pre>log4j.logger.org.apache.kafka.streams.processor.internals.TaskManager=ALL</pre> <p>Refer to Application Logging Using log4j.</p>
------------	--

Creating TaskManager Instance

`TaskManager` takes the following to be created:

- [ChangelogReader](#)
- Process ID
- `logPrefix`
- Kafka "restore" Consumer (`Consumer<byte[], byte[]>`)
- `StreamsMetadataState`
- [AbstractTaskCreator](#) of `StreamTasks` (`StreamThread.AbstractTaskCreator<StreamTask>`)
- [AbstractTaskCreator](#) of `StandbyTasks` (`StreamThread.AbstractTaskCreator<StandbyTask>`)
- [AdminClient](#)
- [AssignedStreamsTasks](#)
- [AssignedStandbyTasks](#)

`TaskManager` initializes the [internal properties](#).

TaskManager and AdminClient

When [created](#), `TaskManager` is given a Kafka [AdminClient](#).

`TaskManager` uses the `AdminClient` when requested to [try to purge \(delete\) committed records](#).

The `AdminClient` is also used for the following:

- `StreamsPartitionAssignor` is requested to [configure](#) (and creates an [InternalTopicManager](#))
- `StreamThread` is requested for the [adminClientMetrics](#)

Notifying StreamsMetadataState About Changes in Cluster Metadata — `setPartitionsByHostState` Method

```
void setPartitionsByHostState(Map<HostInfo, Set<TopicPartition>> partitionsByHostState)
```

`setPartitionsByHostState` simply notifies the `StreamsMetadataState` that [cluster metadata has changed](#).

Note	<code>setPartitionsByHostState</code> is used when <code>streamsPartitionAssignor</code> is requested to assign and onAssignment .
------	--

Setting Assignment Metadata with Active and Standby Tasks — `setAssignmentMetadata` Method

```
setAssignmentMetadata(
    Map<TaskId, Set<TopicPartition>> activeTasks,
    Map<TaskId, Set<TopicPartition>> standbyTasks)
```

`setAssignmentMetadata` simply initializes the [assignedActiveTasks](#) and [assignedStandbyTasks](#) internal registries with the given `activeTasks` and `standbyTasks`, respectively.

Note	<code>setAssignmentMetadata</code> is used exclusively when <code>streamsPartitionAssignor</code> is requested to handle assignment from the consumer group leader .
------	--

updateSubscriptionsFromAssignment Method

```
void updateSubscriptionsFromAssignment(List<TopicPartition> partitions)
```

updateSubscriptionsFromAssignment ...FIXME

Note	updateSubscriptionsFromAssignment is used exclusively when StreamsPartitionAssignor is requested to handle assignment from the leader
------	---

updateSubscriptionsFromMetadata Method

```
void updateSubscriptionsFromMetadata(Set<String> topics)
```

updateSubscriptionsFromMetadata ...FIXME

Note	updateSubscriptionsFromMetadata is used when StreamsPartitionAssignor is requested to subscription.
------	---

Suspending All (Active and Standby) Tasks And State — suspendTasksAndState Method

```
void suspendTasksAndState()
```

suspendTasksAndState ...FIXME

Note	suspendTasksAndState is used exclusively when RebalanceListener is requested to handle onPartitionsAssigned event.
------	--

updateNewAndRestoringTasks Method

```
boolean updateNewAndRestoringTasks()
```

updateNewAndRestoringTasks returns true only when the AssignedStreamsTasks tasks are all running and AssignedStandbyTasks tasks are all running in the end. Otherwise, updateNewAndRestoringTasks returns false .

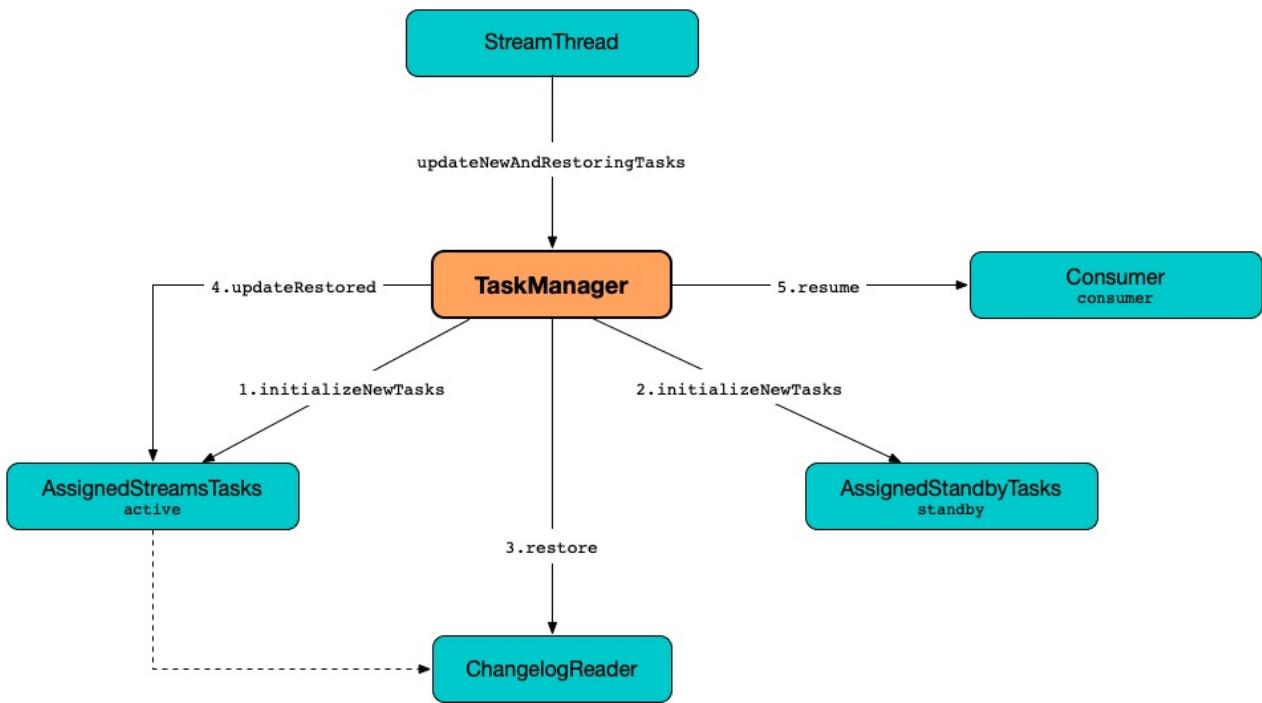


Figure 2. TaskManager and `updateNewAndRestoringTasks`

Internally, `updateNewAndRestoringTasks` requests the `AssignedStreamsTasks` and the `AssignedStandbyTasks` tasks to initialize new tasks.

`updateNewAndRestoringTasks` requests the `ChangelogReader` to restore the `AssignedStreamsTasks` tasks (*restored*).

`updateNewAndRestoringTasks` requests the `AssignedStreamsTasks` tasks to updateRestored (from the `ChangelogReader`).

`updateNewAndRestoringTasks` requests the `AssignedStreamsTasks` tasks to check whether all StreamTasks are running.

If not all active StreamTasks are running, `updateNewAndRestoringTasks` simply returns `false`.

Otherwise, if all active StreamTasks are running, `updateNewAndRestoringTasks` requests the Kafka consumer for the partition assignment (using `Consumer.assignment`) and prints out the following TRACE message to the logs:

```
Resuming partitions [assignment]
```

`updateNewAndRestoringTasks` then requests the Kafka consumer to resume partitions (using `Consumer.resume`), `assignStandbyPartitions` and returns `true`.

Note	<code>updateNewAndRestoringTasks</code> is used exclusively when <code>StreamThread</code> is requested to poll records once and process them using active stream tasks (while running the main record processing loop in <code>PARTITIONS_ASSIGNED</code> state).
------	--

assignStandbyPartitions Internal Method

```
void assignStandbyPartitions()
```

assignStandbyPartitions ...FIXME

Note	assignStandbyPartitions is used when...FIXME
------	--

Looking Up Stream Task Per Partition — activeTask Method

```
StreamTask activeTask(TopicPartition partition)
```

activeTask simply requests the [AssignedStreamsTasks](#) for the running StreamTask for the input [partition](#).

Note	activeTask is used exclusively when StreamThread is requested to add records to active stream tasks.
------	--

hasActiveRunningTasks Method

```
boolean hasActiveRunningTasks()
```

hasActiveRunningTasks simply asks the [AssignedStreamsTasks](#) whether it [has any running stream tasks](#) or not.



Figure 3. TaskManager and AssignedStreamsTasks

Note	hasActiveRunningTasks is used exclusively when StreamThread is requested to poll records once and process them using active stream tasks (and there are records to be processed).
------	---

hasStandbyRunningTasks Method

```
boolean hasStandbyRunningTasks()
```

`hasStandbyRunningTasks` simply asks the `AssignedStandbyTasks` whether it has any running standby tasks or not.

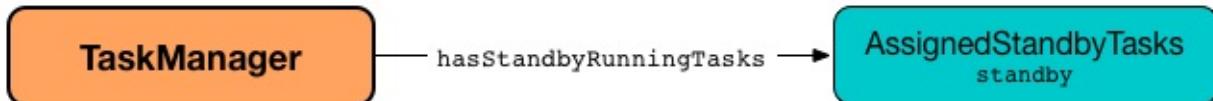


Figure 4. TaskManager and AssignedStandbyTasks

Note

`hasStandbyRunningTasks` is used exclusively when `StreamThread` is requested to `maybeUpdateStandbyTasks` (while `poll` records once and process them using active stream tasks).

Creating Tasks for Assigned Partitions — `createTasks` Method

```
void createTasks(final Collection<TopicPartition> assignment)
```

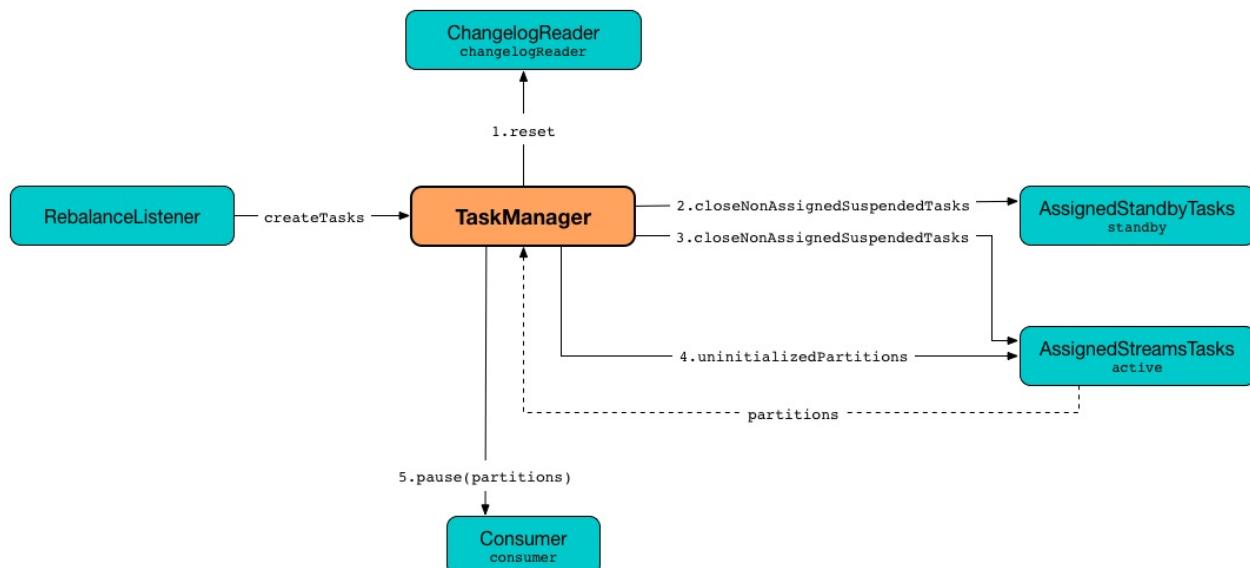


Figure 5. TaskManager.createTasks

`createTasks` requests the `AssignedStandbyTasks` and the `AssignedStreamsTasks` to `closeNonAssignedSuspendedTasks` (with the `assignedStandbyTasks` and the `assignedActiveTasks`, respectively).

`createTasks` (re)creates the stream tasks for the input `assignment` partitions.

`createTasks` `addStandbyTasks`.

`createTasks` prints out the following TRACE message to the logs:

```
Pausing partitions: [assignment]
```

In the end, `createTasks` requests the Kafka consumer to pause the `assignment` partitions.

Note

`createTasks` triggers `Consumer.pause` method that suspends fetching records from partitions until they have been resumed using `Consumer.resume`.

`createTasks` reports an `IllegalStateException` if the `consumer` is not defined (`null`):

stream-thread [threadClientId] consumer has not been initialized while adding stream tasks. This should not happen.

Note

`createTasks` is used exclusively when `RebalanceListener` is requested to handles an `onPartitionsAssigned` event.

(Re)Creating Stream Tasks Per Assigned Partitions

— `addStreamTasks` Internal Method

```
void addStreamTasks(  
    Collection<TopicPartition> assignment)
```

`addStreamTasks` registers new stream tasks.

Note

`addStreamTasks` does nothing (and simply exits) unless `assignedActiveTasks` has at least one task id.

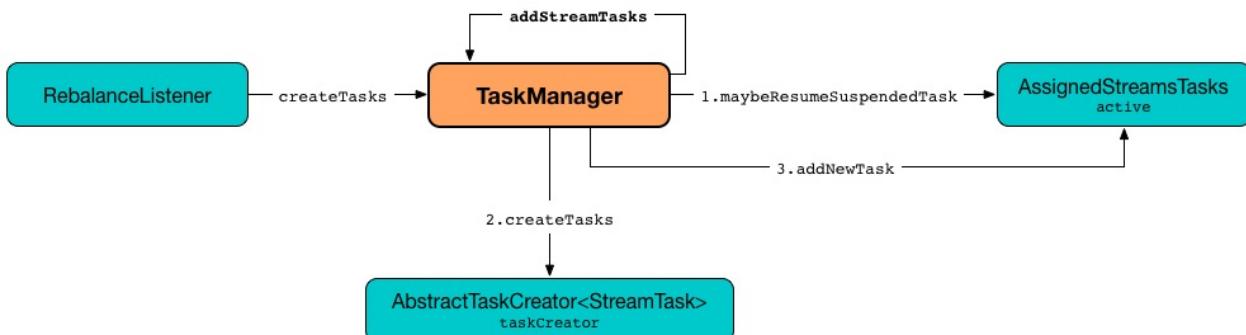


Figure 6. `TaskManager.addStreamTasks`

`addStreamTasks` prints out the following DEBUG message to the logs:

```
Adding assigned tasks as active: [assignedActiveTasks]
```

For every task id in `assignedActiveTasks`, if the partitions of a task are all included in the input `assignment` partitions `addStreamTasks` requests `AssignedStreamsTasks` to `maybeResumeSuspendedTask` (passing in the task id and partitions). If negative, `addStreamTasks` records the task and partitions in a local registry of new tasks to be created.

If the partitions of a task are not all included in the input `assignment` partitions `addStreamTasks` prints out the following WARN message to the logs:

```
Task [taskId] owned partitions [partitions] are not contained in the assignment [assignment]
```

When there are new tasks to be created, `addStreamTasks` prints out the following TRACE message to the logs:

```
New active tasks to be created: [newTasks]
```

`addStreamTasks` then requests `StreamThread.AbstractTaskCreator<StreamTask>` to `createTasks` for every new task (with the `Kafka Consumer`) and requests `AssignedStreamsTasks` to register a new task.

Note	<code>addStreamTasks</code> is used exclusively when <code>TaskManager</code> is requested to create tasks for assigned topic partitions.
------	---

Adding Assigned Standby Tasks — `addStandbyTasks` Internal Method

```
void addStandbyTasks()
```

`addStandbyTasks` registers new standby tasks.

Note	<code>addStandbyTasks</code> does nothing and simply exits when the <code>assignedStandbyTasks</code> internal registry has no standby tasks assigned.
------	--

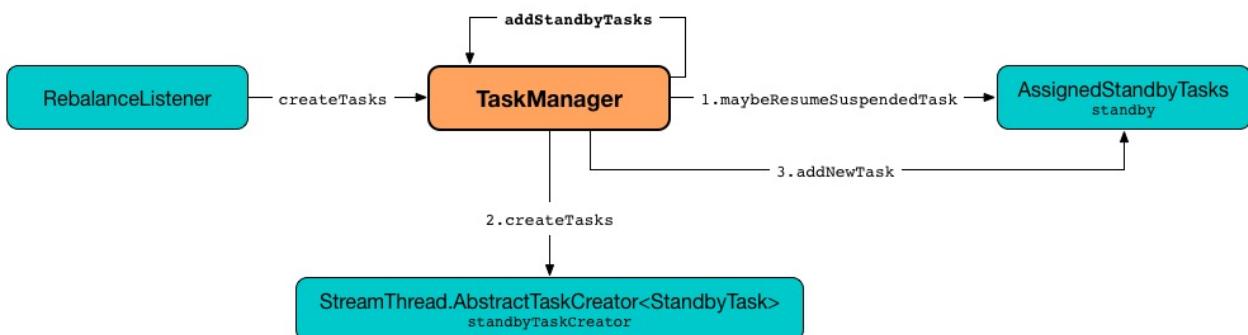


Figure 7. `TaskManager.addStandbyTasks`

`addStandbyTasks` prints out the following DEBUG message to the logs:

```
Adding assigned standby tasks [assignedStandbyTasks]
```

For every task (id and partitions) in the `assignedStandbyTasks` registry, `addStandbyTasks` requests `AssignedStandbyTasks` to `maybeResumeSuspendedTask` and, if negative, adds the task to tasks to be created in standby mode.

If no new tasks should be in standby mode, `addStandbyTasks` simply exits.

When there are new tasks to be in standby mode, `addStandbyTasks` prints out the following TRACE message to the logs:

```
New standby tasks to be created: [newStandbyTasks]
```

`addStandbyTasks` then requests `StreamThread.AbstractTaskCreator<StandbyTask>` to `createTasks` for every new standby task (with the `Kafka Consumer`) and requests `AssignedStandbyTasks` to register a new task.

Note	<code>addStandbyTasks</code> is used exclusively when <code>TaskManager</code> is requested to <code>create tasks for assigned partitions</code> .
------	--

Describing Itself (Textual Representation) — `toString` Method

```
String toString(final String indent)
```

`toString` gives a text representation with the following:

- "Active tasks:" followed by the text representation of `AssignedStreamsTasks`
- "Standby tasks:" followed by the text representation of `AssignedStandbyTasks`

```
FIXME toString in action
```

Note	<code>toString</code> is used exclusively when <code>StreamThread</code> is requested to <code>describe itself</code> .
------	---

Attempting to Purge (Delete) Committed Records of Repartition Topics — `maybePurgeCommittedRecords` Method

```
void maybePurgeCommittedRecords()
```

In essence, `maybePurgeCommittedRecords` requests the `AssignedStreamsTasks` for the `purgable offsets of the repartition topics (of a topology)` and then the `AdminClient` to delete the records (whose offset is smaller than the given offset of the corresponding partition).

```
maybePurgeCommittedRecords ...FIXME
```

Note

`maybePurgeCommittedRecords` is used exclusively when `StreamThread` is requested to [commit all tasks \(when commit interval elapsed\)](#) (when `StreamThread` is requested to [poll records once and process them using active stream tasks](#) in the [main record processing loop](#)).

Processing Records by Running Stream Tasks (One Record Per Task) — `process` Method

```
int process()
```

`process` simply requests [AssignedStreamsTasks](#) to request the running stream tasks to [process a single record \(per task\)](#).

In the end, `process` gives the number of [stream tasks](#) that processed a record.

Note

`process` is used exclusively when `StreamThread` is requested to [process records \(with optional commit\)](#) (when requested to [poll records once and process them using active stream tasks](#)).

Committing Active Running Stream Tasks that Requested It — `maybeCommitActiveTasks` Method

```
int maybeCommitActiveTasks()
```

`maybeCommitActiveTasks` simply requests [AssignedStreamsTasks](#) to [commit running stream tasks that requested it](#).

In the end, `maybeCommitActiveTasks` gives the number of running stream tasks that [needed a commit](#).

Note

`maybeCommitActiveTasks` is used exclusively when `StreamThread` is requested to [processAndMaybeCommit](#).

Punctuating Stream Tasks — `punctuate` Method

```
int punctuate()
```

`punctuate` simply requests the [AssignedStreamsTasks](#) to [punctuate running stream tasks \(by stream and system time\)](#).

Note

`punctuate` is used exclusively when `StreamThread` is requested to attempt to [punctuate](#).

Committing All Active (Stream and Standby) Tasks — `commitAll` Method

```
int commitAll()
```

`commitAll` ...FIXME

Note

`commitAll` is used exclusively when `StreamThread` is requested to [commit all tasks \(when commit interval elapsed\)](#).

All Active Tasks — `activeTaskIds` Method

```
Set<TaskId> activeTaskIds()
```

`activeTaskIds` simply requests the [AssignedStreamsTasks](#) for the [assigned task IDs](#).

Note

`activeTaskIds` is used when:

- `StreamThread` is requested to [commit all active tasks \(when commit interval elapsed\)](#)
- `RebalanceListener` is requested to handle [onPartitionsAssigned](#) and [onPartitionsRevoked](#) events

`standbyTaskIds` Method

```
Set<TaskId> standbyTaskIds()
```

`standbyTaskIds` ...FIXME

Note

`standbyTaskIds` is used when...FIXME

`cachedTaskIds` Method

```
Set<TaskId> cachedTaskIds()
```

`cachedTasksIds` requests the [StreamTask creator](#) for the [StateDirectory](#) that is in turn requested for the [task](#) directories.

`cachedTasksIds` collects the [TaskIds](#) ([parsing the names of the directories](#)) that correspond to task directories with [.checkpoint](#) file.

Note

`cachedTasksIds` is used exclusively when `StreamsPartitionAssignor` is requested to [subscription](#).

maybeCommitActiveTasksPerUserRequested Method

```
int maybeCommitActiveTasksPerUserRequested()
```

`maybeCommitActiveTasksPerUserRequested` simply requests the [AssignedStreamsTasks](#) to [maybeCommitPerUserRequested](#).

Note

`maybeCommitActiveTasksPerUserRequested` is used when `StreamThread` is requested to [poll records once and process them using active stream tasks](#) (and any of the active stream tasks have processed a record) and [commit all tasks](#) (when commit interval elapsed).

Shutting Down — shutdown Method

```
void shutdown(boolean clean)
```

`shutdown` ...FIXME

Note

`shutdown` is used when...FIXME

Looking Up StandbyTask Per Partition — standbyTask Method

```
StandbyTask standbyTask(TopicPartition partition)
```

`standbyTask` simply requests the [AssignedStandbyTasks](#) for the [running StandbyTask](#) for the input [partition](#).

Note

`standbyTask` is used exclusively when `StreamThread` is requested to [attempt to update running StandbyTasks](#).

Previously Active Tasks — `prevActiveTaskIds` Method

```
Set<TaskId> prevActiveTaskIds()
```

`prevActiveTaskIds` simply requests the [AssignedStreamsTasks](#) for the previously active tasks.

Note	<code>prevActiveTaskIds</code> is used when...FIXME
------	---

Internal Properties

Name	Description
assignedActiveTasks	<p><code>Map<TaskId, Set<TopicPartition>> assignedActiveTasks</code></p> <p>Assigned active tasks with the partitions per task id</p> <p>Initialized when setting assignment metadata with active and standby tasks</p> <p>Used exclusively when <code>TaskManager</code> is requested to create tasks for the assigned partitions</p>
assignedStandbyTasks	<p><code>Map<TaskId, Set<TopicPartition>> assignedStandbyTasks</code></p> <p>Assigned standby tasks (as Kafka TopicPartitions per task id)</p> <ul style="list-style-type: none"> • Set when setting assignment metadata with active and standby tasks • Used when creating processor tasks for assigned topic partitions (and addStandbyTasks)
cluster	<p>Cluster metadata, i.e. Kafka Cluster with topic partitions</p> <ul style="list-style-type: none"> • Set when <code>StreamsPartitionAssignor</code> does assign and onAssignment
consumer	<p>Kafka Consumer (<code>Consumer<byte[], byte[]></code>)</p> <p>Assigned right when <code>StreamThread</code> is created (and corresponds to the Kafka consumer from the KafkaClientSupplier that was used to create the KafkaStreams)</p>
deleteRecordsResult	<p>Kafka's <code>DeleteRecordsResult</code> (after maybePurgeCommittedRecords and requesting the AdminClient to delete records)</p> <p>Used in maybePurgeCommittedRecords for informative purposes</p>

TaskCreator — Factory of Stream Processor Tasks (StreamTasks)

`TaskCreator` is a concrete [task factory](#) of [stream processor tasks \(StreamTasks\)](#).

`TaskCreator` is [created](#) exclusively for [StreamThread \(activeTaskCreator\)](#) for the only purpose of creating the [TaskManager](#).

Creating TaskCreator Instance

`TaskCreator` takes the following to be created:

- [InternalTopologyBuilder](#)
- [StreamsConfig](#)
- [StreamsMetricsThreadImpl](#)
- [StateDirectory](#)
- [ChangelogReader](#)
- [ThreadCache](#)
- `Time`
- [KafkaClientSupplier](#)
- [Kafka Producer](#) (`Producer<byte[], byte[]>`)
- `threadClientId`
- `Logger`

Creating Stream Task — `createTask` Factory Method

```
StreamTask createTask(
    Consumer<byte[], byte[]> consumer,
    TaskId taskId,
    Set<TopicPartition> partitions)
```

Note

`createTask` is part of [AbstractTaskCreator Contract](#) to create a concrete [stream processor task](#).

`createTask` requests the [taskCreatedSensor](#) to record this execution.

`createTask` then creates a [StreamTask](#) (for a given topic group ID and a [producer client for the task](#)).

createProducer Internal Method

```
Producer<byte[], byte[]> createProducer(final TaskId id)
```

`createProducer` ...FIXME

Note

`createProducer` is used exclusively when `TaskCreator` is requested to [create a stream processor task](#) (for a task ID).

StandbyTaskCreator — Factory of Standby Tasks (StandbyTasks)

`StandbyTaskCreator` is a concrete [task factory](#) of [standby tasks \(StandbyTasks\)](#).

`StandbyTaskCreator` is [created](#) exclusively for [StreamThread](#) (*standbyTaskCreator*) for the only purpose of creating the [TaskManager](#).

`StandbyTaskCreator` always records when requested to [create a task](#) for monitoring purposes, but creates a [standby task](#) only for [processor task topologies](#) (for [which it was created](#)) with at least one [state store](#).

Creating StandbyTaskCreator Instance

`StandbyTaskCreator` takes the following to be created:

- [InternalTopologyBuilder](#)
- [StreamsConfig](#)
- [StreamsMetricsThreadImpl](#)
- [StateDirectory](#)
- [ChangelogReader](#)
- `Time`
- `Logger`

Creating Standby Task — `createTask` Factory Method

```
StandbyTask createTask(
    final Consumer<byte[], byte[]> consumer,
    final TaskId taskId,
    final Set<TopicPartition> partitions)
```

Note

`createTask` is part of [AbstractTaskCreator Contract](#) to create a [stream processor task](#).

`createTask` records execution (occurrence) for monitoring purposes and creates a [StandbyTask](#) only if the processor task topology has at least one [state store](#).

Internally, `createTask` first requests the [StreamsMetricsThreadImpl](#) for the [taskCreatedSensor](#) and records execution (i.e. increments the counter of how many times `createTask` was executed).

`createTask` requests the [InternalTopologyBuilder](#) to build the processor task topology for the [topicGroupId](#) of the given `TaskId`.

`createTask` creates a [StandbyTask](#) only if the `ProcessorTopology` has at least one [state store](#).

If the `ProcessorTopology` has no [state stores](#), `createTask` simply prints out the following TRACE message to the logs and returns `null`.

```
Skipped standby task [taskId] with assigned partitions [partitions] since it does not have any state stores to materialize
```

AbstractTaskCreator

`AbstractTaskCreator` is the [base](#) of task creators (*factories*) that can [create stream processor tasks](#).

Table 1. AbstractTaskCreator Contract

Method	Description
<code>createTask</code>	<pre>T createTask(Consumer<byte[], byte[]> consumer, TaskId id, Set<TopicPartition> partitions)</pre> <p>Creates a task Used exclusively when <code>AbstractTaskCreator</code> is requested to create tasks (when <code>TaskManager</code> is requested for standby or stream tasks)</p>
Note	<code>AbstractTaskCreator</code> is a Java abstract class and cannot be created directly. It is created indirectly when the concrete AbstractTaskCreators are.

Table 2. AbstractTaskCreators

AbstractTaskCreator	Description
StandbyTaskCreator	Creates standby tasks
TaskCreator	Creates stream tasks

Creating Tasks — `createTasks` Method

```
Collection<T> createTasks(
    Consumer<byte[], byte[]> consumer,
    Map<TaskId, Set<TopicPartition>> tasksToBeCreated)
```

`createTasks` ...FIXME

Note	<code>createTasks</code> is used when <code>TaskManager</code> is requested for standby and stream tasks.
------	---

Creating AbstractTaskCreator Instance

`AbstractTaskCreator` takes the following when created:

- [InternalTopologyBuilder](#)
- [StreamsConfig](#)
- [StreamsMetrics](#)
- [StateDirectory](#)
- Kafka `Sensor` (to monitor task created)
- [ChangelogReader](#)
- `Time`
- `Logger`

StreamThread — Stream Processor Thread

`StreamThread` is a **stream processor thread** (a Java Thread) that runs the main record processing loop when started.

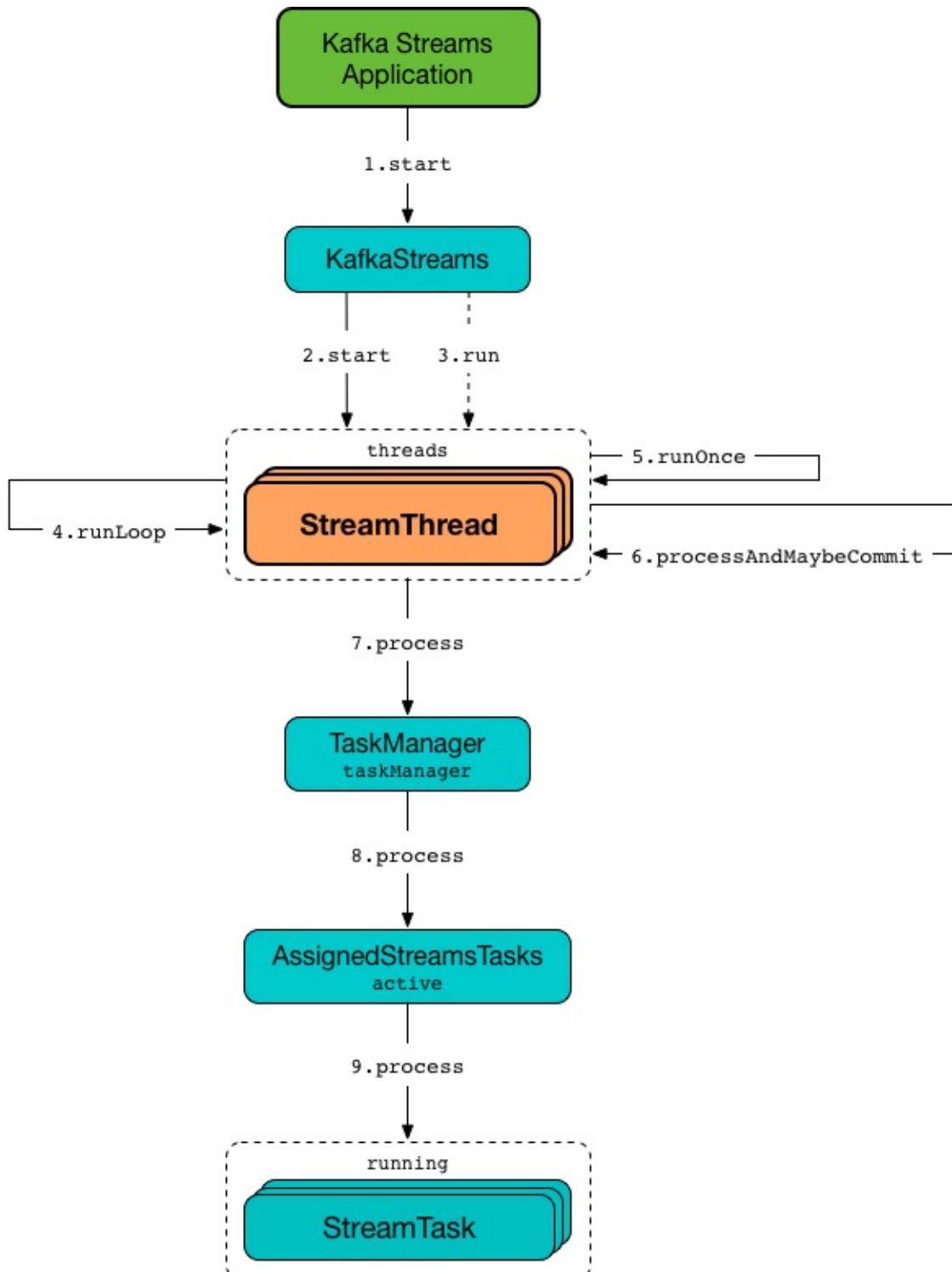


Figure 1. StreamThread and Stream Processing

`StreamThread` is created exclusively alongside **KafkaStreams** (which is one of the main entities that a Kafka Streams developer uses in a Kafka Streams application).

Note

`KafkaStreams` uses `num.stream.threads` configuration property for the number of `StreamThreads` to create (default: 1).

`StreamThread` uses a [Kafka Consumer](#) and a custom [ConsumerRebalanceListener](#) (with the [TaskManager](#)) when subscribing to source topics (when requested to run the main record processing loop and enforce a rebalance). `StreamThread` uses an [InternalTopologyBuilder](#) for the [source topics](#) to subscribe to.

Note

When [partitions get assigned](#), the custom [ConsumerRebalanceListener](#) requests the [TaskManager](#) to [create tasks for the assigned partitions](#).

`StreamThread` uses a [Kafka Consumer](#) and a [TaskManager](#) that are both created when `StreamThread` object is requested to create an instance of itself. That is when the `StreamThread` sets the `TASK_MANAGER_FOR_PARTITION_ASSIGNOR` internal property and indirectly associates the [TaskManager](#) with [StreamsPartitionAssignor](#). `StreamThread` also requests the given [KafkaClientSupplier](#) to create a [KafkaConsumer](#) (with the [TaskManager](#)) and so when the `partition.assignment.strategy` configuration property is picked up, `StreamsPartitionAssignor` is [created](#) and eventually [configured](#) (that will use the `TASK_MANAGER_FOR_PARTITION_ASSIGNOR` internal property).

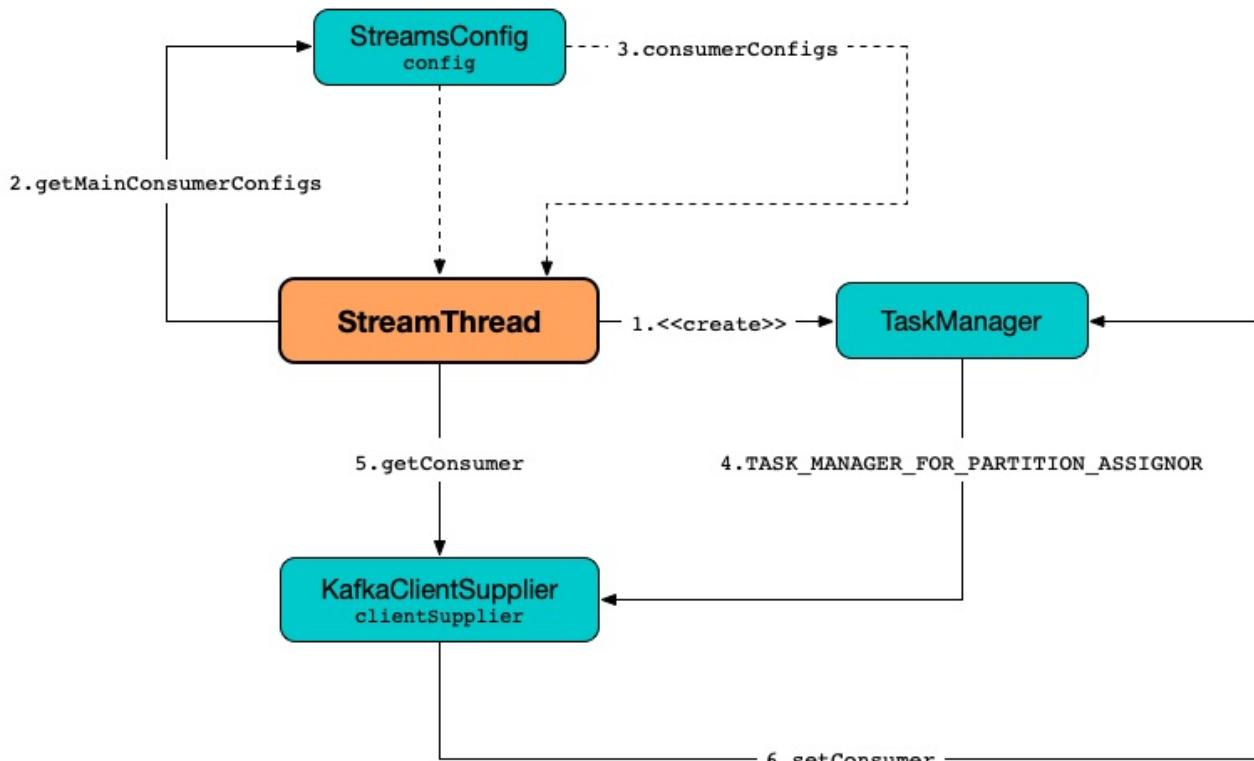


Figure 2. StreamThread and Registering TaskManager under `TASK_MANAGER_FOR_PARTITION_ASSIGNOR`

`StreamThread` uses the `poll.ms` configuration property (default: 100 ms) as the **polling interval** when requested to poll records once and process them using active stream tasks.

`StreamThread` uses the `commit.interval.ms` configuration property as the **flush interval** for persisting the position of a processor (when [polling records once and processing them using active stream tasks](#) and [maybeCommit](#)).

`StreamThread` uses the [Kafka Consumer](#) to:

- Subscribe to topics (with [RebalanceListener](#)) right after `StreamThread` has been requested to [run the main record processing loop](#)
- Poll the topics subscribed (and fetch records if available) right after `StreamThread` has been requested to [get the next batch of records by polling](#)
- `resetInvalidOffsets` (when an `InvalidOffsetException` is reported while [polling the topics for records](#))

`StreamThread` uses `stream-thread [[clientId]-StreamThread-[STREAM_THREAD_ID]]` for the logging prefix.

`StreamThread` requires an [InternalTopologyBuilder](#) to be [created](#) and uses it for the following:

- Creating a [TaskCreator](#) and a [StandbyTaskCreator](#)
- [Running the main record processing loop](#) (and subscribing to the source topics)
- `resetInvalidOffsets`

`StreamThread` uses the [TaskManager](#) for the following:

- **FIXME**

Tip	<p>Enable any of <code>ALL</code> logging levels for <code>org.apache.kafka.streams.processor.internals.StreamThread</code> logger to see what happens inside.</p> <p>Add the following line to <code>log4j.properties</code> :</p> <pre>log4j.logger.org.apache.kafka.streams.processor.internals.StreamThread=ALL</pre> <p>Refer to Application Logging Using log4j.</p>
------------	--

Creating StreamThread Instance

`StreamThread` takes the following to be created:

- `Time`
- `StreamsConfig`

- Kafka Producer (`Producer<byte[], byte[]>`)
- Kafka "restore" Consumer (`Consumer<byte[], byte[]>`)
- `Consumer<byte[], byte[]>`
- `originalReset`
- TaskManager
- StreamsMetricsThreadImpl
- InternalTopologyBuilder
- Thread Client ID
- LogContext
- Assignment Error Code (`AtomicInteger`)

StreamThread initializes the internal properties.

StreamThread and Kafka Consumer

When created, StreamThread is given a KafkaClientSupplier.

StreamThread uses the KafkaClientSupplier to get a Kafka consumer with the consumer-specific configuration.

The Kafka Consumer is then assigned to the TaskManager and to create the StreamThread.

The Kafka Consumer is used in the following:

- runLoop (to subscribe to source topics with the ConsumerRebalanceListener)
- enforceRebalance (by unsubscribing from the source topics and (re)subscribing)
- pollRequests (to poll records)
- resetInvalidOffsets (to seek to the beginning or the end of partitions assigned)
- completeShutdown (to close the consumer)
- consumerMetrics (for the consumer metrics)

StreamThread and AdminClient

When created, StreamThread is given an AdminClient that is only used to create the TaskManager.

Creating StreamThread Instance — `create` Factory Method

```
StreamThread create(
    InternalTopologyBuilder builder,
    StreamsConfig config,
    KafkaClientSupplier clientSupplier,
    AdminClient adminClient,
    UUID processId,
    String clientId,
    Metrics metrics,
    Time time,
    StreamsMetadataState streamsMetadataState,
    long cacheSizeBytes,
    StateDirectory stateDirectory,
    StateRestoreListener userStateRestoreListener,
    int threadIdx)
```

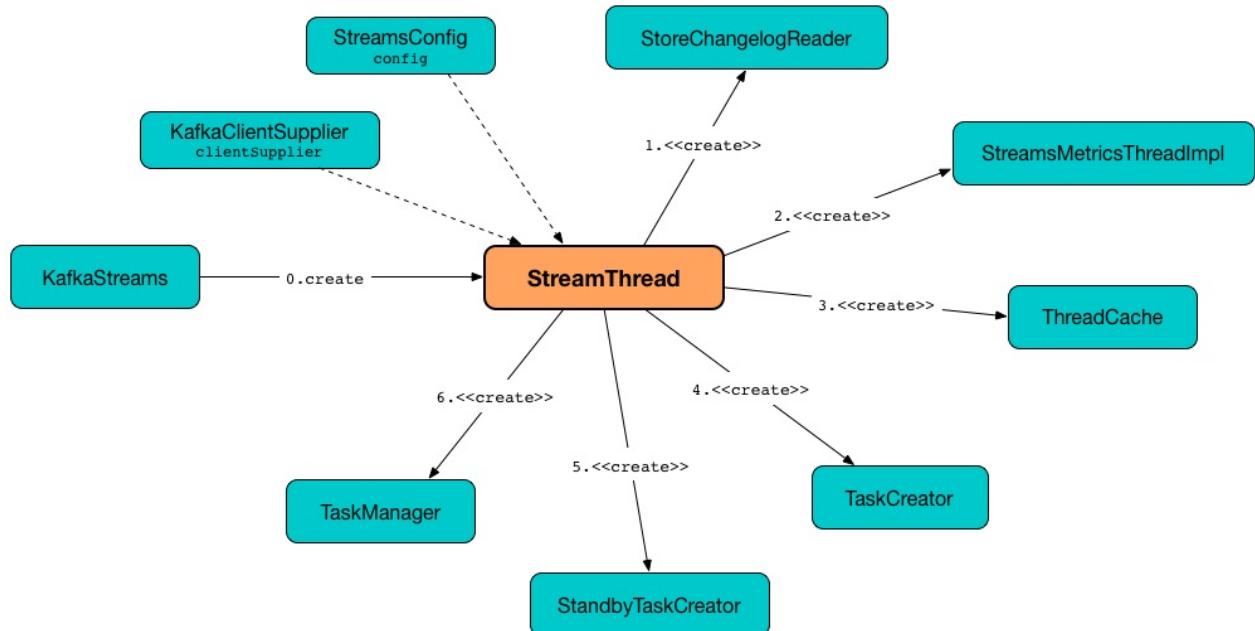


Figure 3. Creating StreamThread

`create` prints out the following INFO message to the logs:

```
Creating restore consumer client
```

`create` requests the input `StreamsConfig` for `getRestoreConsumerConfigs` for a new `threadClientId` (of the format `[clientId]-StreamThread-[STREAM_THREAD_ID]`).

`create` requests the given `KafkaClientSupplier` for `getRestoreConsumer` for the `restoreConsumerConfigs` .

`create` creates a [StoreChangelogReader](#) (with the `restoreConsumer`, the given [StateRestoreListener](#) and the configured `poll.ms`).

Note	The input StateRestoreListener is a DelegatingStateRestoreListener actually.
------	--

(Only with eos enabled) `create` ...FIXME

`create` creates a [StreamsMetricsThreadImpl](#) with the following:

- the input `Metrics`
- **stream-metrics** group name
- `thread.[clientId]-StreamThread-[STREAM_THREAD_ID]` `prefix`
- [Tags](#) with one entry with `client-id` and the `[clientId]-StreamThread-[STREAM_THREAD_ID]` value.

`create` creates a [ThreadCache](#) (with `cacheSizeBytes` for the `maxCacheSizeBytes` and the [StreamsMetricsThreadImpl](#)).

`create` creates a [TaskCreator](#) and a [StandbyTaskCreator](#) that are used exclusively to create a [TaskManager](#) (with a new [AssignedStreamsTasks](#) and [AssignedStandbyTasks](#) as well as the given [StreamsMetadataState](#)).

`create` prints out the following INFO message to the logs:

```
Creating consumer client
```

`create` requests the input `streamsConfig` for [application.id](#) configuration property.

`create` requests the input `streamsConfig` for the [configuration of a Kafka Consumer](#) for the application ID and the `threadClientId` (of the format `[clientId]-StreamThread-[STREAM_THREAD_ID]`) and adds the following internal properties:

- [TASK_MANAGER_FOR_PARTITION_ASSIGNOR](#) to be the `TaskManager` just created
- [ASSIGNMENT_ERROR_CODE](#) to be a new `AtomicInteger`

(Only with non-empty `latestResetTopicsPattern` and `earliestResetTopicsPattern` patterns)

`create` ...FIXME

`create` requests the given `KafkaClientSupplier` for a [Kafka Consumer](#) (with the `consumerConfigs`) and [associates](#) it with the `TaskManager`.

In the end, `create` creates a [StreamThread](#).

Note	<code>create</code> is used exclusively when <code>KafkaStreams</code> is created .
------	---

Starting Stream Thread — `run` Method

```
void run()
```

Note `run` is part of Java's [Thread Contract](#) to be executed by a JVM thread.

`run` prints out the following INFO message to the logs.

```
Starting
```

`run` sets the state to [RUNNING](#) and runs the main record processing loop.

At the end, `run` shuts down (per `cleanRun` flag that says whether running the main loop stopped cleanly or not).

`run` re-throws any `KafkaException`.

`run` prints out the following ERROR message to the logs for any other `Exception`:

```
Encountered the following error during processing: [exception]
```

Note `run` is used exclusively when `kafkaStreams` is requested to [start](#).

Life Cycle of StreamThread — StreamThread's States

`StreamThread` can be in exactly one of the following **states** at any given point in time:

0. `CREATED` - The initial state of `StreamThread` right after it was [created](#)
1. `RUNNING` - `StreamThread` was requested for the following:
 - `run`
 - [Polling records once and processing them using active stream tasks](#) when `StreamThread` is in `PARTITIONS_ASSIGNED` state and `TaskManager` was positive after `updateNewAndRestoringTasks`
 - [Polling records once and processing them using active stream tasks](#) when `StreamThread` polled for records and happened to transition to `PARTITIONS_ASSIGNED` state, but (again) only when `TaskManager` was positive after `updateNewAndRestoringTasks`
2. `STARTING`

3. PARTITIONS_REVOKED - RebalanceListener was requested to handle partition revocation
4. PARTITIONS_ASSIGNED - RebalanceListener was requested to handle partition assignment
5. PENDING_SHUTDOWN - StreamThread was requested to shutdown or completeShutdown
6. DEAD - StreamThread is requested to completeShutdown

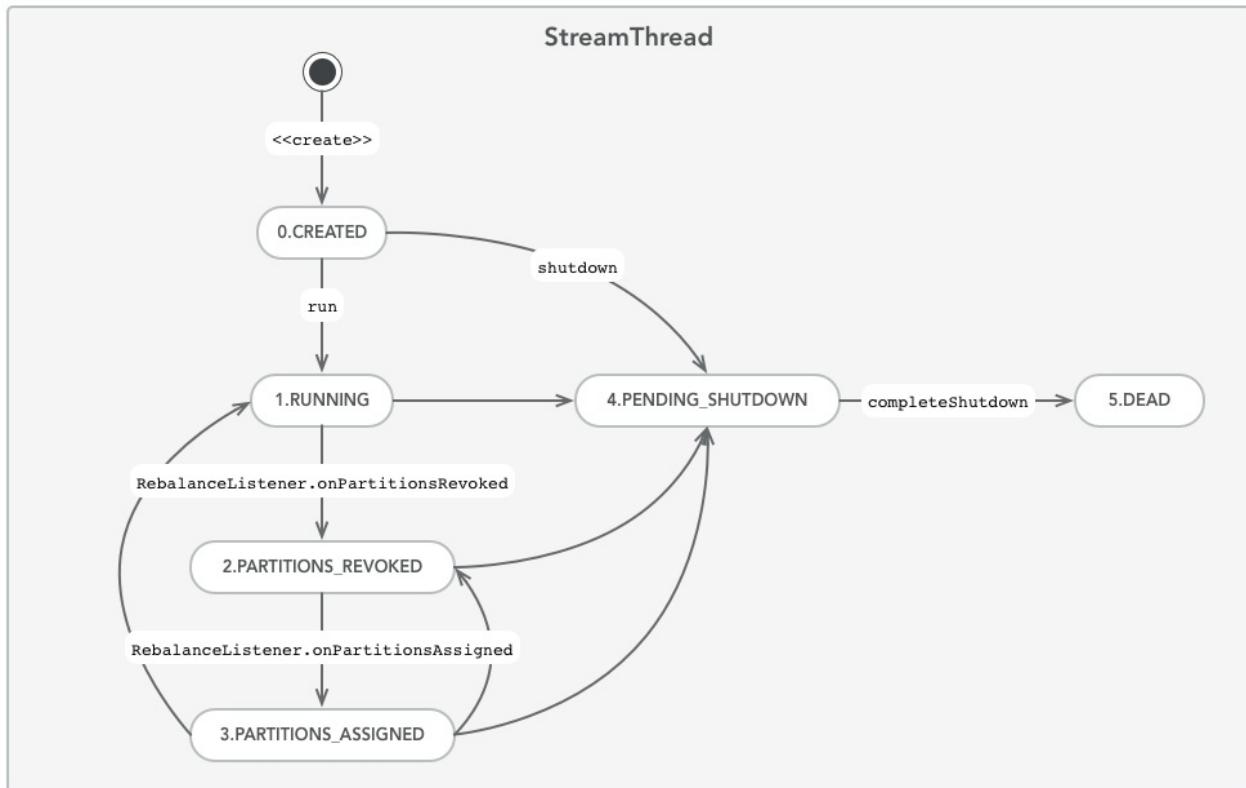


Figure 4. StreamThread's Life Cycle

StreamThread can be transitioned to another state by executing `setState`.

Note

StreamThread defines a Java enumeration `State` with the states above ordered by ordinal. When created, they are assigned the state ordinals that could transition to. You can check whether a transition is valid or not using `State.isValidTransition`.

```

import org.apache.kafka.streams.processor.internals.StreamThread.State._

// CREATED is the 0th state
assert(CREATED.ordinal == 0)

// RUNNING is the next possible state after CREATED
assert(CREATED.isValidTransition(RUNNING))

// DEAD cannot be the next possible state after CREATED
assert(CREATED.isValidTransition(DEAD) == false)
  
```

Shutting Down Stream Thread — `shutdown` Method

```
void shutdown()
```

`shutdown` prints out the following INFO message to the logs:

```
Informed to shut down
```

`shutdown` tries to transition the current state to `PENDING_SHUTDOWN`.

(only when transitioning from `CREATED` state) `shutdown completeShutdown` (with `cleanRun` flag enabled).

	<p><code>shutdown</code> is used when:</p> <ul style="list-style-type: none"> • <code>KafkaStreams</code> is requested to <code>close</code> • <code>RebalanceListener</code> is requested to <code>handle partition assignment</code> (and failed due to <code>INCOMPLETE_SOURCE_TOPIC_METADATA</code> error).
Note	

Polling Records Once And Processing Them Using Active Stream Tasks — `runOnce` Method

```
void runOnce()
```

In essence, `runOnce` requests the `Consumer` to poll records, adds the records to active stream tasks and requests the `TaskManager` to process the records by running stream tasks.

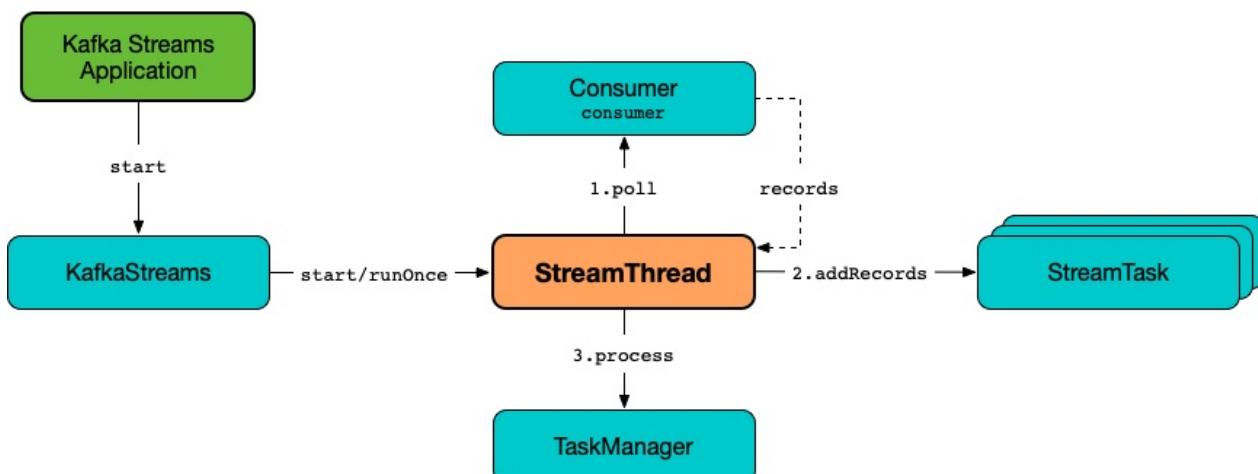


Figure 5. StreamThread and Polling Records Once And Processing Them Using Active Stream Tasks

Note

`runOnce` uses the `StreamsMetricsThreadImpl` to access `sensors` and record metrics.

Internally, `runOnce` `pollRequests` with different poll times as follows:

- `0L` when in `PARTITIONS_ASSIGNED` state
- `pollTime` when in `PARTITIONS_REVOKED`, `STARTING` or `RUNNING` state

Note

When in the other `states` (when `pollRequests` above), `runOnce` prints out the following `ERROR` message to the logs followed by throwing a `StreamsException`:

Unexpected state [state] during normal iteration

`runOnce` `advanceNowAndComputeLatency`.

With records polled, `runOnce` requests the `StreamsMetricsThreadImpl` for the `pollTimeSensor` and requests it to record the above `pollLatency` followed by adding the records polled to active stream tasks.

If in `PARTITIONS_ASSIGNED` state, `runOnce` requests the `TaskManager` to `updateNewAndRestoringTasks` and (when all stream tasks are running) changes to the `RUNNING` state.

`runOnce` `advanceNowAndComputeLatency`.

`runOnce` requests the `TaskManager` to `check out if hasActiveRunningTasks` and if so...
FIXME

In the end, `runOnce` `maybeUpdateStandbyTasks` followed by `maybeCommit`.

Note

`runOnce` is used exclusively when `StreamThread` is requested to run the main record processing loop.

Polling Records — `pollRequests` Internal Method

```
ConsumerRecords<byte[], byte[]> pollRequests(  
    Duration pollTime)
```

`pollRequests` simply requests the `Kafka Consumer` to poll record with the given `pollTime`.

In case of an `InvalidOffsetException`, `pollRequests` `resetInvalidOffsets`.

In case of a `rebalanceException`, `pollRequests` re-throws it as a `TaskMigratedException` or a `StreamsException`.

Note

`pollRequests` is used exclusively when `streamThread` is requested to `poll` records once and process them using active stream tasks.

resetInvalidOffsets Internal Method

```
void resetInvalidOffsets(  
    InvalidOffsetException e)
```

`resetInvalidOffsets` ...FIXME

Note

`resetInvalidOffsets` is used exclusively when `streamThread` is requested to `pollRequests` (and an `InvalidOffsetException` is reported).

**Attempting to Update Running StandbyTasks
— maybeUpdateStandbyTasks Internal Method**

```
void maybeUpdateStandbyTasks()
```

`maybeUpdateStandbyTasks` ...FIXME

`maybeUpdateStandbyTasks` does nothing and simply returns when `streamThread` is not in `RUNNING` state or the `TaskManager` has no `hasStandbyRunningTasks`.

Note

`maybeUpdateStandbyTasks` is used exclusively when `streamThread` is requested to `poll` records once and process them using active stream tasks.

Running Main Record Processing Loop — runLoop Internal Method

```
void runLoop()
```

`runLoop` simply requests the `Consumer` to subscribe to the `source topics` (with the custom `ConsumerRebalanceListener`) and keeps `polling` records and processing them using active stream tasks until the `isRunning` flag is off.

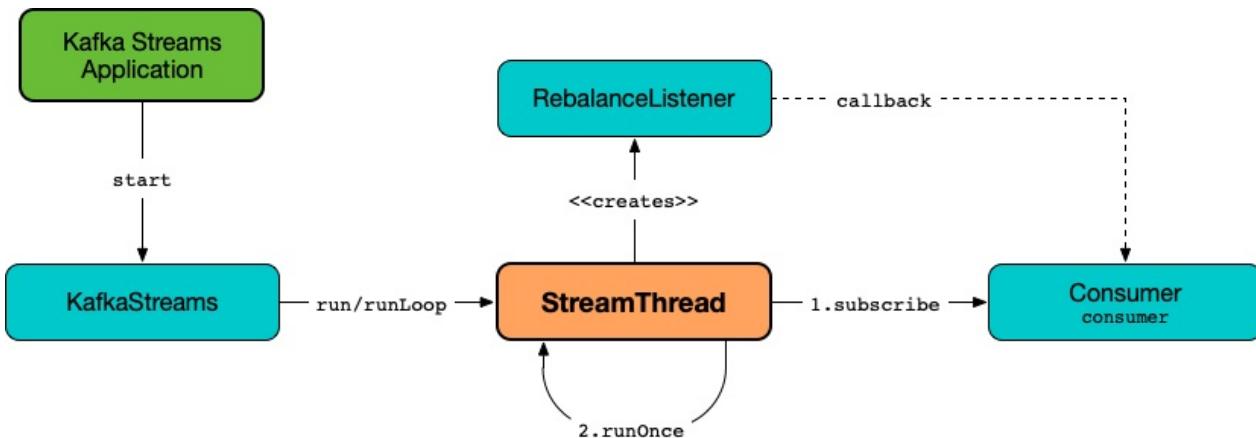


Figure 6. StreamThread and Running Main Record Processing Loop

`runLoop` requests the `Consumer` to subscribe to the `source topics` (from the `InternalTopologyBuilder`) with the custom `ConsumerRebalanceListener`.

`runLoop` then keeps polling records and processing them using active stream tasks until the `isRunning` flag is off.

In case of the `assignmentErrorCode` set to `VERSION_PROBING`, `runLoop` prints out the following INFO message to the logs followed by [enforcing a rebalance](#).

```
Version probing detected. Triggering new rebalance.
```

In case of `TaskMigratedException`, `runLoop` prints out the following WARN message to the logs followed by [enforcing a rebalance](#).

```
Detected task [taskId] that got migrated to another thread. This implies that this thread missed a rebalance and dropped out of the consumer group. Will try to rejoin the consumer group. Below is the detailed description of the task:  
[migratedTask]
```

Note	<code>runLoop</code> is used exclusively when <code>StreamThread</code> is requested to start .
------	---

Setting New State— `setState` Method

```
State setState(  
    State newState)
```

```
setState ...FIXME
```

Note	<code>setState</code> is used when...FIXME
------	--

`setRebalanceException` Internal Method

```
void setRebalanceException(
    Throwable rebalanceException)
```

`setRebalanceException ...FIXME`

Note

`setRebalanceException` is used when...FIXME

Describing Itself (Textual Representation) — `toString` Method

```
String toString() (1)
String toString(
    String indent)
```

1. Uses an empty indent

`toString` gives a text representation with "StreamsThread threadId:" and the thread name followed by the [text representation](#) of the [TaskManager](#).

`FIXME toString in action`

Checking If StreamThread Is Running — `isRunning` Method

```
boolean isRunning()
```

`isRunning` is `true` when `streamThread` is in one of the following [states](#):

- [RUNNING](#)
- [STARTING](#)
- [PARTITIONS_REVOKED](#)
- [PARTITIONS_ASSIGNED](#)

Otherwise, `isRunning` is `false`.

Note

`isRunning` is simply a pass-through variant of [State.isRunning](#).

Note	<p><code>isRunning</code> is used when:</p> <ul style="list-style-type: none"> • <code>StreamThread</code> is requested to run the main record processing loop • <code>KafkaStreams</code> is requested to close.
------	---

adminClientMetrics Method

```
Map<MetricName, Metric> adminClientMetrics()
```

`adminClientMetrics` ...FIXME

Note	<p><code>adminClientMetrics</code> is used exclusively when <code>KafkaStreams</code> is requested for the metrics.</p>
------	---

consumerMetrics Method

```
Map<MetricName, Metric> consumerMetrics()
```

`consumerMetrics` ...FIXME

Note	<p><code>consumerMetrics</code> is used when...FIXME</p>
------	--

producerMetrics Method

```
Map<MetricName, Metric> producerMetrics()
```

`producerMetrics` ...FIXME

Note	<p><code>producerMetrics</code> is used when...FIXME</p>
------	--

getConsumerClientId Static Method

```
String getConsumerClientId(
    String threadClientId)
```

`getConsumerClientId` ...FIXME

Note	<p><code>getConsumerClientId</code> is used when...FIXME</p>
------	--

getRestoreConsumerClientId Static Method

```
String getRestoreConsumerClientId(  
    String threadClientId)
```

getRestoreConsumerClientId ...FIXME

Note

getRestoreConsumerClientId is used when...FIXME

getSharedAdminClientId Static Method

```
String getSharedAdminClientId(  
    String clientId)
```

getSharedAdminClientId ...FIXME

Note

getSharedAdminClientId is used when...FIXME

tasks Method

```
Map<TaskId, StreamTask> tasks()
```

tasks ...FIXME

Note

tasks is used when...FIXME

getTaskProducerClientId Internal Static Method

```
String getTaskProducerClientId(  
    String threadClientId,  
    TaskId taskId)
```

getTaskProducerClientId ...FIXME

Note

getTaskProducerClientId is used when...FIXME

getThreadProducerClientId Internal Static Method

```
String getThreadProducerClientId(
    String threadClientId)
```

getThreadProducerClientId ...FIXME

Note

getThreadProducerClientId is used when...FIXME

Adding Records to Active Stream Tasks — addRecordsToTasks Internal Method

```
void addRecordsToTasks(
    ConsumerRecords<byte[], byte[]> records)
```

For every **partition** of the input **records** `addRecordsToTasks` requests the **TaskManager** for the **active stream processor task** responsible for the partition.

Note

The input records may (and often will) be from different partitions or even topics. Unless you use as many `StreamThread` instances as there are partitions (among the source topics), `addRecordsToTasks` will be given records from many partitions.

With the **StreamTask**, `addRecordsToTasks` requests the input mixed-partition **ConsumerRecords** for the **records for the given partition only** and then requests the **StreamTask** to **buffer the new records** (for the partition).

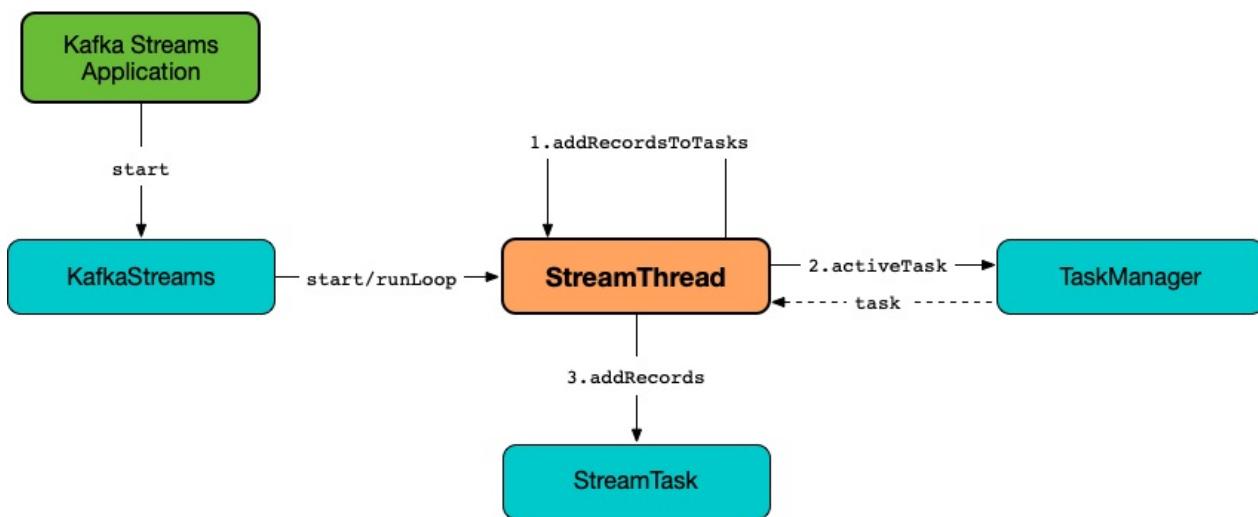


Figure 7. StreamThread and Adding Records to Active Stream Tasks

Note

ConsumerRecords is a container that holds the list of **ConsumerRecord** per partition for a particular topic. There is one **ConsumerRecord** list for every topic partition returned by a `Consumer.poll(long)` operation.

Note

`addRecordsToTasks` is used exclusively when `StreamThread` is requested to [poll records once and process them using active stream tasks](#).

Enforcing Rebalance — `enforceRebalance` Internal Method

```
void enforceRebalance()
```

`enforceRebalance` ...FIXME

Note

`enforceRebalance` is used when...FIXME

Committing All Active and Standby Tasks (When Commit Interval Elapsed) — `maybeCommit` Method

```
boolean maybeCommit()
```

`maybeCommit` commits all tasks (owned by this [TaskManager](#)) if the [commit interval](#) has elapsed (i.e. the commit interval is non-negative and the time since the [last commit](#) is long gone).

Internally, `maybeCommit` prints out the following TRACE message to the logs:

```
Committing all active tasks [activeTaskIds] and standby tasks [standbyTaskIds] since [time]ms has elapsed (commit interval is [commitTimeMs]ms)
```

`maybeCommit` requests the [TaskManager](#) to [commitAll](#).

Only if there are still running active and standby tasks, `maybeCommit` does the following:

1. Requests the [StreamsMetricsThreadImpl](#) for the [commitTimeSensor](#) and records the commit time (as the latency of committing all the tasks by their number)
2. Requests the [TaskManager](#) to [maybePurgeCommittedRecords](#)

`maybeCommit` prints out the following DEBUG message to the logs:

```
Committed all active tasks [activeTaskIds] and standby tasks [standbyTaskIds] in [duration]ms
```

`maybeCommit` updates the [lastCommitMs](#) internal counter with the input `now` time.

`maybeCommit` turns the [processStandbyRecords](#) flag on.

Note

`maybeCommit` is used exclusively when `StreamThread` is requested to [poll records once and process them using active stream tasks](#).

Attempting to Punctuate (Running Stream Tasks) — `maybePunctuate` Internal Method

```
boolean maybePunctuate()
```

`maybePunctuate` requests the [TaskManager](#) to punctuate stream tasks.

If the punctuate returned a positive number (greater than `0`), `maybePunctuate` [advanceNowAndComputeLatency](#) and requests the [StreamsMetricsThreadImpl](#) for the [punctuateTimeSensor](#) to record the punctuate time.

In the end, `maybePunctuate` returns whether the punctuate returned a positive number (`true`) or not (`false`).

Note

`maybePunctuate` is used exclusively when `StreamThread` is requested to [poll records once and process them using active stream tasks](#).

addToResetList Internal Method

```
void addToResetList(
    TopicPartition partition,
    Set<TopicPartition> partitions,
    String logMessage,
    String resetPolicy,
    Set<String> loggedTopics)
```

`addToResetList` ...FIXME

Note

`addToResetList` is used when `StreamThread` ...FIXME

Computing Latency — `advanceNowAndComputeLatency` Internal Method

```
long advanceNowAndComputeLatency()
```

`advanceNowAndComputeLatency` updates (*advances*) the "now" timestamp to be the current timestamp and returns the timestamp difference (*latency*).

Note

`advanceNowAndComputeLatency` is used when `StreamThread` is requested to `poll records once and process them using active stream tasks`, `maybePunctuate`, `maybeCommit`, and `maybeUpdateStandbyTasks`.

clearStandbyRecords Internal Method

```
void clearStandbyRecords()
```

`clearStandbyRecords` ...FIXME

Note

`clearStandbyRecords` is used when `StreamThread` ...FIXME

completeShutdown Internal Method

```
void completeShutdown(
    boolean cleanRun)
```

`completeShutdown` ...FIXME

Note

`completeShutdown` is used when `StreamThread` is requested to `run` and `shutdown`.

updateThreadMetadata Internal Method

```
void updateThreadMetadata(
    Map<TaskId, StreamTask> activeTasks,
    Map<TaskId, StandbyTask> standbyTasks)
StreamThread updateThreadMetadata(
    String adminClientId)
```

`updateThreadMetadata` ...FIXME

Note

`updateThreadMetadata` is used when `StreamThread` ...FIXME

Internal Properties

Name	Description
<code>builder</code>	<code>InternalTopologyBuilder</code>

lastCommitMs	Time of the last commit		
numIterations	<p>Number of iterations when the TaskManager is requested to process records by running stream tasks (one record per task) (while <code>StreamThread</code> is polling records once and processing them using active stream tasks)</p> <p>Default: <code>1</code></p> <p>Incremented while polling records once and processing them using active stream tasks</p> <p>Decrement by half while polling records once and processing them using active stream tasks</p>		
processStandbyRecords	<p>Flag to control whether to maybeUpdateStandbyTasks after maybeCommit</p> <p>Default: <code>false</code></p> <p>Turned off (<code>false</code>) in maybeUpdateStandbyTasks (after requesting the <code>StandbyTasks</code> to update)</p> <p>Turned on (<code>true</code>) when attempting to commit (and the time to commit has come per <code>commit.interval.ms</code> configuration property)</p>		
now	"now" timestamp		
rebalanceListener	<p>RebalanceListener</p> <ul style="list-style-type: none"> Used exclusively when <code>StreamThread</code> is requested to run the main record processing loop (and requests the Kafka Consumer to subscribe to get dynamically assigned partitions of topics matching specified pattern) <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 2px;">Note</td> <td style="padding: 2px;"><code>StreamThread</code> requests InternalTopologyBuilder for the source topic pattern to subscribe to.</td> </tr> </table> </div>	Note	<code>StreamThread</code> requests InternalTopologyBuilder for the source topic pattern to subscribe to.
Note	<code>StreamThread</code> requests InternalTopologyBuilder for the source topic pattern to subscribe to.		
standbyRecords			
stateListener	<p>StateListener</p> <p>Used when <code>StreamThread</code> is requested to change a state</p> <p>Set when <code>KafkaStreams</code> is created</p> <p>Reset (<code>null</code>) when:</p> <ul style="list-style-type: none"> <code>KafkaStreams</code> is requested to close <code>RebalanceListener</code> is requested to handle a partition assignment (and there was <code>INCOMPLETE_SOURCE_TOPIC_METADATA</code> error) 		

timerStartedMs	The timestamp when the timer has started

RebalanceListener — Kafka ConsumerRebalanceListener for Partition Assignment Among Processor Tasks

`RebalanceListener` is a Apache Kafka [ConsumerRebalanceListener](#) callback interface that listens to changes to the partitions assigned to a single `stream processor thread` (aka `StreamThread`).

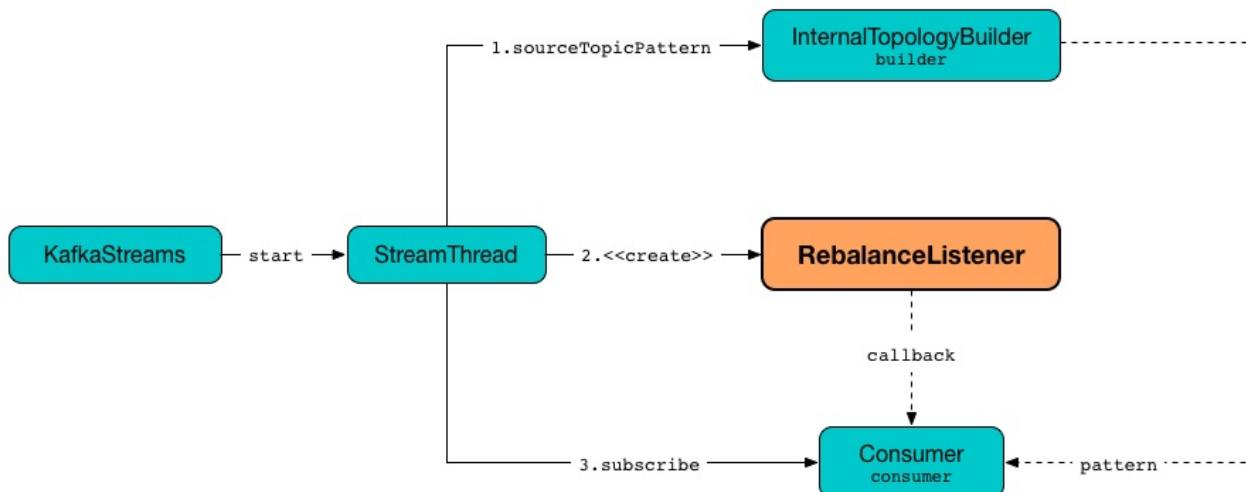


Figure 1. RebalanceListener, StreamThread and KafkaStreams

Note

The number of stream processor threads per `KafkaStreams` instance is controlled by `num.stream.threads` configuration property with the default being 1 thread.

From the documentation of

[org.apache.kafka.clients.consumer.ConsumerRebalanceListener](#):

`ConsumerRebalanceListener` is a callback interface that the user can implement to trigger custom actions when the set of partitions assigned to the consumer changes.

When Kafka is managing the group membership, a partition re-assignment will be triggered any time the members of the group change or the subscription of the members changes. This can occur when processes die, new process instances are added or old instances come back to life after failure. Rebalances can also be triggered by changes affecting the subscribed topics (e.g. when the number of partitions is administratively adjusted).

It is guaranteed that all consumer processes will invoke `onPartitionsRevoked` prior to any process invoking `onPartitionsAssigned` .

`RebalanceListener` is [created](#) exclusively when `StreamThread` is [created](#).

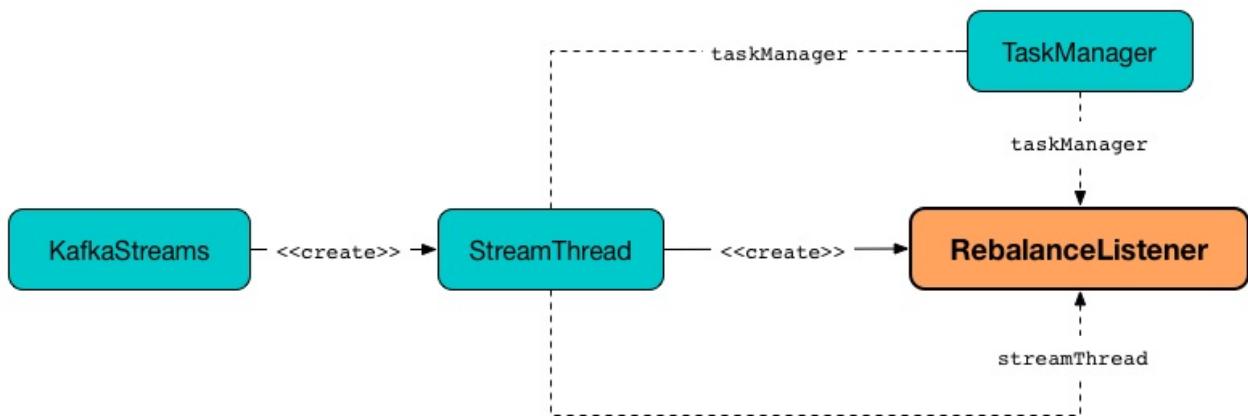


Figure 2. Creating RebalanceListener

At the completion of a successful partition re-assignment (i.e. `onPartitionsAssigned` event) `RebalanceListener` requests `TaskManager` to creating processor tasks for assigned topic partitions.

At the start of a rebalance operation (i.e. `onPartitionsRevoked` event) `RebalanceListener` requests `TaskManager` to suspending all stream tasks and state.

`RebalanceListener` uses `StreamThread` for the following:

1. Changing the state (of the stream processor thread) at `partition assignment` and `partition revocation`
2. Clearing standby records at the end of `partition revocation`
3. Notifying about the error caught at `partition assignment` and `partition revocation`

`RebalanceListener` uses the `logger` from the owning `StreamThread` for logging.

Enable `DEBUG` logging level for
`org.apache.kafka.streams.processor.internals.StreamThread` logger to see what happens inside.

Tip Add the following line to `log4j.properties` :

```
log4j.logger.org.apache.kafka.streams.processor.internals.StreamThread=DEBUG
```

Refer to [Application Logging Using log4j](#).

Handling Partition Assignment

— `onPartitionsAssigned` Handler Method

```
void onPartitionsAssigned(
    Collection<TopicPartition> assignment)
```

Note

`onPartitionsAssigned` is part of `ConsumerRebalanceListener` Contract in Apache Kafka to...FIXME.

Internally, `onPartitionsAssigned` first prints out the following DEBUG message to the logs:

```
at state [state]: partitions [assignment] assigned at the end of consumer rebalance.  
current suspended active tasks: [taskIds]  
current suspended standby tasks: [taskIds]
```

`onPartitionsAssigned` requests `StreamThread` to set the state to `PARTITIONS_ASSIGNED` followed by requesting `TaskManager` for processor tasks for assigned topic partitions.

In the end, `onPartitionsAssigned` prints out the following INFO message to the logs:

```
partition assignment took [duration] ms.  
current active tasks: [activeTaskIds]  
current standby tasks: [standbyTaskIds]  
previous active tasks: [prevActiveTaskIds]
```

`onPartitionsAssigned` does nothing (i.e. prints the messages to the logs) when the state transition was invalid.

Handling Partition Revocation — `onPartitionsRevoked` Handler Method

```
void onPartitionsRevoked(  
    Collection<TopicPartition> assignment)
```

Note

`onPartitionsRevoked` is part of `ConsumerRebalanceListener` Contract in Apache Kafka to...FIXME.

`onPartitionsRevoked` ...FIXME

Creating RebalanceListener Instance

`RebalanceListener` takes the following when created:

- `Time`
- `TaskManager`
- `StreamThread`
- `Logger`

StateListener

StateListener is...FIXME

StreamsMetadataState

`StreamsMetadataState` manages the [Kafka cluster metadata](#) for a [KafkaStreams](#) instance.

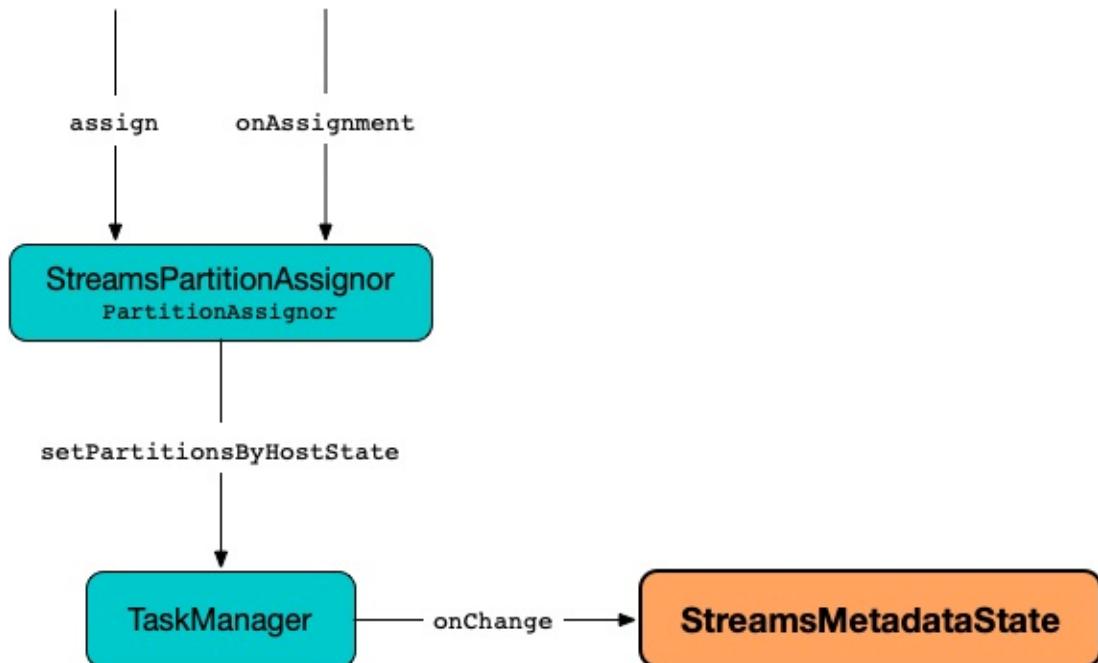


Figure 1. `StreamsMetadataState` and Cluster Metadata Changes (from `StreamsPartitionAssignor`)

`StreamsMetadataState` comes with [getAllMetadata](#) method that returns `StreamsMetadata`s for all streams client instances in a [multi-instance Kafka Streams application](#).

`StreamsMetadataState` is [created](#) exclusively for the `TaskManager` of the `stream processor threads` of a [KafkaStreams](#) instance.

`StreamsMetadata` represents the metadata of a [KafkaStreams](#) stream processing node (process) in a Kafka Streams application:

- A user-defined endpoint (as a pair of a host and a port)
- Names of state stores
- Kafka [TopicPartitions](#) (that this instance is assigned to)

Table 1. StreamsMetadataState's Internal Properties (e.g. Registries, Counters and Flags)

Name	Description
allMetadata	<p>StreamsMetadata for all instances in a multi-instance Kafka Streams application</p> <p>StreamsMetadata are added exclusively when StreamsMetadataState is requested to rebuildMetadata (after all entries are first removed)</p> <p>allMetadata is accessed when StreamsMetadataState is requested for a textual representation, getAllMetadataForStore, getMetadataWithKey, getStreamsMetadataForKey</p>
clusterMetadata	<p>Cluster metadata, i.e. a subset of the nodes, topics, and partitions in the Kafka cluster (as a Kafka Cluster) that is given when StreamsMetadataState is requested to handle cluster metadata changes</p> <p>Used when:</p> <ul style="list-style-type: none"> getMetadataWithKey isInitialized toString
globalStores	<p>Names of the global StateStores</p> <p>Used when...FIXME</p>
myMetadata	StreamsMetadata of the current node

Creating StreamsMetadataState Instance

StreamsMetadataState takes the following when created:

- InternalTopologyBuilder
- HostInfo (with the host and port)

StreamsMetadataState initializes the internal registries and counters.

getAllMetadataForStore Method

```
Collection<StreamsMetadata> getAllMetadataForStore(final String storeName)
```

getAllMetadataForStore ...FIXME

Note

`getAllMetadataForStore` is used exclusively when `KafkaStreams` is requested to `allMetadataForStore`.

Handling Cluster Metadata Changes — `onChange` Method

```
void onChange(
    final Map<HostInfo, Set<TopicPartition>> currentState,
    final Cluster clusterMetadata)
```

`onChange` simply records the given `cluster` as the `Cluster` and `rebuildMetadata` (with the given current state of Kafka `TopicPartitions` per `HostInfo`).

Note

`onChange` is used exclusively when `TaskManager` is requested to `notify the StreamsMetadataState about cluster metadata changes`.

Rebuilding Internal StreamsMetadata — `rebuildMetadata` Internal Method

```
void rebuildMetadata(final Map<HostInfo, Set<TopicPartition>> currentState)
```

`rebuildMetadata` firstly clears the `allMetadata` internal registry.

`rebuildMetadata` does nothing else and returns if the given `currentState` is empty.

`rebuildMetadata` requests the `InternalTopologyBuilder` for `stateStoreNameToSourceTopics`.

For every entry in the input `currentState`, `rebuildMetadata` ...FIXME

Note

`rebuildMetadata` is used exclusively when `StreamsMetadataState` is requested to handle cluster metadata changes.

Describing Itself (Textual Representation) — `toString` Method

```
String toString() (1)
String toString(final String indent)
```

1. Uses an empty indent

`toString` ...FIXME

Note

`toString` is used when `TaskManager` is requested to `describe itself`.

isInitialized Internal Method

```
boolean isInitialized()
```

`isInitialized` ...FIXME

Note	<code>isInitialized</code> is used when...FIXME
------	---

getMetadataWithKey Method

```
StreamsMetadata getMetadataWithKey(
    final String storeName,
    final K key,
    final Serializer<K> keySerializer)
StreamsMetadata getMetadataWithKey(
    final String storeName,
    final K key,
    final StreamPartitioner<? super K, ?> partitioner)
```

`getMetadataWithKey` ...FIXME

Note	<code>getMetadataWithKey</code> is used exclusively when <code>KafkaStreams</code> is requested for <code>metadataForKey</code> .
------	---

hasPartitionsForAnyTopics Internal Method

```
boolean hasPartitionsForAnyTopics(
    final List<String> topicNames,
    final Set<TopicPartition> partitionForHost)
```

`hasPartitionsForAnyTopics` is enabled (returns `true`) when the input `topicNames` contain any of the topics in the input `partitionForHost`. Otherwise, `hasPartitionsForAnyTopics` is disabled (`false`).

Note	<code>hasPartitionsForAnyTopics</code> is used exclusively when <code>StreamsMetadataState</code> is requested to <code>rebuildMetadata</code> .
------	--

getAllMetadata Method

```
Collection<StreamsMetadata> getAllMetadata()
```

`getAllMetadata` simply returns the [allMetadata](#) internal registry.

Note

`getAllMetadata` is used exclusively when `KafkaStreams` is requested for StreamsMetadatas for all KafkaStreams instances (in a multi-instance Kafka Streams application).

StreamsPartitionAssignor — Dynamic Partition Assignment Strategy

`StreamsPartitionAssignor` is a custom `PartitionAssignor` (from the Kafka Consumer API) that is used to [assign partitions dynamically](#) to the stream processor threads of a Kafka Streams application (identified by the required `StreamsConfig.APPLICATION_ID_CONFIG` configuration property with the number of stream processor threads per `StreamsConfig.NUM_STREAM_THREADS_CONFIG` configuration property).

Tip

`PartitionAssignor` is used for a dynamic partition assignment and distributing partition ownership across the members of a consumer group.

Read up [Kafka Client-side Assignment Proposal](#) on the group management in Apache Kafka's Consumer API.

Read up on `PartitionAssignor` Contract in [The Internals of Apache Kafka](#).

From [Kafka Client-side Assignment Proposal](#) (with extra formatting of mine):

To support client-side assignment, we propose to split the group management protocol into two phases: **group membership** and **state synchronization**.

Group Membership phase is used to set the active members of the group and to elect a group leader.

State Synchronization phase is used to enable the group leader to synchronize member state in the group (in other words to assign each member's state).

From the perspective of the consumer, group membership phase is used to collect member subscriptions, while state synchronization phase is used to propagate partition assignments.

The elected leader in the join group phase is responsible for setting the assignments for the whole group.

Given the above, there will be just one elected `StreamsPartitionAssignor` among the members of a Kafka Streams application (so you should see the [log messages](#) from just a single thread).

`StreamsPartitionAssignor` (as a fully-qualified class name) is registered under **partition.assignment.strategy** (`ConsumerConfig.PARTITION_ASSIGNMENT_STRATEGY_CONFIG`) configuration property when `StreamsConfig` is requested for the [configuration for the main Kafka Consumer](#) (when `StreamThread` is requested to [create a new StreamThread instance](#) and requests the `KafkaClientSupplier` for a [Kafka Consumer](#)).

`StreamsPartitionAssignor` is a Kafka Configurable that is instantiated by reflection with configuration parameters. That is when `StreamsPartitionAssignor` finds the current TaskManager (using `TASK_MANAGER_FOR_PARTITION_ASSIGNOR` internal property set when the `streamThread` is created).

`StreamsPartitionAssignor` uses `stream-thread [client.id]` for the `logPrefix` (that uses `client.id` configuration property).

`StreamsPartitionAssignor` uses `num.standby.replicas` configuration property for the number of standby replicas per processing task when performing partition assignment (and requesting `StickyTaskAssignor` to assign tasks).

	<p>Enable ALL logging level for <code>org.apache.kafka.streams.processor.internals.StreamsPartitionAssignor</code> logger to see what happens inside.</p> <p>Tip Add the following line to <code>log4j.properties</code> :</p> <pre>log4j.logger.org.apache.kafka.streams.processor.internals.StreamsPartitionAssignor=ALL</pre> <p>Refer to Application Logging Using log4j.</p>
--	--

subscription Method

`Subscription subscription(Set<String> topics)`

Note	<code>subscription</code> is part of Kafka Consumer's <code>PartitionAssignor</code> contract to return a local member's subscription for the topics subscribed to (e.g. using <code>KafkaConsumer.subscribe</code>).
-------------	--

`subscription` requests the TaskManager for the previously active tasks.

`subscription` requests the TaskManager for the cached task IDs (aka *standbyTasks*) and removes all of the previously active tasks.

`subscription` requests the TaskManager to `updateSubscriptionsFromMetadata` with the given `topics`.

In the end, `subscription` creates a `SubscriptionInfo` (with the `usedSubscriptionMetadataVersion`, the process ID from the TaskManager, previously active and standby tasks and the `userEndPoint`), requests the `SubscriptionInfo` for the encoded version and creates a `Subscription` (with the topics and the encoded version).

Handling Partition Assignment From Group Leader

— `onAssignment` Method

```
void onAssignment(Assignment assignment)
```

Note

`onAssignment` is part of Kafka Consumer's `PartitionAssignor` contract for a group member to handle assignment from the leader.

`onAssignment` is executed when a group member has successfully joined a group.

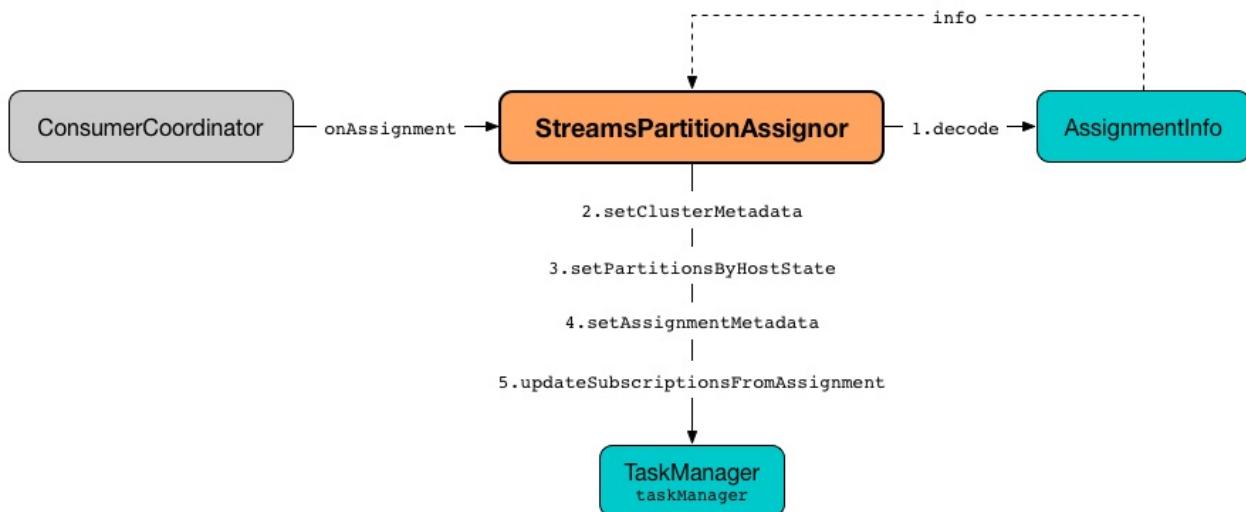


Figure 1. StreamsPartitionAssignor.onAssignment

`onAssignment` takes the partitions (from the given `assignment`) and sorts them by topic and partition.

`onAssignment` requests `AssignmentInfo` helper object to `decode` the additional metadata (i.e. the `userData` from the input `assignment`).

Caution

FIXME Finish me

In the end, `onAssignment` requests the `TaskManager` for the following:

1. Setting cluster metadata
2. Notifying `StreamsMetadataState` about Cluster Metadata Changes
3. Setting task assignment metadata with active and standby tasks
4. `updateSubscriptionsFromAssignment` with the assigned partitions

`onAssignment` reports an `TaskAssignmentException` if the numbers of partitions and active tasks are not equal.

```
Number of assigned partitions [partitions] is not equal to the number of active taskIds [activeTasks], assignmentInfo=[info]
```

Initialization (Configuring Partition Assignor)

— configure Method

```
void configure(final Map<String, ?> configs)
```

Note `configure` is part of Kafka Consumer's `Configurable` contract for classes that are instantiated by reflection and need to take configuration parameters.

`configure` creates a `InternalStreamsConfig` for the given `configs`.

`configure` initializes the `logPrefix` (with `CLIENT_ID_CONFIG` configuration property).

`configure` uses `UPGRADE_FROM_CONFIG` configuration property for...FIXME

`configure` sets the `TaskManager` per `TASK_MANAGER_FOR_PARTITION_ASSIGNOR` internal property (in the given `configs`). If not found or of a different type than `TaskManager`, `configure` throws a `KafkaException`:

```
TaskManager is not specified
```

```
[className] is not an instance of org.apache.kafka.streams.processor.internals.TaskManager
```

`configure` sets the `assignmentErrorCode` per `ASSIGNMENT_ERROR_CODE` internal property (in the given `configs`). If not found or of a different type than `AtomicInteger`, `configure` throws a `KafkaException`:

```
assignmentErrorCode is not specified
```

```
[className] is not an instance of java.util.concurrent.atomic.AtomicInteger
```

`configure` sets the `numStandbyReplicas` per `NUM_STANDBY_REPLICAS_CONFIG` configuration property (in the given `configs`).

`configure` sets the `PartitionGrouper` per `PARTITION_GROUPER_CLASS_CONFIG` configuration property (in the given `configs`).

`configure` sets the `userEndPoint` if `APPLICATION_SERVER_CONFIG` configuration property was defined.

`configure` creates a new `InternalTopicManager` (with the Kafka `AdminClient` of the `TaskManager` and the `InternalStreamsConfig` created earlier).

`configure` creates a new `CopartitionedTopicsValidator` (with the `logPrefix`).

prepareTopic Internal Method

```
void prepareTopic(final Map<String, InternalTopicMetadata> topicPartitions)
```

`prepareTopic` prints out the following DEBUG message to the logs:

```
Starting to validate internal topics [topicPartitions] in partition assignor.
```

For every `InternalTopicMetadata` (in the given `topicPartitions` collection), `prepareTopic` makes sure that the number of partition is defined, i.e. is `0` or more. If not, `prepareTopic` throws a `StreamsException`:

```
stream-thread [client.id] Topic [[name]] number of partitions not defined
```

In the end, `prepareTopic` requests the `InternalTopicManager` to `makeReady` the topics and prints out the following DEBUG message to the logs:

```
Completed validating internal topics [topicPartitions] in partition assignor.
```

Note	<code>prepareTopic</code> is used exclusively when <code>StreamsPartitionAssignor</code> is requested to <code>perform partition assignment</code> (and finds repartition source and change log topics).
------	--

Performing Group Assignment (Assigning Tasks To Consumer Clients) — `assign` Method

```
Map<String, Assignment> assign(
    Cluster metadata,
    Map<String, Subscription> subscriptions)
```

Note

`assign` is part of Kafka Consumer's `PartitionAssignor` contract to perform **group assignment** given the member subscriptions and current cluster metadata.

Note

The input `Map<String, Subscription>` contains bindings of a consumer ID and the Kafka `Subscription` with a list of topics (their names) and a user data encoded (as a `java.nio.ByteBuffer`).

`assign` constructs the client metadata (as `Map<UUID, ClientMetadata>`) from the decoded `subscription info` (from the user data).

1. `assign` takes consumer IDs with subscriptions (from `subscriptions`).
2. `assign` requests `SubscriptionInfo` to `decode` the user data of the subscription (aka `metadata`) and makes sure that the version is supported, i.e. up to 2 currently.
3. `assign` finds the client metadata (by the process ID) and creates one if not available.
4. `assign` requests the `ClientMetadata` to `addConsumer`.

`assign` prints out the following INFO message to the logs only if the minimal version received is smaller than the latest supported version (i.e. `4`).

```
Downgrading metadata to version [minReceivedMetadataVersion]. Latest supported version is 4.
```

`assign` prints out the following DEBUG message to the logs:

```
Constructed client metadata [clientsMetadata] from the member subscriptions.
```

Caution

FIXME

`assign` reports a `IllegalStateException` when the subscription version is unsupported.

```
Unknown metadata version: [usedVersion]; latest supported version: " + SubscriptionInfo.LATEST_SUPPORTED_VERSION
```

Creating Empty Partition Assignment with Error Code — `errorAssignment` Method

```
Map<String, Assignment> errorAssignment(
    final Map<UUID, ClientMetadata> clientsMetadata,
    final String topic,
    final int errorCode)
```

`errorAssignment` prints out the following ERROR message to the logs:

```
[topic] is unknown yet during rebalance, please make sure they have been pre-created before starting the Streams application.
```

In the end, `errorAssignment` returns a new map of client IDs with empty `Assignment` per `consumers` in the [ClientMetadata](#) from the input `clientsMetadata`.

Note

`errorAssignment` is used exclusively when `StreamsPartitionAssignor` is requested to [perform partition assignment](#).

versionProbingAssignment Internal Method

```
Map<String, Assignment> versionProbingAssignment(
    final Map<UUID, ClientMetadata> clientsMetadata,
    final Map<TaskId, Set<TopicPartition>> partitionsForTask,
    final Map<HostInfo, Set<TopicPartition>> partitionsByHostState,
    final Set<String> futureConsumers,
    final int minUserMetadataVersion)
```

`versionProbingAssignment` ...FIXME

Note

`versionProbingAssignment` is used exclusively when `StreamsPartitionAssignor` is requested to [perform partition assignment](#).

computeNewAssignment Internal Method

```
Map<String, Assignment> computeNewAssignment(
    final Map<UUID, ClientMetadata> clientsMetadata,
    final Map<TaskId, Set<TopicPartition>> partitionsForTask,
    final Map<HostInfo, Set<TopicPartition>> partitionsByHostState,
    final int minUserMetadataVersion)
```

`computeNewAssignment` ...FIXME

Note

`computeNewAssignment` is used exclusively when `StreamsPartitionAssignor` is requested to [perform partition assignment](#).

processVersionOneAssignment Internal Method

```
void processVersionOneAssignment(  
    AssignmentInfo info,  
    List<TopicPartition> partitions,  
    Map<TaskId, Set<TopicPartition>> activeTasks)
```

processVersionOneAssignment ...FIXME

Note	processVersionOneAssignment is used when StreamsPartitionAssignor is requested to handle partition assignment from a group leader (for version 1) and process partition assignment (version 2).
------	---

processVersionTwoAssignment Internal Method

```
void processVersionTwoAssignment(  
    AssignmentInfo info,  
    List<TopicPartition> partitions,  
    Map<TaskId, Set<TopicPartition>> activeTasks,  
    Map<TopicPartition, PartitionInfo> topicToPartitionInfo)
```

processVersionTwoAssignment ...FIXME

Note	processVersionTwoAssignment is used when...FIXME
------	--

Internal Properties

Name	Description		
assignmentErrorCode	<code>java.util.concurrent.atomic.AtomicInteger</code>		
copartitionedTopicsValidator	<code>CopartitionedTopicsValidator</code>		
internalTopicManager	<p><code>InternalTopicManager</code></p> <p>Initialized when <code>StreamsPartitionAssignor</code> is requested to configure itself</p> <p>Used exclusively when <code>StreamsPartitionAssignor</code> is requested to prepareTopic</p>		
partitionGrouper	<code>PartitionGrouper</code>		
supportedVersions	<p>Supported versions</p> <p>All bindings removed (cleared) and then populated in assign</p> <table border="1"> <tr> <td>Note</td><td><code>supportedVersions</code> does not seem to be used except to hold the entries.</td></tr> </table>	Note	<code>supportedVersions</code> does not seem to be used except to hold the entries.
Note	<code>supportedVersions</code> does not seem to be used except to hold the entries.		
taskManager	<code>TaskManager</code>		
usedSubscriptionMetadataVersion			
userEndPoint	<p>User-defined endpoint in the format of <code>host:port</code> as configured by <code>application.server</code> configuration property (default: empty).</p> <p>Used when <code>streamsPartitionAssignor</code> is requested to subscription (and create a <code>SubscriptionInfo</code>)</p>		

InternalTopicManager

`InternalTopicManager` is created exclusively for the `StreamsPartitionAssignor` so it can make repartition and state changelog internal topics ready.

`InternalTopicManager` uses a Kafka `AdminClient` to create and describe topics.

`InternalTopicManager` uses stream-thread [threadName] for the `logContext`.

Table 1. InternalTopicManager's Internal Properties (e.g. Registries, Counters and Flags)

Name	Description
<code>defaultTopicConfigs</code>	Default topic configurations (<code>Map<String, String></code>) that is filled out with new entries (with topic. prefix) when <code>InternalTopicManager</code> is created
<code>log</code>	Logger
<code>replicationFactor</code>	Value of <code>replication.factor</code> configuration property
<code>retries</code>	Value of AdminClient's <code>retries</code> configuration property (to resend any request that fails with a potentially transient error)
<code>windowChangeLogAdditionalRetention</code>	Value of <code>windowstore.changelog.additional.retention.ms</code> configuration property
Tip	<p>Enable any of <code>DEBUG</code> logging level for <code>org.apache.kafka.streams.processor.internals.InternalTopicManager</code> logger to see what happens inside.</p> <p>Add the following line to <code>log4j.properties</code>:</p> <pre>log4j.logger.org.apache.kafka.streams.processor.internals.InternalTopicManager=DEBUG</pre> <p>Refer to Application Logging Using log4j.</p>

Creating InternalTopicManager Instance

`InternalTopicManager` takes the following when created:

- Kafka `AdminClient`

- StreamsConfig

`InternalTopicManager` initializes the [internal registries and counters](#).

While being created, `InternalTopicManager` prints out the following DEBUG message to the logs:

```
Configs:
retries = [retries]
replication.factor = [replicationFactor]
windowstore.changelog.additional.retention.ms = [windowChangeLogAdditionalRetention]
```

In the end, `InternalTopicManager` requests the [StreamsConfig](#) for [originalsWithPrefix](#) with topic. prefix and adds every non-empty configuration to the [defaultTopicConfigs](#).

makeReady Method

```
void makeReady(final Map<String, InternalTopicConfig> topics)
```

`makeReady` ...FIXME

Note	<code>makeReady</code> is used exclusively when <code>StreamsPartitionAssignor</code> is requested to prepareTopic (when requested to perform group assignment).
------	---

getNumPartitions Method

```
Map<String, Integer> getNumPartitions(final Set<String> topics)
```

`getNumPartitions` ...FIXME

Note	<code>getNumPartitions</code> is used when <code>InternalTopicManager</code> is requested to makeReady .
------	--

AssignmentInfo

AssignmentInfo is...FIXME

Decoding User Data — decode Static Method

```
static AssignmentInfo decode(ByteBuffer data)
```

decode ...FIXME

Note

decode is used exclusively when StreamsPartitionAssignor is requested to handle partition assignment from the group leader.

decodeVersionOneData Internal Static Method

```
static void decodeVersionOneData(  
    final AssignmentInfo assignmentInfo,  
    final DataInputStream in) throws IOException
```

decodeVersionOneData ...FIXME

Note

decodeVersionOneData is used exclusively when AssignmentInfo is requested to decode the user data.

encodeActiveAndStandbyTaskAssignment Internal Method

```
void encodeActiveAndStandbyTaskAssignment(final DataOutputStream out)
```

encodeActiveAndStandbyTaskAssignment ...FIXME

Note

encodeActiveAndStandbyTaskAssignment is used when AssignmentInfo is requested to encodeVersionOne, encodeVersionTwo, encodeVersionThree, and encodeVersionFour.

decodeActiveTasks Internal Factory Method

```
static void decodeActiveTasks(
    final AssignmentInfo assignmentInfo,
    final DataInputStream in)
```

decodeActiveTasks ...FIXME

Note

decodeActiveTasks is used when...FIXME

decodeStandbyTasks Internal Factory Method

```
static void decodeStandbyTasks(
    final AssignmentInfo assignmentInfo,
    final DataInputStream in)
```

decodeStandbyTasks ...FIXME

Note

decodeStandbyTasks is used when...FIXME

encodeVersionFour Internal Method

```
void encodeVersionFour(final DataOutputStream out)
```

encodeVersionFour ...FIXME

Note

encodeVersionFour is used exclusively when AssignmentInfo is requested to encode (for the metadata version as 4).

encodeVersionThree Internal Method

```
void encodeVersionThree(final DataOutputStream out)
```

encodeVersionThree ...FIXME

Note

encodeVersionThree is used when...FIXME

encodeVersionTwo Internal Method

```
void encodeVersionTwo(final DataOutputStream out)
```

encodeVersionTwo ...FIXME

Note

encodeVersionTwo is used when...FIXME

encodeVersionOne Internal Method

```
void encodeVersionOne(final DataOutputStream out)
```

encodeVersionOne ...FIXME

Note

encodeVersionOne is used when...FIXME

encode Method

```
ByteBuffer encode()
```

encode ...FIXME

Note

encode is used exclusively when StreamsPartitionAssignor is requested to errorAssignment, computeNewAssignment, and versionProbingAssignment.

SubscriptionInfo

SubscriptionInfo is...FIXME

SubscriptionInfo uses 4 for the latest supported subscription version.

decode Method

```
SubscriptionInfo decode(final ByteBuffer data)
```

decode ...FIXME

Note	decode is used when...FIXME
------	-----------------------------

encodeTasks Method

```
void encodeTasks(  
    final ByteBuffer buf,  
    final Collection<TaskId> taskIds)
```

encodeTasks ...FIXME

Note	encodeTasks is used when SubscriptionInfo is requested to encodeVersionOne, encodeVersionTwo, encodeVersionThree, and encodeVersionFour.
------	--

decodeTasks Factory Method

```
static void decodeTasks(  
    final SubscriptionInfo subscriptionInfo,  
    final ByteBuffer data)
```

decodeTasks ...FIXME

Note	decodeTasks is used when...FIXME
------	----------------------------------

encodeVersionFour Internal Method

```
ByteBuffer encodeVersionFour()
```

encodeVersionFour ...FIXME

Note	encodeVersionFour is used exclusively when SubscriptionInfo is requested to encode (for the metadata version as 4).
------	--

encodeVersionThree Internal Method

```
ByteBuffer encodeVersionThree()
```

encodeVersionThree ...FIXME

Note	encodeVersionThree is used when...FIXME
------	---

encodeVersionTwo Internal Method

```
ByteBuffer encodeVersionTwo()
```

encodeVersionTwo ...FIXME

Note	encodeVersionTwo is used when...FIXME
------	---------------------------------------

encodeVersionOne Internal Method

```
ByteBuffer encodeVersionOne()
```

encodeVersionOne ...FIXME

Note	encodeVersionOne is used when...FIXME
------	---------------------------------------

encode Method

```
ByteBuffer encode()
```

encode ...FIXME

Note	encode is used exclusively when StreamsPartitionAssignor is requested to subscription.
------	--

ClientMetadata

ClientMetadata is...FIXME

Table 1. ClientMetadata's Internal Properties (e.g. Registries, Counters and Flags)

Name	Description
consumers	Member IDs Used when...FIXME

addConsumer Method

```
void addConsumer(  
    final String consumerMemberId,  
    final SubscriptionInfo info)
```

addConsumer ...FIXME

Note	addConsumer is used when...FIXME
------	----------------------------------

TaskAssignor Contract

`TaskAssignor` is the abstraction of [task assignors](#) that can [assign](#).

Table 1. TaskAssignor Contract

Method	Description
<code>assign</code>	<pre>void assign(int numStandbyReplicas)</pre> <p>Assigns tasks to clients with a given number of standby replicas Used exclusively when <code>StreamsPartitionAssignor</code> is requested to perform group assignment</p>
Note	<p>StickyTaskAssignor is the one and only known implementation of the TaskAssignor Contract.</p>

StickyTaskAssignor

StickyTaskAssignor is...FIXME

InternalProcessorContext Contract

`InternalProcessorContext` is the [extension](#) of the [ProcessorContext contract](#) for [internal processor contexts](#).

Table 1. InternalProcessorContext Contract

Method	Description
<code>currentNode</code>	<pre>ProcessorNode currentNode()</pre> <p><code>ProcessorNode</code></p> <p>Used when...FIXME</p>
<code>getCache</code>	<pre>ThreadCache getCache()</pre> <p><code>ThreadCache</code></p> <p>Used when...FIXME</p>
<code>initialize</code>	<pre>void initialize()</pre> <p>Initializes the processor context</p> <p>Used when:</p> <ul style="list-style-type: none"> <code>GlobalStateUpdateTask</code> is requested to initialize <code>StandbyTask</code> is requested to initialize state stores <code>StreamTask</code> is requested to initialize a topology
<code>metrics</code>	<pre>StreamsMetricsImpl metrics()</pre> <p><code>StreamsMetricsImpl</code></p> <p>Used when...FIXME</p>
<code>recordContext</code>	<pre>ProcessorRecordContext recordContext()</pre> <p><code>ProcessorRecordContext</code></p> <p>Used when...FIXME</p>

<code>setCurrentNode</code>	<pre>void setCurrentNode(ProcessorNode currentNode)</pre> <p>Sets the current ProcessorNode Used when...FIXME</p>
<code>setRecordContext</code>	<pre>void setRecordContext(ProcessorRecordContext recordContext)</pre> <p>Sets the current ProcessorRecordContext Used when...FIXME</p>
<code>streamTime</code>	<pre>long streamTime()</pre> <p>Stream time Used when...FIXME</p>
<code>uninitialize</code>	<pre>void uninitialize()</pre> <p>Uninitializes the processor context Used when: <ul style="list-style-type: none"> AbstractStateManager is requested to reinitializeStateStoresForPartitions AbstractTask is requested to registerStateStores </p>
Note	<p>AbstractProcessorContext is the base implementation of the InternalProcessorContext Contract.</p>

AbstractProcessorContext — Base Of Internal Processor Contexts

`AbstractProcessorContext` is the base implementation of the [InternalProcessorContext contract](#) for processor contexts that [FIXME](#).

Note	<code>AbstractProcessorContext</code> is a Java abstract class and cannot be created directly . It is created indirectly for the concrete AbstractProcessorContexts .
------	---

Table 1. AbstractProcessorContexts

AbstractProcessorContext	Description
GlobalProcessorContextImpl	
ProcessorContextImpl	
StandbyContextImpl	

Table 2. AbstractProcessorContext's Internal Properties (e.g. Registries, Counters and Flags)

Name	Description
<code>applicationId</code>	APPLICATION_ID_CONFIG
<code>valueSerde</code>	Default <code>Serde</code> for values (from the StreamsConfig)
<code>keySerde</code>	Default <code>Serde</code> for keys (from the StreamsConfig)
<code>initialized</code>	<p>Flag that indicates whether the <code>AbstractProcessorContext</code> was initialized (<code>true</code>) or not (<code>false</code>)</p> <p>Turned off (<code>false</code>) when <code>AbstractProcessorContext</code> is requested to uninitialize</p> <p>Used when <code>AbstractProcessorContext</code> is requested to register a state store</p>

stateDir Method

```
File stateDir()
```

Note	<code>stateDir</code> is part of the ProcessorContext Contract to... FIXME .
------	--

```
stateDir ...FIXME
```

Registering State Store (And StateRestoreCallback) — register Method

```
void register(  
    StateStore store,  
    StateRestoreCallback stateRestoreCallback)
```

Note

`register` is part of the [ProcessorContext Contract](#) to register a [state store](#) (and a [StateRestoreCallback](#)).

`register` simply requests the [StateManager](#) to [register the state store](#) (and the [StateRestoreCallback](#)).

When [initialized](#) already, `register` throws an [IllegalStateException](#) :

```
Can only create state stores during initialization.
```

uninitialize Method

```
void uninitialize()
```

Note

`uninitialize` is part of the [InternalProcessorContext Contract](#) to...FIXME.

`uninitialize` marks the content uninitialized (and simply turns the [initialized](#) flag off).

Creating AbstractProcessorContext Instance

`AbstractProcessorContext` takes the following when created:

- [TaskId](#)
- [StreamsConfig](#)
- [StreamsMetricsImpl](#)
- [StateManager](#)
- [ThreadCache](#)

`AbstractProcessorContext` initializes the [internal registries and counters](#).

Note

`AbstractProcessorContext` is a Java abstract class and cannot be [created directly](#). It is created indirectly for the [concrete AbstractProcessorContexts](#).

GlobalProcessorContextImpl

GlobalProcessorContextImpl is a concrete [AbstractProcessorContext](#) that...FIXME

getStateStore Method

```
StateStore getStateStore(final String name)
```

Note	getStateStore is part of the ProcessorContext Contract to...FIXME.
------	--

getStateStore simply requests the [StateManager](#) to get the global state store by the given name.

forward Method

```
void forward(final K key, final V value)
```

Note	forward is part of the ProcessorContext Contract to...FIXME.
------	--

forward ...FIXME

StandbyContextImpl

StandbyContextImpl is...FIXME

StandbyContextImpl creates a [RecordCollector](#) that simply does nothing when executed (i.e. all methods are no-ops).

ForwardingDisabledProcessorContext

ForwardingDisabledProcessorContext is...FIXME

GlobalStreamThread

`GlobalStreamThread` is...FIXME

`GlobalStreamThread` is [created](#) exclusively when `KafkaStreams` is [created](#).

Table 1. GlobalStreamThread's Internal Properties (e.g. Registries, Counters and Flags)

Name	Description
<code>cache</code>	ThreadCache

initialize Internal Method

`StateConsumer initialize()`

`initialize` ...FIXME

Note	<code>initialize</code> is used exclusively when <code>GlobalStreamThread</code> is started .
------	---

Starting Thread — run Method

`void run()`

Note	<code>run</code> is part of Java's Thread Contract to be executed by a JVM thread.
------	--

`run` ...FIXME

Note	<code>run</code> is used exclusively when <code>KafkaStreams</code> is started .
------	--

Creating GlobalStreamThread Instance

`GlobalStreamThread` takes the following when created:

- [ProcessorTopology](#)
- [StreamsConfig](#)
- Kafka [Consumer](#) (`Consumer<byte[], byte[]>`)
- [StateDirectory](#)

- `cacheSizeBytes`
- `Metrics`
- `Time`
- `threadClientId`
- `StateRestoreListener`

`GlobalStreamThread` initializes the [internal registries and counters](#).

StateConsumer

`StateConsumer` is...FIXME

`StateConsumer` is created exclusively when `GlobalStreamThread` is requested to initialize.

`StateConsumer` uses global Kafka consumer that is the restore Kafka consumer from `KafkaClientSupplier` (which by default is `DefaultKafkaClientSupplier`).

`StateConsumer` uses Kafka Consumer's manual topic assignment feature when initializing itself, i.e. uses `Consumer.assign` and `Consumer.seek` methods to assign topics and seek to offsets, respectively.

`StateConsumer` uses `poll.ms` configuration property as the polling interval while polling the topics assigned for records.

`StateConsumer` uses `commit.interval.ms` configuration property as the flush interval while polling the topics assigned for records.

Polling Topics Assigned for Records and Flushing State — `pollAndUpdate` Method

```
void pollAndUpdate()
```

`pollAndUpdate` requests the global Kafka Consumer to fetch records from the topics and partitions assigned (upon initialization).

`pollAndUpdate` polls for records for the polling interval that is configured by `poll.ms` configuration property.

For every record received, `pollAndUpdate` requests `GlobalStateMaintainer` to update.

If the time between the last flush is longer than the flush interval `pollAndUpdate` requests `GlobalStateMaintainer` to flushState and records the current time in last flush internal registry.

In case of `InvalidOffsetException` `pollAndUpdate` prints out the following ERROR message and reports a `StreamsException`.

```
Updating global state failed. You can restart KafkaStreams to recover from this error.
```

Note

`pollAndUpdate` is used exclusively when `GlobalStreamThread` is up and running.

Initializing StateConsumer (Initializing GlobalStateMaintainer and Setting Offsets) — `initialize` Method

```
void initialize()
```

`initialize` requests `GlobalStateMaintainer` to `initialize` (and give topic partitions and offsets).

`initialize` then assigns the topic partitions to the `global Kafka Consumer` and sets the fetch offsets that the consumer will use (on the next `poll`).

In the end, `initialize` records the current time in `lastFlush`.

Note

`initialize` is used exclusively when `GlobalStreamThread` is requested to `initialize`.

Creating StateConsumer Instance

`StateConsumer` takes the following when created:

- `LogContext`
- (Global) Kafka `Consumer` (`Consumer<byte[], byte[]>`)
- `GlobalStateMaintainer`
- `Time`
- Polling interval (in milliseconds)
- Flush interval (in milliseconds)

GlobalStateMaintainer

`GlobalStateMaintainer` is the [contract](#) for managing global state stores.

```
package org.apache.kafka.streams.processor.internals;

interface GlobalStateMaintainer {
    void close() throws IOException;
    void flushState();
    Map<TopicPartition, Long> initialize();
    void update(ConsumerRecord<byte[], byte[]> record);
}
```

Table 1. GlobalStateMaintainer Contract

Method	Description
<code>close</code>	<code>void close() throws IOException</code> Used when...FIXME
<code>flushState</code>	Used when...FIXME
<code>initialize</code>	Used when...FIXME
<code>update</code>	Used when...FIXME

Note	<code>GlobalStateUpdateTask</code> is the one and only <code>GlobalStateMaintainer</code> .
------	---

GlobalStateUpdateTask — The Default GlobalStateMaintainer

`GlobalStateUpdateTask` is the one and only [GlobalStateMaintainer](#) that...FIXME

`GlobalStateUpdateTask` is [created](#) exclusively when `GlobalStreamThread` is requested to [initialize](#) (and creates a [StateConsumer](#)).

`GlobalStateUpdateTask` keeps track of the **offsets** of the records (by the topics and the partitions) that have already been [processed](#). The offsets are then [flushed to the global store](#) when requested.

`GlobalStateUpdateTask` keeps track of the **deserializers** for...FIXME

Initializing — `initialize` Method

```
Map<TopicPartition, Long> initialize()
```

Note	<code>initialize</code> is part of GlobalStateMaintainer Contract to...FIXME.
------	---

`initialize` ...FIXME

update Method

```
void update(final ConsumerRecord<byte[], byte[]> record)
```

Note	<code>update</code> is part of GlobalStateMaintainer Contract to...FIXME.
------	---

`update` looks up the [RecordDeserializer](#) for the topic (of the record) in the [deserializers](#) internal registry.

Note	<code>RecordDeserializer</code> is the metadata of a SourceNode that knows how to deserialize a raw record in a given <code>ProcessorContext</code> .
------	---

`update` then uses the `RecordDeserializer` to deserialize the record (to the proper types of the key and the value).

When the record has been successfully deserialized, `update` does the following:

- Creates a [ProcessorRecordContext](#) for the deserialized record

2. Requests `InternalProcessorContext` to use the record context
3. Requests `InternalProcessorContext` to use the source node of the `RecordDeserializer`
4. Requests the `SourceNode` (of the deserialized record) to process the deserialized record

In the end, `update` saves the topic and the partition of the record with the offset in the `offsets` internal registry.

Flushing State Stores — `flushState` Method

```
void flushState()
```

Note

`flushState` is part of `GlobalStateMaintainer Contract` to...FIXME.

`flushState` requests `GlobalStateManager` to `flush` followed by `checkpointing` the `offsets`.

Closing State Manager — `close` Method

```
void close() throws IOException
```

Note

`close` is part of `StateManager Contract` to...FIXME.

`close` simply requests `GlobalStateManager` to `close` (passing on the `offsets`).

Creating GlobalStateUpdateTask Instance

`GlobalStateUpdateTask` takes the following when created:

- `ProcessorTopology`
- `InternalProcessorContext`
- `GlobalStateManager`
- `DeserializationExceptionHandler`
- `LogContext`

`GlobalStateUpdateTask` initializes the `offsets` and `deserializers` internal registries.

`initTopology` Internal Method

```
void initTopology()
```

initTopology ...FIXME

Note

initTopology is used exclusively when GlobalStateUpdateTask is requested to initialize.

RecordCollector Contract

`RecordCollector` is the [contract](#) of record collectors that can [send a record to a topic](#).

Table 1. RecordCollector Contract

Method	Description
<code>close</code>	<pre>void close()</pre> <p>Closes the <code>RecordCollector</code> Used when...FIXME</p>
<code>flush</code>	<pre>void flush()</pre> <p>Flushes the <code>RecordCollector</code> (and the internal Kafka producer) Used exclusively when <code>StreamTask</code> is requested to flush state stores and the internal Kafka producer</p>
<code>init</code>	<pre>void init(Producer<byte[], byte[]> producer)</pre> <p>Initializes the <code>RecordCollector</code> (with a Kafka Producer) Used when <code>StreamTask</code> is created and resumed</p>
<code>offsets</code>	<pre>Map<TopicPartition, Long> offsets()</pre> <p>Used exclusively when <code>StreamTask</code> is requested to activeTaskCheckpointableOffsets</p>

send

```
<K, V> void send(
    String topic,
    K key,
    V value,
    Headers headers,
    Integer partition,
    Long timestamp,
    Serializer<K> keySerializer,
    Serializer<V> valueSerializer)
<K, V> void send(
    String topic,
    K key,
    V value,
    Headers headers,
    Long timestamp,
    Serializer<K> keySerializer,
    Serializer<V> valueSerializer,
    StreamPartitioner<? super K, ? super V> partitioner)
```

Sends a record to a topic

Used when:

- `RecordCollectorImpl` is requested to [send a record](#)
- `InMemoryTimeOrderedKeyValueBuffer` is requested to [flush](#)
- `StoreChangeLogger` is requested to [logChange](#)
- `SinkNode` is requested to [process a record](#)

Table 2. RecordCollectors

RecordCollector	Description
NO_OP_COLLECTOR	
RecordCollectorImpl	

Supplier Contract and `recordCollector` Method

`RecordCollector` defines a `Supplier` that is used to access the [RecordCollector](#).

```
RecordCollector recordCollector()
```

Note

`recordCollector` is used when:

- `SinkNode` is requested to [process a record](#)
- `InMemoryTimeOrderedKeyValueBuffer` is requested to [initialize](#)
- `StoreChangeLogger` is [created](#)

RecordCollectorImpl

`RecordCollectorImpl` is a concrete `RecordCollector` that is [created](#) and immediately [initialized](#) exclusively when `StreamTask` is created.

`RecordCollectorImpl` uses `task [streamTaskId]` for the `logPrefix` (that uses the specified `streamTaskId`).

Creating RecordCollectorImpl Instance

`RecordCollectorImpl` takes the following to be created:

- StreamTask ID
- `LogContext`
- `ProductionExceptionHandler`
- **skippedRecords** sensor

`RecordCollectorImpl` initializes the [internal properties](#).

send Method

```
<K, V> void send(  
    final String topic,  
    final K key,  
    final V value,  
    final Long timestamp,  
    final Serializer<K> keySerializer,  
    final Serializer<V> valueSerializer,  
    final StreamPartitioner<? super K, ? super V> partitioner)
```

Note	<code>send</code> is part of RecordCollector Contract to...FIXME.
------	---

Flushing Kafka Producer— flush Method

```
void flush()
```

Note

`flush` is part of the [RecordCollector Contract](#) to flush the internal Kafka producer.

`flush` prints out the following DEBUG message to the logs:

```
Flushing producer
```

`flush` requests the Kafka Producer to flush the buffered (*accumulated*) records (via `Producer.flush`).

In the end, `flush` [checkForException](#).

Closing RecordCollector— `close` Method

```
void close()
```

Note

`close` is part of the [RecordCollector Contract](#) to close a `RecordCollector`.

`close` prints out the following DEBUG message to the logs:

```
Closing producer
```

`close` requests the internal Producer to close.

In the end, `close` [throws any KafkaException that may have been reported \(but not yet thrown\)](#).

checkForException Internal Method

```
void checkForException()
```

`checkForException` checks the `sendException` internal registry and throws the exception if specified.

Note

`checkForException` is used when...FIXME

Internal Properties

Name	Description
sendException	KafkaException
offsets	<p>Map<TopicPartition, Long> offsets</p> <p>Initialized (as empty) when RecordCollectorImpl is created</p> <p>A new pair of a Kafka TopicPartition with the offset added (or updated) after RecordCollectorImpl has finished sending a record</p>
producer	<p>Kafka Producer (Producer<byte[], byte[]>)</p> <p>Used when...FIXME</p>

ThreadCache

`ThreadCache` is a [collection of NamedCaches](#) per [task ID](#) and state store with caching enabled.

`ThreadCache` is [created](#) when:

- `GlobalStreamThread` is [created](#)
- `StreamThread` is [created](#) (to create a [TaskCreator](#))
- `StandbyContextImpl` is [created](#)

Tip Enable `ALL` logging level for `org.apache.kafka.streams.state.internals.ThreadCache` logger to see what happens inside.

Add the following line to `log4j.properties`:

```
log4j.logger.org.apache.kafka.streams.state.internals.ThreadCache=ALL
```

Refer to [Application Logging Using log4j](#).

Creating ThreadCache Instance

`ThreadCache` takes the following to be created:

- `LogContext`
- `maxCacheSizeBytes`
- `StreamsMetricsImpl`

`ThreadCache` initializes the [internal properties](#).

StreamsConfig.CACHE_MAX_BYTES_BUFFERING_CONFIG Configuration Property

When [created](#), `ThreadCache` is given the `maxCacheSizeBytes` that is configured using `StreamsConfig.CACHE_MAX_BYTES_BUFFERING_CONFIG` configuration property.

The `maxCacheSizeBytes` is used when `ThreadCache` is requested to [maybeEvict](#) (when requested to [put](#) and [putIfAbsent](#)).

Building Cache Namespace

— `nameSpaceFromTaskIdAndStore` Static Method

```
String nameSpaceFromtaskIdAndStore(
    String taskIdString,
    String underlyingStoreName)
```

`nameSpaceFromtaskIdAndStore` builds a cache namespace to be of the format:

`[taskIdString]-[underlyingStoreName]`

Note

`nameSpaceFromtaskIdAndStore` is used exclusively when `CachingKeyValueStore` is requested to [initialize](#) (via `initInternal`).

Registering DirtyEntryFlushListener For Namespace

— `addDirtyEntryFlushListener` Method

```
void addDirtyEntryFlushListener(
    String namespace,
    DirtyEntryFlushListener listener)
```

`addDirtyEntryFlushListener` gets or creates a new named cache (by the namespace) and requests it to [set the flush listener](#) with the `DirtyEntryFlushListener`.

Note

`addDirtyEntryFlushListener` is used when the [state stores with caching enabled](#) (i.e. `CachingKeyValueStore`, `CachingSessionStore`, and `CachingWindowStore`) are requested to initialize.

Closing NamedCache by Namespace — `close` Method

```
void close(String namespace)
```

`close` ...FIXME

Note

`close` is used when...FIXME

Putting Key And Value Into Namespace — `put` Method

```
void put(
    String namespace,
    Bytes key,
    LRUCacheEntry value)
```

put ...FIXME

Note	put is used when...FIXME
------	--------------------------

putIfAbsent Method

```
LRUCacheEntry putIfAbsent(
    String namespace,
    Bytes key,
    LRUCacheEntry value)
```

putIfAbsent ...FIXME

Note	putIfAbsent is used when...FIXME
------	----------------------------------

flush Method

```
void flush(String namespace)
```

flush ...FIXME

Note	flush is used when...FIXME
------	----------------------------

Getting Or Creating New Named Cache by Namespace — getOrCreateCache Internal Method

```
NamedCache getOrCreateCache(String name)
```

getOrCreateCache ...FIXME

Note	getOrCreateCache is used when ThreadCache is requested to addDirtyEntryFlushListener, put a key and a value into a namespace, putIfAbsent, and maybeEvict.
------	--

maybeEvict Internal Method

```
void maybeEvict(String namespace)
```

maybeEvict ...FIXME

Note	maybeEvict is used when ThreadCache is requested to put and putIfAbsent.
------	--

sizeBytes Method

```
long sizeBytes()
```

sizeBytes ...FIXME

In the end, sizeBytes prints out the following TRACE message to the logs:

```
Evicted [numEvicted] entries from cache [namespace]
```

Note	sizeBytes is used exclusively when ThreadCache is requested to maybeEvict (when requested to put and putIfAbsent).
------	--

Internal Properties

Name	Description
caches	<p>Collection of NamedCaches by namespace (Map<String, NamedCache>)</p> <ul style="list-style-type: none"> A new NamedCache is added when requested to get or create a new NamedCache by namespace A NamedCache is removed in close <p>Used when...FIXME</p>
numPuts	<p>Number of put and putIfAbsent</p> <p>Default: 0</p> <p>Used exclusively in flush to print the current value with TRACE logging level enabled</p>

NamedCache

NamedCache is...FIXME

NamedCache is created exclusively when ThreadCache is requested to get or create a new named cache by namespace.

Creating NamedCache Instance

NamedCache takes the following to be created:

- Name
- StreamsMetricsImpl

NamedCache initializes the internal properties.

NamedCacheMetrics — Performance Metrics of NamedCache

NamedCache uses the following performance metrics...FIXME

setListener Method

```
void setListener(  
    ThreadCache.DirtyEntryFlushListener listener)
```

setListener simply sets the listener internal registry to be the ThreadCache.DirtyEntryFlushListener .

Note

setListener is used exclusively when ThreadCache is requested to register a DirtyEntryFlushListener for a namespace.

evict Method

```
void evict()
```

evict ...FIXME

Note

`evict` is used exclusively when `ThreadCache` is requested to `maybeEvict` a `namespace`

close Method

```
void close()
```

`close` ...FIXME

Note

`close` is used when...FIXME

flush Method

```
void flush() (1)
// Private API
void flush(LRUNode evicted)
```

1. Uses `null` for the evicted `LRUNode`

`flush` ...FIXME

Note

`flush` is used when...FIXME

Internal Properties

Name	Description
<code>listener</code>	<p><code>ThreadCache.DirtyEntryFlushListener</code></p> <ul style="list-style-type: none"> • Initialized in <code>setListener</code> • Removed (<i>nullified</i>) in <code>close</code> <p>Used exclusively when <code>flush</code> an evicted <code>LRUNode</code></p>

Stamped — Orderable Value At Timestamp

`Stamped` represents a [value](#) at a given [timestamp](#).

`Stamped` takes the following when created:

- Value (of type `v`)
- Timestamp

`Stamped` values can be compared (and hence ordered) by [timestamp](#) (in ascending order).

```
import org.apache.kafka.streams.processor.internals.Stamped

// three stampeds in ascending order
val s1 = new Stamped("a", 0)
val s2 = new Stamped("b", 1)
val s3 = new Stamped("c", 2)

// Adding the stampeds in a random order
// TreeSet is a concrete SortedSet
import java.util.TreeSet
import collection.JavaConverters._
val stampeds = new TreeSet(Seq(s3, s1, s2).asJava)

assert(stampeds.asScala == Set(s1, s2, s3))
```

Table 1. Stampeds

Stamped	Description
StampedRecord	Uses Kafka <code>ConsumerRecords</code> for values
PunctuationSchedule	Uses <code>ProcessorNodes</code> for values

PartitionGroup

`PartitionGroup` is a collection of `RecordQueues` (one per every `partition` assigned to a `StreamTask`).

`PartitionGroup` is `created` exclusively for a `StreamTask`.

Creating PartitionGroup Instance

`PartitionGroup` takes the following to be created:

- `RecordQueues` per Kafka `TopicPartition` (`Map<TopicPartition, RecordQueue>`)
- `recordLateness` sensor

`PartitionGroup` initializes the `internal properties`.

Retrieving Next Stamped Record (Record with Timestamp) and RecordQueue — `nextRecord` Method

```
StampedRecord nextRecord(final RecordInfo info)
```

`nextRecord` ...FIXME

Note	<code>nextRecord</code> is used exclusively when <code>StreamTask</code> is requested to process a single record.
------	---

Adding Records to RecordQueue For Given Partition — `addRawRecords` Method

```
int addRawRecords(
    final TopicPartition partition,
    final Iterable<ConsumerRecord<byte[], byte[]>> rawRecords)
```

`addRawRecords` looks up the `RecordQueue` for the input Kafka `TopicPartition` (in the `partitionQueues`).

`addRawRecords` requests the `RecordQueue` for the `size` and to `add Kafka ConsumerRecords (as StampedRecords)`.

`addRawRecords` inserts the `RecordQueue` into the `queuesByTime` priority queue only if the `RecordQueue` was empty before.

`addRawRecords` adds the difference between the old and current sizes to `totalBuffered`.

In the end, `addRawRecords` returns the current size of the `RecordQueue`.

Note

`addRawRecords` is used exclusively when `streamTask` is requested to `buffer new records`.

clear Method

```
void clear()
```

`clear` removes all of the elements from the `queuesByTime` and requests every `RecordQueue` (in the internal `partitionQueues` registry) to `clear` itself.

Note

`clear` is used exclusively when `streamTask` is requested to `closeTopology`.

numBuffered Method

```
int numBuffered(final TopicPartition partition)
```

`numBuffered` ...FIXME

Note

`numBuffered` is used when...FIXME

Minimum Partition Timestamp Across All Partitions — timestamp Method

```
long timestamp()
```

`timestamp` simply iterates over the `TopicPartitions` and requests every `RecordQueue` for the `partition timestamp`.

In the end, `timestamp` returns the oldest partition timestamp (time-wise).

Note

`timestamp` is used exclusively when `StreamTask` is requested to `maybePunctuateStreamTime`

Internal Properties

Name	Description
allBuffered	<p>Flag that indicates whether all the RecordQueues have at least one record buffered (<code>true</code>) or not (<code>false</code>)</p> <p>Default: <code>false</code></p> <p>Enabled (<code>true</code>) when the size of the nonEmptyQueuesByTime is exactly the size of the partitionQueues (when requested to add records to the RecordQueue for a given partition)</p> <p>Disabled (<code>false</code>) when one of the RecordQueues has been drained (when requested for the next record)</p> <p>Used (as allPartitionsBuffered method) exclusively when StreamTask is requested to isProcessable</p>
nonEmptyQueuesByTime	<p>Java PriorityQueue that orders RecordQueues per tracked partition timestamp (and the initial capacity as the number of partitions in the partitionQueues) <code>(PriorityQueue<RecordQueue>)</code></p> <p>Used when PartitionGroup is requested for the following:</p> <ul style="list-style-type: none"> * Next record * Add records to the RecordQueue for a given partition <p>Cleared when PartitionGroup is requested to clear</p>
queuesByTime	Java PriorityQueue that is an unbounded priority queue that uses partition timestamps for ordering.
streamTime	Default: UNKNOWN
totalBuffered	<p>Number of...FIXME</p> <p>Default: <code>0</code></p>

RecordInfo

RecordInfo is...FIXME

Getting Processor Node — node Method

```
ProcessorNode node()
```

node ...FIXME

Note

node is used when...FIXME

Getting RecordQueue — queue Method

```
RecordQueue queue()
```

queue ...FIXME

Note

queue is used when...FIXME

Getting Topic Partition — partition Method

```
TopicPartition partition()
```

partition ...FIXME

Note

partition is used when...FIXME

RecordQueue — FIFO Queue of Buffered Stamped Records

`RecordQueue` is a [FIFO queue of StampedRecords](#) from a [Kafka partition](#) (associated with a [SourceNode](#)).

`RecordQueue` is [created](#) along with a [StreamTask](#) exclusively (for every [partition assigned](#)).

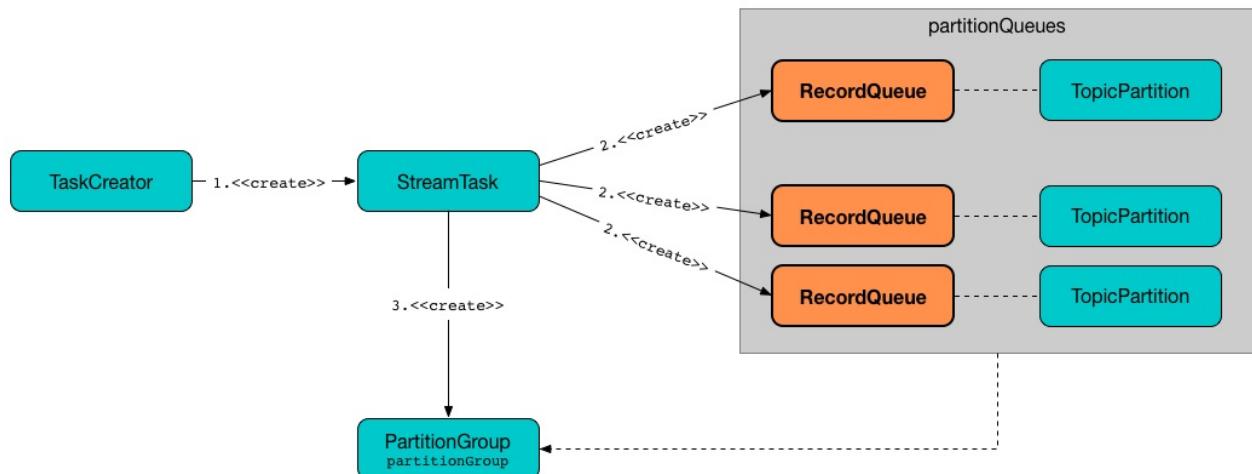


Figure 1. RecordQueue, StreamTask and TaskCreator

`RecordQueue` acts as a buffer of Kafka [ConsumerRecords](#).

Note

`StreamTask` uses a [PartitionGroup](#) to manage the `RecordQueues` per partition assigned.

`RecordQueue` defines the **UNKNOWN** constant to be `ConsumerRecord.NO_TIMESTAMP`.

Tip Enable `ALL` logging level for `org.apache.kafka.streams.processor.internals.RecordQueue` logger to see what happens inside.

Add the following line to `log4j.properties`:

```
log4j.logger.org.apache.kafka.streams.processor.internals.RecordQueue=ALL
```

Refer to [Application Logging Using log4j](#).

Creating RecordQueue Instance

`RecordQueue` takes the following to be created:

- Kafka [TopicPartition](#)

- `SourceNode`
- `TimestampExtractor`
- `DeserializationExceptionHandler`
- `InternalProcessorContext`
- `LogContext`

`RecordQueue` initializes the [internal properties](#).

Adding Kafka ConsumerRecords (as StampedRecords)

— `addRawRecords` Method

```
int addRawRecords(final Iterable<ConsumerRecord<byte[], byte[]>> rawRecords)
```

For every Kafka [ConsumerRecord](#) in the input `rawRecords`, `addRawRecords` does the following:

1. Requests the [RecordDeserializer](#) to [deserialize](#) the record (with the [ProcessorContext](#))
2. [FIXME](#)
3. Creates a [StampedRecord](#) for the record and the record timestamp
4. Inserts the `StampedRecord` at the end of the [fifoQueue](#)
5. [FIXME](#)

While processing `ConsumerRecords`, `addRawRecords` prints out the following TRACE message to the logs:

```
Source node [name] extracted timestamp [timestamp] for record [record]
```

With all `ConsumerRecords` processed, `addRawRecords` ...[FIXME](#)

In the end, `addRawRecords` returns the [number of ConsumerRecords](#) in the [queue](#).

Note	<p><code>addRawRecords</code> skips (drops) Kafka <code>ConsumerRecords</code> when either condition holds:</p> <ul style="list-style-type: none"> • RecordDeserializer could not deserialize and the <code>DeserializationExceptionHandler</code> is not set to fail upon a deserialization error • Extracted timestamp is negative (and hence invalid)
-------------	--

Note

`addRawRecords` is used exclusively when `PartitionGroup` is requested to add records to a RecordQueue for a Kafka partition.

Clearing (Resetting) RecordQueue — `clear` Method

```
void clear()
```

`clear ...FIXME`

In the end, `clear` (re)sets the `partitionTime` to `NOT_KNOWN`.

Note

`clear` is used exclusively when `PartitionGroup` is requested to `clear`.

Requesting Number of Records in Queue — `size` Method

```
int size()
```

`size` simply returns the number of records in the `fifoQueue`.

Note

`size` is used when:

- `PartitionGroup` is requested to add records to a RecordQueue for a Kafka partition and `numBuffered`
- `RecordQueue` is requested to add Kafka ConsumerRecords (as `StampedRecords`)
- `StreamTask` is requested to process a single record

Internal Properties

Table 1. RecordQueue's Internal Properties (e.g. Registries, Counters and Flags)

Name	Description
<code>fifoQueue</code>	Java's <code>java.util.ArrayDeque</code> of <code>StampedRecords</code> (i.e. orderable Kafka <code>ConsumerRecords</code> with a timestamp) Used when...FIXME
<code>partitionTime</code>	
<code>recordDeserializer</code>	RecordDeserializer (for the <code>SourceNode</code> and <code>DeserializationExceptionHandler</code>) Used when...FIXME

StampedRecord — Orderable Kafka ConsumerRecords At Timestamp

`StampedRecord` is a `Stamped` with [Kafka ConsumerRecords](#) (as [values](#)).

In other words, `StampedRecord` represents a [Kafka ConsumerRecord](#) at a given [timestamp](#) and can be ordered in ascending order.

`StampedRecord` is [created](#) when:

- `RecordQueue` is requested to [add Kafka ConsumerRecords](#) (as [StampedRecords](#))
- `StreamTask` is requested to [punctuate](#)

`StampedRecord` takes the following when created:

- [Kafka ConsumerRecord](#)
- [Timestamp](#)

`StampedRecord` gives access to the properties of a [ConsumerRecord](#):

- `topic`
- `partition`
- `key`
- `value`
- `offset`

The text representation of a `StampedRecord` is of the format `value, timestamp = [timestamp]`.

```
import org.apache.kafka.streams.processor.internals.StampedRecord
import org.apache.kafka.clients.consumer.ConsumerRecord
import org.apache.kafka.clients.consumer.ConsumerRecord.{NULL_CHECKSUM, NULL_SIZE}
import org.apache.kafka.common.record.TimestampType

val timestamp1 = 1L
val cr1 = new ConsumerRecord("topic", 0, 0L,
    timestamp1,
    TimestampType.NO_TIMESTAMP_TYPE, NULL_CHECKSUM, NULL_SIZE, NULL_SIZE, 0, 0)
    .asInstanceOf[ConsumerRecord[Object, Object]] // wish there were a better way
val sr1 = new StampedRecord(cr1, timestamp1)

val timestamp2 = 2L
val cr2 = new ConsumerRecord("topic", 0, 0L,
    timestamp2,
    TimestampType.NO_TIMESTAMP_TYPE, NULL_CHECKSUM, NULL_SIZE, NULL_SIZE, 0, 0)
    .asInstanceOf[ConsumerRecord[Object, Object]] // wish there were a better way
val sr2 = new StampedRecord(cr2, timestamp2)

val timestamp3 = 3L
val cr3 = new ConsumerRecord("topic", 0, 0L,
    timestamp3,
    TimestampType.NO_TIMESTAMP_TYPE, NULL_CHECKSUM, NULL_SIZE, NULL_SIZE, 0, 0)
    .asInstanceOf[ConsumerRecord[Object, Object]] // wish there were a better way
val sr3 = new StampedRecord(cr3, timestamp3)

// Adding the stampeds in a random order
// TreeSet is a concrete SortedSet
import java.util.TreeSet
import collection.JavaConverters.-
val srs = new TreeSet(Seq(sr3, sr1, sr2).asJava)

assert(srs.asScala == Set(sr1, sr2, sr3))
```

PunctuationQueue

`PunctuationQueue` manages a [java.util.PriorityQueue](#) of [PunctuationSchedules](#).

Attempting to Punctuate — `mayPunctuate` Method

```
boolean mayPunctuate(
    long timestamp,
    PunctuationType type,
    ProcessorNodePunctuator processorNodePunctuator)
```

`mayPunctuate` takes the [PunctuationSchedules](#) off the [PriorityQueue](#) for which the `timestamp` is older (*smaller*) than the given `timestamp`.

`mayPunctuate` then requests the given [ProcessorNodePunctuator](#) to punctuate (with the `node` and the `punctuator` of every [PunctuationSchedule](#), the given `timestamp` and the `PunctuationType`).

In the end, `mayPunctuate` returns whether a [PunctuationSchedule](#) was punctuated (`true`) or not (`false`).

Note	<code>mayPunctuate</code> is used when <code>StreamTask</code> is requested to attempt to punctuate by <code>stream</code> and <code>system</code> time (with itself as the ProcessorNodePunctuator).
------	--

Scheduling Cancellable Periodic Action (Punctuator) — `schedule` Method

```
Cancellable schedule(PunctuationSchedule sched)
```

`schedule ...FIXME`

Note	<code>schedule</code> is used when...FIXME
------	--

PunctuationSchedule — Orderable ProcessorNodes At Timestamp

`PunctuationSchedule` is a [Stamped](#) with [ProcessorNodes](#) (as [values](#)).

In other words, `PunctuationSchedule` represents a [ProcessorNode](#) at a given [timestamp](#) and can be ordered in ascending order.

`PunctuationSchedule` is [created](#) when:

- `StreamTask` is requested to [schedule](#)
- `PunctuationSchedule` is requested for the [next PunctuationSchedule](#)

`PunctuationSchedule` takes the following when created:

- [ProcessorNode](#)
- [Timestamp](#)
- [Interval](#)
- [Punctuator](#)
- [RepointableCancellable](#)

next Method

```
PunctuationSchedule next(final long currTimestamp)
```

`next` ...FIXME

Note

`next` is used exclusively when `PunctuationQueue` is requested to [mayPunctuate](#).

QueryableStoreProvider

`QueryableStoreProvider` is created along with `KafkaStreams` to `getStore` when requested to store.

Creating QueryableStoreProvider Instance

`QueryableStoreProvider` takes the following when created:

- `StateStoreProviders`
- `GlobalStateStoreProvider`

getStore Method

```
<T> T getStore(final String storeName, final QueryableStoreType<T> queryableStoreType)
```

getStore ...FIXME

Note	<code>getStore</code> is used exclusively when <code>KafkaStreams</code> is requested to <code>store</code> .
------	---

StateStoreProvider

`StateStoreProvider` is the [contract](#) of [StateStoreProviders](#) that [stores](#).

```
package org.apache.kafka.streams.state.internals;

interface StateStoreProvider {
    <T> List<T> stores(String storeName, QueryableStoreType<T> queryableStoreType);
}
```

Table 1. StateStoreProvider Contract

Method	Description
<code>stores</code>	Used when...FIXME

Table 2. StateStoreProviders

StateStoreProvider	Description
GlobalStateStoreProvider	
StreamThreadStateStoreProvider	
WrappingStoreProvider	

StreamThreadStateStoreProvider

StreamThreadStateStoreProvider is...FIXME

stores Method

```
List<T> stores(  
    final String storeName,  
    final QueryableStoreType<T> queryableStoreType)
```

Note

stores is part of [StateStoreProvider Contract](#) to...FIXME.

stores ...FIXME

GlobalStateStoreProvider

`GlobalStateStoreProvider` is a [StateStoreProvider](#) that...FIXME

`GlobalStateStoreProvider` is created along with [KafkaStreams](#) (when creating the [QueryableStoreProvider](#)).

`GlobalStateStoreProvider` takes a collection of [StateStores](#) by their names when created.

stores Method

```
List<T> stores(  
    final String storeName,  
    final QueryableStoreType<T> queryableStoreType)
```

Note

`stores` is part of [StateStoreProvider Contract](#) to...FIXME.

`stores` ...FIXME

WrappingStoreProvider

WrappingStoreProvider is...FIXME

RecordDeserializer

`RecordDeserializer` is the metadata of a `SourceNode` that knows how to `deserialize` a raw `ConsumerRecord`.

`RecordDeserializer` takes the following when created:

- `SourceNode`
- `DeserializationExceptionHandler`
- `LogContext`

Deserializing Kafka ConsumerRecord (in ProcessorContext) — `deserialize` Method

```
ConsumerRecord<Object, Object> deserialize(  
    final ProcessorContext processorContext,  
    final ConsumerRecord<byte[], byte[]> rawRecord)
```

`deserialize` ...FIXME

Note	<code>deserialize</code> is used when...FIXME
------	---

StateDirectory

`StateDirectory` is...FIXME

`StateDirectory` is [created](#) when `KafkaStreams` is [created](#).

`StateDirectory` uses [state.dir](#) and [application.id](#) configuration properties for the location of the state store.

Note	<code>state.dir</code> configuration property defaults to <code>/tmp/kafka-streams</code> .
------	---

Tip	<p>Enable <code>ALL</code> logging level for <code>org.apache.kafka.streams.processor.internals.StateDirectory</code> logger to see what happens inside.</p>
-----	--

Add the following line to `log4j.properties` :

```
log4j.logger.org.apache.kafka.streams.processor.internals.StateDirectory=ALL
```

Refer to [Application Logging Using log4j](#).

clean Method

```
synchronized void clean()
```

`clean` ...FIXME

Note	<code>clean</code> is used when...FIXME
------	---

cleanRemovedTasks Method

```
void cleanRemovedTasks(final long cleanupDelayMs) (1)

// private
private void cleanRemovedTasks(
    final long cleanupDelayMs,
    final boolean manualUserCall) throws Exception
```

1. Turns the `manualUserCall` flag off

`cleanRemovedTasks` ...FIXME

Note	cleanRemovedTasks is used when...FIXME
------	--

Creating StateDirectory Instance

StateDirectory takes the following when created:

- StreamsConfig
- Time

StateDirectory initializes the internal registries and counters.

listTaskDirectories Method

```
File[] listTaskDirectories()
```

listTaskDirectories ...FIXME

Note	listTaskDirectories is used when...FIXME
------	--

Locking State Directory For Task — lock Method

```
boolean lock(TaskId taskId)
```

lock ...FIXME

Note	lock is used when...FIXME
------	---------------------------

ProcessorRecordContext — Record Metadata

`ProcessorRecordContext` is a `RecordContext` that is the metadata of a record:

- Timestamp
- Offset
- Partition
- Topic

`ProcessorRecordContext` is created when...FIXME

CopartitionedTopicsValidator

CopartitionedTopicsValidator is...FIXME

StateManager Contract — State Store Managers

`StateManager` is the contract of `state store managers` that manage `state stores` and are `checkpointable`.

Table 1. StateManager Contract

Method	Description
<code>baseDir</code>	<pre>File baseDir()</pre> <p>Base directory</p> <p>Used exclusively when <code>AbstractProcessorContext</code> is requested for the <code>state directory</code></p>
<code>close</code>	<pre>void close(Map<TopicPartition, Long> offsets)</pre> <p>Closes a state manager</p> <p>Used when:</p> <ul style="list-style-type: none"> • <code>AbstractTask</code> is requested to <code>close the state manager</code> • <code>GlobalStateUpdateTask</code> is requested to <code>close the state manager</code>
<code>flush</code>	<pre>void flush()</pre> <p>Flushes all <code>state stores</code> registered (in the state manager)</p> <p>Used when:</p> <ul style="list-style-type: none"> • <code>AbstractTask</code> and <code>GlobalStateUpdateTask</code> are requested to flush state stores • <code>StandbyTask</code> is requested to <code>flush and checkpoint state stores</code>
	<pre>StateStore getGlobalStore(String name)</pre> <p>Accessing the global <code>state store</code> by name</p>

getGlobalStore	<p>Used when:</p> <ul style="list-style-type: none"> • <code>GlobalProcessorContextImpl</code> and <code>ProcessorContextImpl</code> are requested to get the global state store by name • <code>GlobalStateManagerImpl</code> is requested to <code>get state store by name</code>
getStore	<pre>StateStore getStore(String name)</pre> <p>Accessing the state store by name</p> <p>Used when <code>AbstractTask</code> and <code>ProcessorContextImpl</code> are requested to get the state store by name</p>
register	<pre>void register(StateStore store, StateRestoreCallback stateRestoreCallback)</pre> <p>Registers a state store (with an associated <code>StateRestoreCallback</code>)</p> <p>Used exclusively when <code>AbstractProcessorContext</code> is requested to register a state store</p>
reinitializeStateStoresForPartitions	<pre>void reinitializeStateStoresForPartitions(Collection<TopicPartition> partitions, InternalProcessorContext processorContext)</pre> <p>Used when:</p> <ul style="list-style-type: none"> • <code>AbstractTask</code> is requested to <code>reinitializeStateStoresForPartitions</code> • <code>GlobalStateManagerImpl</code> is requested to <code>restoreState</code>

Table 2. StateManagers (Direct Extensions)

StateManager	Description
<code>AbstractStateManager</code>	
<code>GlobalStateManager</code>	

AbstractStateManager — Base State Manager

`AbstractStateManager` is the base implementation of the [StateManager](#) contract for [state managers](#) that use [offset checkpointing](#).

Note

`AbstractStateManager` is a Java abstract class and cannot be [created](#) directly. It is created indirectly for the [concrete AbstractStateManagers](#).

`AbstractStateManager` uses `.checkpoint` for the name of the [checkpoint](#) offset file.

Table 1. AbstractStateManagers

AbstractStateManager	Description
GlobalStateManagerImpl	
ProcessorStateManager	

Creating AbstractStateManager Instance

`AbstractStateManager` takes the following to be created:

- Base directory
- `eosEnabled` flag

`AbstractStateManager` initializes the [internal properties](#).

Note

`AbstractStateManager` is a Java abstract class and cannot be [created](#) directly. It is created indirectly for the [concrete AbstractStateManagers](#).

reinitializeStateStoresForPartitions Method

```
void reinitializeStateStoresForPartitions(
    Logger log,
    Map<String, StateStore> stateStores,
    Map<String, String> storeToChangelogTopic,
    Collection<TopicPartition> partitions,
    InternalProcessorContext processorContext)
```

`reinitializeStateStoresForPartitions ...FIXME`

Note

- `reinitializeStateStoresForPartitions` is used when:
- `GlobalStateManagerImpl` is requested to `reinitializeStateStoresForPartitions`
 - `ProcessorStateManager` is requested to `reinitializeStateStoresForPartitions`

inverseOneToOneMap Internal Method

```
Map<String, String> inverseOneToOneMap(Map<String, String> origin)
```

`inverseOneToOneMap ...FIXME`

Note

`inverseOneToOneMap` is used exclusively when `AbstractStateManager` is requested to `reinitializeStateStoresForPartitions`.

converterForStore Static Method

```
RecordConverter converterForStore(StateStore store)
```

`converterForStore ...FIXME`

Note

`converterForStore` is used when...FIXME

Internal Properties

Name	Description
<code>checkpoint</code>	<p><code>OffsetCheckpoint</code> with the <code>.checkpoint</code> file in the <code>base directory</code>.</p> <p>Initialized to a new <code>OffsetCheckpoint</code> when <code>AbstractStateManager</code> is <code>created</code></p> <p>Used when:</p> <ul style="list-style-type: none"> • <code>AbstractStateManager</code> is requested to <code>reinitializeStateStoresForPartitions</code> (to write the <code>checkpointableOffsets</code> with <code>exactly-once support</code> off) • <code>GlobalStateManagerImpl</code> is requested to <code>initialize</code> and <code>checkpoint</code> • <code>ProcessorStateManager</code> is <code>created</code> and requested to <code>checkpoint</code>

	<p>Collection of Kafka TopicPartitions with offsets $(\text{Map} < \text{TopicPartition}, \text{Long} >)$</p> <p>Entries (partitions with offsets) added when:</p> <ul style="list-style-type: none"> • <code>GlobalStateManagerImpl</code> is requested to initialize, restoreState and checkpoint • <code>ProcessorStateManager</code> is created and requested to checkpoint
globalStores	<p>Global state stores by name (managed by GlobalStateManagerImpl mostly)</p> <p>New state stores added when:</p> <ul style="list-style-type: none"> • <code>GlobalStateManagerImpl</code> is requested to register a state store • <code>ProcessorStateManager</code> is requested to registerGlobalStateStores <p>State stores removed when:</p> <ul style="list-style-type: none"> • <code>GlobalStateManagerImpl</code> is requested to close
stores	<p>State stores by name (managed by ProcessorStateManager)</p> <p>New state stores added when:</p> <ul style="list-style-type: none"> • <code>ProcessorStateManager</code> is requested to register a state store <p>State stores removed when:</p> <ul style="list-style-type: none"> • <code>ProcessorStateManager</code> is requested to close

ProcessorStateManager

`ProcessorStateManager` is a concrete `StateManager` (as a `AbstractStateManager`) that...

FIXME

`ProcessorStateManager` is created exclusively when `AbstractTask` is created (for `StandbyTask` and `StreamTask` tasks).

Note

`ProcessorStateManager` is created with the `isStandby` flag to indicate whether used by `StandbyTask` (`true`) or `StreamTask` (`false`).

`ProcessorStateManager` uses the `-changelog` suffix for the name of `storeChangelogTopic`.

`ProcessorStateManager` uses...FIXME...for the base directory.

`ProcessorStateManager` manages **changelog partitions** that are the Kafka `partitions` of the registered `StateStores`.

Tip

Enable `ALL` logging levels for `org.apache.kafka.streams.processor.internals.ProcessorStateManager` logger to see what happens inside.

Add the following line to `log4j.properties`:

```
log4j.logger.org.apache.kafka.streams.processor.internals.ProcessorStateManager=
```

Refer to [Application Logging Using log4j](#).

Creating ProcessorStateManager Instance

`ProcessorStateManager` takes the following when created:

- Task ID
- Source Kafka [TopicPartitions](#)
- `isStandby` flag (to indicate whether used by `StandbyTask`, i.e. `true` or `StreamTask`, i.e. `false` that is used exclusively when [registering a state store](#))
- [StateDirectory](#)
- Names of the state stores and the names of the corresponding changelog topics (from the [ProcessorTopology](#))
- [ChangelogReader](#)

- `eosEnabled` Flag (that corresponds to `processing.guarantee` configuration property)
- `LogContext`

`ProcessorStateManager` initializes the internal registries and counters.

Registering State Store (and StateRestoreCallback)

— `register` Method

```
void register(
    StateStore store,
    StateRestoreCallback stateRestoreCallback)
```

Note	<code>register</code> is part of <code>StateManager Contract</code> to register a state store (with an associated <code>StateRestoreCallback</code>).
------	--

`register` prints out the following DEBUG message to the logs:

```
Registering state store [storeName] to its state manager
```

`register` looks up the given `StateStore` (by its name) in the `storeToChangelogTopic` internal registry.

If not found, `register` adds the `StateStore` in the `registeredStores`.

If the changelog topic is found, `register` finds the partition ID by the changelog topic and creates a Kafka `TopicPartition`. `register` finds the `RecordConverter` for the `StateStore`.

If the `ProcessorStateManager` is in `standby` mode, `register` prints out the following TRACE message to the logs and registers the topic with the input `StateRestoreCallback` and the `RecordConverter` in the `restoreCallbacks` and `recordConverters` registries, respectively.

```
Preparing standby replica of persistent state store [storeName] with changelog topic [topic]
```

If the `ProcessorStateManager` is not in `standby` mode, `register` prints out the following TRACE message to the logs, creates a `StateRestorer` and requests the `ChangelogReader` to register it.

```
Restoring state store [storeName] from changelog topic [topic] at checkpoint [offset]
```

`register` creates a `StateRestorer` and requests the `ChangelogReader` to register it.

`register` then adds the (store) partition to the `changelogPartitions` internal registry.

In the end, `register` registers the `stateStore` (in the `registeredStores` registry).

`register` throws an `IllegalArgumentException` if the name of the `StateStore` is `.checkpoint`.

```
[logPrefix]Illegal store name: .checkpoint
```

`register` throws an `IllegalArgumentException` if the `stateStore` is already registered (in the `registeredStores` registry).

```
[logPrefix]Store [storeName] has already been registered.
```

Closing ProcessorStateManager — `close` Method

```
void close(Map<TopicPartition, Long> ackedOffsets)
```

Note

`close` is part of `StateManager Contract` to...FIXME.

`close` ...FIXME

Flushing All State Stores — `flush` Method

```
void flush()
```

Note

`flush` is part of the `StateManager Contract` to flush all `state stores` registered with the state manager.

`flush` ...FIXME

Requesting Global StateStore By Name — `getGlobalStore` Method

```
StateStore getGlobalStore(String name)
```

Note

`getGlobalStore` is part of `StateManager Contract` to...FIXME.

`getGlobalStore` ...FIXME

Requesting StateStore By Name — `getStore` Method

```
StateStore getStore(String name)
```

Note

`getStore` is part of [StateManager Contract](#) to...FIXME.

`getStore` ...FIXME

reinitializeStateStoresForPartitions Method

```
void reinitializeStateStoresForPartitions(
    Collection<TopicPartition> partitions,
    InternalProcessorContext processorContext)
```

Note

`reinitializeStateStoresForPartitions` is part of [StateManager Contract](#) to...
FIXME.

`reinitializeStateStoresForPartitions` ...FIXME

storeChangelogTopic Static Method

```
static String storeChangelogTopic(
    String applicationId,
    String storeName)
```

`storeChangelogTopic` simply returns the following topic name:

```
[applicationId]-[storeName][STATE_CHANGELOG_TOPIC_SUFFIX]
```

Note

`storeChangelogTopic` is used when:

- `InternalTopologyBuilder` is requested to [buildProcessorNode](#) and [topicGroups](#)
- `CachingSessionStore`, `ChangeLoggingKeyValueBytesStore`, `ChangeLoggingSessionBytesStore`, `ChangeLoggingWindowBytesStore`, `InMemoryKeyValueStore`, `MemoryLRUCache`, `MeteredKeyValueBytesStore`, `MeteredSessionStore`, `MeteredWindowStore`, `RocksDBSegmentedBytesStore`, `RocksDBSessionStore`, `RocksDBWindowStore`, `CachingKeyValueStore` and `CachingWindowStore` are requested to `init`
- `StoreChangeLogger` is [created](#)

Finding Partition ID For Topic (Name) — `getPartition` Internal Method

```
int getPartition(String topic)
```

`getPartition` tries to find the [TopicPartition](#) for the input `topic` name (in the [partitionForTopic](#) internal registry).

If found, `getPartition` returns the [partition](#) of the [TopicPartition](#).

Otherwise, `getPartition` returns the partition of the [TaskId](#).

Note	<code>getPartition</code> is used when <code>ProcessorStateManager</code> is requested to register a StateStore , checkpointed and checkpoint .
------	---

checkpointed Method

```
Map<TopicPartition, Long> checkpointed()
```

Note	<code>checkpointed</code> is part of the Checkpointable Contract to...FIXME.
------	--

`checkpointed` ...FIXME

updateStandbyStates Method

```
List<ConsumerRecord<byte[], byte[]>> updateStandbyStates(
    TopicPartition storePartition,
    List<ConsumerRecord<byte[], byte[]>> records)
```

`updateStandbyStates` ...FIXME

Note	<code>updateStandbyStates</code> is used exclusively when <code>StandbyTask</code> is requested to update standby replicas of the state store .
------	---

Checkpointing Offsets (Writing Offsets to Checkpoint File) — `checkpoint` Method

```
void checkpoint(Map<TopicPartition, Long> checkpointableOffsets)
```

Note	<code>checkpoint</code> is part of the Checkpointable Contract to checkpoint offsets.
------	---

`checkpoint` requests the [ChangelogReader](#) for [restoredOffsets](#) and adds them to the [checkpointableOffsets](#) registry.

For every state store (in the [stores](#) internal registry), `checkpoint` ...FIXME

`checkpoint` creates a new [OffsetCheckpoint](#) (with the `.checkpoint` file in the [base directory](#)) unless [it was done already](#).

`checkpoint` prints out the following TRACE message to the logs:

```
Writing checkpoint: [checkpointableOffsets]
```

In the end, `checkpoint` requests the [OffsetCheckpoint](#) to [write](#) the [checkpointableOffsets](#).

registerGlobalStateStores Method

```
void registerGlobalStateStores(List<StateStore> stateStores)
```

`registerGlobalStateStores` ...FIXME

Note	<code>registerGlobalStateStores</code> is used exclusively when <code>StreamTask</code> is created .
------	--

Internal Properties

Name	Description
<code>restoreCallbacks</code>	<code>StateRestoreCallback</code> by changelog topic name (<code>Map<String, StateRestoreCallback></code>) Used when...FIXME
<code>partitionForTopic</code>	Kafka TopicPartitions by changelog topic name (<code>Map<String, TopicPartition></code>) Initialized and filled in when <code>ProcessorStateManager</code> is created Used exclusively when <code>ProcessorStateManager</code> is requested to find the partition ID for a topic (name) .
<code>recordConverters</code>	
<code>registeredStores</code>	

GlobalStateManager

GlobalStateManager is the [contract](#) for custom StateManagers that are [checkpointable](#) and...FIXME

```
package org.apache.kafka.streams.processor.internals;

interface GlobalStateManager extends StateManager {
    Set<String> initialize();
    void setGlobalProcessorContext(final InternalProcessorContext processorContext);
}
```

Table 1. GlobalStateManager Contract

Method	Description
initialize	Used when...FIXME
setGlobalProcessorContext	Used when...FIXME

GlobalStateManagerImpl

`GlobalStateManagerImpl` is a concrete [GlobalStateManager](#) (and a [AbstractStateManager](#)) that...FIXME

`GlobalStateManagerImpl` is created exclusively when `GlobalStreamThread` is requested to initialize (when `GlobalStreamThread` is started with [KafkaStreams](#)).

initialize Method

```
Set<String> initialize()
```

Note	<code>initialize</code> is part of GlobalStateManager Contract to...FIXME.
------	--

`initialize` ...FIXME

checkpoint Method

```
void checkpoint(final Map<TopicPartition, Long> offsets)
```

Note	<code>checkpoint</code> is part of Checkpointable Contract to...FIXME.
------	--

`checkpoint` ...FIXME

Creating GlobalStateManagerImpl Instance

`GlobalStateManagerImpl` takes the following when created:

- `LogContext`
- [ProcessorTopology](#)
- (Global) Kafka [Consumer](#) (`Consumer<byte[], byte[]>`)
- [StateDirectory](#)
- [StateRestoreListener](#)
- [StreamsConfig](#)

`GlobalStateManagerImpl` initializes the [internal registries and counters](#).

reinitializeStateStoresForPartitions Method

```
void reinitializeStateStoresForPartitions(
    final Collection<TopicPartition> partitions,
    final InternalProcessorContext processorContext)
```

Note

`reinitializeStateStoresForPartitions` is part of [StateManager Contract](#) to...
FIXME.

`reinitializeStateStoresForPartitions` ...FIXME

Flushing All State Stores — flush Method

```
void flush()
```

Note

`flush` is part of the [StateManager Contract](#) to flush all state stores registered with the state manager.

`flush` ...FIXME

restoreState Internal Method

```
void restoreState(
    StateRestoreCallback stateRestoreCallback,
    List<TopicPartition> topicPartitions,
    Map<TopicPartition, Long> highWatermarks,
    String storeName)
```

`restoreState` ...FIXME

Note

`restoreState` is used exclusively when `GlobalStateManagerImpl` is requested to register a state store.

Registering State Store — register Method

```
void register(
    final StateStore store,
    final StateRestoreCallback stateRestoreCallback)
```

Note

`register` is part of the [StateManager Contract](#) to register a state store.

```
register ...FIXME
```

Closing State Manager— close Method

```
void close(final Map<TopicPartition, Long> offsets) throws IOException
```

Note

`close` is part of the [StateManager Contract](#) to close the state manager.

```
close ...FIXME
```

Checkpointable Internal Contract

`Checkpointable` is the internal [contract](#) for [objects](#) with associated partition offsets that can be [checkpointed](#).

Note

[StateManager](#) is the one and only known direct extension contract of the [Checkpointable Contract](#).

Table 1. Checkpointable Contract

Method	Description
<code>checkpoint</code>	<pre>void checkpoint(final Map<TopicPartition, Long> offsets)</pre> <p>Checkpointing offsets Used when:</p> <ul style="list-style-type: none"> • <code>GlobalStateUpdateTask</code> is requested to flush state stores • <code>StandbyTask</code> is requested to flushAndCheckpointState (when requested to commit and suspend) • <code>StreamTask</code> is requested to commit (with exactly-once support disabled) and suspend (with exactly-once support enabled)
<code>checkpointed</code>	<pre>Map<TopicPartition, Long> checkpointed()</pre> <p>Used when:</p> <ul style="list-style-type: none"> • <code>GlobalStateUpdateTask</code> is requested to initialize • <code>StandbyTask</code> is requested to initialize state stores

OffsetCheckpoint

`offsetCheckpoint` is...FIXME

`offsetCheckpoint` is created when:

1. `AbstractStateManager` is created
2. `ProcessorStateManager` is requested to `checkpoint`

`offsetCheckpoint` only takes the **offset checkpoint file** when created.

Note	The default name of the file is <code>.checkpoint</code> .
------	--

`offsetCheckpoint` uses the internal **version** to maintain compatibility between checkpointing protocols. The current version is 0.

Loading Offset Checkpoint File — `read` Method

```
Map<TopicPartition, Long> read() throws IOException
```

`read` ...FIXME

	<code>read</code> is used when:
Note	<ul style="list-style-type: none"> • <code>GlobalStateManagerImpl</code> is requested to <code>initialize</code> • <code>ProcessorStateManager</code> is created

Writing Offsets to Checkpoint File — `write` Method

```
void write(final Map<TopicPartition, Long> offsets) throws IOException
```

`write` ...FIXME

	<code>write</code> is used when:
Note	<ul style="list-style-type: none"> • <code>AbstractStateManager</code> is requested to <code>reinitializeStateStoresForPartitions</code> • <code>GlobalStateManagerImpl</code> is requested to <code>checkpoint</code> • <code>ProcessorStateManager</code> is requested to <code>checkpoint</code>

Deleting Offset Checkpoint File — `delete` Method

```
void delete() throws IOException
```

`delete` simply deletes the [offset checkpoint file](#) if exists.

Note

`delete` is used exclusively when `ProcessorStateManager` is [created](#) (with [Exactly-Once Support \(EOS\)](#) enabled).

ChangelogReader Internal Contract

ChangelogReader is the internal contract for changelog readers that...FIXME

Table 1. ChangelogReader Contract

Method	Description
register	<pre>void register(StateRestorer restorer)</pre> <p>Registers a StateRestorer (for a changelog partition) Used exclusively when ProcessorStateManager is requested to register a persistent state store (with an underlying changelog topic)</p>
reset	<pre>void reset()</pre> <p>Used exclusively when TaskManager is requested to suspend all (active and standby) tasks and state (when RebalanceListener is requested to handle onPartitionsRevoked event (at the beginning of consumer rebalance))</p>
restore	<pre>Collection<TopicPartition> restore(RestoringTasks active)</pre> <p>Restores logging-enabled state stores using the RestoringTasks Used exclusively when TaskManager is requested to updateNewAndRestoringTasks (when the owning StreamThread is requested to poll records once and process them using active stream tasks while running the main record processing loop)</p>
restoredOffsets	<pre>Map<TopicPartition, Long> restoredOffsets()</pre> <p>Restores offsets (for persistent state stores) Used exclusively when ProcessorStateManager is requested to checkpoint offsets</p>
Note	StoreChangelogReader is the default and only known implementation of the ChangelogReader Contract in Kafka Streams.

StoreChangelogReader

`StoreChangelogReader` is the default [ChangelogReader](#).

`StoreChangelogReader` is [created](#) for a [StreamThread](#) for the only purpose of creating the [TaskCreator](#), the [StandbyTaskCreator](#) and the [TaskManager](#).

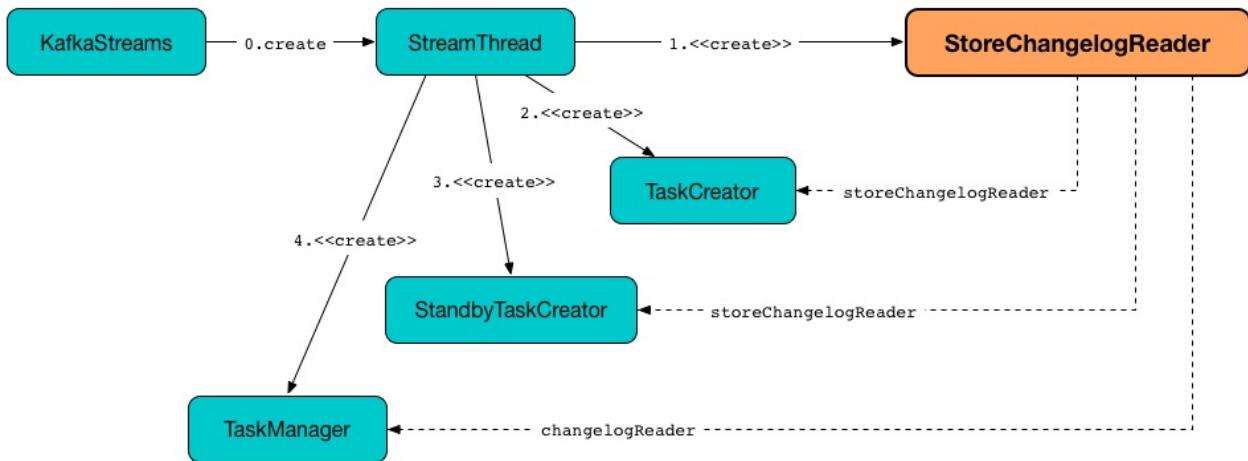


Figure 1. `StoreChangelogReader`

`StoreChangelogReader` is used for the following:

- [Tasks](#) to create a [ProcessorStateManager](#)
- [TaskManager](#) when [suspending](#) all (active and standby) tasks and [state](#) (at the beginning of consumer rebalance) and [updateNewAndRestoringTasks](#) (at the [end](#) of consumer rebalance)

Tip	<p>Enable <code>ALL</code> logging level for <code>org.apache.kafka.streams.processor.internals.StoreChangelogReader</code> logger to see what happens inside.</p> <p>Add the following line to <code>log4j.properties</code>:</p> <pre>log4j.logger.org.apache.kafka.streams.processor.internals.StoreChangelogReader=A</pre> <p>Refer to Application Logging Using log4j.</p>
------------	---

Creating `StoreChangelogReader` Instance

`StoreChangelogReader` takes the following to be created:

- **Restore Consumer** (`Consumer<byte[], byte[]>`)
- [Poll duration](#)

- User-Defined StateRestoreListener
- LogContext

`StoreChangelogReader` initializes the internal properties.

Poll Duration and StreamsConfig.POLL_MS_CONFIG

When created, `StoreChangelogReader` is given the poll duration that is configured using `StreamsConfig.POLL_MS_CONFIG` configuration property.

The poll duration is used exclusively for the restore Consumer to fetch data for changelog partitions (using `Consumer.poll`) when restoring active StreamTasks (from the changelog partitions).

User-Defined StateRestoreListener

When created, `StoreChangelogReader` is given a user-defined StateRestoreListener.

`StoreChangelogReader` uses the StateRestoreListener exclusively when registering a StateRestorer for a changelog partition so that the StateRestorer to be registered is associated with the listener.

Registering StateRestorer For Changelog Partition — register Method

```
void register(StateRestorer restorer)
```

Note	<code>register</code> is part of <code>ChangelogReader Contract</code> to register a <code>StateRestorer</code> (for a changelog partition).
------	--

`register` requests the given `StateRestorer` for the `partition`.

If the `stateRestorers` internal registry does not have the partition registered, `register` requests the `StateRestorer` to `setUserRestoreListener` with the `userStateRestoreListener` and adds it to the `stateRestorers` registry. `register` prints out the following TRACE message to the logs:

```
Added restorer for changelog [partition]
```

In the end, `register` adds the partition to the `needsInitializing` internal registry.

reset Method

```
void reset()
```

Note

`reset` is part of [ChangelogReader Contract](#) to...FIXME.

`reset` ...FIXME

restoredOffsets Method

```
Map<TopicPartition, Long> restoredOffsets()
```

Note

`restoredOffsets` is part of [ChangelogReader Contract](#) to restore offsets.

`restoredOffsets` returns pairs of `TopicPartition` and `restoredOffset` (from the associated `StateRestorer` that is [persistent](#) for the `state store` that is [persistent](#)).

Internally, for every pair of `TopicPartition` and `StateRestorer` (in the `stateRestorers` internal registry), `restoredOffsets` requests the `StateRestorer` to `restoredOffset` when the restorer is [persistent](#) (i.e. when the [associated state store](#) is [persistent](#)).

Restoring Active StreamTasks (From Changelog Partitions)

— restore Method

```
Collection<TopicPartition> restore(
    RestoringTasks active)
```

Note

`restore` is part of the [ChangelogReader Contract](#) to restore logging-enabled `state stores` using the `RestoringTasks`.

`restore` initializes (with the given active `RestoringTasks`) when the `needsInitializing` internal registry has at least one changelog partition (that was added when [registering a StateRestorer for a changelog partition](#)).

`restore` simply finishes with the [completely-restored changelog partitions](#) when there is no [changelog partitions that need restoring](#). `restore` also requests the `restore Consumer` to unsubscribe from changelog topics (`KafkaConsumer.unsubscribe`).

With at least one [changelog partition that needs restoring](#), `restore` requests the `restore Consumer` to fetch data for the changelog partitions (`KafkaConsumer.poll`) with the `poll duration`.

```
restore ...FIXME
```

`restore` removes all changelog partitions that are [completed](#) from the [needsRestoring](#) internal registry.

In the end, `restore` requests the [restore Consumer](#) to unsubscribe from changelog topics (`KafkaConsumer.unsubscribe`) when the [needsRestoring](#) internal registry has no changelog partition and simply finishes with the [completely-restored changelog partitions](#).

In case of `InvalidOffsetException`, `restore ...FIXME`

initialize Internal Method

```
void initialize(RestoringTasks active)
```

```
initialize ...FIXME
```

Note	<code>initialize</code> is used exclusively when <code>StateRestorer</code> is requested to restore (and there are needsInitializing changelog partitions).
------	---

Restoring State Stores From Changelog Topics

— startRestoration Internal Method

```
void startRestoration(
    Set<TopicPartition> initialized,
    RestoringTasks active)
```

`startRestoration` prints out the following DEBUG message to the logs:

```
Start restoring state stores from changelog topics [initialized]
```

`startRestoration` requests the [restore Consumer](#) for partition assignment, adds the `initialized` partitions and finally requests the [restore Consumer](#) to use the partitions only (aka *manual partition assignment*).

For every `initialized` partition, `startRestoration` uses the [stateRestorers](#) internal registry to find the associated `StateRestorer` that is then requested for the [checkpoint offset](#).

There are two possible cases of the checkpoint offsets.

When the checkpoint offset is [known](#), `startRestoration` prints out the following TRACE message to the logs:

```
Found checkpoint [checkpoint] from changelog [partition] for store [storeName].
```

`startRestoration` requests the [restore Consumer](#) to seek (*the fetch offsets*) for the partition to the checkpoint.

`startRestoration` looks up the partition in the [endOffsets](#) internal registry and prints out the following DEBUG message to the logs:

```
Restoring partition [partition] from offset [startingOffset] to endOffset [endOffset]
```

`startRestoration` requests the [StateRestorer](#) to [set the starting offset](#) (with the offset of the next record to be fetched for the partition using the [restore Consumer](#)).

`startRestoration` requests the [StateRestorer](#) to [restoreStarted](#).

When the checkpoint offset is [unknown](#), `startRestoration` prints out the following TRACE message to the logs:

```
Did not find checkpoint from changelog [partition] for store [storeName], rewinding to beginning.
```

`startRestoration` requests the [restore Consumer](#) to seek to the beginning (`KafkaConsumer.seekToBeginning`) for the partition.

`startRestoration` adds the partition to [needsPositionUpdate](#) local registry.

For every [stateRestorer](#) in the `startRestoration` local registry (for which the checkpoint offset was unknown), `startRestoration` requests the [StateRestorer](#) for the [partition](#).

`startRestoration` requests the given active [RestoringTasks](#) for the [restoring StreamTask](#) of the [changelog partition](#).

There are two possible cases of the restoring [StreamTask](#).

With [Exactly-Once Support enabled](#), `startRestoration` prints out the following INFO message to the logs:

```
No checkpoint found for task [id] state store [storeName] changelog [partition] with E OS turned on. Reinitializing the task and restore its state from the beginning.
```

`startRestoration` removes the partition from the [needsInitializing](#) internal registry (and the [initialized](#) local registry).

`startRestoration` requests the `StateRestorer` to [set the checkpoint offset](#) (with the offset of the next record to be fetched for the partition using the [restore Consumer](#)).

`startRestoration` requests the `StreamTask` to [reinitializeStateStoresForPartitions](#) with the partition.

With [Exactly-Once Support disabled](#), `startRestoration` prints out the following INFO message to the logs:

```
Restoring task [id]'s state store [storeName] from beginning of the changelog [partition]
```

`startRestoration` requests the [restore Consumer](#) for the offset of the next record to be fetched (*position*) for the partition to the `StateRestorer`.

`startRestoration` looks up the partition of the `StateRestorer` in the `endOffsets` internal registry and prints out the following DEBUG message to the logs:

```
Restoring partition [partition] from offset [position] to endOffset [endOffset]
```

`startRestoration` requests the `StateRestorer` to [set the starting offset](#) to the position (of the [restore Consumer](#)).

`startRestoration` requests the `StateRestorer` to [restoreStarted](#).

In the end, `startRestoration` adds all `initialized` partitions to the `needsRestoring` internal registry.

Note	<code>startRestoration</code> is used exclusively when <code>StoreChangelogReader</code> is requested to initialize (when requested to restore).
------	---

processNext Internal Method

```
long processNext(
    List<ConsumerRecord<byte[], byte[]>> records,
    StateRestorer restorer,
    Long endOffset)
```

`processNext` ...FIXME

Note	<code>processNext</code> is used exclusively when <code>StoreChangelogReader</code> is requested to restore active StreamTasks (from changelog partitions) .
------	--

Internal Properties

Name	Description
completedRestorers	Completely-restored changelog partitions (<code>Set<TopicPartition></code>)
endOffsets	
needsInitializing	<p>Changelog partitions (of <code>StateRestorers</code>) that need initializing (<code>Set<TopicPartition></code>)</p> <ul style="list-style-type: none"> • New changelog partitions added while registering a new StateRestorer • A changelog partition removed in initialize (restore and startRestoration) • All changelog partitions removed in reset <p>Used in restore</p>
needsRestoring	<p>Changelog partitions (of <code>StateRestorers</code>) that need restoring (<code>Set<TopicPartition></code>)</p> <ul style="list-style-type: none"> • New changelog partitions added while restoring state stores from their changelog topics (while initialize while restore) • All changelog partitions removed in reset <p>Used in restore</p>
partitionInfo	
stateRestorers	<p><code>StateRestorers</code> per partition of changelog topic of a state store (<code>Map<TopicPartition, StateRestorer></code>)</p> <ul style="list-style-type: none"> • New <code>StateRestorer</code> added in register • All <code>StateRestorers</code> removed in reset <p>Used in restore, initialize, and restoredOffsets</p>

StateRestorer

`StateRestorer` is a concrete `BatchingStateRestoreCallback` and a [StateRestoreListener](#).

`StateRestorer` uses a [CompositeRestoreListener](#) to propagate the events: `restoreStarted`, `restoreDone`, `restoreBatchCompleted`, and `restore`.

`StateRestorer` is [created](#) exclusively when `ProcessorStateManager` is requested to [register](#) a state store.

`StateRestorer` uses `-1` as a marker for an uninitialized [partition offset](#).

Creating StateRestorer Instance

`StateRestorer` takes the following to be created:

- Kafka [TopicPartition](#)
- [CompositeRestoreListener](#)
- Checkpoint offset
- `offsetLimit`
- `persistent` flag
- Name of the state store
- `RecordConverter`

`StateRestorer` initializes the [internal properties](#).

restoreBatchCompleted Method

```
void restoreBatchCompleted(long currentRestoredOffset, int numRestored)
```

`restoreBatchCompleted` ...FIXME

Note

`restoreBatchCompleted` is used exclusively when `storeChangelogReader` is requested to [processNext](#) (when requested to `restore`).

restoreStarted Method

```
void restoreStarted()
```

restoreStarted ...FIXME

Note

`restoreStarted` is used exclusively when `StoreChangelogReader` is requested to `startRestoration` (when requested to `initialize` as part of `restore`).

restoreDone Method

```
void restoreDone()
```

restoreDone ...FIXME

Note

`restoreDone` is used exclusively when `StoreChangelogReader` is requested to `processNext` (when requested to `restore`).

restore Method

```
void restore(final Collection<KeyValue<byte[], byte[]>> records)
```

restore ...FIXME

Note

`restore` is used exclusively when `StoreChangelogReader` is requested to `restore`.

Setting StateRestoreListener

— setUserRestoreListener Method

```
void setUserRestoreListener(StateRestoreListener userRestoreListener)
```

`setUserRestoreListener` simply requests the `CompositeRestoreListener` to use the input `StateRestoreListener`.

Note

`setUserRestoreListener` is used exclusively when `StoreChangelogReader` is requested to `register a StateRestorer`.

isPersistent Method

```
boolean isPersistent()
```

`isPersistent` returns the [persistent](#) flag.

Note	<code>isPersistent</code> is used exclusively when <code>StoreChangelogReader</code> is requested to restoredOffsets .
------	--

setStartingOffset Method

```
void setStartingOffset(long startingOffset)
```

`setStartingOffset` simply sets the [startingOffset](#) internal registry to the minimum value of the [offsetLimit](#) and the given `startingOffset`.

Note	<code>setStartingoffset</code> is used when...FIXME
------	---

Internal Properties

Name	Description
<code>checkpointOffset</code>	Checkpoint offset <ul style="list-style-type: none"> Initialized to the given checkpoint if defined or <code>-1</code> Used when...FIXME
<code>startingOffset</code>	Used when...FIXME

RestoringTasks Contract

`RestoringTasks` is the abstraction of restoring task managers that can find the restoring `StreamTask` for a partition.

Table 1. RestoringTasks Contract

Method	Description
<code>restoringTaskFor</code>	<pre>StreamTask restoringTaskFor(TopicPartition partition)</pre> <p>Finds the restoring <code>StreamTask</code> for a partition</p> <p>Used exclusively when <code>storeChangelogReader</code> is requested to <code>restore</code></p>
Note	<p><code>AssignedStreamsTasks</code> is the default and only known implementation of the <code>RestoringTasks Contract</code> in Kafka Streams.</p>

RecordBatchingStateRestoreCallback

RecordBatchingStateRestoreCallback is...FIXME

CompositeRestoreListener

`CompositeRestoreListener` is a concrete `BatchingStateRestoreCallback` and a [StateRestoreListener](#).

`CompositeRestoreListener` is created exclusively when `ProcessorStateManager` is requested to [register a state store](#) (and restores it from the changelog topic).

`CompositeRestoreListener` uses a [StateRestoreListener](#) in `onRestoreStart`, `onBatchRestored` and `onRestoreEnd` callbacks. A `stateRestoreListener` can be assigned using `setUserRestoreListener` method.

`CompositeRestoreListener` takes a single [StateRestoreCallback](#) when created.

Setting `StateRestoreListener` — `setUserRestoreListener` Method

```
void setUserRestoreListener(final StateRestoreListener userRestoreListener)
```

`setUserRestoreListener` simply sets the `userRestoreListener` to be the given [StateRestoreListener](#).

Note	<code>setUserRestoreListener</code> is used exclusively when <code>StateRestorer</code> is requested to set a StateRestoreListener .
------	--

StateRestoreCallbackAdapter — Converting StateRestoreCallbacks To RecordBatchingStateRestoreCallbacks

`StateRestoreCallbackAdapter` is a factory object (*adapter*) of `RecordBatchingStateRestoreCallbacks` given `StateRestoreCallbacks`.

Creating RecordBatchingStateRestoreCallback For StateRestoreCallback — `adapt` Factory Method

```
RecordBatchingStateRestoreCallback adapt(  
    StateRestoreCallback restoreCallback)
```

`adapt` simply creates a `RecordBatchingStateRestoreCallback` for a `StateRestoreCallback`.

Internally, `adapt` ...FIXME

Note	<p><code>adapt</code> is used when:</p> <ul style="list-style-type: none">• <code>CompositeRestoreListener</code> is created• <code>GlobalStateManagerImpl</code> is requested to <code>restoreState</code>• <code>ProcessorStateManager</code> is requested to <code>updateStandbyStates</code>
------	--