
Table of Contents

Introduction	1.1
Overview of Kafka	1.2

Kafka Clients / Producer API

KafkaProducer	2.1
Producer	2.1.1
ProducerConfig	2.2
ProducerRecord	2.3
Callback	2.4
Partitioner	2.5
ProducerInterceptor	2.6

Internals of Kafka Producer

Sender — Kafka Producer I/O Thread	3.1
RecordAccumulator	3.2
ProducerBatch	3.2.1
ProducerInterceptors	3.3
DefaultPartitioner	3.4
TransactionManager	3.5

Kafka Broker Services

KafkaServer — Kafka Broker	4.1
Kafka Server and Periodic Tasks	4.2
AdminManager	4.3
Authorizer	4.4
DelegationTokenManager	4.5
DynamicConfigManager	4.6

ConfigHandler	4.6.1
BrokerConfigHandler	4.6.2
ClientIdConfigHandler	4.6.3
TopicConfigHandler	4.6.4
UserConfigHandler	4.6.5
DynamicBrokerConfig	4.7
BrokerReconfigurable	4.7.1
DynamicConnectionQuota	4.7.1.1
DynamicListenerConfig	4.7.1.2
DynamicThreadPool	4.7.1.3
DynamicClientQuotaCallback	4.7.2
DynamicLogConfig	4.7.3
DynamicMetricsReporters	4.7.4
GroupCoordinator	4.8
GroupMetadataManager	4.8.1
GroupMetadata	4.8.2
Kafka	4.9
KafkaApis	4.10
KafkaController	4.11
KafkaHealthcheck	4.12
KafkaServerStartable	4.13
KafkaConfig	4.14
KafkaMetricsReporter	4.15
KafkaRequestHandlerPool	4.16
KafkaRequestHandler	4.16.1
LogManager	4.17
Log	4.17.1
LogSegment	4.17.2
LogCleanerManager	4.17.3
LogCleaner	4.17.4
CleanerThread	4.17.5
ProducerStateManager	4.17.6
LogConfig	4.17.7
MetadataCache	4.18

OffsetConfig	4.19
ReplicaManager	4.20
ReplicaFetcherManager	4.20.1
ReplicaFetcherThread	4.20.1.1
ReplicaAlterLogDirsManager	4.20.2
ReplicaAlterLogDirsThread	4.20.2.1
AbstractFetcherManager	4.20.3
AbstractFetcherThread	4.20.3.1
ReplicaFetcherBlockingSend	4.20.4
ReplicationQuotaManager	4.20.5
LogDirFailureHandler	4.20.6
Selector	4.21
Selectable	4.21.1
ShutdownableThread	4.22
SocketServer	4.23
Network Processor Thread	4.23.1
RequestChannel	4.24
RequestChannel.Request	4.24.1
TransactionCoordinator	4.25
TransactionMarkerChannelManager	4.25.1
InterBrokerSendThread	4.25.2
TransactionStateManager	4.26
QuotaManagers	4.27
ZkUtils	4.28
ZKRebalancerListener	4.29

Kafka Controller

ControllerContext	5.1
ControllerEventManager	5.2
ControllerEventThread	5.2.1
ControllerEvent	5.3
AutoPreferredReplicaLeaderElection	5.3.1

Startup	5.3.2
Reelect	5.3.3
TopicDeletion	5.3.4
ControllerState	5.4
ControllerChannelManager	5.5
ControllerBrokerRequestBatch	5.6
TopicDeletionManager	5.7
ReplicaStateMachine	5.8
PartitionStateMachine	5.9

Kafka Cluster

Partition	6.1
Replica	6.2
ReplicationUtils	6.3

Kafka Performance Metrics

KafkaMetricsGroup	7.1
BrokerTopicStats	7.2
BrokerTopicMetrics	7.3

Kafka Operations and Administration

ConfigCommand	8.1
ReassignPartitionsCommand — Partition Reassignment on Command Line	8.2
TopicCommand — Topic Management on Command Line	8.3
kafka-consumer-groups.sh	8.4
ConsumerGroupCommand	8.4.1
KafkaConsumerGroupService	8.4.2
ConsumerGroupService	8.4.2.1
KafkaAdminClient	8.5
AdminClient	8.5.1
AdminMetadataManager	8.5.2

AdminClientRunnable	8.5.3
AdminUtils	8.6

Kafka Features

Kafka Controller Election	9.1
Topic Replication	9.2
Topic Deletion	9.3
Transactional Producer	9.4
Idempotent Producer	9.5

Kafka Clients / Consumer API

Consumer Contract — Kafka Clients for Consuming Records	10.1
KafkaConsumer	10.1.1
MockConsumer	10.1.2
ConsumerRecord	10.2
OffsetAndMetadata	10.3
OffsetAndTimestamp	10.4
OffsetCommitCallback	10.5
ConsumerRebalanceListener	10.6
ConsumerConfig — Configuration Properties for KafkaConsumer	10.7
CommitFailedException	10.8
InvalidOffsetException	10.9
NoOffsetForPartitionException	10.10
OffsetOutOfRangeException	10.11
RetriableCommitFailedException	10.12
ConsumerInterceptor	10.13
PartitionAssignor Contract	10.14
RangeAssignor	10.14.1
RoundRobinAssignor	10.14.2
StickyAssignor	10.14.3
AbstractPartitionAssignor	10.14.4
ConsumerCoordinator	10.15

AbstractCoordinator Contract	10.15.1
HeartbeatThread	10.15.2
GroupCoordinatorMetrics	10.15.3
ConsumerNetworkClient	10.16
ConsumerMetrics	10.17
Fetcher	10.18
RequestFutureListener	10.19
SubscriptionState	10.20
RequestFuture	10.21
RequestFutureAdapter Contract	10.22
CoordinatorResponseHandler Contract	10.22.1
FindCoordinatorResponseHandler	10.22.2
HeartbeatResponseHandler	10.22.3
JoinGroupResponseHandler	10.22.4
OffsetCommitResponseHandler	10.22.5
SyncGroupResponseHandler	10.22.6

Kafka Architecture

Broker Nodes — Kafka Servers	11.1
Broker	11.1.1
Topics	11.2
Messages	11.3
Kafka Clients	11.4
Producers	11.4.1
Consumers	11.4.2
Clusters	11.5

Kafka Monitoring (Metrics)

Metrics	12.1
Sensor	12.2
MetricsReporter	12.3
ProducerMetrics	12.4

SenderMetrics	12.5
---------------	------

Kafka Tools

Kafka Tools	13.1
kafka-configs.sh	13.1.1
kafka-topics.sh	13.1.2

Kafka Configuration

Properties	14.1
bootstrap.servers	14.1.1
client.id	14.1.2
enable.auto.commit	14.1.3
group.id	14.1.4
retry.backoff.ms	14.1.5
Logging	14.2

Tips and Tricks

Gradle Tips	15.1
Zookeeper Tips	15.2
Kafka in Scala REPL for Interactive Exploration	15.3
Running Kafka Broker in Docker	15.4

Kafka Clients

KafkaClient	16.1
NetworkClient — Non-Blocking Network KafkaClient	16.1.1
RequestCompletionHandler Contract	16.2
MetadataUpdater	16.3
DefaultMetadataUpdater	16.3.1
Metadata	16.4
Listener Contract — Intercepting Metadata Updates	16.4.1

ClientRequest	16.5
ClientResponse	16.6
StaleMetadataException	16.7
NetworkClientUtils	16.8

Kafka Common

Cluster	17.1
Cluster (deprecated)	17.1.1
ClusterConnectionStates	17.2
ClusterResourceListener (and ClusterResourceListeners Collection)	17.3
NotificationHandler Contract	17.4
ZkNodeChangeNotificationListener	17.5
Configurable Contract	17.6
Reconfigurable	17.7
MemoryRecordsBuilder	17.8

Kafka Common / Requests

AbstractRequestResponse Contract	18.1
AbstractRequest Contract	18.1.1
AbstractRequest.Builder Contract	18.1.2
AbstractResponse	18.1.3
DescribeLogDirsRequest	18.2
FindCoordinatorRequest	18.3
FindCoordinatorResponse	18.4
HeartbeatRequest	18.5
JoinGroupRequest	18.6
JoinGroupResponse	18.7
LeaderAndIsrRequest	18.8
MetadataRequest	18.9
MetadataResponse	18.10
OffsetCommitRequest	18.11
ProduceRequest	18.12

SyncGroupRequest	18.13
UpdateMetadataRequest	18.14
RequestContext	18.15

Kafka Common / Serialization

Serializer Contract	19.1
Deserializer Contract	19.2
Serde Contract	19.3
Serdess Factory Object	19.4

Kafka Security

SimpleAclAuthorizer	20.1
-------------------------------------	------

Varia / Misc

KafkaScheduler	21.1
Scheduler	21.2
ZooKeeperClient	21.3
KafkaZkClient — Higher-Level Kafka-Specific ZooKeeper Client	21.4
AdminZkClient	21.5
ZkAclChangeStore	21.6

Kafka Connect

WorkerGroupMember	22.1
ConnectDistributed	22.2

Kafka Demos

Demo: Kafka Controller Election	23.1
---	------

Appendix

Further reading or watching

24.1

The Internals of Apache Kafka 2.3.0

Welcome to **The Internals of Apache Kafka** gitbook! I'm very excited to have you here and hope you will enjoy exploring the internals of Apache Kafka 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 Spark](#), [Spark SQL](#), [The Internals of Spark Structured Streaming](#), and [Kafka Streams](#).

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

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

Overview of Kafka

[Apache Kafka](#) is an open source project for a distributed publish-subscribe messaging system rethought as a distributed commit log.

Kafka stores messages in [topics](#) that are [partitioned](#) and replicated across multiple [brokers](#) in a cluster. [Producers](#) send messages to topics from which [consumers](#) read.

Language Agnostic — producers and consumers use binary protocol to talk to a Kafka cluster.

Messages are byte arrays (with String, JSON, and Avro being the most common formats). If a message has a key, Kafka makes sure that all messages of the same key are in the same partition.

Consumers may be grouped in a [consumer group](#) with multiple consumers. Each consumer in a consumer group will read messages from a unique subset of partitions in each topic they subscribe to. Each message is delivered to one consumer in the group, and all messages with the same key arrive at the same consumer.

Durability — Kafka does not track which messages were read by each consumer. Kafka keeps all messages for a finite amount of time, and it is consumers' responsibility to track their location per topic, i.e. [offsets](#).

It is worth to note that Kafka is often compared to the following open source projects:

1. [Apache ActiveMQ](#) and [RabbitMQ](#) given they are message broker systems, too.
2. [Apache Flume](#) for its ingestion capabilities designed to send data to [HDFS](#) and [Apache HBase](#).

KafkaProducer — Main Class For Kafka Producers

`KafkaProducer` is the default [Producer](#) client in Apache Kafka.

`KafkaProducer` is the class that a Kafka developer uses to [send messages](#) to a Kafka cluster.

```
// Necessary imports
import org.apache.kafka.clients.producer.KafkaProducer
import org.apache.kafka.clients.producer.ProducerConfig
import org.apache.kafka.common.serialization.StringSerializer

// Creating a KafkaProducer
import java.util.Properties
val props = new Properties()
props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, classOf[StringSerializer].getName)
props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, classOf[StringSerializer].getName)
props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, ":9092")
val producer = new KafkaProducer[String, String](props)

// Creating a record to be sent
import org.apache.kafka.clients.producer.ProducerRecord
val r = new ProducerRecord[String, String]("0", "this is a message")

// Sending the record (with no Callback)
import java.util.concurrent.Future
import org.apache.kafka.clients.producer.RecordMetadata
val metadataF: Future[RecordMetadata] = producer.send(r)
```

`KafkaProducer` uses the [ProducerConfig.INTERCEPTOR_CLASSES_CONFIG](#) configuration property to define [ProducerInterceptors](#) that are notified (about successes and failures) every time `KafkaProducer` is requested to [send a record to a topic](#).

`KafkaProducer` uses the [ProducerConfig.PARTITIONER_CLASS_CONFIG](#) configuration property to define the [Partitioner](#) that is used (to [compute the partition for a record](#)) when `KafkaProducer` is requested to [partition](#) (when requested to [send a ProducerRecord to a Kafka cluster asynchronously](#)).

Internally, `KafkaProducer` uses the [Kafka producer I/O thread](#) that is responsible for sending produce requests to a Kafka cluster (on [kafka-producer-network-thread](#) daemon thread of execution). The thread is started right when `KafkaProducer` is [created](#).

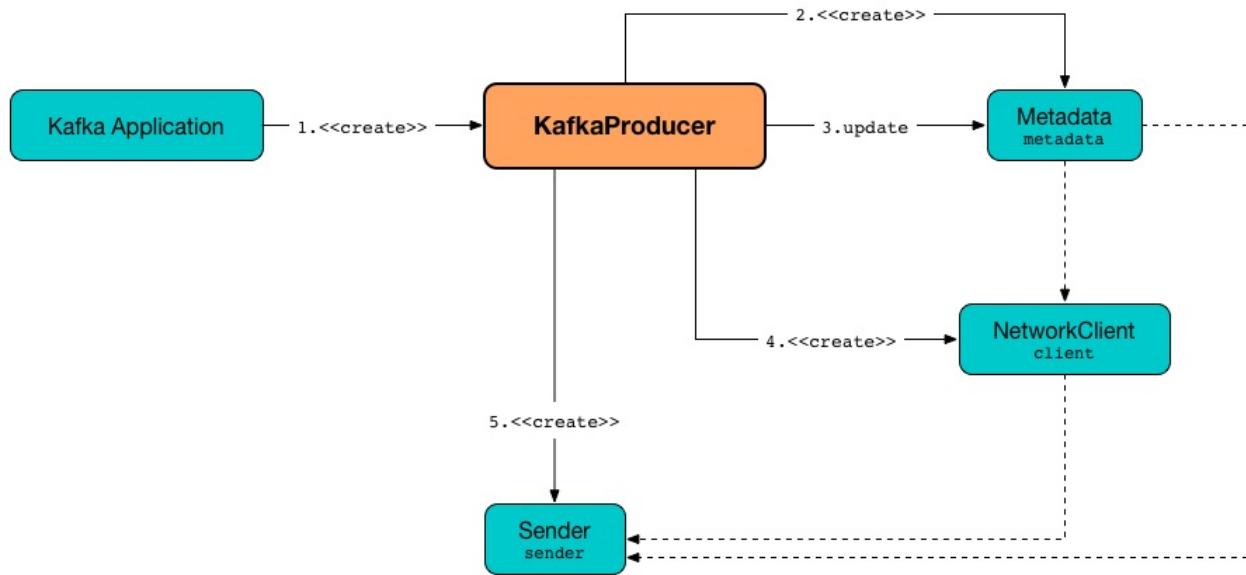


Figure 1. KafkaProducer

KafkaProducer uses **producer-metrics** for...FIXME

Tip

Enable ALL logging levels for `org.apache.kafka.clients.producer.KafkaProducer` logger to see what happens inside.

Add the following line to `config/log4j.properties` :

```
log4j.logger.org.apache.kafka.clients.producer.KafkaProducer=ALL
```

Refer to [Logging](#).

Performance Metrics

KafkaProducer exposes performance metrics.

Table 1. KafkaProducer's Metrics

Metric Name	Recording Level	Description
buffer-exhausted-records	INFO	Recorded when KafkaProducer is requested to send a ProducerRecord to a Kafka cluster asynchronously and BufferExhaustedException is reported.
errors		Recorded when KafkaProducer is requested to send a ProducerRecord to a Kafka cluster asynchronously and there were exceptions reported (e.g. ApiException , InterruptedException , BufferExhaustedException and KafkaException)
produce-throttle-time-avg		Recorded when NetworkClient is requested to parseStructMaybeUpdateThrottleTimeMetrics (and a request was throttled due to quota violation)
produce-throttle-time-max		Recorded when NetworkClient is requested to parseStructMaybeUpdateThrottleTimeMetrics (and a request was throttled due to quota violation)

Sending ProducerRecord to Kafka Cluster Asynchronously With Producer Interceptors — send Method

```
Future<RecordMetadata> send(
    ProducerRecord<K, V> record) (1)
Future<RecordMetadata> send(
    ProducerRecord<K, V> record,
    Callback callback)
```

1. Uses no callback (null)

Note

send is a part of Producer Contract to send a ProducerRecord to a Kafka cluster asynchronously.

send requests the ProducerInterceptors to pass the ProducerRecord through a chain of registered producer interceptors.

In the end, send sends a ProducerRecord to a Kafka cluster asynchronously (with the intercepted ProducerRecord and the Callback).

Sending ProducerRecord to Kafka Cluster Asynchronously

— `doSend` Internal Method

```
Future<RecordMetadata> doSend(
    ProducerRecord<K, V> record,
    Callback callback)
```

`doSend` ...FIXME

`doSend` requests the [RecordAccumulator](#) to append the serialized key and value.

`doSend` ...FIXME

Note	<code>doSend</code> is used exclusively when <code>KafkaProducer</code> is requested to send a ProducerRecord to a Kafka cluster asynchronously with producer interceptors.
------	---

Configuring ClusterResourceListeners

— `configureClusterResourceListeners` Internal Method

```
ClusterResourceListeners configureClusterResourceListeners(
    Serializer<K> keySerializer,
    Serializer<V> valueSerializer,
    List<?>... candidateLists)
```

`configureClusterResourceListeners` creates a [ClusterResourceListeners](#) and registers `ClusterResourceListener` instances from the input `candidateLists`, `keySerializer` and `valueSerializer`.

Note	<code>configureClusterResourceListeners</code> is used exclusively when <code>KafkaProducer</code> is created (to create the Metadata) with the following input arguments: <ul style="list-style-type: none"> • <code>key</code> and <code>value</code> serializers (defined when <code>KafkaProducer</code> is created) • <code>ProducerInterceptors</code> from <code>interceptor.classes</code> Kafka property • <code>MetricsReporters</code> from <code>metric.reporters</code> Kafka property
------	---

Requesting Partitions for Topic — `partitionsFor` Method

```
List<PartitionInfo> partitionsFor(String topic)
```

Note	<code>partitionsFor</code> is a part of Producer Contract .
------	---

`partitionsFor` waits on cluster metadata for the input `topic` and `max.block.ms` time. Once retrieved, `partitionsFor` requests cluster for the `partitions`.

Waiting for Cluster Metadata (with Partitions for Topic) — `waitOnMetadata` Internal Recursive Method

```
ClusterAndWaitTime waitOnMetadata(
    String topic,
    Integer partition,
    long maxWaitMs) throws InterruptedException
```

`waitOnMetadata` adds the input `topic` to [Metadata](#).

`waitOnMetadata` first checks if the available cluster metadata could be current enough.

`waitOnMetadata` requests [Metadata](#) for the [current cluster information](#) and then requests the cluster for the [number of partitions](#) of the input `topic`.

If the cluster metadata is not current enough (i.e. the number of partitions is unavailable or the `partition` is above the current count), `waitOnMetadata` prints out the following TRACE message to the logs:

```
Requesting metadata update for topic [topic].
```

`waitOnMetadata` requests [Metadata](#) for [update](#) and requests [Sender](#) to [wake up](#).

`waitOnMetadata` then requests [Metadata](#) to [wait for a metadata update](#) and then [Metadata](#) for the [current cluster information](#).

`waitOnMetadata` keeps doing it until the [number of partitions](#) of the input `topic` is available.

`waitOnMetadata` reports a `TimeoutException` when `maxWaitMs` has elapsed.

```
Failed to update metadata after [maxWaitMs] ms.
```

`waitOnMetadata` reports a `TopicAuthorizationException` when the access to the `topic` is unauthorized.

`waitOnMetadata` reports a `KafkaException` when the `partition` is above the number of available partitions.

```
Invalid partition given with record: [partition] is not in the range [0...[partitionsCount]).
```

Note

`waitOnMetadata` is used when `KafkaProducer` is requested for the partitions of a topic and to send a ProducerRecord to a Kafka cluster asynchronously.

Creating KafkaProducer Instance

`KafkaProducer` takes the following when created:

- `ProducerConfig`
- `Serializer` for keys
- `Serializer` for values
- `Metadata`
- `KafkaClient`

`KafkaProducer` initializes the internal registries and counters.

While being created, `KafkaProducer` saves the `ProducerConfig` in the `producerConfig` internal registry and the `time` becomes `SYSTEM`.

`KafkaProducer` sets the `clientId` as the `ProducerConfig.CLIENT_ID_CONFIG` or uses `producer-[id]`.

`KafkaProducer` prints out the following `TRACE` message to the logs:

```
Starting the Kafka producer
```

`KafkaProducer` creates a `MetricConfig` with the following:

- Number of samples as `ProducerConfig.METRICS_NUM_SAMPLES_CONFIG`
- Time window of `ProducerConfig.METRICS_SAMPLE_WINDOW_MS_CONFIG` milliseconds
- Recording level as `ProducerConfig.METRICS_RECORDING_LEVEL_CONFIG`
- Metrics tags with a single pair of `client-id` and the `clientId`

`KafkaProducer` uses the `ProducerConfig.METRIC_REPORTER_CLASSES_CONFIG` as the `MetricsReporters` and adds the `JmxReporter` (with `kafka.producer` prefix).

KafkaProducer sets the metrics as a new Metrics (with the MetricConfig , the MetricsReporters and the time).

KafkaProducer sets the partitioner as ProducerConfig.PARTITIONER_CLASS_CONFIG.

KafkaProducer sets the keySerializer as follows:

- ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG when the input keySerializer was not defined and immediately requests the serializer to configure itself
- The input keySerializer

KafkaProducer sets the valueSerializer as follows:

- ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG when the input keySerializer was not defined and immediately requests the serializer to configure itself
- The input valueSerializer

KafkaProducer sets the interceptors as ProducerConfig.INTERCEPTOR_CLASSES_CONFIG.

KafkaProducer sets the following:

- maxRequestSize as ProducerConfig.MAX_REQUEST_SIZE_CONFIG
- totalMemorySize as ProducerConfig.BUFFER_MEMORY_CONFIG
- compressionType as ProducerConfig.COMPRESSION_TYPE_CONFIG
- maxBlockTimeMs as ProducerConfig.MAX_BLOCK_MS_CONFIG
- requestTimeoutMs as ProducerConfig.REQUEST_TIMEOUT_MS_CONFIG

KafkaProducer creates a new ApiVersions for the apiVersions.

KafkaProducer creates a new RecordAccumulator for the accumulator with the following configuration properties:

- ProducerConfig.BATCH_SIZE_CONFIG
- totalMemorySize, i.e. ProducerConfig.BUFFER_MEMORY_CONFIG
- compressionType, i.e. ProducerConfig.COMPRESSION_TYPE_CONFIG
- ProducerConfig.LINGER_MS_CONFIG
- ProducerConfig.RETRY_BACKOFF_MS_CONFIG

KafkaProducer sets the metadata as follows:

- Creates a new `Metadata` (with `ProducerConfig.RETRY_BACKOFF_MS_CONFIG`, `ProducerConfig.METADATA_MAX_AGE_CONFIG` and `configureClusterResourceListeners`) and immediately requests the `Metadata` to update (with `ProducerConfig.BOOTSTRAP_SERVERS_CONFIG`)
- The input `metadata` if given

`KafkaProducer` creates a new `NetworkClient` (unless the input `KafkaClient` was given) with the following configuration properties:

- `ProducerConfig.CONNECTIONS_MAX_IDLE_MS_CONFIG`
- `ProducerConfig.RECONNECT_BACKOFF_MS_CONFIG`
- `ProducerConfig.RECONNECT_BACKOFF_MAX_MS_CONFIG`
- `ProducerConfig.SEND_BUFFER_CONFIG`
- `ProducerConfig.RECEIVE_BUFFER_CONFIG`
- `ProducerConfig.REQUEST_TIMEOUT_MS_CONFIG`

`KafkaProducer` creates a new `Sender` as the `sender` with the following configuration properties:

- `ProducerConfig.MAX_REQUEST_SIZE_CONFIG`
- `ProducerConfig.REQUEST_TIMEOUT_MS_CONFIG`
- `ProducerConfig.RETRY_BACKOFF_MS_CONFIG`

`KafkaProducer` sets the `transactionManager` as `configureTransactionState`.

`KafkaProducer` `configureRetries` for the number of retries.

`KafkaProducer` `configureInflightRequests` for the maximum number of in-flight requests.

`KafkaProducer` `configureAcks` for acks.

`KafkaProducer` creates a new `ProducerMetrics` (with the `metrics`).

`KafkaProducer` starts the `kafka-producer-network-thread` daemon thread of execution for the `sender`.

`KafkaProducer` requests the `ProducerConfig` to `logUnused`.

`KafkaProducer` registers the `AppInfo MBean` (with `kafka.producer` JMX prefix, the `clientId` and the `metrics`).

In the end, `KafkaProducer` prints out the following DEBUG message to the logs:

```
Kafka producer started
```

In case of any errors, `KafkaProducer` [closes](#) itself with `0` millis timeout and throws a `KafkaException`:

```
Failed to construct kafka producer
```

Computing Partition For ProducerRecord — `partition` Internal Method

```
int partition(
    ProducerRecord<K, V> record,
    byte[] serializedKey,
    byte[] serializedValue,
    Cluster cluster)
```

`partition` requests the `ProducerRecord` for the [partition](#) and return it when specified. Otherwise, `partition` requests the [Partitioner](#) for the `partition`.

Note	<code>partition</code> is used exclusively when <code>KafkaProducer</code> is requested to send a <code>ProducerRecord</code> to a Kafka cluster asynchronously.
------	--

beginTransaction Method

```
void beginTransaction() throws ProducerFencedException
```

Note	<code>beginTransaction</code> is part of the Producer Contract to...FIXME.
------	--

`beginTransaction` simply requests the internal [TransactionManager](#) to `beginTransaction`.

`beginTransaction` throws an `IllegalStateException` when the [TransactionManager](#) is undefined (`null`).

```
Cannot use transactional methods without enabling transactions by setting the transactional.id configuration property
```

configureTransactionState Static Internal Method

```
TransactionManager configureTransactionState(
    ProducerConfig config,
    LogContext logContext,
    Logger log)
```

configureTransactionState ...FIXME

Note	configureTransactionState is used exclusively when KafkaProducer is created (and initializes a TransactionManager).
------	---

Closing Kafka Producer— close Method

```
void close() (1)
void close(Duration timeout) (2)
// private API
void close(Duration timeout, boolean swallowException)
```

1. Uses the maximum timeout (Long.MAX_VALUE)
2. Disables swallowException flag (false)

Note	close is a part of Producer Contract .
------	--

close ...FIXME

Flushing Accumulated Records— flush Method

```
void flush()
```

Note	flush is a part of Producer Contract .
------	--

flush ...FIXME

initTransactions Method

```
void initTransactions()
```

Note	initTransactions is a part of Producer Contract .
------	---

initTransactions ...FIXME

sendOffsetsToTransaction Method

```
void sendOffsetsToTransaction(
    Map<TopicPartition, OffsetAndMetadata> offsets,
    String consumerGroupId)
throws ProducerFencedException
```

Note

`sendOffsetsToTransaction` is a part of [Producer Contract](#).

`sendOffsetsToTransaction` ...FIXME

commitTransaction Method

```
void commitTransaction()
throws ProducerFencedException
```

Note

`commitTransaction` is a part of [Producer Contract](#).

`commitTransaction` ...FIXME

abortTransaction Method

```
void abortTransaction()
throws ProducerFencedException
```

Note

`abortTransaction` is a part of [Producer Contract](#).

`abortTransaction` ...FIXME

Internal Properties

Name	Description
accumulator	<p><code>RecordAccumulator</code></p> <p>Used when:</p> <ul style="list-style-type: none"> • Sending a <code>ProducerRecord</code> to a Kafka cluster asynchronously • Creating a new Sender • Flushing accumulated records

	<p>Client ID per CLIENT_ID_CONFIG (if defined) or <code>producer-[number]</code></p>		
<code>clientId</code>	<p>Used when:</p> <ul style="list-style-type: none"> • <code>KafkaProducer</code> is requested for logging • NetworkClient is created 		
<code>interceptors</code>	<p>ProducerInterceptors that are notified (about successes and failures) when <code>KafkaProducer</code> is requested to send a record to a topic</p> <p>The <code>ProducerInterceptors</code> are initialized when the <code>KafkaProducer</code> is created using ProducerConfig.INTERCEPTOR_CLASSES_CONFIG configuration property.</p>		
<code>ioThread</code>	<p>kafka-producer-network-thread daemon thread of execution for the sender</p>		
<code>maxBlockTimeMs</code>	<p>Time <code>KafkaProducer</code> uses to block when requesting partitions for a topic.</p> <table border="1"> <tr> <td>Note</td><td>Use max.block.ms Kafka property to set the value.</td></tr> </table>	Note	Use max.block.ms Kafka property to set the value.
Note	Use max.block.ms Kafka property to set the value.		
<code>metadata</code>	<p>Metadata</p> <ul style="list-style-type: none"> • Created when <code>KafkaProducer</code> is created with the following properties: <ul style="list-style-type: none"> ◦ retry.backoff.ms for refreshBackoffMs ◦ metadata.max.age.ms for metadataExpireMs ◦ allowAutoTopicCreation flag enabled ◦ topicExpiryEnabled flag enabled • Updated with a bootstrap cluster when <code>KafkaProducer</code> is created • Used in waitOnMetadata 		
<code>partitioner</code>	<p>Partitioner that is used (to compute the partition for a record) when <code>KafkaProducer</code> is requested to partition (when requested to send a ProducerRecord to a Kafka cluster asynchronously)</p> <p>The <code>Partitioner</code> is initialized when the <code>KafkaProducer</code> is created using ProducerConfig.PARTITIONER_CLASS_CONFIG configuration property.</p>		

producerConfig	ProducerConfig
sender	Kafka producer I/O thread (aka <code>Sender</code>) that is started when <code>KafkaProducer</code> is created.
time	Time abstraction (with <code>SYSTEM</code> being the default).
transactionManager	<p><code>TransactionManager</code></p> <p>Used when:</p> <ul style="list-style-type: none">• <code>KafkaProducer</code> is created (and creates a <code>RecordAccumulator</code> and a <code>Sender</code>)• For transactional public methods: <code>initTransactions</code>, <code>beginTransaction</code>, <code>sendOffsetsToTransaction</code>, <code>commitTransaction</code>, <code>abortTransaction</code>• Sending a <code>ProducerRecord</code> to a Kafka cluster asynchronously

Producer Contract

Producer is the [contract](#) of Kafka producers.

Note	KafkaProducer is the one and only known implementation of the Producer Contract in Apache Kafka.
------	--

Table 1. Producer Contract

Method	Description
abortTransaction	<pre>void abortTransaction() throws ProducerFencedException</pre> <p>Used when...FIXME</p>
beginTransaction	<pre>void beginTransaction() throws ProducerFencedException</pre> <p>Used when...FIXME</p>
close	<pre>void close() void close(long timeout, TimeUnit unit)</pre> <p>Used when...FIXME</p>
commitTransaction	<pre>void commitTransaction() throws ProducerFencedException</pre> <p>Used when...FIXME</p>
flush	<pre>void flush()</pre> <p>Used when...FIXME</p>
initTransactions	<pre>void initTransactions()</pre> <p>Initializes transactions (when the <code>transactional.id</code> is configured)</p>
metrics	<pre>Map<MetricName, ? extends Metric> metrics()</pre>

	The performance metrics
partitionsFor	<pre>List<PartitionInfo> partitionsFor(String topic)</pre> <p>The partition metadata for the given topic. This can be used for custom partitioning.</p> <p>Used when...FIXME</p>
send	<pre>Future<RecordMetadata> send(ProducerRecord<K, V> record) Future<RecordMetadata> send(ProducerRecord<K, V> record, Callback callback)</pre> <p>Sending a <code>ProducerRecord</code> to a Kafka cluster asynchronously</p> <p>Used when...FIXME</p>
sendOffsetsToTransaction	<pre>void sendOffsetsToTransaction(Map<TopicPartition, OffsetAndMetadata> offsets, String consumerGroupId) throws ProducerFencedException</pre> <p>Used when...FIXME</p>

ProducerConfig

`ProducerConfig` is the configuration of a [Kafka Producer](#).

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

- ...

Table 1. ProducerConfig's Configuration Properties

Name / Default Value / Property	Description
<code>ACKS_CONFIG</code>	<p>The number of acknowledgments a producer requires the leader to have received before considering a request complete. This controls the durability of records that are sent.</p> <p>Default: <code>1</code></p> <p>Property: <code>acks</code></p> <p>The following settings are allowed:</p> <ul style="list-style-type: none"> • <code>0</code> - a producer will not wait for any acknowledgment from the server at all. The record will be immediately added to the socket buffer and considered sent. No guarantee can be made that the server has received the record in this case, and the <code>retries</code> configuration will not take effect (as the client won't generally know of any failures). The offset given back for each record will always be set to <code>-1</code>. • <code>1</code> (default) - the leader will write the record to its local log but will respond without awaiting full acknowledgement from all followers. In this case should the leader fail immediately after acknowledging the record but before the followers have replicated it then the record will be lost. • <code>all</code> or <code>-1</code> - the leader will wait for the full set of in-sync replicas to acknowledge the record. This guarantees that the record will not be lost as long as at least one in-sync replica remains alive. This is the strongest available guarantee.
	Host:port pairs to use to establish the initial connection to a Kafka cluster. The client will make use of all servers irrespective of which

	<p>servers are specified here for bootstrapping, i.e. this list only impacts the initial hosts used to discover the full set of servers.</p> <p>Default: (empty)</p> <p>BOOTSTRAP_SERVERS_CONFIG</p> <p>Property: <code>bootstrap.servers</code></p> <p>This list should be in the form <code>host1:port1,host2:port2,...</code></p> <p>Since these servers are just used for the initial connection to discover the full cluster membership (which may change dynamically), this list need not contain the full set of servers (you may want more than one, though, in case server is down).</p>
BATCH_SIZE_CONFIG	<p>Batch size (in bytes)</p> <p>Default: 16384</p> <p>Property: <code>batch.size</code></p> <p>A Kafka producer will attempt to batch records together into fewer requests whenever multiple records are being sent to the same partition. This helps performance on both the client and the server.</p> <p>No attempt will be made to batch records larger than this size.</p> <p>Requests sent to brokers will contain multiple batches, one for each partition with data available to be sent.</p> <p>A small batch size will make batching less common and may reduce throughput (a batch size of zero will disable batching entirely).</p> <p>A very large batch size may use memory a bit more wastefully as we will always allocate a buffer of the specified batch size in anticipation of additional records.</p> <p>Must be at least 0</p>
	<p>The total bytes of memory a Kafka producer can use to buffer records waiting to be sent to the server.</p> <p>Default: <code>32 * 1024 * 1024</code></p> <p>Property: <code>buffer.memory</code></p>

BUFFER_MEMORY_CONFIG	<p>If records are sent faster than they can be delivered to the server the producer will block for <code>MAX_BLOCK_MS_CONFIG</code> after which it will throw an exception.</p> <p>This setting should correspond roughly to the total memory the producer will use, but is not a hard bound since not all memory the producer uses is used for buffering. Some additional memory will be used for compression (if compression is enabled) as well as for maintaining in-flight requests.</p>
CLIENT_ID_CONFIG	<p>An id string to pass to the server when making requests. The purpose of this is to be able to track the source of requests beyond just ip/port by allowing a logical application name to be included in server-side request logging.</p> <p>Default: (empty)</p> <p>Property: <code>client.id</code></p>
COMPRESSION_TYPE_CONFIG	<p>The compression type for all data generated by the producer. The default is <code>none</code> (i.e. no compression).</p> <p>Default: <code>none</code></p> <p>Property: <code>compression.type</code></p> <p>Valid values are <code>none</code>, <code>gzip</code>, <code>snappy</code>, or <code>lz4</code></p> <p>Compression is of full batches of data, so the efficacy of batching will also impact the compression ratio (more batching means better compression).</p>
CONNECTIONS_MAX_IDLE_MS_CONFIG	<p>Close idle connections after the number of milliseconds specified by this config.</p> <p>Default: <code>9 * 60 * 1000</code></p> <p>Property: <code>connections.max.idle.ms</code></p>
	<p>An upper bound on the time to report a success or a failure after a call to <code>send()</code> returns. This limits the total time that a record will be delayed prior to sending, the time to await acknowledgement from the broker (if expected) and the time allowed for retriable send failures.</p> <p>Default: <code>120 * 1000</code></p> <p>Property: <code>delivery.timeout.ms</code></p>

DELIVERY_TIMEOUT_MS_CONFIG	<p>A Kafka producer may report failure to send a record earlier than this config if either an unrecoverable error is encountered, the retries have been exhausted, or the record is added to batch which reached an earlier delivery expiration deadline.</p> <p>The value of this config should be greater than or equal to the sum of <code>request.timeout.ms</code> and <code>linger.ms</code>.</p> <p>Must be at least <code>0</code></p>
ENABLE_IDEMPOTENCE_CONFIG	<p>When enabled (<code>true</code>), a producer will ensure that exactly one copy of a message is written to the stream.</p> <p>Default: <code>false</code></p> <p>Property: <code>enable.idempotence</code></p> <p>When disabled (<code>false</code>), it is acceptable that a producer may write duplicates of a message to the stream (e.g. due to broker failures and retries).</p> <p>Enabling idempotence requires:</p> <ul style="list-style-type: none"> • <code>max.in.flight.requests.per.connection</code> to be less than or equal to <code>5</code> • <code>retries</code> to be greater than <code>0</code> • <code>acks</code> must be <code>all</code> <p>If these values are not explicitly set, suitable values will be chosen. If incompatible values are set, a <code>configException</code> will be thrown.</p>
INTERCEPTOR_CLASSES_CONFIG	<p>ProducerInterceptors to use to intercept (and possibly mutate) the records sent out by the producer before they are published to the Kafka cluster.</p> <p>Default: (empty)</p> <p>Property: <code>interceptor.classes</code></p>
KEY_SERIALIZER_CLASS_CONFIG	<p>Serializer class for keys that implements the org.apache.kafka.common.serialization.Serializer interface.</p> <p>Default: (empty)</p> <p>Property: <code>key.serializer</code></p>

LINGER_MS_CONFIG	<p>The producer groups together any records that arrive in between request transmissions into a single batched request. Normally this occurs or under load when records arrive faster than they can be sent out. However in some circumstances the client may want to reduce the number of requests even under moderate load. This setting accomplishes this by adding a small amount of artificial delay, i.e. rather than immediately sending out a record the producer will wait for up to the given delay to allow other records to be sent so that the sends can be batched together. This can be thought of as analogous to Nagle's algorithm in TCP. This setting gives the upper bound on the delay for batching: once we get BATCH_SIZE_CONFIG worth of records for a partition it will be sent immediately regardless of this setting, however if we have fewer than this many bytes accumulated for this partition we will 'linger' for the specified time waiting for more records to show up.</p> <p>Default: 0</p> <p>Property: <code>linger.ms</code></p> <p>Must be at least 0</p>
MAX_IN_FLIGHT_REQUESTS_PER_CONNECTION	<p>The maximum number of unacknowledged requests a client will send on a single connection before blocking.</p> <p>Default: 5</p> <p>Property: <code>max.in.flight.requests.per.connection</code></p> <p>Note that if this setting is set to be greater than 1 and there are failed sends, there is a risk of message re-ordering due to retries (i.e. if retries are enabled).</p> <p>Must be at least 1</p>
	<p>A value greater than 0 will cause the client to resend any record whose send fails with a potentially transient error.</p> <p>Default: <code>Integer.MAX_VALUE</code></p> <p>Property: <code>retries</code></p> <p>Note that this retry is no different than if the client resent the record upon receiving the error.</p>

RETRIES_CONFIG	<p>Allowing retries without setting <code>max.in.flight.requests.per.connection</code> to 1 will potentially change the ordering of records because if two batches are sent to a single partition, and the first fails and is retried but the second succeeds, then the records in the second batch may appear first.</p> <p>Note also that produce requests will be failed before the number of retries has been exhausted if the timeout configured by <code>delivery.timeout.ms</code> expires first before successful acknowledgement.</p> <p>Users should generally prefer to leave this configuration unset and instead use <code>delivery.timeout.ms</code> to control retry behavior.</p> <p>Must be at least 0</p>
VALUE_SERIALIZER_CLASS_CONFIG	<p>Serializer class for values that implements the <code>org.apache.kafka.common.serialization.Serializer</code> interface.</p> <p>Property: <code>value.serializer</code></p>
MAX_BLOCK_MS_CONFIG	<p>How long can <code>KafkaProducer.send()</code> and <code>KafkaProducer.partitionsFor()</code> block.</p> <p>Default: 60 * 1000</p> <p>Property: <code>max.block.ms</code></p> <p>These methods can be blocked either because the buffer is full or metadata unavailable. Blocking in the user-supplied serializers or partitioner will not be counted against this timeout.</p> <p>Must be at least 0</p>
MAX_REQUEST_SIZE_CONFIG	<p>The maximum size of a request in bytes. This setting will limit the number of record batches the producer will send in a single request to avoid sending huge requests. This is also effectively a cap on the maximum record batch size.</p> <p>Default: 1024 * 1024</p> <p>Property: <code>max.request.size</code></p> <p>Note that the server has its own cap on record batch size which may be different.</p> <p>Must be at least 0</p>

METADATA_MAX_AGE_CONFIG	The period of time (in milliseconds) after which we force a refresh of metadata even if we haven't seen any partition leadership changes to proactively discover any new brokers or partitions. Default: <code>5 * 60 * 1000</code> Property: <code>metadata.max.age.ms</code> Must be at least <code>0</code>
METRICS_NUM_SAMPLES_CONFIG	The number of samples maintained to compute metrics (for Kafka producers). Must be at least <code>1</code> . Property: <code>metrics.num.samples</code>
METRICS_RECORDING_LEVEL_CONFIG	The name of highest recording level for metrics Property: <code>metrics.recording.level</code> Must be one of the following: <code>INFO</code> or <code>DEBUG</code> .
METRIC_REPORTER_CLASSES_CONFIG	The class names of the MetricsReporters that will be notified of new metric creation. Property: <code>metric.reporters</code> The JmxReporter is always included to register JMX statistics.
METRICS_SAMPLE_WINDOW_MS_CONFIG	The window of time a metrics sample is computed over (for Kafka producers). Property: <code>metrics.sample.window.ms</code>
PARTITIONER_CLASS_CONFIG	The Partitioner to compute the partition for a record when <code>KafkaProducer</code> is requested to send a record to topic . Default: <code>DefaultPartitioner</code> Property: <code>partitioner.class</code>
RECONNECT_BACKOFF_MAX_MS_CONFIG	
RECONNECT_BACKOFF_MS_CONFIG	
RECEIVE_BUFFER_CONFIG	
	Maximum amount of time the client will wait for the response of a request. If the response is not received before the timeout elapses the client will

	<p>resend the request if necessary or fail the request if retries are exhausted.</p> <p>Default: <code>30 * 1000</code></p> <p>Property: <code>request.timeout.ms</code></p> <p>This should be larger than <code>replica.lag.time.max.ms</code> (a broker configuration) to reduce the possibility of message duplication due to unnecessary producer retries.</p> <p>Must be at least <code>0</code></p>
<code>RETRY_BACKOFF_MS_CONFIG</code>	<p>How long to wait (back off) before attempting to retry a failed request to a given topic partition. This avoids repeatedly sending requests in a tight loop under some failure scenarios.</p> <p>Default: <code>100</code></p> <p>Property: <code>retry.backoff.ms</code></p> <p>Must be at least <code>0</code></p> <p>Used when <code>KafkaProducer</code> is created (for a <code>RecordAccumulator</code>, a <code>Metadata</code>, a <code>Sender</code>, and a <code>TransactionManager</code>)</p>
<code>SEND_BUFFER_CONFIG</code>	
<code>TRANSACTION_TIMEOUT_CONFIG</code>	<p>The maximum amount of time (in ms) that the transaction coordinator will wait for a transaction status update from a producer before proactive aborting the ongoing transaction.</p> <p>Property: <code>transaction.timeout.ms</code></p> <p>If this value is larger than <code>transaction.max.timeout.ms</code> configuration (of Kafka brokers), the request will fail with an <code>InvalidTransactionTimeout</code> error.</p>
	<p>User-defined transactional ID to use for transactional delivery. This enables reliability semantics which span multiple producer sessions since it allows the client to guarantee that transactions using the same Transactional ID have been completed prior to starting any new transactions.</p> <p>Default: (empty)</p> <p>Property: <code>transactional.id</code></p>

TRANSACTIONAL_ID_CONFIG

With no Transactional ID provided, the produce is limited to idempotent delivery.

`enable.idempotence` must be enabled (`true`) if a Transactional ID is configured.

The default means transactions cannot be used.

Note that transactions requires a cluster of at least 3 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 .
```

logUnused Method

```
void logUnused()
```

logUnused ...FIXME

Note	logUnused is used when...FIXME
------	--------------------------------

ProducerRecord

ProducerRecord is...FIXME

Callback Contract

Callback is...FIXME

Partitioner Contract — Partitioning Strategy

`Partitioner` is the [contract](#) of [partitioners](#) (aka *partitioning strategies*) that can [compute the partition for a producer record](#).

`Partitioner` is also a [Configurable](#) that can be created by reflection and need to take configuration parameters.

Note

[DefaultPartitioner](#) is the one and only known implementation of the [Partitioner Contract](#) in Apache Kafka.

Table 1. Partitioner Contract

Method	Description
close	<pre>void close()</pre> <p>Closes the partitioner</p>
partition	<pre>int partition(String topic, Object key, byte[] keyBytes, Object value, byte[] valueBytes, Cluster cluster)</pre> <p>Computes the partition ID for a record (described by a topic, a key, a value, the serialized key and value, and the Cluster metadata)</p> <p>Used when <code>KafkaProducer</code> is requested to compute the partition for a ProducerRecord</p>

ProducerInterceptor

ProducerInterceptor is...FIXME

Sender—Kafka Producer I/O Thread

`Sender` is a **Kafka producer I/O thread** (of execution) that runs indefinitely and sends `ProduceRequests` to a Kafka cluster.

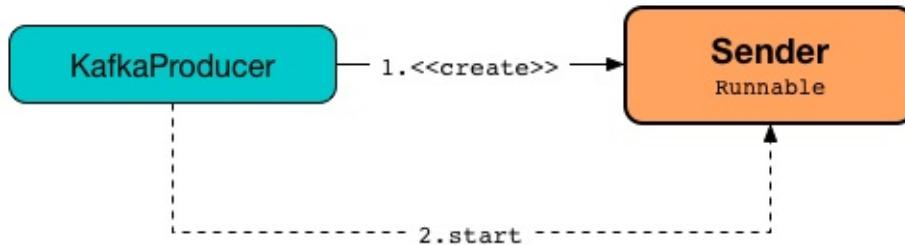


Figure 1. Sender and KafkaProducer

`Sender` is created exclusively when `KafkaProducer` is created and immediately started under the name **kafka-producer-network-thread | [clientId]** (as a `daemon thread`).

Every `runOnce`, `Sender` takes (*drains*) up to `max.request.size` messages from the `record accumulator` (that batches records) and eventually `sendProduceRequests` to every active Kafka broker for which there is at least one `ProducerBatch`.

`KafkaProducer` uses the following configuration properties to create a `Sender`:

- `max.in.flight.requests.per.connection` for the `guaranteeMessageOrder`
- `max.request.size` for the `maxRequestSize`
- `acks` for the `acks`
- `retries` for the `retries`
- `request.timeout.ms` for the `requestTimeoutMs`
- `retry.backoff.ms` for the `retryBackoffMs`

Tip Enable `ALL` logging level for `org.apache.kafka.clients.producer.internals.Sender` logger to see what happens inside.

Add the following line to `config/tools-log4j.properties`:

```
log4j.logger.org.apache.kafka.clients.producer.internals.Sender=ALL
```

Refer to [Logging](#).

Creating Sender Instance

`Sender` takes the following to be created:

- `LogContext`
- `KafkaClient`
- `Metadata`
- `RecordAccumulator`
- `guaranteeMessageOrder` flag
- `maxRequestSize`
- `acks`
- The number of retries
- `SenderMetricsRegistry`
- `Time`
- `requestTimeout`
- `retryBackoffMs`
- `TransactionManager`
- `ApiVersions`

`Sender` initializes the [internal properties](#).

Starting Thread of Execution — `run` Method (of Java's `Runnable`)

```
void run()
```

Note	<code>run</code> is a part of <code>java.lang.Runnable</code> that is executed when the thread is started.
------	--

`run` first prints out the following DEBUG message to the logs:

```
Starting Kafka producer I/O thread.
```

`run` keeps running until the `running` internal flag is off.

Once the `running` internal flag is off, `run` prints out the following DEBUG message to the logs:

```
Beginning shutdown of Kafka producer I/O thread, sending remaining records.
```

`run` may wait until...FIXME (`(!forceClose...)`)

In the end, `run` requests the [KafkaClient](#) to [close](#) and prints out the following DEBUG message to the logs:

```
Shutdown of Kafka producer I/O thread has completed.
```

Running Single Iteration of Sending — `runOnce` Method

```
void runOnce()
```

`runOnce` simply [sendProducerData](#) followed by requesting the [KafkaClient](#) to [poll](#).

With a [transactionManager](#) assigned, `runOnce` ...FIXME

Note	<code>run</code> is used exclusively when <code>Sender</code> is requested to <code>run</code> (as a thread of execution).
------	--

`sendProducerData` Internal Method

```
long sendProducerData(long now)
```

`sendProducerData` requests the [ProducerMetadata](#) to get the current cluster metadata ([without blocking](#)).

`sendProducerData` requests the [RecordAccumulator](#) to get the current cluster metadata ([without blocking](#)).

`sendProducerData` requests the [RecordAccumulator](#) to drain (create [ProducerBatches](#) per [TopicPartition](#) for ready brokers).

`sendProducerData` adds the batches to [inflight batches](#).

`sendProducerData` ...FIXME (`guaranteeMessageOrder`)

`sendProducerData` requests the [SenderMetrics](#) to [updateProduceRequestMetrics](#).

`sendProducerData` ...FIXME

In the end, `sendProducerData` [sendProduceRequests](#) (for the batches).

Note

`sendProducerData` is used exclusively when `Sender` is requested to run a single iteration of sending.

Sending ProduceRequests — `sendProduceRequests` Internal Method

```
void sendProduceRequests(
    Map<Integer, List<ProducerBatch>> collated,
    long now)
```

`sendProduceRequests` simply goes over the given `collated` collection (of `ProducerBatches` per partition ID) and `sendProduceRequest` for every pair.

Note

`sendProduceRequests` is used exclusively when `Sender` is requested to `sendProducerData`.

Sending ProduceRequest to Broker — `sendProduceRequest` Internal Method

```
void sendProduceRequest(
    long now,
    int destination,
    short acks,
    int timeout,
    List<ProducerBatch> batches)
```

`sendProduceRequest` ...FIXME

`sendProduceRequest` requests the `ProduceRequest.Builder` to create a `ProduceRequest.Builder` instance (for a given magic number).

`sendProduceRequest` creates a new `RequestCompletionHandler` that `handleProduceResponse` when a request is complete.

`sendProduceRequest` requests the `KafkaClient` to create a new `ClientRequest` (for the `ProduceRequest.Builder` and `RequestCompletionHandler`).

Note

`sendProduceRequest` creates a new `ClientRequest` with the `expectResponse` flag on when `acks` argument is non-`0`.

`sendProduceRequest` requests the `KafkaClient` to send the `ClientRequest`.

In the end, `sendProduceRequest` prints out the following TRACE message to the logs:

```
Sent produce request to [nodeId]: [requestBuilder]
```

Note

`sendProduceRequest` is used exclusively when `Sender` is requested to [sendProduceRequests](#).

Handling ProduceResponse — `handleProduceResponse` Internal Method

```
void handleProduceResponse(  
    ClientResponse response,  
    Map<TopicPartition, ProducerBatch> batches,  
    long now)
```

```
handleProduceResponse ...FIXME
```

Note

`handleProduceResponse` is used exclusively when `sender` is requested to [send a ProduceRequest](#).

completeBatch Internal Method

```
void completeBatch(  
    ProducerBatch batch,  
    ProduceResponse.PartitionResponse response,  
    long correlationId,  
    long now,  
    long throttleUntilTimeMs)
```

```
completeBatch ...FIXME
```

Note

`completeBatch` is used exclusively when `sender` is requested to [handle a ProduceResponse](#).

addToInflightBatches Method

```
void addToInflightBatches(Map<Integer, List<ProducerBatch>> batches)
```

```
addToInflightBatches ...FIXME
```

Note

`addToInflightBatches` is used exclusively when `sender` is requested to [sendProducerData](#).

maybeSendTransactionalRequest Internal Method

```
boolean maybeSendTransactionalRequest(long now)
```

maybeSendTransactionalRequest ...FIXME

Note

maybeSendTransactionalRequest is used exclusively when Sender is running.

maybeWaitForProducerId Internal Method

```
void maybeWaitForProducerId()
```

maybeWaitForProducerId ...FIXME

Note

maybeWaitForProducerId is used exclusively when Sender is running.

awaitLeastLoadedNodeReady Internal Method

```
Node awaitLeastLoadedNodeReady(long remainingTimeMs)
```

awaitLeastLoadedNodeReady ...FIXME

Note

awaitLeastLoadedNodeReady is used when Sender is requested to maybeSendTransactionalRequest and maybeWaitForProducerId.

initiateClose Method

```
void initiateClose()
```

initiateClose requests the RecordAccumulator to close.

In the end, initiateClose turns the running internal flag off followed by waking up the Kafka client.

Note

initiateClose is used when:

- KafkaProducer is requested to close
- Sender is requested to forceClose

wakeup Method

```
void wakeup()
```

wakeup merely requests the KafkaClient to wakeup.

Note

wakeup is used when:

- KafkaProducer is requested to `initTransactions`, `sendOffsetsToTransaction`, `commitTransaction`, `abortTransaction`, `doSend`, `waitOnMetadata`, and `flush`
- Sender is requested to `initiateClose`

forceClose Method

```
void forceClose()
```

forceClose ...FIXME

Note

forceClose is used exclusively when KafkaProducer is requested to close.

reenqueueBatch Internal Method

```
void reenqueueBatch(  
    ProducerBatch batch,  
    long currentTimeMs)
```

reenqueueBatch ...FIXME

Note

reenqueueBatch is used when...FIXME

Internal Properties

Name	Description
<code>running</code>	<p>Flag that controls whether run should stop (<code>false</code>) or not (<code>true</code>)</p> <ul style="list-style-type: none"> • Enabled (<code>true</code>) by default when sender is created • Disabled (<code>false</code>) when sender is requested to <code>initiateClose</code>

RecordAccumulator

RecordAccumulator is created exclusively when KafkaProducer is created.

KafkaProducer uses the following configuration properties to create a RecordAccumulator :

- batch.size for the batchSize
- linger.ms for the lingerMs
- retry.backoff.ms for the retryBackoffMs

Tip	<p>Enable ALL logging level for org.apache.kafka.clients.producer.internals.RecordAccumulator logger to see what happens inside.</p> <p>Add the following line to config/tools-log4j.properties :</p> <pre>log4j.logger.org.apache.kafka.clients.producer.internals.RecordAccumulator=ALL</pre> <p>Refer to Logging.</p>
-----	--

Creating RecordAccumulator Instance

RecordAccumulator takes the following to be created:

- LogContext
- batchSize
- CompressionType
- lingerMs
- retryBackoffMs
- deliveryTimeoutMs
- Metrics
- Metric group name (e.g. producer-metrics)
- Time
- ApiVersions
- TransactionManager

- `BufferPool`

`RecordAccumulator` initializes the [internal properties](#).

When created, `RecordAccumulator` `registerMetrics` (with the [Metrics](#) and the [metric group name](#)).

registerMetrics Method

```
void registerMetrics()
```

`registerMetrics` ...[FIXME](#)

Note	<code>registerMetrics</code> is used exclusively when <code>RecordAccumulator</code> is created .
------	---

close Method

```
void close()
```

`close` ...[FIXME](#)

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

append Method

```
RecordAppendResult append(
    TopicPartition tp,
    long timestamp,
    byte[] key,
    byte[] value,
    Header[] headers,
    Callback callback,
    long maxTimeToBlock) throws InterruptedException
```

`append` looks up or creates a new Deque of ProducerBatches for a given TopicPartition.

`append` [tryAppend](#) and returns the `RecordAppendResult` if available (not `null`).

`append` prints out the following TRACE message to the logs:

```
Allocating a new [size] byte message buffer for topic [topic] partition [partition]
```

Note	The <code>size</code> in the above TRACE message is controlled by <code>batch.size</code> producer configuration property (default: <code>16384</code> which is <code>16 * 1024</code>).
------	---

`append` once again `tryAppend` and returns the `RecordAppendResult` if available (which they say "*should not happen often*").

`append` creates a new `MemoryRecordsBuilder` and a new `ProducerBatch` (for the given `TopicPartition` and the new `MemoryRecordsBuilder`).

`append` requests the `ProducerBatch` to `tryAppend` (that gives a `FutureRecordMetadata`).

`append` adds the `ProducerBatch` to the `Deque<ProducerBatch>` and the `IncompleteBatches` internal registry.

In the end, `append` creates a new `RecordAppendResult` (with the `FutureRecordMetadata`).

Note	<code>append</code> is used exclusively when <code>KafkaProducer</code> is requested to send a <code>ProducerRecord</code> to a Kafka Cluster asynchronously.
------	---

tryAppend Internal Method

```
RecordAppendResult tryAppend(
    long timestamp,
    byte[] key,
    byte[] value,
    Header[] headers,
    Callback callback,
    Deque<ProducerBatch> deque)
```

`tryAppend` ...FIXME

Note	<code>tryAppend</code> is used exclusively when <code>RecordAccumulator</code> is requested to <code>append</code> .
------	--

Creating MemoryRecordsBuilder Instance (With TimestampType.CREATE_TIME) — recordsBuilder Internal Method

```
MemoryRecordsBuilder recordsBuilder(
    ByteBuffer buffer,
    byte maxUsableMagic)
```

`recordsBuilder` simply builds a new `MemoryRecordsBuilder` (with `TimestampType.CREATE_TIME`).

`recordsBuilder` throws an `UnsupportedVersionException` when the `TransactionManager` is defined and the `maxUsableMagic` magic number is lower than 2 :

Attempting to use idempotence with a broker which does not support the required message format (v2). The broker must be version 0.11 or later.

Note

`recordsBuilder` is used exclusively when `RecordAccumulator` is requested to [append](#).

Aborting Incomplete Batches — `abortIncompleteBatches` Method

`void abortIncompleteBatches()`

`abortIncompleteBatches ...FIXME`

Note

`abortIncompleteBatches` is used exclusively when `Sender` is requested to [run](#) (and forced to close while shutting down).

`reenqueue` Method

```
void reenqueue(
    ProducerBatch batch,
    long now)
```

`reenqueue ...FIXME`

Note

`reenqueue` is used exclusively when `Sender` is requested to [reenqueueBatch](#).

`splitAndReenqueue` Method

`int splitAndReenqueue(ProducerBatch bigBatch)`

`splitAndReenqueue ...FIXME`

Note

`splitAndReenqueue` is used exclusively when `Sender` is requested to [completeBatch](#).

Looking Up Or Creating New Deque Of ProducerBatches For TopicPartition — `getOrCreateDeque` Internal Method

```
Deque<ProducerBatch> getOrCreateDeque(TopicPartition tp)
```

`getOrCreateDeque` ...FIXME

Note

`getOrCreateDeque` is used when `RecordAccumulator` is requested to `append`, `reenqueue`, and `splitAndReenqueue`.

expiredBatches Method

```
List<ProducerBatch> expiredBatches(long now)
```

`expiredBatches` ...FIXME

Note

`expiredBatches` is used exclusively when `Sender` is requested to `sendProducerData`.

ready Method

```
ReadyCheckResult ready(
    Cluster cluster,
    long nowMs)
```

`ready` ...FIXME

Note

`ready` is used exclusively when `sender` is requested to `sendProducerData` (when requested to `run a single iteration of sending`).

drain Method

```
Map<Integer, List<ProducerBatch>> drain(
    Cluster cluster,
    Set<Node> nodes,
    int maxSize,
    long now)
```

`drain` ...FIXME

Note

`drain` is used exclusively when `sender` is requested to `sendProducerData` (when requested to `run a single iteration of sending`).

drainBatchesForOneNode Internal Method

```
List<ProducerBatch> drainBatchesForOneNode(
    Cluster cluster,
    Node node,
    int maxSize,
    long now)
```

drainBatchesForOneNode ...FIXME

Note

`drainBatchesForOneNode` is used exclusively when `RecordAccumulator` is requested to `drain`.

Internal Properties

Name	Description
appendsInProgress	<p><code>java.util.concurrent.atomic.AtomicInteger</code> to keep track of the number of <code>appending threads</code></p> <ul style="list-style-type: none"> Starts at <code>0</code> when <code>RecordAccumulator</code> is <code>created</code> Increments and decrements when <code>RecordAccumulator</code> is requested to <code>append</code> (just after it is requested and right before it finishes) <p>Used exclusively when <code>RecordAccumulator</code> is requested to <code>abort incomplete batches</code></p>
batches	<code>ProducerBatches per TopicPartition</code> <code>(ConcurrentMap<TopicPartition, Deque<ProducerBatch>>)</code>
incomplete	<code>IncompleteBatches</code>

ProducerBatch

`ProducerBatch` is...FIXME

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

- `RecordAccumulator` is requested to [append](#)
- `ProducerBatch` is requested to [split](#)

Tip Enable `ALL` logging level for
`org.apache.kafka.clients.producer.internals.ProducerBatch` logger to see what happens inside.

Add the following line to `config/tools-log4j.properties`:

```
log4j.logger.org.apache.kafka.clients.producer.internals.ProducerBatch=ALL
```

Refer to [Logging](#).

Creating ProducerBatch Instance

`ProducerBatch` takes the following to be created:

- `TopicPartition`
- [MemoryRecordsBuilder](#)
- `createdMs`
- `isSplitBatch` flag (default: `false`)

`ProducerBatch` initializes the [internal properties](#).

split Method

```
Deque<ProducerBatch> split(int splitBatchSize)
```

`split` ...FIXME

Note `split` is used exclusively when `RecordAccumulator` is requested to [splitAndReenqueue](#).

createBatchOffAccumulatorForRecord Internal Method

```
ProducerBatch createBatchOffAccumulatorForRecord(  
    Record record,  
    int batchSize)
```

createBatchOffAccumulatorForRecord ...FIXME

Note	createBatchOffAccumulatorForRecord is used exclusively when ProducerBatch is requested to split.
------	--

is used exclusively when ProducerBatch is requested to [split](#).

ProducerInterceptors

ProducerInterceptors is...FIXME

Passing ProducerRecord Through Chain Of Registered Producer Interceptors — onSend Method

```
ProducerRecord<K, V> onSend(ProducerRecord<K, V> record)
```

onSend ...FIXME

Note

onSend is used when...FIXME

DefaultPartitioner — Default Partitioning Strategy

`DefaultPartitioner` is a [Partitioner](#) that uses a 32-bit murmur2 hash to compute the [partition](#) for a record (with the key defined) or chooses a partition in a round-robin fashion (per the available partitions of the topic).

`DefaultPartitioner` is the default partitioning strategy (per [ProducerConfig.PARTITIONER_CLASS_CONFIG](#) configuration property).

`DefaultPartitioner` is a [Configurable](#) that needs no [configuration](#) while being created.

`DefaultPartitioner` does nothing when requested to [close](#).

Computing Partition For Record — `partition` Method

```
int partition(
    String topic,
    Object key,
    byte[] keyBytes,
    Object value,
    byte[] valueBytes,
    Cluster cluster)
```

Note

`partition` is part of the [Partitioner Contract](#) to compute the partition for a record.

`partition` requests the input `cluster` for the [partitions](#) for the given topic.

If there is the `keyBytes` serialized key present (i.e. non-`null`), `partition` simply generates a 32-bit murmur2 hash (from the byte array) and applies the modulo operation by the number of partitions.

If there is no `keyBytes` serialized key defined (i.e. `null`), `partition` requests the input `cluster` for the [availablePartitionsForTopic](#) and chooses a partition in a round-robin fashion.

nextValue Internal Method

```
int nextValue(String topic)
```

`nextValue ...FIXME`

Note

`nextValue` is used exclusively when `DefaultPartitioner` is requested for the partition for a record.

TransactionManager

`TransactionManager` is created exclusively for a [transactional KafkaProducer](#) (when a user-configured transaction or idempotence are enabled).

Tip

A [transactional KafkaProducer](#) uses **[Producer clientId=[clientId], transactionalId=[transactionalId]]** prefix for [logging](#).

Tip

You should see one of the following INFO messages in the logs with a [transactional KafkaProducer](#):

Instantiated a transactional producer.

or

Instantiated an idempotent producer.

```
// Enable TRACE logging level for KafkaProducer
val props = new java.util.Properties()
props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, classOf[StringSerializer].getName)
props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, classOf[StringSerializer].getName)
props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, ":9092")
val clientId = this.getClass.getSimpleName.replace("$", "")
props.put(ProducerConfig.CLIENT_ID_CONFIG, clientId)
props.put(ProducerConfig.TRANSACTIONAL_ID_CONFIG, "txId")
val producer = new KafkaProducer[String, String](props)

// Observe the logs
```

`KafkaProducer` uses the following configuration properties when creating a `TransactionManager`:

- `retry.backoff.ms` for `retryBackoffMs`
- `transactional.id` for `transactionalId`
- `transaction.timeout.ms` for `transactionTimeoutMs`

`TransactionManager` is **transactional** when the `Transactional Id` is assigned.

`TransactionManager` is in one of the following states:

- UNINITIALIZED when created
- INITIALIZING
- READY
- IN_TRANSACTION
- COMMITTING_TRANSACTION
- ABORTING_TRANSACTION
- ABORTABLE_ERROR
- FATAL_ERROR

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

Name	Description
consumerGroupCoordinator	Kafka Node Used when...FIXME
inflightBatchesBySequence	Map<TopicPartition, PriorityQueue<ProducerBatch>> Used when...FIXME
lastAckedOffset	Map<TopicPartition, Long> Used when...FIXME
lastAckedSequence	Map<TopicPartition, Integer> Used when...FIXME
newPartitionsInTransaction	Set<TopicPartition> Used when...FIXME
nextSequence	Map<TopicPartition, Integer> Used when...FIXME
partitionsInTransaction	Set<TopicPartition> Used when...FIXME
partitionsWithUnresolvedSequences	Set<TopicPartition> Used when...FIXME
pendingPartitionsInTransaction	Set<TopicPartition> Used when...FIXME

<code>pendingRequests</code>	<code>java.util.PriorityQueue</code> of <code>TxnRequestHandlers</code> Used when...FIXME
<code>pendingTxnOffsetCommits</code>	<code>Map<TopicPartition, CommittedOffset></code> Used when...FIXME
<code>producerIdAndEpoch</code>	<code>ProducerIdAndEpoch</code> Used when...FIXME
<code>transactionCoordinator</code>	<code>Kafka Node</code> Used when...FIXME

beginTransaction Method

```
void beginTransaction()
```

`beginTransaction` ...FIXME

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

Creating TransactionManager Instance

`TransactionManager` takes the following to be created:

- `LogContext`
- Transactional ID (as `transactional.id`)
- `transactionTimeoutMs` (as `transaction.timeout.ms`)
- `retryBackoffMs` (as `retry.backoff.ms`)

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

initializeTransactions Method

```
TransactionalRequestResult initializeTransactions()
```

`initializeTransactions` ...FIXME

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

sendOffsetsToTransaction Method

```
TransactionalRequestResult sendOffsetsToTransaction(  
    Map<TopicPartition, OffsetAndMetadata> offsets,  
    String consumerGroupId)
```

sendOffsetsToTransaction ...FIXME

Note

sendOffsetsToTransaction is used when...FIXME

beginCommit Method

```
TransactionalRequestResult beginCommit()
```

beginCommit ...FIXME

Note

beginCommit is used when...FIXME

beginAbort Method

```
TransactionalRequestResult beginAbort()
```

beginAbort ...FIXME

Note

beginAbort is used when...FIXME

maybeAddPartitionToTransaction Method

```
void maybeAddPartitionToTransaction(TopicPartition topicPartition)
```

maybeAddPartitionToTransaction ...FIXME

Note

maybeAddPartitionToTransaction is used when...FIXME

KafkaServer — Kafka Broker

`KafkaServer` is a Kafka broker that manages Kafka services.

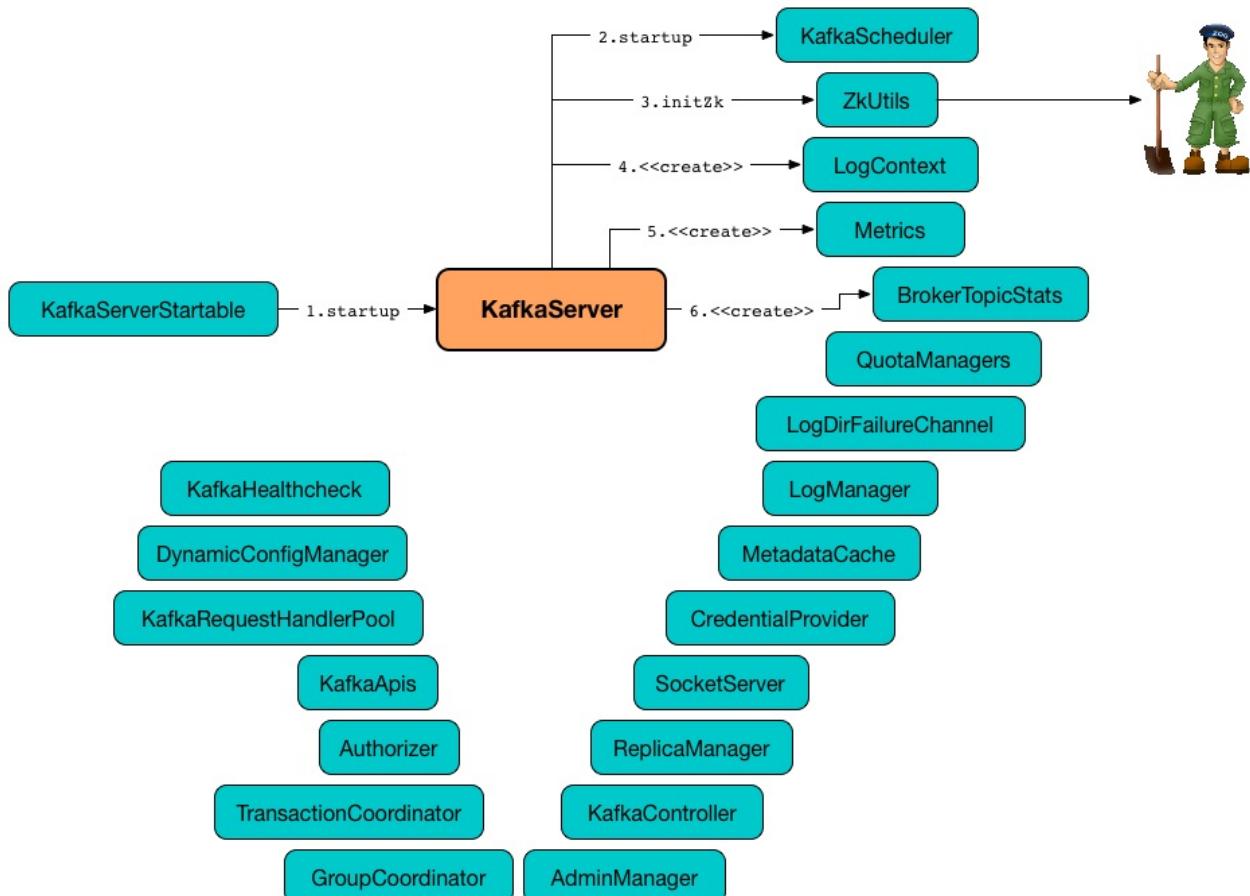


Figure 1. KafkaServer's Startup and Auxiliary Services

When `created`, `KafkaServer` registers itself in the JMX system under the `kafka.server` node.

Tip	<p>Enable <code>ALL</code> logging level for <code>kafka.server.KafkaServer</code> logger to see what happens inside.</p> <p>Add the following line to <code>config/log4j.properties</code> :</p> <pre>log4j.logger.kafka.server.KafkaServer=ALL</pre> <p>Refer to Logging.</p>
------------	---

Creating KafkaServer Instance

`KafkaServer` takes the following when created:

- `KafkaConfig`

- `Time` (defaults to `Time.SYSTEM`)
- Optional thread name prefix
- A collection of [KafkaMetricsReporters](#) (defaults to no reporters)

Caution	FIXME
Note	<code>KafkaServer</code> is created when KafkaServerStartable is created.

Starting Up — `startup` Method

```
startup(): Unit
```

`startup` starts the Kafka broker.

Internally, `startup` first prints out the following INFO message to the logs:

```
starting
```

`startup` sets [BrokerState](#) as `Starting`.

`startup` requests [KafkaScheduler](#) to [start](#).

`startup` connects to Zookeeper (and initializes [ZkUtils](#)).

`startup` [getOrGenerateClusterId](#) (that is recorded as [cluster id](#)).

You should see the following INFO message in the logs:

```
Cluster ID = [clusterId]
```

`startup` gets the broker ID and initial offline directories.

`startup` requests the [KafkaConfig](#) to set the brokerId to the broker ID.

`startup` creates the `LogContext` with **[KafkaServer id=[brokerId]]** prefix.

`startup` requests the [KafkaConfig](#) for the [DynamicBrokerConfig](#) that is in turn requested to initialize (fetching broker configuration from Zookeeper).

`startup` creates a new [KafkaScheduler](#) (with the number of threads as specified by [background.threads](#) configuration property) and immediately requests it to [start up](#).

`startup` creates and configures metrics.

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

`startup` registers broker topic metrics (by initializing `BrokerTopicStats`).

`startup` initializes `QuotaManagers`.

`startup` notifies cluster resource listeners (i.e. `KafkaMetricsReporters` and the configured instances of metric reporters).

`startup` creates the `LogDirFailureChannel`

`startup` creates the `LogManager` and requests it to `start up`.

`startup` creates the `MetadataCache` (for the broker ID).

`startup` creates the `CredentialProvider` (per `sasl.enabled.mechanisms` property).

`startup` creates the `SocketServer` (for `KafkaConfig`, `Metrics` and `CredentialProvider`) and requests it to `start up`.

`startup` creates the `ReplicaManager` and requests it to `start up`.

`startup` creates the `KafkaController` (for `KafkaConfig`, `ZkUtils`, `Metrics` and the optional `threadNamePrefix`) and requests it to `start up`.

`startup` creates the `AdminManager` (for `KafkaConfig`, `Metrics`, `MetadataCache` and `ZkUtils`).

`startup` creates the `GroupCoordinator` (for `KafkaConfig`, `ZkUtils` and `ReplicaManager`) and requests it to `start up`.

`startup` creates the `TransactionCoordinator` (for `KafkaConfig`, `ReplicaManager`, a new dedicated `KafkaScheduler` with `transaction-log-manager-` thread name prefix, `ZkUtils`, `Metrics` and `MetadataCache`) and requests it to `start up`.

`startup` creates a `Authorizer` (if defined using `authorizer.class.name` property) and configures it.

`startup` creates the `KafkaApis` (for `SocketServer`, `ReplicaManager`, `AdminManager`, `GroupCoordinator`, `TransactionCoordinator`, `KafkaController`, `ZkUtils`, broker ID, `KafkaConfig`, `MetadataCache`, `Metrics`, `Authorizer`, `QuotaManagers`, `BrokerTopicStats`, cluster ID).

Note	At this point <code>KafkaServer</code> may start processing requests.
------	---

`startup` creates the `KafkaRequestHandlerPool` (for broker ID, `SocketServer`, `KafkaApis` and `num.io.threads`).

`startup` starts the HTTP interface of mx4j (if configured).

`startup` creates the `DynamicConfigManager` (for `ZkUtils` and `dynamicConfigHandlers`) and requests it to `start up`.

`startup` configures the advertised listeners (if defined).

`startup` creates the `KafkaHealthcheck` (for broker ID, the advertised listeners, `ZkUtils`, `broker.rack` and `inter.broker.protocol.version` Kafka properties) and requests it to `start up`.

`startup` checkpoints the broker ID.

`startup` sets `BrokerState` as `RunningAsBroker`, creates the `CountDownLatch`, enables the `startupComplete` flag, disables `isStartingUp` flag

`startup` registers `AppInfo` as an MBean with the MBean server as `kafka.server:type=app-info,id=[brokerId]`.

In the end, you should see the following INFO message in the logs:

```
INFO [Kafka Server [brokerId]], started (kafka.server.KafkaServer)
```

Note	The INFO message above uses so-called log ident with the value of <code>broker.id</code> property and is always in the format [Kafka Server [brokerId]], after a Kafka server has fully started.
------	---

Note	<code>startup</code> is used exclusively when <code>KafkaServerStartable</code> is requested to <code>starts up</code> .
------	--

Sending Updated Cluster Metadata to ClusterResourceListeners — `notifyClusterListeners` Internal Method

```
notifyClusterListeners(clusterListeners: Seq[AnyRef]): Unit
```

`notifyClusterListeners` creates a `ClusterResourceListeners` (with the objects from the input `clusterListeners` of type `ClusterResourceListener`) and sends the updated cluster metadata to them.

Note

- `notifyClusterListeners` is used when:
- `KafkaServer` is requested to `start up` (with `clusterListeners` as `kafkaMetricsReporters` and the `MetricsReporter` reporters from `metric.reporters` Kafka property)
 - `DynamicMetricsReporters` is requested to `createReporters` (when `created` and requested to `reconfigure`)

Creating ReplicaManager — `createReplicaManager` Internal Method

```
createReplicaManager(isShuttingDown: AtomicBoolean): ReplicaManager
```

`createReplicaManager` simply `creates` the `ReplicaManager` (passing in the references to the services, e.g. `Metrics`, `KafkaScheduler`, `LogManager`, `QuotaManagers`, `MetadataCache`, `LogDirFailureChannel`).

Note

`createReplicaManager` is used exclusively when `KafkaServer` is requested to `start up`.

getOrGenerateClusterId Internal Method

```
getOrGenerateClusterId(zkClient: KafkaZkClient): String
```

`getOrGenerateClusterId` simply requests the given `KafkaZkClient` for the `cluster ID` or `createOrGetClusterId` with a random UUID (as Base64).

Note

`getOrGenerateClusterId` is used exclusively when `KafkaServer` is requested to `start up` (and initializes the internal `cluster ID`).

Shutting Down — `shutdown` Method

```
shutdown(): Unit
```

`shutdown` ...FIXME

Note

`shutdown` is used when:

- `KafkaServer` is requested to [start up](#) (and there was an exception)
- `KafkaServerStartable` is requested to [shut down](#)

initZkClient Internal Method

```
initZkClient(time: Time): Unit
```

`initZkClient` prints out the following INFO message to the logs:

```
Connecting to zookeeper on [zkConnect]
```

(only if the chroot path is used) `initZkClient` ...FIXME

`initZkClient` ...FIXME (`secureAclsEnabled`)

`initZkClient` creates a [KafkaZkClient](#) (with the following configuration properties: `KafkaConfig.zkConnect`, `KafkaConfig.secureAclsEnabled`, `KafkaConfig.zkSessionTimeoutMs`, `KafkaConfig.zkConnectionTimeoutMs`, `KafkaConfig.zkMaxInFlightRequests`).

In the end, `initZkClient` requests the [KafkaZkClient](#) to [createTopLevelPaths](#).

Note

`initZkClient` is used exclusively when `KafkaServer` is requested to [start up](#).

controlledShutdown Internal Method

```
controlledShutdown(): Unit
```

`controlledShutdown` ...FIXME

Note

`controlledShutdown` is used when...FIXME

Checkpointing Broker — checkpointBrokerId Internal Method

```
checkpointBrokerId(brokerId: Int): Unit
```

For every directory in `KafkaConfig.logDirs` that is `isLogDirOnline` (according to the `LogManager`), `checkpointBrokerId` finds the corresponding `BrokerMetadataCheckpoint` (with the path to the `meta.properties` file) in the `brokerMetadataCheckpoints` registry and requests it to `read` it.

Unless the `meta.properties` file was already available, `checkpointBrokerId` requests the `BrokerMetadataCheckpoints` (of the log directories with no meta files) to `write` the broker metadata.

Note

`checkpointBrokerId` is used exclusively when `KafkaServer` is requested to `start up`.

Getting Broker ID and Initial Offline Directories — `getBrokerIdAndOfflineDirs` Internal Method

```
getBrokerIdAndOfflineDirs: (Int, Seq[String])
```

```
getBrokerIdAndOfflineDirs ...FIXME
```

Note

`getBrokerIdAndOfflineDirs` is used exclusively when `KafkaServer` is requested to `start up`.

generateBrokerId Internal Method

```
generateBrokerId: Int
```

```
generateBrokerId ...FIXME
```

Note

`generateBrokerId` is used exclusively when `KafkaServer` is requested to `getBrokerIdAndOfflineDirs`.

createBrokerInfo Internal Method

```
createBrokerInfo: BrokerInfo
```

```
createBrokerInfo ...FIXME
```

Note	<p><code>createBrokerInfo</code> is used when:</p> <ul style="list-style-type: none"> • <code>KafkaServer</code> is requested to start up • <code>DynamicListenerConfig</code> is requested to reconfigure
-------------	--

Cluster ID — `_clusterId` Internal Property

```
_clusterId: String
```

`KafkaServer` uses **Cluster ID** that is a random UUID (encoded to Base64).

When requested to [start up](#), `KafkaServer` initializes the internal `_clusterId` which is immediately printed out as an INFO message to the logs:

```
Cluster ID = [clusterId]
```

Cluster ID is persisted in Zookeeper in `/cluster/id` znode (in JSON format).

Cluster ID is registered as `kafka.server:type=KafkaServer,name=ClusterId` MBean in the JMX system.

Cluster ID is used for the following:

- Creating [KafkaApis](#) (for `dataPlaneRequestProcessor` and `controlPlaneRequestChannelOpt`) at [startup](#)
- [Sending an updated cluster metadata to ClusterResourceListeners](#)

Internal Properties

Name	Description
<code>adminManager</code>	AdminManager
<code>apis</code>	KafkaApis
<code>authorizer</code>	Authorizer
<code>brokerMetadataCheckpoints</code>	
<code>brokerState</code>	BrokerState
	BrokerTopicStats

<code>_brokerTopicStats</code>	Created when <code>KafkaServer</code> is requested to start up Used (as <code>brokerTopicStats</code> method) to create the data-plane KafkaApis , the control-plane KafkaApis , the ReplicaManager , the LogManager when <code>KafkaServer</code> is requested to start up
<code>controlPlaneRequestHandlerPool</code>	Control-plane KafkaRequestHandlerPool for the control-plane KafkaApis
<code>controlPlaneRequestProcessor</code>	Control-plane KafkaApis request handler for the optional control-plane KafkaRequestHandlerPool of the SocketServer
<code>credentialProvider</code>	<code>CredentialProvider</code>
<code>dataPlaneRequestProcessor</code>	Data-plane KafkaApis request handler for the data-plane RequestChannel of the SocketServer
<code>dataPlaneRequestHandlerPool</code>	Data-plane KafkaRequestHandlerPool for the data-plane KafkaApis
<code>dynamicConfigHandlers</code>	ConfigHandlers by name: <ul style="list-style-type: none">• TopicConfigHandler as topics• ClientIdConfigHandler as clients• UserConfigHandler as users• BrokerConfigHandler as brokers
<code>dynamicConfigManager</code>	Initialized when <code>KafkaServer</code> is requested to start up for the only purpose of creating the DynamicConfigManager .
<code>groupCoordinator</code>	GroupCoordinator (for the only purpose of creating the KafkaApis) Created and immediately started up when <code>KafkaServer</code> is requested to start up Shut down when <code>KafkaServer</code> is requested to shut down
<code>isStartingUp</code>	Flag for...FIXME
<code>kafkaController</code>	KafkaController
<code>kafkaHealthcheck</code>	KafkaHealthcheck

kafkaScheduler	<code>KafkaScheduler</code> with the number of daemon threads as configured using <code>background.threads</code> configuration property (default: 10)
logContext	<code>LogContext</code>
logDirFailureChannel	<code>LogDirFailureChannel</code>
logManager	<p><code>LogManager</code></p> <p>Used when:</p> <ul style="list-style-type: none"> • <code>checkpointBrokerId</code> (when <code>KafkaServer</code> is requested to <code>start up</code>) • <code>DynamicBrokerConfig</code> is requested to <code>addReconfigurables</code> • <code>KafkaServer</code> is requested to <code>start up</code> (and creates a <code>TopicConfigHandler</code> for topics for <code>dynamicConfigHandlers</code> and a <code>ReplicaManager</code>) <hr/> <p><code>Created</code> and immediately <code>started</code> when <code>KafkaServer</code> is requested to <code>start up</code>.</p> <p><code>Shut down</code> when <code>KafkaServer</code> is requested to <code>shut down</code>.</p>
metadataCache	<p><code>MetadataCache</code> that is created for the sake of creating the following services (at <code>startup</code>):</p> <ul style="list-style-type: none"> • <code>AdminManager</code> • <code>KafkaApis</code> • <code>ReplicaManager</code> • <code>TransactionCoordinator</code>
replicaManager	<p><code>ReplicaManager</code> used to create:</p> <ul style="list-style-type: none"> • <code>KafkaApis</code> • <code>GroupCoordinator</code> • <code>TransactionCoordinator</code> <hr/> <ul style="list-style-type: none"> • <code>Created</code> (and <code>started</code> immediately) when <code>KafkaServer</code> is requested to <code>start up</code>

	<ul style="list-style-type: none">• Shut down when <code>KafkaServer</code> shuts down
reporters	Collection of MetricsReporter Used when...FIXME
<code>requestHandlerPool</code>	KafkaRequestHandlerPool
<code>socketServer</code>	SocketServer
<code>transactionCoordinator</code>	TransactionCoordinator
<code>quotaManagers</code>	QuotaManagers
<code>shutdownLatch</code>	java.util.concurrent.CountDownLatch
<code>startupComplete</code>	Flag for...FIXME
<code>zkUtils</code>	ZkUtils

Kafka Server and Periodic Tasks

Apache Kafka uses the [Scheduler](#) to [schedule periodic tasks](#). Kafka internal services use the interface to schedule maintenance tasks.

The page describes the [tasks](#) (that should further improve comprehension of the interaction of the internal services and how Kafka server works internally).

Table 1. Tasks

Kafka Service	Task Name	Task Method
KafkaController	delete-expired-tokens	expireTokens
	auto-leader-rebalance-task	Sends AutoPreferredReplicaLeaderChosen controller event to ControllerEventManager
GroupMetadataManager	delete-expired-group-metadata	cleanupGroupMetadata
	__consumer_offsets-[offsetsPartition]	loadGroupsAndOffsets
	__consumer_offsets-[offsetsPartition]	removeGroupsAndOffsets
	handleTxnCompletion-[producerId]	handleTxnCompletion
TransactionCoordinator	transaction-abort	abortTimedOutTransactions
TransactionStateManager	transactionalId-expiration	enableTransactionalIdExpiration
	load-tns-for-partition-[topicPartition]	loadTransactions
	remove-tns-for-partition-[topicPartition]	removeTransactions
Log	PeriodicProducerExpirationCheck	removeExpiredProducers
	flush-log	flush
	delete-file	deleteSeg
	kafka-log-retention	cleanupLogs

LogManager	kafka-log-flusher	flushDirtyLogs
	kafka-recovery-point-checkpoint	checkpointLogRecovery
	kafka-log-start-offset-checkpoint	checkpointLogStartOffs
	kafka-delete-logs	deleteLogs
ReplicaManager	highwatermark-checkpoint	checkpointHighWatermark
	isr-expiration	maybeShrinkIsr
	isr-change-propagation	maybePropagateIsrChange
	shutdown-idle-replica-alter-log-dirs-thread	shutdownIdleReplicaAlterLogDirsThread

AdminManager

`AdminManager` is...FIXME

`AdminManager` is [created](#) exclusively when `KafkaServer` is [started](#).

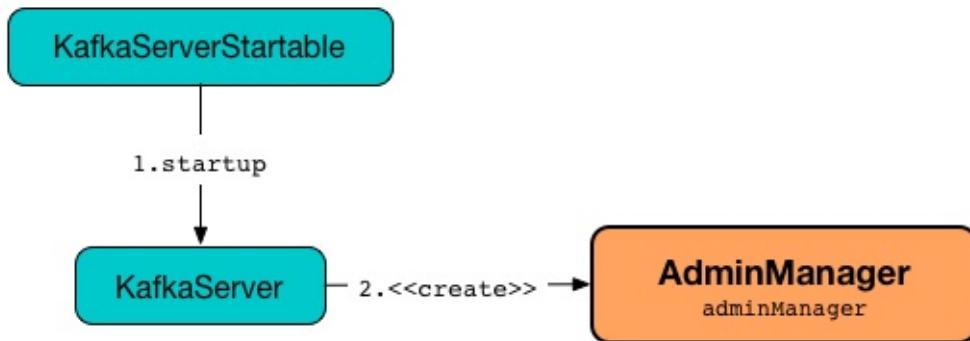


Figure 1. AdminManager

`AdminManager` uses the [MetadataCache](#) for the following:

- [createTopics](#) (to get the [getAliveBrokers](#))
- [describeConfigs](#) for `TOPIC` resource type (to [check whether a topic is available or not](#))

Table 1. AdminManager's Internal Properties (e.g. Registries and Counters)

Name	Description
<code>adminZkClient</code>	
<code>alterConfigPolicy</code>	
<code>createTopicPolicy</code>	
<code>topicPurgatory</code>	

`AdminManager` uses **[Admin Manager on Broker [brokerId]]** as the logging prefix (aka `logIdent`).

Tip	<p>Enable <code>INFO</code>, <code>DEBUG</code>, <code>TRACE</code> logging levels for <code>kafka.server.AdminManager</code> logger to see what happens inside.</p> <p>Add the following line to <code>config/log4j.properties</code> :</p> <pre>log4j.logger.kafka.server.AdminManager=TRACE</pre> <p>Refer to Logging.</p>

Creating Topics — `createTopics` Method

```
createTopics(
  timeout: Int,
  validateOnly: Boolean,
  createInfo: Map[String, CreateTopicsRequest.TopicDetails],
  responseCallback: (Map[String, ApiError]) => Unit
```

`createTopics` ...FIXME

Note

`createTopics` is used exclusively when `KafkaApis` is requested to handle a `CREATE_TOPICS` request.

Creating AdminManager Instance

`AdminManager` takes the following when created:

- `KafkaConfig`
- `Metrics`
- `MetadataCache`
- `KafkaZkClient`

`AdminManager` initializes the internal registries and counters.

`describeConfigs` Method

```
describeConfigs(
  resourceToConfigNames: Map[ConfigResource, Option[Set[String]]],
  includeSynonyms: Boolean): Map[ConfigResource, DescribeConfigsResponse.Config]
```

`describeConfigs` ...FIXME

Note

`describeConfigs` is used when...FIXME

`alterConfigs` Method

```
alterConfigs(
  configs: Map[ConfigResource, AlterConfigsRequest.Config],
  validateOnly: Boolean): Map[ConfigResource, ApiError]
```

```
alterConfigs ...FIXME
```

Note

`alterConfigs` is used exclusively when `KafkaApis` is requested to [handleAlterConfigsRequest](#).

configSynonyms Internal Method

```
configSynonyms(  
    name: String,  
    synonyms: List[String],  
    isSensitive: Boolean): List[DescribeConfigsResponse.ConfigSynonym]
```

```
configSynonyms ...FIXME
```

Note

`configSynonyms` is used when...FIXME

createPartitions Method

```
createPartitions(  
    timeout: Int,  
    newPartitions: Map[String, PartitionDetails],  
    validateOnly: Boolean,  
    listenerName: ListenerName,  
    callback: Map[String, ApiError] => Unit): Unit
```

```
createPartitions ...FIXME
```

Note

`createPartitions` is used when...FIXME

Authorizer

Authorizer is...FIXME

configure Method

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

DelegationTokenManager

DelegationTokenManager is...FIXME

DynamicConfigManager uses the [TokenChangedNotificationHandler](#) that...FIXME

expireTokens Method

expireTokens(): Unit

expireTokens ...FIXME

Note	expireTokens is used when...FIXME
------	-----------------------------------

TokenChangedNotificationHandler Object

TokenChangedNotificationHandler is a [NotificationHandler](#) that processNotification.

processNotification Method

processNotification(tokenIdBytes: Array[Byte]): Unit

Note	processNotification is part of the NotificationHandler Contract to...FIXME.
------	---

processNotification ...FIXME

DynamicConfigManager — Dynamic Configuration Management

`DynamicConfigManager` monitors (indirectly through the [ZkNodeChangeNotificationListener](#)) for any child node changes under `/config/changes` path in Zookeeper.

`DynamicConfigManager` uses the [ZkNodeChangeNotificationListener](#) (that in turn uses the [NotificationHandler](#)) to process configuration change notifications using [ConfigHandlers](#).

`DynamicConfigManager` is created and immediately started up when `KafkaServer` is requested to start up.

When created, `DynamicConfigManager` is given the [ConfigHandlers](#) to use (directly from [KafkaServer](#)).

`DynamicConfigManager` uses the feature of Zookeeper called Sequence Nodes with `config_change` sequence number prefix.

Tip	Read up on Sequence Nodes in the official documentation of Apache Zookeeper.
-----	--

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

Name	Description
<code>adminZkClient</code>	<p><code>AdminZkClient</code> (for the KafkaZkClient)</p> <p>Used when:</p> <ul style="list-style-type: none"> • <code>ConfigChangedNotificationHandler</code> is requested to processEntityConfigChangeVersion1 and processEntityConfigChangeVersion2 • <code>DynamicConfigManager</code> is requested to start up
<code>configChangeListener</code>	<p><code>ZkNodeChangeNotificationListener</code> (for the KafkaZkClient and <code>/config/changes</code> path in Zookeeper, <code>config_change</code> sequence number prefix and the ConfigChangedNotificationHandler)</p> <p>Initialized when <code>DynamicConfigManager</code> is requested to start up</p> <p>Closed when <code>DynamicConfigManager</code> is requested to shut down</p>

startup Method

<p><code>startup</code></p>

```
startup ...FIXME
```

Note	<code>startup</code> is used exclusively when <code>KafkaServer</code> is requested to start up.
------	--

Creating DynamicConfigManager Instance

`DynamicConfigManager` takes the following when created:

- `KafkaZkClient`
- `ConfigHandlers` by name
- `changeExpirationMs` (default: 15 mins)
- `Time`

Note	<code>changeExpirationMs</code> seems never used.
------	---

`DynamicConfigManager` initializes the internal registries and counters.

ConfigChangedNotificationHandler Object

`ConfigChangedNotificationHandler` is a `NotificationHandler` that `processNotification`.

Processing Config Change Notifications

— `processNotification` Method

```
processNotification(jsonBytes: Array[Byte]): Unit
```

Note	<code>processNotification</code> is part of the <code>NotificationHandler Contract</code> to handle a change notification.
------	--

```
processNotification ...FIXME
```

processEntityConfigChangeVersion1 Internal Method

```
processEntityConfigChangeVersion1(jsonBytes: Array[Byte], js: JsonObject): Unit
```

```
processEntityConfigChangeVersion1 ...FIXME
```

Note	<code>processEntityConfigChangeVersion1</code> is used when...FIXME
------	---

processEntityConfigChangeVersion2 Internal Method

```
processEntityConfigChangeVersion2(jsonBytes: Array[Byte], js: JsonObject): Unit
```

processEntityConfigChangeVersion2 ...FIXME

Note

processEntityConfigChangeVersion2 is used when...FIXME

Shutting Down— shutdown Method

```
shutdown(): Unit
```

shutdown simply requests the [ZkNodeChangeNotificationListener](#) to close.

Note

shutdown is used exclusively when KafkaServer is requested to shut down.

ConfigHandler Contract — Configuration Change Handlers

`ConfigHandler` is the [contract](#) of [config change handlers](#) that can [process configuration change notifications](#) (from [DynamicConfigManager](#)).

Table 1. ConfigHandler Contract

Method	Description
<code>processConfigChanges</code>	<p><code>processConfigChanges(entityName: String, value: Properties)</code></p> <p>Processes configuration changes</p> <p>Used when:</p> <ul style="list-style-type: none"> • <code>ConfigChangedNotificationHandler</code> is requested to <code>processEntityConfigChangeVersion1</code> and <code>processEntityConfigChangeVersion2</code> (that is a notification handler registered when <code>DynamicConfigManager</code> is created) • <code>DynamicConfigManager</code> is requested to start up

Table 2. ConfigHandlers

ConfigHandler	Description
BrokerConfigHandler	
ClientIdConfigHandler	
TopicConfigHandler	
UserConfigHandler	

BrokerConfigHandler

`BrokerConfigHandler` is a [ConfigHandler](#) that can [process configuration changes](#) for **brokers** entity type.

`BrokerConfigHandler` is [created](#) exclusively when `KafkaServer` is requested to [start up](#) (to create a [DynamicConfigManager](#)).

`BrokerConfigHandler` takes the following to be created:

- [KafkaConfig](#)
- [QuotaManagers](#)

When requested to [process configuration changes](#) (for all brokers or the current one), `BrokerConfigHandler` simply propagates the changes to the [DynamicBrokerConfig](#) (of the [KafkaConfig](#)).

Processing Configuration Changes — `processConfigChanges` Method

```
processConfigChanges(brokerId: String, properties: Properties): Unit
```

Note

`processConfigChanges` is part of the [ConfigHandler Contract](#) to process configuration changes.

`processConfigChanges` branches off per the given `brokerId`.

All Brokers

For changes to all brokers (i.e. `brokerId` is [`<default>`](#)), `processConfigChanges` requests the [KafkaConfig](#) for the [DynamicBrokerConfig](#) that is in turn requested to [updateDefaultConfig](#) (with the `Properties`).

Current Broker

For changes to the current broker (i.e. `brokerId` is exactly `brokerId`), `processConfigChanges` requests the [KafkaConfig](#) for the [DynamicBrokerConfig](#) that is in turn requested to [updateBrokerConfig](#) (with the `brokerId` and the `Properties`).

`processConfigChanges` requests the [QuotaManagers](#) for the [leader ReplicationQuotaManager](#) that is in turn requested to [updateQuota](#).

`processConfigChanges` requests the [QuotaManagers](#) for the [follower ReplicationQuotaManager](#) that is in turn requested to [updateQuota](#).

`processConfigChanges` requests the [QuotaManagers](#) for the [alterLogDirs ReplicationQuotaManager](#) that is in turn requested to [updateQuota](#).

ClientIdConfigHandler

ClientIdConfigHandler is...FIXME

processConfigChanges Method

```
processConfigChanges(brokerId: String, properties: Properties): Unit
```

Note	processConfigChanges is part of the ConfigHandler Contract to...FIXME.
------	--

processConfigChanges ...FIXME

TopicConfigHandler

`TopicConfigHandler` is a [ConfigHandler](#) that...FIXME

`TopicConfigHandler` is [created](#) when...FIXME

Creating TopicConfigHandler Instance

`TopicConfigHandler` takes the following when created:

- [LogManager](#)
- [KafkaConfig](#)
- [QuotaManagers](#)

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

UserConfigHandler

UserConfigHandler is...FIXME

processConfigChanges Method

```
processConfigChanges(brokerId: String, properties: Properties): Unit
```

Note	processConfigChanges is part of the ConfigHandler Contract to...FIXME.
------	--

processConfigChanges ...FIXME

DynamicBrokerConfig

`DynamicBrokerConfig` is [created](#) exclusively when `KafkaConfig` is [created](#).

`DynamicBrokerConfig` is used to create a [KafkaConfig](#).

Note	<code>DynamicBrokerConfig</code> is created when KafkaConfig is, but the reverse is also true (!) Isn't it a dependency cycle? Not really since <code>KafkaConfig</code> will create a <code>DynamicBrokerConfig</code> unless one is provided.
------	---

`DynamicBrokerConfig` takes a single [KafkaConfig](#) when created.

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

Name	Description
<code>brokerReconfigurables</code>	<p>BrokerReconfigurables</p> <p>New reconfigurables registered in addBrokerReconfigurable</p> <p>All removed in clear</p> <p>Used in processReconfiguration</p>
<code>currentConfig</code>	<p>Current KafkaConfig</p> <p>Initialized with the input KafkaConfig.</p> <p>Used when...FIXME</p>
<code>dynamicDefaultConfigs</code>	<p>Dynamic configurations</p> <p>Cleared and immediately updated in updateDefaultConfig</p> <p>Used in validatedKafkaProps and updateCurrentConfig</p> <p>Cleared when <code>DynamicBrokerConfig</code> is requested to clear</p> <p>Available using currentDynamicDefaultConfigs</p>

Tip	<p>Enable <code>INFO</code>, <code>DEBUG</code> or <code>TRACE</code> logging levels for <code>kafka.server.DynamicBrokerConfig</code> logger to see what happens inside.</p>
-----	---

Add the following line to `config/log4j.properties`:

```
log4j.logger.kafka.server.DynamicBrokerConfig=TRACE
```

Refer to [Logging](#).

addBrokerReconfigurable Method

```
addBrokerReconfigurable(reconfigurable: BrokerReconfigurable): Unit
```

addBrokerReconfigurable ...FIXME

Note	addBrokerReconfigurable is used when...FIXME
------	--

processReconfiguration Internal Method

```
processReconfiguration(
  newProps: Map[String, String],
  validateOnly: Boolean): (KafkaConfig, List[BrokerReconfigurable])
```

processReconfiguration ...FIXME

Note	processReconfiguration is used when DynamicBrokerConfig is requested to validate (with validateOnly flag enabled) and updateCurrentConfig (with validateOnly flag disabled).
------	--

updateCurrentConfig Internal Method

```
updateCurrentConfig(): Unit
```

updateCurrentConfig ...FIXME

Note	updateCurrentConfig is used when DynamicBrokerConfig is requested to updateBrokerConfig and updateDefaultConfig.
------	--

currentDynamicDefaultConfigs Method

```
currentDynamicDefaultConfigs: Map[String, String]
```

currentDynamicDefaultConfigs simply clones the dynamicDefaultConfigs.

Note	currentDynamicDefaultConfigs is used when AdminManager is requested to configSynonyms and describeConfigs.
------	--

Initializing (Fetching Broker Configuration From Zookeeper) — initialize Method

```
initialize(zkClient: KafkaZkClient): Unit
```

`initialize` creates a new [KafkaConfig](#) (with the [properties](#) of the initial [KafkaConfig](#)) that becomes the [currentConfig](#).

`initialize` creates a [AdminZkClient](#) (with the input [KafkaZkClient](#)).

`initialize` requests the [AdminZkClient](#) to [fetch the default configuration](#) for [brokers](#) entities and [updateDefaultConfig](#)

`initialize` requests the [AdminZkClient](#) to [fetch the configuration](#) for the broker by the [broker.id](#).

`initialize` [maybeReEncodePasswords](#) in the broker configuration and [updateBrokerConfig](#) with the [broker.id](#).

Note	<code>initialize</code> is used exclusively when KafkaServer is requested to start up .
------	---

updateDefaultConfig Method

```
updateDefaultConfig(persistentProps: Properties): Unit
```

`updateDefaultConfig` ...FIXME

Note	<code>updateDefaultConfig</code> is used when:
------	--

- `BrokerConfigHandler` is requested to [processConfigChanges](#)
- `DynamicBrokerConfig` is requested to [initialize](#)

updateBrokerConfig Method

```
updateBrokerConfig(brokerId: Int, persistentProps: Properties): Unit
```

`updateBrokerConfig` ...FIXME

Note	<code>updateBrokerConfig</code> is used when:
------	---

- `BrokerConfigHandler` is requested to [processConfigChanges](#)
- `DynamicBrokerConfig` is requested to [initialize](#)

maybeReEncodePasswords Internal Method

```
maybeReEncodePasswords(persistentProps: Properties, adminZkClient: AdminZkClient): Properties
```

maybeReEncodePasswords ...FIXME

Note

maybeReEncodePasswords is used when...FIXME

Registering (Adding) Reconfigurables For KafkaServer — addReconfigurables Method

```
addReconfigurables(kafkaServer: KafkaServer): Unit
```

addReconfigurables registers (*adds*) broker and regular reconfigurables.

Internally, addReconfigurables creates a DynamicThreadPool with the input KafkaServer and addBrokerReconfigurable.

addReconfigurables addBrokerReconfigurable with the LogCleaner (if configured).

addReconfigurables creates a DynamicThreadPool with the LogManager and the input KafkaServer, and addReconfigurable.

addReconfigurables creates a DynamicMetricsReporters for the broker and addReconfigurable.

addReconfigurables creates a DynamicClientQuotaCallback for the broker and addReconfigurable.

addReconfigurables creates a DynamicListenerConfig with the input KafkaServer and addBrokerReconfigurable.

Note

addReconfigurables is used exclusively when KafkaServer is requested to start up.

validate Method

```
validate(props: Properties, perBrokerConfig: Boolean): Unit
```

validate ...FIXME

Note

`validate` is used exclusively when `AdminManager` is requested to [alterConfigs](#) (when `KafkaApis` is requested to [handleAlterConfigsRequest](#)).

maybeReconfigure Method

```
maybeReconfigure(  
    reconfigurable: Reconfigurable,  
    oldConfig: KafkaConfig,  
    newConfig: util.Map[String, _]): Unit
```

maybeReconfigure ...FIXME

Note

`maybeReconfigure` is used when:

- `DynamicMetricsReporters` is requested to [reconfigure](#)
- `DynamicClientQuotaCallback` is requested to [reconfigure](#)

processReconfigurable Method

```
processReconfigurable(  
    reconfigurable: Reconfigurable,  
    updatedConfigNames: Set[String],  
    allNewConfigs: util.Map[String, _],  
    newCustomConfigs: util.Map[String, Object],  
    validateOnly: Boolean): Unit
```

processReconfigurable ...FIXME

Note

`processReconfigurable` is used when `DynamicBrokerConfig` is requested to [processReconfiguration](#) and [processListenerReconfigurable](#).

processListenerReconfigurable Internal Method

```
processListenerReconfigurable(  
    listenerReconfigurable: ListenerReconfigurable,  
    newConfig: KafkaConfig,  
    customConfigs: util.Map[String, Object],  
    validateOnly: Boolean,  
    reloadOnly: Boolean): Unit
```

processListenerReconfigurable ...FIXME

Note	<code>processListenerReconfigurable</code> is used when <code>DynamicBrokerConfig</code> is requested to <code>reloadUpdatedFilesWithoutConfigChange</code> and <code>processReconfiguration</code> .
------	---

reloadUpdatedFilesWithoutConfigChange Internal Method

```
reloadUpdatedFilesWithoutConfigChange(newProps: Properties): Unit
```

```
reloadUpdatedFilesWithoutConfigChange ...FIXME
```

Note	<code>reloadUpdatedFilesWithoutConfigChange</code> is used exclusively when <code>AdminManager</code> is requested to <code>alterConfigs</code> (when <code>KafkaApis</code> is requested to handle a <code>AlterConfigs</code> request).
------	---

clear Method

```
clear(): Unit
```

```
clear ...FIXME
```

Note	<code>clear</code> is used exclusively when <code>KafkaServer</code> is requested to <code>shut down</code> .
------	---

validatedKafkaProps Method

```
validatedKafkaProps(  
  propsOverride: Properties,  
  perBrokerConfig: Boolean): Map[String, String]
```

```
validatedKafkaProps ...FIXME
```

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

BrokerReconfigurable Contract — Reconfigurable Broker Services

`BrokerReconfigurable` is the contract of [reconfigurable broker services](#) that allow for [self-reconfiguration](#).

`BrokerReconfigurable` interface is used exclusively by [DynamicBrokerConfig](#).

Table 1. BrokerReconfigurable Contract

Property	Description
<code>reconfigurableConfigs</code>	<pre>reconfigurableConfigs: Set[String]</pre> <p>Dynamic configurations Used when <code>DynamicBrokerConfig</code> is requested to addBrokerReconfigurable and processReconfiguration</p>
<code>reconfigure</code>	<pre>reconfigure(oldConfig: KafkaConfig, newConfig: KafkaConfig)</pre> <p>Used exclusively when <code>DynamicBrokerConfig</code> is requested to updateCurrentConfig</p>
<code>validateReconfiguration</code>	<pre>validateReconfiguration(newConfig: KafkaConfig): Unit</pre> <p>Used exclusively when <code>DynamicBrokerConfig</code> is requested to processReconfiguration</p>

Table 2. BrokerReconfigurables

BrokerReconfigurable	Dynamic Configurations
<code>DynamicConnectionQuota</code>	<code>max.connections.per.ip</code> <code>max.connections.per.ip.overrides</code>
	<code>advertised.listeners</code> <code>listeners</code> <code>listener.security.protocol.map</code> <code>principal.builder.class</code> <code>ssl.protocol</code> <code>ssl.provider</code>

DynamicListenerConfig	ssl.cipher.suites ssl.enabled.protocols ssl.keystore.type ssl.keystore.location ssl.keystore.password ssl.key.password ssl.truststore.type ssl.truststore.location ssl.truststore.password ssl.keymanager.algorithm ssl.trustmanager.algorithm ssl.endpoint.identification.algorithm ssl.secure.random.implementation ssl.client.auth sasl.mechanism.inter.broker.protocol sasl.jaas.config sasl.enabled.mechanisms sasl.kerberos.service.name sasl.kerberos.kinit.cmd sasl.kerberos.ticket.renew.window.factor sasl.kerberos.ticket.renew.jitter sasl.kerberos.min.time.before.relogin sasl.kerberos.principal.to.local.rules sasl.login.refresh.window.factor sasl.login.refresh.window.jitter sasl.login.refresh.min.period.seconds sasl.login.refresh.buffer.seconds
DynamicThreadPool	num.io.threads num.network.threads num.replica.fetchers num.recovery.threads.per.data.dir

	background.threads
LogCleaner	log.cleaner.threads log.cleaner.dedupe.buffer.size log.cleaner.io.buffer.load.factor log.cleaner.io.buffer.size log.cleaner.io.max.bytes.per.second log.cleaner.backoff.ms message.max.bytes

DynamicConnectionQuota

`DynamicConnectionQuota` is a [BrokerReconfigurable](#) for the following [dynamic configurations](#):

- [max.connections.per.ip](#)
- [max.connections.per.ip.overrides](#)

`DynamicConnectionQuota` is [created](#) exclusively when `DynamicBrokerConfig` is requested to [addReconfigurables](#) (when `KafkaServer` is requested to [start up](#)).

`DynamicConnectionQuota` takes a single [KafkaServer](#) to be created.

Reconfiguring — `reconfigure` Method

```
reconfigure(oldConfig: KafkaConfig, newConfig: KafkaConfig): Unit
```

Note	<code>reconfigure</code> is part of the BrokerReconfigurable Contract to change (<code>reconfigure</code>) the value of a Kafka dynamic configuration.
------	--

`reconfigure` requests the [KafkaServer](#) for the [SocketServer](#) that is in turn requested to [updateMaxConnectionsPerIpOverride](#) with the new value of [max.connections.per.ip.overrides](#).

If the values of the current and the new [max.connections.per.ip](#) are different, `reconfigure` requests the [KafkaServer](#) for the [SocketServer](#) that is in turn requested to [updateMaxConnectionsPerIp](#) with the new value.

DynamicListenerConfig

`DynamicListenerConfig` is a [BrokerReconfigurable](#) for the following [dynamic configurations](#):

- `advertised.listeners`
- `listeners`
- `listener.security.protocol.map`
- `principal.builder.class`
- `ssl.protocol`
- `ssl.provider`
- `ssl.cipher.suites`
- `ssl.enabled.protocols`
- `ssl.keystore.type`
- `ssl.keystore.location`
- `ssl.keystore.password`
- `ssl.key.password`
- `ssl.truststore.type`
- `ssl.truststore.location`
- `ssl.truststore.password`
- `ssl.keymanager.algorithm`
- `ssl.trustmanager.algorithm`
- `ssl.endpoint.identification.algorithm`
- `ssl.secure.random.implementation`
- `ssl.client.auth`
- `sasl.mechanism.inter.broker.protocol`
- `sasl.jaas.config`
- `sasl.enabled.mechanisms`

- `sasl.kerberos.service.name`
- `sasl.kerberos.kinit.cmd`
- `sasl.kerberos.ticket.renew.window.factor`
- `sasl.kerberos.ticket.renew.jitter`
- `sasl.kerberos.min.time.before.relogin`
- `sasl.kerberos.principal.to.local.rules`
- `sasl.login.refresh.window.factor`
- `sasl.login.refresh.window.jitter`
- `sasl.login.refresh.min.period.seconds`
- `sasl.login.refresh.buffer.seconds`

`DynamicListenerConfig` is created exclusively when `DynamicBrokerConfig` is requested to `addReconfigurables` (when `KafkaServer` is requested to start up).

`DynamicListenerConfig` takes a single `KafkaServer` to be created.

Reconfiguring — `reconfigure` Method

```
reconfigure(oldConfig: KafkaConfig, newConfig: KafkaConfig): Unit
```

Note

`reconfigure` is part of the `BrokerReconfigurable Contract` to change (`reconfigure`) the value of a Kafka dynamic configuration.

`reconfigure ...FIXME`

If there are any listeners added or removed, `reconfigure` requests the `LoginManager` to `closeAll`.

`reconfigure` requests the `KafkaServer` for the `SocketServer` that is in turn requested to `removeListeners` with the listeners removed.

`reconfigure` requests the `KafkaServer` for the `SocketServer` that is in turn requested to `addListeners` with the listeners added.

In the end, `reconfigure` requests the `KafkaServer` for the `KafkaController` that is in turn requested to `updateBrokerInfo` with the new `createBrokerInfo`.

`validateReconfiguration` Method

```
validateReconfiguration(newConfig: KafkaConfig): Unit
```

Note

`validateReconfiguration` is part of the [BrokerReconfigurable Contract](#) to...
FIXME.

```
validateReconfiguration ...FIXME
```

DynamicThreadPool

`DynamicThreadPool` is a [BrokerReconfigurable](#) for the following [dynamic configurations](#):

- `num.io.threads`
- `num.network.threads`
- `num.replica.fetchers`
- `num.recovery.threads.per.data.dir`
- `background.threads`

Tip

Use `kafka-configs` shell script to alter a dynamic configuration:

```
$ ./bin/kafka-configs.sh \
--bootstrap-server :9092 \
--alter \
--entity-type brokers \
--entity-name 0 \
--add-config num.recovery.threads.per.data.dir=2
```

`DynamicThreadPool` is [created](#) exclusively when `DynamicBrokerConfig` is requested to [addReconfigurables](#) (when `KafkaServer` is requested to [start up](#)).

`DynamicThreadPool` takes a single [KafkaServer](#) to be created.

Reconfiguring — `reconfigure` Method

```
reconfigure(oldConfig: KafkaConfig, newConfig: KafkaConfig): Unit
```

Note

`reconfigure` is part of the [BrokerReconfigurable Contract](#) to change ([reconfigure](#)) the value of a Kafka dynamic configuration.

`reconfigure ...FIXME`

DynamicClientQuotaCallback

DynamicClientQuotaCallback is...FIXME

reconfigure Method

```
reconfigure(configs: util.Map[String, _]): Unit
```

Note

reconfigure is part of the [Reconfigurable Contract](#) to...FIXME.

reconfigure ...FIXME

DynamicLogConfig

`DynamicLogConfig` is a [Reconfigurable](#) that...FIXME

`DynamicLogConfig` is created exclusively when `DynamicBrokerConfig` is requested to [register Reconfigurables for a KafkaServer](#) (when requested to [start up](#)).

Creating DynamicLogConfig Instance

`DynamicLogConfig` takes the following when created:

- [LogManager](#)
- [KafkaServer](#)

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

reconfigure Method

```
reconfigure(configs: util.Map[String, _]): Unit
```

Note

`reconfigure` is part of the [Reconfigurable Contract](#) to...FIXME.

`reconfigure` ...FIXME

DynamicMetricsReporters

`DynamicMetricsReporters` is...FIXME

Creating DynamicMetricsReporters Instance

`DynamicMetricsReporters` takes the following to be created:

- Broker ID
- [KafkaServer](#)

`DynamicMetricsReporters` initializes the [internal properties](#).

reconfigure Method

```
reconfigure(configs: util.Map[String, _]): Unit
```

Note

`reconfigure` is part of the [Reconfigurable Contract](#) to reconfigure the [Reconfigurable](#).

`reconfigure` ...FIXME

createReporters Internal Method

```
createReporters(
  reporterClasses: util.List[String],
  updatedConfigs: util.Map[String, _]): Unit
```

`createReporters` ...FIXME

Note

`createReporters` is used when `DynamicMetricsReporters` is created and [reconfigure](#).

Internal Properties

Name	Description
<code>dynamicConfig</code>	DynamicBrokerConfig Used when...FIXME

GroupCoordinator

`GroupCoordinator` is used (in `KafkaApis`) for `handleGroupImmigration`, `handleGroupEmigration`, `handleDeletedPartitions`, `handleCommitOffsets`, `handleFetchOffsets`, `handleDescribeGroup`, `handleListGroups`, `handleJoinGroup`, `handleSyncGroup`, `handleDeleteGroups`, `handleLeaveGroup`, `handleHeartbeat`, `handleTxnCommitOffsets` and `scheduleHandleTxnCompletion` (that all simply request the `GroupMetadataManager` to handle them).

`GroupCoordinator` is also used for `partitionFor`.

`GroupCoordinator` is created and immediately started when `KafkaServer` is requested to start up for the only purpose of creating the `KafkaApis`.

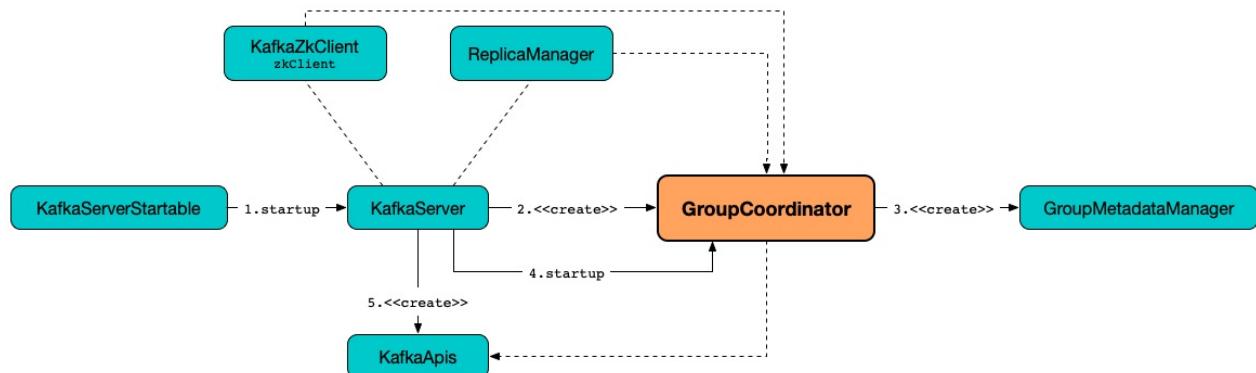


Figure 1. GroupCoordinator's Startup

`GroupCoordinator` manages the `GroupMetadataManager`. `GroupCoordinator` uses `isActive` flag to control whether the `GroupMetadataManager` was requested to start up (when `GroupCoordinator` was) that is used in `handleListGroups` and `validateGroupStatus`.

`GroupCoordinator` uses **[GroupCoordinator [brokerId]]** as the logging prefix (aka `logIdent`).

Tip	<p>Enable <code>DEBUG</code> logging level for <code>kafka.coordinator.group.GroupCoordinator</code> logger to see what happens inside.</p> <p>Add the following line to <code>config/log4j.properties</code> :</p> <pre>log4j.logger.kafka.coordinator.group.GroupCoordinator=DEBUG</pre> <p>Refer to Logging.</p>
-----	---

Creating GroupCoordinator Instance — apply Factory Method

```

apply(
  config: KafkaConfig,
  zkClient: KafkaZkClient,
  replicaManager: ReplicaManager,
  time: Time): GroupCoordinator (1)
apply(
  config: KafkaConfig,
  zkClient: KafkaZkClient,
  replicaManager: ReplicaManager,
  heartbeatPurgatory: DelayedOperationPurgatory[DelayedHeartbeat],
  joinPurgatory: DelayedOperationPurgatory[DelayedJoin],
  time: Time): GroupCoordinator

```

- Creates a `DelayedOperationPurgatory` and a `DelayedOperationPurgatory`

`apply ...FIXME`

Note	<code>apply</code> is used exclusively when <code>KafkaServer</code> is requested to start up .
------	---

Creating GroupCoordinator Instance

`GroupCoordinator` takes the following to be created:

- Broker ID
- `GroupConfig`
- `OffsetConfig`
- `GroupMetadataManager`
- `DelayedOperationPurgatory[DelayedHeartbeat]`
- `DelayedOperationPurgatory[DelayedJoin]`
- `Time`

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

Starting Up— `startup` Method

```
startup(enableMetadataExpiration: Boolean = true): Unit
```

`startup` first prints out the following INFO message to the logs:

```
Starting up.
```

`startup` requests the [GroupMetadataManager](#) to [startup](#) (with the given `enableMetadataExpiration` flag).

`startup` turns the `isActive` internal flag on.

In the end, `startup` prints out the following INFO message to the logs:

```
Startup complete.
```

Note	<code>startup</code> is used exclusively when <code>KafkaServer</code> is requested to start up .
------	---

partitionFor Method

```
partitionFor(group: String): Int
```

`partitionFor` simply requests the [GroupMetadataManager](#) to [partitionFor](#) the given `group ID`.

Note	<p><code>partitionFor</code> is used when:</p> <ul style="list-style-type: none"> • <code>KafkaApis</code> is requested to handle an AddOffsetsToTxn and FindCoordinatorRequest requests • <code>GroupCoordinator</code> is requested to prepareRebalance and onCompleteJoin
------	--

handleCommitOffsets Method

```
handleCommitOffsets(
  groupId: String,
  memberId: String,
  generationId: Int,
  offsetMetadata: immutable.Map[TopicPartition, OffsetAndMetadata],
  responseCallback: immutable.Map[TopicPartition, Errors] => Unit)
```

`handleCommitOffsets` firstly [validateGroupStatus](#) (for the given `groupId` and `OFFSET_COMMIT` API key).

`handleCommitOffsets` requests the [GroupMetadataManager](#) to [get the metadata of the group](#) (by the given `groupId`) and then [doCommitOffsets](#).

If the [GroupMetadataManager](#) could not [getGroup](#), `handleCommitOffsets` ...FIXME

In case of an error while [validateGroupStatus](#), `handleCommitOffsets` ...FIXME

Note

`handleCommitOffsets` is used exclusively when `KafkaApis` is requested to handle an `OffsetCommitRequest`.

doCommitOffsets Internal Method

```
doCommitOffsets(
    group: GroupMetadata,
    memberId: String,
    generationId: Int,
    producerId: Long,
    producerEpoch: Short,
    offsetMetadata: immutable.Map[TopicPartition, OffsetAndMetadata],
    responseCallback: immutable.Map[TopicPartition, Errors] => Unit)
```

`doCommitOffsets` ...FIXME

Note

`doCommitOffsets` is used when `GroupCoordinator` is requested to `handleTxnCommitOffsets` and `handleCommitOffsets`.

onCompleteJoin Method

```
onCompleteJoin(group: GroupMetadata): Unit
```

`onCompleteJoin` ...FIXME

Note

`onCompleteJoin` is used exclusively when `DelayedJoin` delayed operation is requested to `onComplete`.

doSyncGroup Internal Method

```
doSyncGroup(
    group: GroupMetadata,
    generationId: Int,
    memberId: String,
    groupAssignment: Map[String, Array[Byte]],
    responseCallback: SyncCallback): Unit
```

`doSyncGroup` ...FIXME

Note

`doSyncGroup` is used when...FIXME

handleDescribeGroup Method

```
handleDescribeGroup(groupId: String): (Errors, GroupSummary)
```

```
handleDescribeGroup ...FIXME
```

Note

`handleDescribeGroup` is used exclusively when `KafkaApis` is requested to [handleDescribeGroupRequest](#).

handleGroupImmigration Method

```
handleGroupImmigration(offsetTopicPartitionId: Int): Unit
```

`handleGroupImmigration` simply requests the [GroupMetadataManager](#) to [scheduleLoadGroupAndOffsets](#) (for the given offset and with the `onGroupLoaded` callback).

Note

`handleGroupImmigration` is used exclusively when `KafkaApis` is requested to handle a [LeaderAndIsrRequest](#).

handleGroupEmigration Method

```
handleGroupEmigration(offsetTopicPartitionId: Int): Unit
```

`handleGroupEmigration` simply requests the [GroupMetadataManager](#) to [removeGroupsForPartition](#) (for the given offset and with the `onGroupUnloaded` callback).

Note

`handleGroupEmigration` is used when `KafkaApis` is requested to handle a [LeaderAndIsrRequest](#) and a [StopReplicaRequest](#).

handleDeletedPartitions Method

```
handleDeletedPartitions(topicPartitions: Seq[TopicPartition]): Unit
```

`handleDeletedPartitions` simply requests the [GroupMetadataManager](#) to [cleanupGroupMetadata](#) and...FIXME

Note

`handleDeletedPartitions` is used when...FIXME

handleFetchOffsets Method

```
handleFetchOffsets(
    groupId: String,
    partitions: Option[Seq[TopicPartition]] = None):  

  (Errors, Map[TopicPartition, OffsetFetchResponse.PartitionData])
```

handleFetchOffsets ...FIXME

Note

handleFetchOffsets is used when...FIXME

handleListGroups Method

```
handleListGroups(): (Errors, List[GroupOverview])
```

handleListGroups ...FIXME

Note

handleListGroups is used when...FIXME

handleJoinGroup Method

```
handleJoinGroup(
    groupId: String,
    memberId: String,
    clientId: String,
    clientHost: String,
    rebalanceTimeoutMs: Int,
    sessionTimeoutMs: Int,
    protocolType: String,
    protocols: List[(String, Array[Byte])],
    responseCallback: JoinCallback): Unit
```

handleJoinGroup starts by validating the status of the group and the coordinator itself. In case of an error, handleJoinGroup uses the given JoinCallback to report it back and returns.

handleJoinGroup validates the group configuration, namely the given sessionTimeoutMs . In case of an error, handleJoinGroup uses the given JoinCallback to report a INVALID_SESSION_TIMEOUT error back and returns.

handleJoinGroup requests the GroupMetadataManager to getGroup by the given groupId .

If the group could not be found and the given memberId is defined (i.e. not empty), handleJoinGroup uses the given JoinCallback to report a UNKNOWN_MEMBER_ID error back and returns.

If the group could not be found and the given `memberId` is undefined (i.e. empty) or simply the group is available, `handleJoinGroup` requests the `GroupMetadataManager` to `addGroup` followed by `doJoinGroup`.

Note

`handleJoinGroup` is used exclusively when `KafkaApis` is requested to handle a `JoinGroupRequest`.

handleSyncGroup Method

```
handleSyncGroup(  
    groupId: String,  
    generation: Int,  
    memberId: String,  
    groupAssignment: Map[String, Array[Byte]],  
    responseCallback: SyncCallback): Unit
```

`handleSyncGroup` ...FIXME

Note

`handleSyncGroup` is used when...FIXME

handleDeleteGroups Method

```
handleDeleteGroups(groupIds: Set[String]): Map[String, Errors]
```

`handleDeleteGroups` ...FIXME

Note

`handleDeleteGroups` is used when...FIXME

handleHeartbeat Method

```
handleHeartbeat(  
    groupId: String,  
    memberId: String,  
    generationId: Int,  
    responseCallback: Errors => Unit)
```

`handleHeartbeat` ...FIXME

Note

`handleHeartbeat` is used when...FIXME

handleLeaveGroup Method

```
handleLeaveGroup(
    groupId: String,
    memberId: String,
    responseCallback: Errors => Unit): Unit
```

handleLeaveGroup ...FIXME

Note

handleLeaveGroup is used when...FIXME

scheduleHandleTxnCompletion Method

```
scheduleHandleTxnCompletion(
    producerId: Long,
    offsetsPartitions: Iterable[TopicPartition],
    transactionResult: TransactionResult): Unit
```

scheduleHandleTxnCompletion ...FIXME

Note

scheduleHandleTxnCompletion is used exclusively when KafkaApis is requested to handleWriteTxnMarkersRequest.

handleTxnCommitOffsets Method

```
handleTxnCommitOffsets(
    groupId: String,
    producerId: Long,
    producerEpoch: Short,
    offsetMetadata: immutable.Map[TopicPartition, OffsetAndMetadata],
    responseCallback: immutable.Map[TopicPartition, Errors] => Unit)
```

handleTxnCommitOffsets ...FIXME

Note

handleTxnCommitOffsets is used when...FIXME

onGroupLoaded Internal Callback

```
onGroupLoaded(group: GroupMetadata): Unit
```

onGroupLoaded ...FIXME

Note

onGroupLoaded is used when...FIXME

onGroupUnloaded Internal Callback

```
onGroupUnloaded(group: GroupMetadata): Unit
```

onGroupUnloaded ...FIXME

Note	onGroupUnloaded is used when...FIXME
------	--------------------------------------

Validating Group Status — validateGroupStatus Internal Method

```
validateGroupStatus(groupId: String, api: ApiKeys): Option[Errors]
```

validateGroupStatus ...FIXME

Note	validateGroupStatus is used when...FIXME
------	--

doJoinGroup Internal Method

```
doJoinGroup(
    group: GroupMetadata,
    memberId: String,
    clientId: String,
    clientHost: String,
    rebalanceTimeoutMs: Int,
    sessionTimeoutMs: Int,
    protocolType: String,
    protocols: List[(String, Array[Byte])],
    responseCallback: JoinCallback): Unit
```

doJoinGroup ...FIXME

Note	doJoinGroup is used exclusively when <code>GroupCoordinator</code> is requested to handleJoinGroup.
------	---

prepareRebalance Internal Method

```
prepareRebalance(group: GroupMetadata, reason: String): Unit
```

prepareRebalance ...FIXME

Note

`prepareRebalance` is used exclusively when `GroupCoordinator` is requested to [maybePrepareRebalance](#).

maybePrepareRebalance Internal Method

```
maybePrepareRebalance(group: GroupMetadata, reason: String): Unit
```

`maybePrepareRebalance` ...FIXME

Note

`maybePrepareRebalance` is used exclusively when `GroupCoordinator` is requested to...FIXME

addMemberAndRebalance Internal Method

```
addMemberAndRebalance(  
    rebalanceTimeoutMs: Int,  
    sessionTimeoutMs: Int,  
    clientId: String,  
    clientHost: String,  
    protocolType: String,  
    protocols: List[(String, Array[Byte])],  
    group: GroupMetadata,  
    callback: JoinCallback): MemberMetadata
```

`addMemberAndRebalance` ...FIXME

Note

`addMemberAndRebalance` is used exclusively when `GroupCoordinator` is requested to [doJoinGroup](#).

removeMemberAndUpdateGroup Internal Method

```
removeMemberAndUpdateGroup(  
    group: GroupMetadata,  
    member: MemberMetadata,  
    reason: String): Unit
```

`removeMemberAndUpdateGroup` ...FIXME

Note

`removeMemberAndUpdateGroup` is used when `GroupCoordinator` is requested to [handleLeaveGroup](#) and [onExpireHeartbeat](#).

onExpireHeartbeat Method

```
onExpireHeartbeat(  
    group: GroupMetadata,  
    member: MemberMetadata,  
    heartbeatDeadline: Long): Unit
```

onExpireHeartbeat ...FIXME

Note	onExpireHeartbeat is used exclusively when DelayedHeartbeat is requested to onExpiration .
------	--

onExpireHeartbeat is used exclusively when DelayedHeartbeat is requested to onExpiration .

GroupMetadataManager

`GroupMetadataManager` is [created](#) exclusively when `GroupCoordinator` is [created](#).

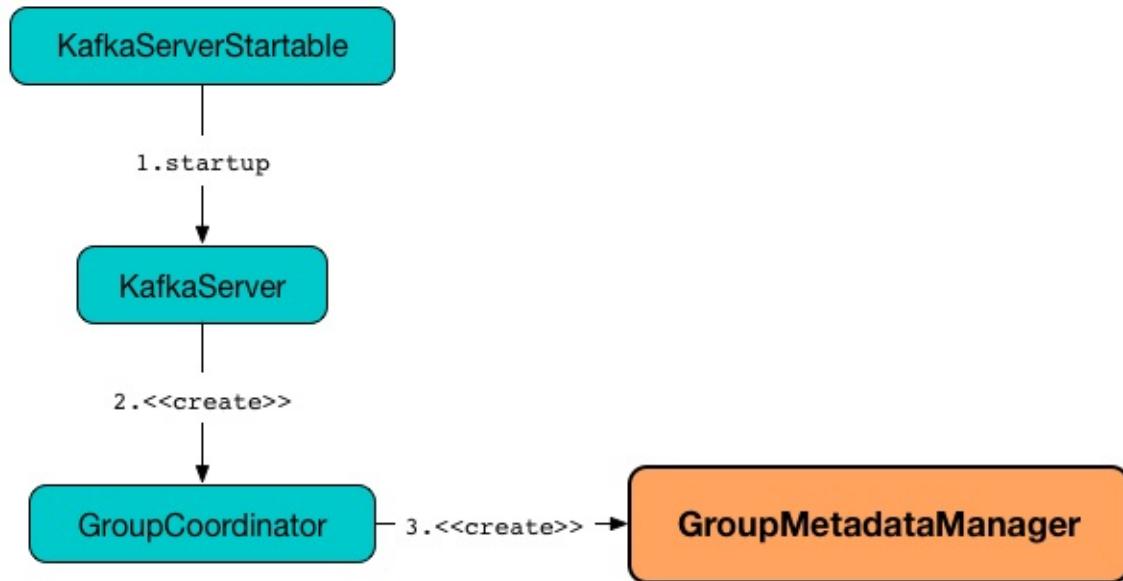


Figure 1. Creating GroupMetadataManager

`GroupMetadataManager` is a [KafkaMetricsGroup](#) and registers [performance metrics](#).

Table 1. GroupMetadataManager's Performance Metrics

Metric Name	Description
<code>NumGroups</code>	
<code>NumGroupsCompletingRebalance</code>	
<code>NumGroupsDead</code>	
<code>NumGroupsEmpty</code>	
<code>NumGroupsPreparingRebalance</code>	
<code>NumGroupsStable</code>	
<code>NumOffsets</code>	

The [performance metrics](#) are registered in
`kafka.coordinator.group:type=GroupMetadataManager` group.

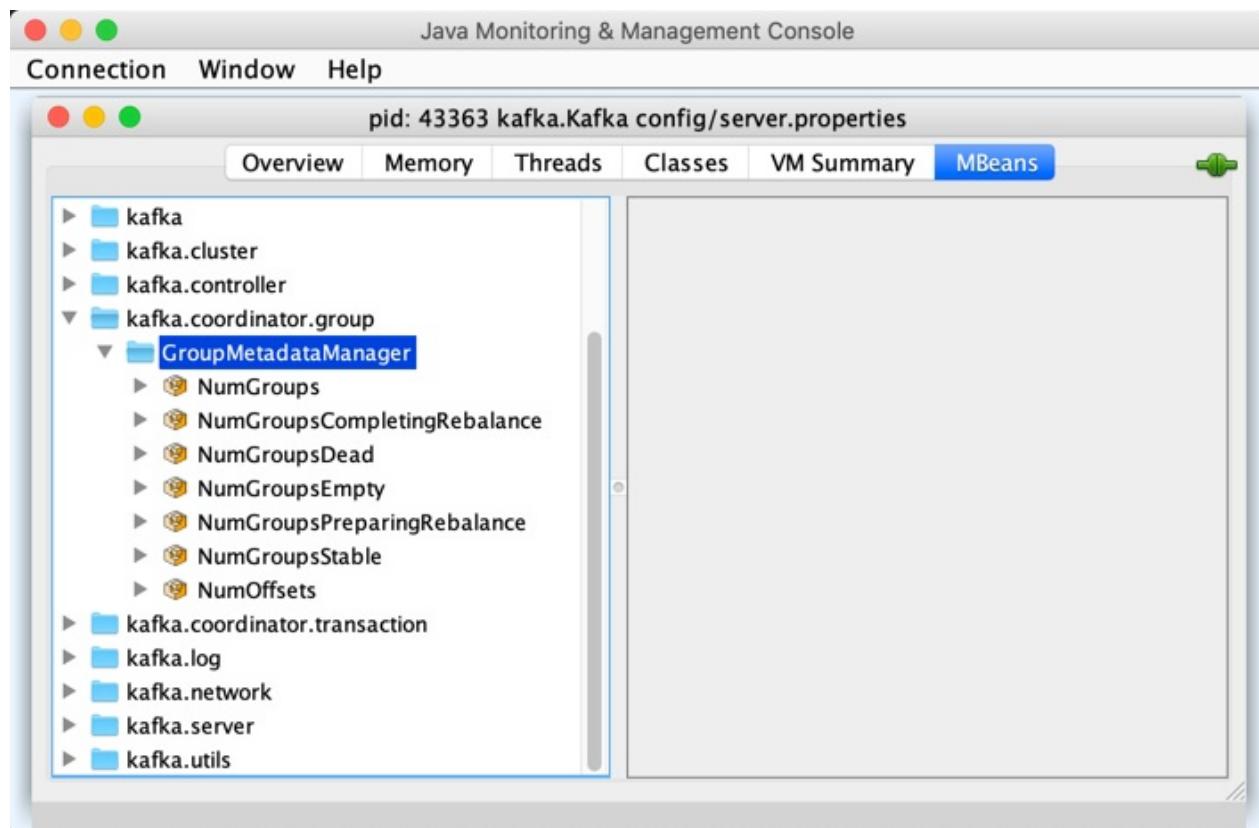


Figure 2. GroupMetadataManager in jconsole

Table 2. GroupMetadataManager's Internal Properties (e.g. Registries and Counters)

Name	Description
compressionType	CompressionType (default: NoCompressionCodec)
groupMetadataCache	GroupMetadata by group ID
groupMetadataTopicPartitionCount	Number of partitions for the __consumer_offsets consumer group metadata topic (default: 50)
scheduler	<p>KafkaScheduler with 1 daemon thread with group-metadata-manager- prefix</p> <p>Started when GroupMetadataManager is requested to start up.</p> <p>Shut down when GroupMetadataManager is requested to shut down.</p> <p>Used when:</p> <ul style="list-style-type: none"> • startup (with enableMetadataExpiration flag on) that schedules the delete-expired-group-metadata task that cleanupGroupMetadata every 600000L millis • scheduleLoadGroupAndOffsets that loads offsets and group metadata from __consumer_offsets topic • removeGroupsForPartition that unloads offsets and group metadata from __consumer_offsets topic • scheduleHandleTxnCompletion that schedules the handleTxnCompletion-[producerId] task that handleTxnCompletion

enableMetadataExpiration Method

```
enableMetadataExpiration(): Unit
```

enableMetadataExpiration requests KafkaScheduler to start.

enableMetadataExpiration schedules **delete-expired-group-metadata** task that **cleanupGroupMetadata** every `offsetsRetentionCheckIntervalMs` milliseconds.

Note

enableMetadataExpiration is used exclusively when GroupCoordinator is started.

Creating GroupMetadataManager Instance

`GroupMetadataManager` takes the following when created:

- Broker ID
- `ApiVersion`
- `OffsetConfig`
- `ReplicaManager`
- `KafkaZkClient`
- `Time`

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

cleanupGroupMetadata Internal Method

```
cleanupGroupMetadata(): Unit  (1)
cleanupGroupMetadata(deletedTopicPartitions: Option[Seq[TopicPartition]]): Unit
```

1. Calls the other `cleanupGroupMetadata` with empty `deletedTopicPartitions` collection

`cleanupGroupMetadata` takes the current time (using `time`) and for every `GroupMetadata` in the `cache` does the following:

1. `FIXME`

In the end, `cleanupGroupMetadata` prints out the following INFO message to the logs:

```
Removed [offsetsRemoved] expired offsets in [duration] milliseconds
```

Note

`cleanupGroupMetadata` is used exclusively when `GroupMetadataManager` is requested to [enableMetadataExpiration](#) (as **delete-expired-group-metadata** task).

Getting Number of Partitions for __consumer_offsets Consumer Group Metadata Topic — `getGroupMetadataTopicPartitionCount` Internal Method

```
getGroupMetadataTopicPartitionCount: Int
```

`getGroupMetadataTopicPartitionCount` requests the [KafkaZkClient](#) for `getTopicPartitionCount` of `__consumer_offsets` consumer group metadata topic.

If not available, `getGroupMetadataTopicPartitionCount` uses the [OffsetConfig](#) for `offsetsTopicNumPartitions` (default: 50).

Note

`getGroupMetadataTopicPartitionCount` is used exclusively when `GroupMetadataManager` is requested for [groupMetadataTopicPartitionCount](#).

shutdown Method

`shutdown(): Unit`

`shutdown ...FIXME`

Note

`shutdown` is used when...FIXME

startup Method

`startup(enableMetadataExpiration: Boolean): Unit`

`startup ...FIXME`

Note

`startup` is used when...FIXME

scheduleLoadGroupAndOffsets Method

`scheduleLoadGroupAndOffsets(offsetsPartition: Int, onGroupLoaded: GroupMetadata => Unit): Unit`

`scheduleLoadGroupAndOffsets ...FIXME`

Note

`scheduleLoadGroupAndOffsets` is used when...FIXME

removeGroupsForPartition Method

`removeGroupsForPartition(
 offsetsPartition: Int,
 onGroupUnloaded: GroupMetadata => Unit): Unit`

```
removeGroupsForPartition ...FIXME
```

Note	removeGroupsForPartition is used when...FIXME
------	---

scheduleHandleTxnCompletion Method

```
scheduleHandleTxnCompletion(  
    producerId: Long,  
    completedPartitions: Set[Int],  
    isCommit: Boolean): Unit
```

```
scheduleHandleTxnCompletion ...FIXME
```

Note	scheduleHandleTxnCompletion is used exclusively when GroupCoordinator is requested to scheduleHandleTxnCompletion.
------	--

loadGroupsAndOffsets Method

```
loadGroupsAndOffsets(  
    topicPartition: TopicPartition,  
    onGroupLoaded: GroupMetadata => Unit): Unit
```

```
loadGroupsAndOffsets ...FIXME
```

Note	loadGroupsAndOffsets is used when...FIXME
------	---

removeGroupsAndOffsets Internal Method

```
removeGroupsAndOffsets(): Unit
```

```
removeGroupsAndOffsets ...FIXME
```

Note	removeGroupsAndOffsets is used when...FIXME
------	---

handleTxnCompletion Method

```
handleTxnCompletion(  
    producerId: Long,  
    completedPartitions: Set[Int],  
    isCommit: Boolean): Unit
```

```
handleTxnCompletion ...FIXME
```

Note	<code>handleTxnCompletion</code> is used exclusively when <code>GroupMetadataManager</code> is requested to scheduleHandleTxnCompletion .
------	---

partitionFor Method

```
partitionFor(groupId: String): Int
```

```
partitionFor ...FIXME
```

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

storeOffsets Method

```
storeOffsets(
    group: GroupMetadata,
    consumerId: String,
    offsetMetadata: immutable.Map[TopicPartition, OffsetAndMetadata],
    responseCallback: immutable.Map[TopicPartition, Errors] => Unit,
    producerId: Long = RecordBatch.NO_PRODUCER_ID,
    producerEpoch: Short = RecordBatch.NO_PRODUCER_EPOCH): Unit
```

```
storeOffsets ...FIXME
```

Note	<code>storeOffsets</code> is used exclusively when <code>GroupCoordinator</code> is requested to doCommitOffsets .
------	--

storeGroup Method

```
storeGroup(
    group: GroupMetadata,
    groupAssignment: Map[String, Array[Byte]],
    responseCallback: Errors => Unit): Unit
```

```
storeGroup ...FIXME
```

Note	<code>storeGroup</code> is used exclusively when <code>GroupCoordinator</code> is requested to doSyncGroup and onCompleteJoin .
------	---

Requesting ReplicaManager to Append Records — appendForGroup Internal Method

```
appendForGroup(
  group: GroupMetadata,
  records: Map[TopicPartition, MemoryRecords],
  callback: Map[TopicPartition, PartitionResponse] => Unit)
```

`appendForGroup` simply requests the `ReplicaManager` to [append records](#).

Note

`appendForGroup` is used exclusively when `GroupMetadataManager` is requested to [storeGroup](#) and [storeOffsets](#).

addGroup Method

```
addGroup(group: GroupMetadata): GroupMetadata
```

`addGroup` ...FIXME

Note

`addGroup` is used when...FIXME

Getting Metadata of Group by Group ID — getGroup Method

```
getGroup(groupId: String): Option[GroupMetadata]
```

`getGroup` finds the `GroupMetadata` for the group ID in the `groupMetadataCache` internal registry.

`getGroup` returns `None` if the metadata could not be found.

Note

`getGroup` is used when:

- `GroupCoordinator` is requested to [handleJoinGroup](#), [handleSyncGroup](#), [handleLeaveGroup](#), [handleDeleteGroups](#), [handleHeartbeat](#), [handleTxnCommitOffsets](#), [handleCommitOffsets](#), and [handleDescribeGroup](#)
- `GroupMetadataManager` is requested to [groupNotExists](#) (when `GroupCoordinator` is requested to [handleDeleteGroups](#)) and [handleTxnCompletion](#) (when `GroupCoordinator` is requested to [scheduleHandleTxnCompletion](#))

readGroupMessageValue Method

```
readGroupMessageValue(  
    groupId: String,  
    buffer: ByteBuffer,  
    time: Time): GroupMetadata
```

`readGroupMessageValue` ...FIXME

Note

- `readGroupMessageValue` is used when:
- `GroupMetadataManager` is requested to `doLoadGroupsAndOffsets`
 - `GroupMetadataMessageFormatter` is requested to `writeTo`
 - `OffsetsMessageParser` is requested to `parseGroupMetadata`

doLoadGroupsAndOffsets Internal Method

```
doLoadGroupsAndOffsets(  
    topicPartition: TopicPartition,  
    onGroupLoaded: GroupMetadata => Unit): Unit
```

`doLoadGroupsAndOffsets` ...FIXME

Note

`doLoadGroupsAndOffsets` is used exclusively when `GroupMetadataManager` is requested to `loadGroupsAndOffsets`.

groupNotExists Method

```
groupNotExists(groupId: String): Boolean
```

`groupNotExists` ...FIXME

Note

`groupNotExists` is used exclusively when `GroupCoordinator` is requested to `handleDeleteGroups`

GroupMetadata — Metadata of Consumer Group

`GroupMetadata` is the metadata of a consumer group.

`GroupMetadata` takes the following to be created:

- Group ID
- `GroupState`
- `Time`

`GroupMetadata` uses the `states` to describe the state of a consumer group.

Table 1. States of Consumer Group

State	Description
<code>CompletingRebalance</code>	<p><code>Transitioned to</code> when:</p> <ul style="list-style-type: none"> • <code>initNextGeneration</code> (and there are any members)
<code>Dead</code>	<p><code>Transitioned to</code> when:</p> <ul style="list-style-type: none"> • <code>GroupCoordinator</code> is requested to <code>handleDeleteGroups</code> (when in <code>Empty</code> state) • <code>GroupCoordinator</code> is requested to <code>onGroupUnloaded</code> • <code>GroupMetadataManager</code> is requested to <code>cleanupGroupMetadata</code>
<code>Empty</code>	<p><code>Transitioned to</code> when:</p> <ul style="list-style-type: none"> • <code>initNextGeneration</code> (and there are no members) • <code>GroupMetadataManager</code> is requested to <code>cleanupGroupMetadata</code> (when in <code>Empty</code> state with no offsets) • <code>GroupMetadataManager</code> is requested to <code>readGroupMessageValue</code> (and there are no members)
<code>PreparingRebalance</code>	<p><code>Transitioned to</code> when:</p> <ul style="list-style-type: none"> • <code>GroupCoordinator</code> is requested to <code>prepareRebalance</code>
<code>Stable</code>	<p><code>Transitioned to</code> when:</p> <ul style="list-style-type: none"> • <code>GroupCoordinator</code> is requested to <code>doSyncGroup</code> (when in <code>CompletingRebalance</code> state and the <code>generationId</code> did not change) • <code>GroupMetadataManager</code> is requested to <code>readGroupMessageValue</code> (and there is at least one member)

`GroupMetadata` defines `validPreviousStates` internal registry of the valid previous states per state.

Table 2. Valid Previous States

State	Valid Previous States
Dead	Stable, PreparingRebalance, CompletingRebalance, Empty, Dead
CompletingRebalance	PreparingRebalance
Stable	CompletingRebalance
PreparingRebalance	Stable, CompletingRebalance, Empty
Empty	PreparingRebalance

`GroupMetadata` is created when:

- `GroupCoordinator` is requested to `handleJoinGroup`, `handleTxnCommitOffsets` and `handleCommitOffsets`
- `GroupMetadata` is requested to `loadGroup`
- `GroupMetadataManager` is requested to `doLoadGroupsAndOffsets`

Table 3. GroupMetadata's Internal Properties (e.g. Registries, Counters and Flags)

Name	Description
<code>generationId</code>	Starts at <code>0</code>
<code>leaderId</code>	ID of the Kafka Consumer that is the leader of the consumer group Starts empty (<code>None</code>) and is initialized in <code>loadGroup</code> , <code>add</code> or <code>remove</code> Used when <code>isLeader</code> and <code>leaderOrNull</code>
<code>members</code>	Members of the consumer group (as <code>MemberMetadata</code> per member ID)

Registering Consumer with Consumer Group — `add` Method

```
add(
  member: MemberMetadata,
  callback: JoinCallback = null): Unit
```

add ...FIXME

Note

`add` is used when:

- `GroupCoordinator` is requested to [addMemberAndRebalance](#)
- `GroupMetadata` is requested to [loadGroup](#)

Loading Consumer Group Metadata — `loadGroup` Factory Method

```
loadGroup(
  groupId: String,
  initialState: GroupState,
  generationId: Int,
  protocolType: String,
  protocol: String,
  leaderId: String,
  currentStateTimestamp: Option[Long],
  members: Iterable[MemberMetadata],
  time: Time): GroupMetadata
```

loadGroup ...FIXME

Note

`loadGroup` is used exclusively when `GroupMetadataManager` is requested to [readGroupMessageValue](#).

Unregistering Consumer from Consumer Group — `remove` Method

```
remove(memberId: String): Unit
```

remove ...FIXME

Note

`remove` is used when `GroupCoordinator` is requested to [removeMemberAndUpdateGroup](#) and [onCompleteJoin](#).

Changing State of Consumer Group — `transitionTo` Method

```
transitionTo(groupState: GroupState): Unit
```

`transitionTo ...FIXME`

Note

`transitionTo` is used when...FIXME

initNextGeneration Method

`initNextGeneration(): Unit`

`initNextGeneration ...FIXME`

Note

`initNextGeneration` is used exclusively when `GroupCoordinator` is requested to [onCompleteJoin](#).

Asserting No Offsets in Use — hasOffsets Method

`hasOffsets: Boolean`

`hasOffsets ...FIXME`

Note

`hasOffsets` is used exclusively when `GroupMetadataManager` is requested to [cleanupGroupMetadata](#)

Kafka — Standalone Command-Line Application

`kafka.Kafka` is a standalone command-line application that [starts a Kafka broker](#).

`kafka.Kafka` is started using [kafka-server-start.sh](#) shell script.

```
// Using sh -xv to trace kafka-server-start.sh
$ sh -xv ./bin/kafka-server-start.sh config/server.properties
...
exec $base_dir/kafka-run-class.sh $EXTRA_ARGS kafka.Kafka "$@"
+ exec ./bin/kafka-run-class.sh -name kafkaServer -loggc kafka.Kafka config/server.properties
...
```

getPropsFromArgs Method

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

Starting Kafka Broker on Command Line — main Method

```
main(args: Array[String]): Unit
```

`main` merges properties and creates a `KafkaServerStartable`.

`main` registers a JVM shutdown hook to shut down `KafkaServerStartable`.

Note	<code>main</code> uses Java's Runtime.addShutdownHook to register the shutdown hook.
------	--

In the end, `main` starts the `KafkaServerStartable` and waits till it finishes.

`main` terminates the JVM with status `0` when `KafkaServerStartable` shuts down properly and with status `1` in case of any exception.

Note	<code>main</code> uses Java's System.exit to terminate a JVM.
------	---

Registering INFO Logging Signal Handlers (for TERM, INT and HUP Signals) — registerLoggingSignalHandler Internal Method

```
registerLoggingSignalHandler(): Unit
```

`registerLoggingSignalHandler` registers signal handlers for `TERM`, `INT` and `HUP` signals so that, once received, it prints out the following INFO message to the logs:

```
Terminating process due to signal [signal]
```

```
$ jps -lm | grep -i kafka
79965 kafka.Kafka config/server.properties

// You could use "pkill -TERM -nf kafka" instead
$ kill -TERM 79965

// In the Kafka server's console
INFO Terminating process due to signal SIGTERM (kafka.Kafka$)
```

Note `registerLoggingSignalHandler` is used exclusively when a Kafka broker is started.

Note `registerLoggingSignalHandler` was added to Kafka 1.0.0 in KAFKA-5679; Add logging for broker termination due to SIGTERM or SIGINT.

KafkaApis — API Request Handler

`KafkaApis` is responsible for handling API requests (by means of `handlers`) to a `KafkaServer` and orchestrating interactions between the services.

Table 1. KafkaApis's API Keys and Handlers

API Key	Handler
AddOffsetsToTxn	handleAddOffsetsToTxnRequest
AlterConfigs	handleAlterConfigsRequest
AlterReplicaLogDirs	handleAlterReplicaLogDirsRequest
ControlledShutdown	handleControlledShutdownRequest
CreatePartitions	handleCreatePartitionsRequest
CreateTopics	handleCreateTopicsRequest
DeleteTopics	handleDeleteTopicsRequest
DescribeGroups	handleDescribeGroupRequest
DescribeLogDirs	handleDescribeLogDirsRequest
DeleteRecords	handleDeleteRecordsRequest
Fetch	handleFetchRequest
FindCoordinator	handleFindCoordinatorRequest
JoinGroup	handleJoinGroupRequest
LeaderAndIsr	handleLeaderAndIsrRequest
ListOffsets	handleListOffsetRequest
Metadata	handleTopicMetadataRequest
OffsetCommit	handleOffsetCommitRequest
OffsetFetch	handleOffsetFetchRequest

OffsetForLeaderEpoch	handleOffsetForLeaderEpochRequest
Produce	handleProduceRequest
StopReplica	handleStopReplicaRequest
UpdateMetadata	handleUpdateMetadataRequest
WriteTxnMarkers	handleWriteTxnMarkersRequest

KafkaApis is created exclusively when KafkaServer is started (and creates KafkaRequestHandlerPool).

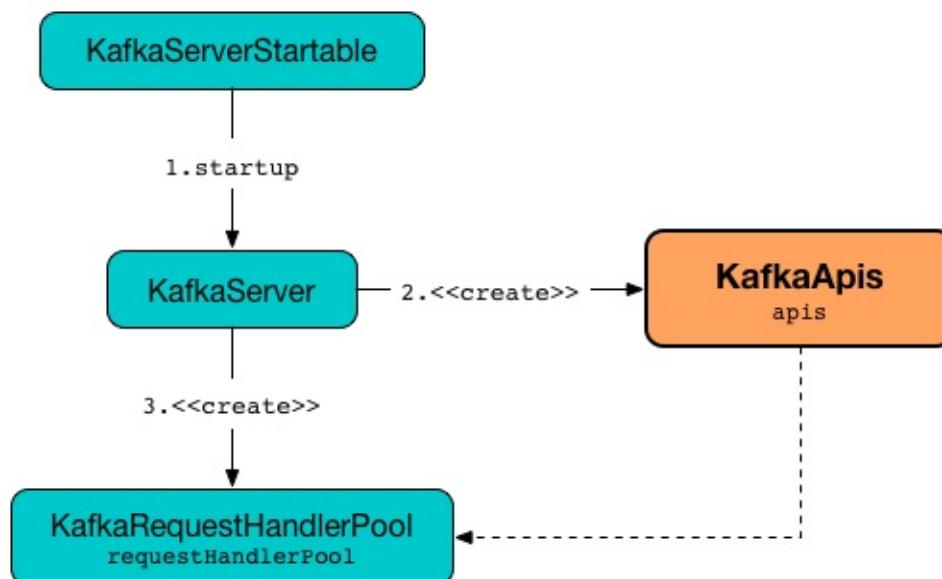


Figure 1. KafkaApis is Created for KafkaRequestHandlerPool when KafkaServer Starts Up
KafkaApis uses the ReplicaManager to handle the following API requests:

- AlterReplicaLogDirs
- DeleteRecords
- DescribeLogDirs
- Fetch
- LeaderAndLsr
- ListOffset
- OffsetForLeaderEpoch
- Produce
- StopReplica

- [UpdateMetadata](#)
- [WriteTxnMarkers](#)

`KafkaApis` uses a [AdminZkClient](#) for...FIXME

`KafkaApis` uses the [MetadataCache](#) for the following:

- [handleTopicMetadataRequest](#) (to get controller ID)
- ...FIXME

Enable ALL logging levels for `kafka.server.KafkaApis` logger to see what happens inside.

Add the following line to `config/log4j.properties`:

```
log4j.logger.kafka.server.KafkaApis=ALL
```

Refer to [Logging](#).

Tip

Please note that Kafka comes with a preconfigured `kafka.server.KafkaApis` logger in `config/log4j.properties`:

```
log4j.appendер.requestAppender=org.apache.log4j.DailyRollingFileAppender
log4j.appendер.requestAppender.DatePattern='.' 'yyyy-MM-dd-HH
log4j.appendер.requestAppender.File=${kafka.logs.dir}/kafka-request.log
log4j.appendер.requestAppender.layout=org.apache.log4j.PatternLayout
log4j.appendер.requestAppender.layout.ConversionPattern=[%d] %p %m (%c)%n

log4j.logger.kafka.server.KafkaApis=TRACE, requestAppender
log4j.additivity.kafka.server.KafkaApis=false
```

That means that the logs of `KafkaApis` go to `logs/kafka-request.log` file at `TRACE` logging level and are not added to the main logs (per `log4j.additivity` being off).

Routing API Requests to Respective Handlers — `handle` Method

```
handle(request: RequestChannel.Request): Unit
```

`handle` first prints out the following TRACE message to the logs:

```
Handling request:[request] from connection [id];securityProtocol:[protocol],principal:[principal]
```

`handle` then relays the input `RequestChannel.Request` to the corresponding `handler` per the `apiKey` (from the header of the input `request`).

Note	<code>handle</code> is used exclusively when <code>KafkaRequestHandler</code> thread is requested to run.
------	---

Handling LeaderAndIsr Request

— `handleLeaderAndIsrRequest` Handler

```
handleLeaderAndIsrRequest(request: RequestChannel.Request): Unit
```

In summary, `handleLeaderAndIsrRequest` requests the `ReplicaManager` to [becomeLeaderOrFollower](#).

`handleLeaderAndIsrRequest` takes the `LeaderAndIsrRequest` from the body (of the `RequestChannel.Request`).

When [authorized cluster action](#), `handleLeaderAndIsrRequest` requests the `ReplicaManager` to [becomeLeaderOrFollower](#) (with the `onLeadershipChange` callback).

When [unauthorized](#), `handleLeaderAndIsrRequest` ...FIXME

Note	<code>handleLeaderAndIsrRequest</code> is used exclusively when <code>KafkaApis</code> is requested to handle a <code>LeaderAndIsr</code> request.
------	--

onLeadershipChange Callback

```
onLeadershipChange(
  updatedLeaders: Iterable[Partition],
  updatedFollowers: Iterable[Partition]): Unit
```

`onLeadershipChange` ...FIXME

Handling AlterReplicaLogDirs Request

— `handleAlterReplicaLogDirsRequest` Handler

```
handleAlterReplicaLogDirsRequest(request: RequestChannel.Request): Unit
```

In summary, `handleAlterReplicaLogDirsRequest` requests the `ReplicaManager` to [alterReplicaLogDirs](#).

`handleAlterReplicaLogDirsRequest` ...FIXME

Note

`handleAlterReplicaLogDirsRequest` is used exclusively when `KafkaApis` is requested to handle a `AlterReplicaLogDirs` request.

Handling CreateTopics Request**— handleCreateTopicsRequest Handler**

```
handleCreateTopicsRequest(request: RequestChannel.Request): Unit
```

`handleCreateTopicsRequest` ...FIXME

`handleCreateTopicsRequest` checks whether `KafkaController` is active...FIXME

`handleCreateTopicsRequest` authorizes the `create` operation for `ClusterResource` ...FIXME

In the end, `handleCreateTopicsRequest` requests `AdminManager` to create the topics.

Note

`handleCreateTopicsRequest` is used exclusively when `KafkaApis` is requested to handle a `CreateTopics` request.

Handling OffsetFetch Request**— handleOffsetFetchRequest Handler**

```
handleOffsetFetchRequest(request: RequestChannel.Request): Unit
```

`handleOffsetFetchRequest` ...FIXME

Note

`handleOffsetFetchRequest` is used exclusively when `KafkaApis` is requested to handle a `OffsetFetch` request.

Handling Fetch Request— handleFetchRequest Handler

```
handleFetchRequest(request: RequestChannel.Request): Unit
```

In summary, `handleFetchRequest` requests the `ReplicaManager` to `fetchMessages`.

`handleFetchRequest` ...FIXME

Note

`handleFetchRequest` is used exclusively when `KafkaApis` is requested to handle a `Fetch` request.

Handling Metadata Request — `handleTopicMetadataRequest` Handler

```
handleTopicMetadataRequest(request: RequestChannel.Request): Unit
```

`handleTopicMetadataRequest` takes the [MetadataRequest](#) from the body (of the [RequestChannel.Request](#)).

`handleTopicMetadataRequest` requests the [MetadataCache](#) for `getAllTopics` or its subset (per `topics` attribute of the `MetadataRequest`).

`handleTopicMetadataRequest` filters out the topics for which the current principal (user) is not authorized to execute `Describe` operation.

For every authorized topic, `handleTopicMetadataRequest` ...[FIXME](#)

`handleTopicMetadataRequest` creates a `MetadataResponse.TopicMetadata` with `TOPIC_AUTHORIZATION_FAILED` for every `unauthorizedForCreateTopics` and `unauthorizedForDescribeTopics`.

`handleTopicMetadataRequest` [getTopicMetadata](#) if there are `authorizedTopics`.

`handleTopicMetadataRequest` prints out the following TRACE message to the logs:

```
Sending topic metadata [completeTopicMetadata] and brokers [brokers] for correlation id [correlationId] to client [clientId]
```

In the end, `handleTopicMetadataRequest` [sendResponseMaybeThrottle](#) with a new [MetadataResponse](#).

Note	<code>handleTopicMetadataRequest</code> is used exclusively when <code>KafkaApis</code> is requested to handle a Metadata request.
------	--

Authorizing Operation on Resource — `authorize` Internal Method

```
authorize(
    session: RequestChannel.Session,
    operation: Operation,
    resource: Resource): Boolean
```

`authorize` simply requests the [Authorizer](#) to [authorize](#) the given `operation` on the given `Resource` in the `RequestChannel.Session`.

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

Handling CreatePartitions Request

— `handleCreatePartitionsRequest` Handler

```
handleCreatePartitionsRequest(request: RequestChannel.Request): Unit
```

`handleCreatePartitionsRequest` ...FIXME

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

Handling DeleteTopics Request

— `handleDeleteTopicsRequest` Handler

```
handleDeleteTopicsRequest(request: RequestChannel.Request): Unit
```

`handleDeleteTopicsRequest` ...FIXME

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

Handling ControlledShutdown Request

— `handleControlledShutdownRequest` Handler

```
handleControlledShutdownRequest(request: RequestChannel.Request): Unit
```

`handleControlledShutdownRequest` ...FIXME

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

Creating KafkaApis Instance

`KafkaApis` takes the following when created:

- `RequestChannel`
- `ReplicaManager`
- `AdminManager`
- `GroupCoordinator`

- TransactionCoordinator
- KafkaController
- KafkaZkClient
- Broker ID
- KafkaConfig
- MetadataCache
- Metrics
- Authorizer
- QuotaManagers
- FetchManager
- BrokerTopicStats
- Cluster ID
- Time
- DelegationTokenManager

`KafkaApis` initializes the internal registries and counters.

fetchOffsetForTimestamp Internal Method

```
fetchOffsetForTimestamp(topicPartition: TopicPartition, timestamp: Long): Option[Time
stampOffset]
```

`fetchOffsetForTimestamp` ...FIXME

Note	<code>fetchOffsetForTimestamp</code> is used exclusively when <code>KafkaApis</code> is requested to <code>handleListOffsetRequestV1AndAbove</code> .
------	---

handleListOffsetRequestV0 Internal Method

```
handleListOffsetRequestV0(
    request : RequestChannel.Request) : Map[TopicPartition, ListOffsetResponse.Partition
Data]
```

`handleListOffsetRequestV0` ...FIXME

Note

`handleListOffsetRequestV0` is used exclusively when `KafkaApis` is requested to `handleListOffsetRequest` (for the API version `0`).

handleListOffsetRequestV1AndAbove Internal Method

```
handleListOffsetRequestV1AndAbove(  
    request: RequestChannel.Request): Map[TopicPartition, ListOffsetResponse.PartitionData]
```

`handleListOffsetRequestV1AndAbove` ...FIXME

Note

`handleListOffsetRequestV1AndAbove` is used exclusively when `KafkaApis` is requested to `handleListOffsetRequest` (for the API version `1` or above).

Handling DescribeLogDirs Request — handleDescribeLogDirsRequest Handler

```
handleDescribeLogDirsRequest(request: RequestChannel.Request): Unit
```

In summary, `handleDescribeLogDirsRequest` requests the `ReplicaManager` to `describeLogDirs`.

Internally, `handleDescribeLogDirsRequest` takes the `DescribeLogDirsRequest` from the body (of the `RequestChannel.Request`).

`handleDescribeLogDirsRequest` branches off per whether the `DescribeLogDirsRequest` was for `isAllTopicPartitions` or not.

- For all `TopicPartitions`, `handleDescribeLogDirsRequest` requests the `ReplicaManager` for the `LogManager` that is requested for all the partition logs and their `TopicPartitions`.
- For specific `TopicPartitions`, `handleDescribeLogDirsRequest` requests them from the `DescribeLogDirsRequest`.

Note

`handleDescribeLogDirsRequest` returns an empty list of log directories when the request is not `authorized`.

`handleDescribeLogDirsRequest` then requests the `ReplicaManager` to `describeLogDirs` with the requested `TopicPartitions`.

In the end, `handleDescribeLogDirsRequest` `sendResponseMaybeThrottle` with a `DescribeLogDirsResponse` and the `LogDirInfos`.

Note

`handleDescribeLogDirsRequest` is used exclusively when `KafkaApis` is requested to handle a [DescribeLogDirs](#) request.

sendResponseMaybeThrottle Internal Method

```
sendResponseMaybeThrottle(  
    request: RequestChannel.Request,  
    createResponse: Int => AbstractResponse,  
    onComplete: Option[Send => Unit] = None): Unit
```

`sendResponseMaybeThrottle` ...FIXME

Note

`sendResponseMaybeThrottle` is used when...FIXME

fetchOffsetsBefore Method

```
fetchOffsetsBefore(log: Log, timestamp: Long, maxNumOffsets: Int): Seq[Long]
```

`fetchOffsetsBefore` ...FIXME

Note

`fetchOffsetsBefore` is used exclusively when `KafkaApis` is requested to [fetchOffsets](#).

fetchOffsets Method

```
fetchOffsets(  
    logManager: LogManager,  
    topicPartition: TopicPartition,  
    timestamp: Long,  
    maxNumOffsets: Int): Seq[Long]
```

`fetchOffsets` ...FIXME

Note

`fetchOffsets` is used exclusively when `KafkaApis` is requested to [handleListOffsetRequestV0](#).

Handling StopReplica Request — `handleStopReplicaRequest` Handler

```
handleStopReplicaRequest(request: RequestChannel.Request): Unit
```

In summary, `handleStopReplicaRequest` requests the `ReplicaManager` to [stopReplicas](#).

`handleStopReplicaRequest ...FIXME`

Note

`handleStopReplicaRequest` is used exclusively when `KafkaApis` is requested to handle a [StopReplica](#) request.

Handling UpdateMetadata Request

— `handleUpdateMetadataRequest` Handler

`handleUpdateMetadataRequest(request: RequestChannel.Request): Unit`

In summary, `handleUpdateMetadataRequest` requests the `ReplicaManager` to [maybeUpdateMetadataCache](#).

`handleUpdateMetadataRequest ...FIXME`

Note

`handleUpdateMetadataRequest` is used exclusively when `KafkaApis` is requested to handle a [UpdateMetadata](#) request.

Handling OffsetCommitRequest

— `handleOffsetCommitRequest` Handler

`handleOffsetCommitRequest(request: RequestChannel.Request): Unit`

`handleOffsetCommitRequest` takes the [OffsetCommitRequest](#) from the body (of the `RequestChannel.Request`).

If [authorized](#), `handleOffsetCommitRequest` simply requests the `GroupCoordinator` to [handleCommitOffsets](#) (with the `sendResponseCallback`).

Note

If [authorized](#), `handleOffsetCommitRequest` branches off per API version (i.e. `0` to store offsets in Zookeeper and `1` and beyond). The API version `0` is not described here.

If not [authorized](#), `handleOffsetCommitRequest ...FIXME`

Note

`handleOffsetCommitRequest` is used exclusively when `KafkaApis` is requested to handle an [OffsetCommit](#) request.

`sendResponseCallback` Method

```
sendResponseCallback(commitStatus: immutable.Map[TopicPartition, Errors]): Unit
```

`sendResponseCallback` prints out the following DEBUG message to the logs for offsets with errors (i.e. unauthorized topics to read or non-existing topics):

```
Offset commit request with correlation id [correlationId] from client [clientId] on partition [topicPartition] failed due to [exceptionName]
```

In the end, `sendResponseCallback` [sendResponseMaybeThrottle](#) a new `OffsetCommitResponse`.

createInternalTopic Internal Method

```
createInternalTopic(topic: String): MetadataResponse.TopicMetadata
```

```
createInternalTopic ...FIXME
```

Note

`createInternalTopic` is used when `KafkaApis` is requested to [getOrCreateInternalTopic](#) and [getTopicMetadata](#).

getOrCreateInternalTopic Internal Method

```
getOrCreateInternalTopic(
    topic: String,
    listenerName: ListenerName): MetadataResponse.TopicMetadata
```

`getOrCreateInternalTopic` requests the [MetadataCache](#) for [getTopicMetadata](#) for the input `topic` (and the `ListenerName`).

In the end, `getOrCreateInternalTopic` returns the `TopicMetadata` if available or [createInternalTopic](#).

Note

`getOrCreateInternalTopic` is used exclusively when `KafkaApis` is requested to [handle a FindCoordinatorRequest](#).

getTopicMetadata Internal Method

```
getTopicMetadata(
    allowAutoTopicCreation: Boolean,
    topics: Set[String],
    listenerName: ListenerName,
    errorUnavailableEndpoints: Boolean,
    errorUnavailableListeners: Boolean): Seq[MetadataResponse.TopicMetadata]
```

getTopicMetadata ...FIXME

Note	getTopicMetadata is used exclusively when KafkaApis is requested to handle Metadata request.
------	--

Handling DescribeGroups Request

— handleDescribeGroupRequest Handler

```
handleDescribeGroupRequest(request: RequestChannel.Request): Unit
```

handleDescribeGroupRequest ...FIXME

Note	handleDescribeGroupRequest is used exclusively when KafkaApis is requested to handle a DescribeGroups request.
------	--

Handling AlterConfigs Request

— handleAlterConfigsRequest Handler

```
handleAlterConfigsRequest(request: RequestChannel.Request): Unit
```

handleAlterConfigsRequest ...FIXME

Note	handleAlterConfigsRequest is used exclusively when KafkaApis is requested to handle a AlterConfigs request.
------	---

createTopic Internal Method

```
createTopic(
    topic: String,
    numPartitions: Int,
    replicationFactor: Int,
    properties: Properties = new Properties()): MetadataResponse.TopicMetadata
```

createTopic ...FIXME

Note

`createTopic` is used when `KafkaApis` is requested to `createInternalTopic` and `getTopicMetadata`.

Handling FindCoordinatorRequest

— `handleFindCoordinatorRequest` Handler

```
handleFindCoordinatorRequest(request: RequestChannel.Request): Unit
```

`handleFindCoordinatorRequest` takes the `FindCoordinatorRequest` from the body (of the `RequestChannel.Request`).

`handleFindCoordinatorRequest` checks permissions...FIXME

For an authorized request, `handleFindCoordinatorRequest` branches off per `CoordinatorType`, i.e. `GROUP` or `TRANSACTION`.

For `GROUP` coordinator type, `handleFindCoordinatorRequest` does the following:

1. Requests the `GroupCoordinator` for `partitionFor` the `coordinator key` (of the `FindCoordinatorRequest`)
2. `getOrCreateInternalTopic` for `__consumer_offsets` topic

For `TRANSACTION` coordinator type, `handleFindCoordinatorRequest` does the following:

1. Requests the `TransactionCoordinator` for `partitionFor` (for the `coordinatorKey` of the `FindCoordinatorRequest`)
2. `getOrCreateInternalTopic` for `__transaction_state` topic

In the end, `handleFindCoordinatorRequest` `sendResponseMaybeThrottle` with a new `FindCoordinatorResponse`.

You should see the following TRACE message in the logs:

```
Sending FindCoordinator response [body] for correlation id [correlationId] to client [clientId].
```

Note

`handleFindCoordinatorRequest` is used exclusively when `KafkaApis` is requested to handle a `FindCoordinator` request.

Handling JoinGroupRequest

— `handleJoinGroupRequest` Handler

```
handleJoinGroupRequest(request: RequestChannel.Request): Unit
```

`handleJoinGroupRequest` takes the [JoinGroupRequest](#) from the body (of the [RequestChannel.Request](#)) and simply requests the [GroupCoordinator](#) to [handleJoinGroup](#) (with [sendResponseCallback](#) to handle the response).

Note

`handleJoinGroupRequest` is used exclusively when [KafkaApis](#) is requested to handle a [JoinGroup](#) request.

Handling JoinGroup Response — `sendResponseCallback` Method

```
sendResponseCallback(joinResult: JoinGroupResult): Unit
```

`sendResponseCallback` creates a new [JoinGroupResponse](#) for the given [JoinGroupResult](#) and prints out the following TRACE message to the logs:

```
Sending join group response [responseBody] for correlation id [correlationId] to client [clientId].
```

In the end, `sendResponseCallback` [sendResponseMaybeThrottle](#) with the new [JoinGroupResponse](#).

Handling AddOffsetsToTxn Request — `handleAddOffsetsToTxnRequest` Handler

```
handleAddOffsetsToTxnRequest(request: RequestChannel.Request): Unit
```

`handleAddOffsetsToTxnRequest` ...FIXME

Note

`handleAddOffsetsToTxnRequest` is used exclusively when [KafkaApis](#) is requested to handle a [AddOffsetsToTxn](#) request.

Handling Produce Request — `handleProduceRequest` Handler

```
handleProduceRequest(request: RequestChannel.Request): Unit
```

In summary, `handleProduceRequest` takes the `ProduceRequest` from the body (of the `RequestChannel.Request`) and requests the `ReplicaManager` to `appendRecords` (with `isFromClient` flag enabled).

Note	<code>internalTopicsAllowed</code> flag (when the <code>ReplicaManager</code> is requested to <code>appendRecords</code>) is enabled (<code>true</code>) only when the client ID is <code>__admin_client</code> .
------	--

`handleProduceRequest ...FIXME`

Note	<code>handleProduceRequest</code> is used exclusively when <code>KafkaApis</code> is requested to handle a <code>Produce</code> request.
------	--

handleWriteTxnMarkersRequest Handler

<code>handleWriteTxnMarkersRequest(request: RequestChannel.Request): Unit</code>
--

In summary, `handleWriteTxnMarkersRequest` requests the `ReplicaManager` to `getMagic` followed by `appendRecords` (with `isFromClient` flag disabled).

`handleWriteTxnMarkersRequest ...FIXME`

Note	<code>handleWriteTxnMarkersRequest</code> is used exclusively when <code>KafkaApis</code> is requested to handle a <code>WriteTxnMarkers</code> request.
------	--

handleDeleteRecordsRequest Handler

<code>handleDeleteRecordsRequest(request: RequestChannel.Request): Unit</code>
--

In summary, `handleDeleteRecordsRequest` requests the `ReplicaManager` to `deleteRecords`.

`handleDeleteRecordsRequest ...FIXME`

Note	<code>handleDeleteRecordsRequest</code> is used exclusively when <code>KafkaApis</code> is requested to handle a <code>DeleteRecords</code> request.
------	--

handleOffsetForLeaderEpochRequest Handler

<code>handleOffsetForLeaderEpochRequest(request: RequestChannel.Request): Unit</code>

In summary, `handleOffsetForLeaderEpochRequest` requests the `ReplicaManager` to `lastOffsetForLeaderEpoch`.

```
handleOffsetForLeaderEpochRequest ...FIXME
```

Note

`handleOffsetForLeaderEpochRequest` is used exclusively when `KafkaApis` is requested to handle a [OffsetForLeaderEpoch](#) request.

handleListOffsetRequest Handler

```
handleListOffsetRequest(request: RequestChannel.Request): Unit
```

In summary, `handleListOffsetRequest` requests the [ReplicaManager](#) to [fetchOffsetForTimestamp](#).

```
handleListOffsetRequest ...FIXME
```

Note

`handleListOffsetRequest` is used exclusively when `KafkaApis` is requested to handle a [ListOffsets](#) request.

isAuthorizedClusterAction Internal Method

```
isAuthorizedClusterAction(request: RequestChannel.Request): Boolean
```

`isAuthorizedClusterAction` simply [authorize](#) with `ClusterAction` operation and `ClusterResource` resource.

Note

`isAuthorizedClusterAction` is used when...FIXME

updateRecordConversionStats Internal Method

```
updateRecordConversionStats(  
    request: RequestChannel.Request,  
    tp: TopicPartition,  
    conversionStats: RecordConversionStats): Unit
```

```
updateRecordConversionStats ...FIXME
```

Note

`updateRecordConversionStats` is used when...FIXME

KafkaController

`KafkaController` is a Kafka service that runs on every Kafka broker.

Only one `KafkaController` can be [active \(elected\)](#) among all the Kafka brokers in a Kafka cluster. The process of promoting a `KafkaController` to be active among brokers is [Kafka Controller Election](#).

`KafkaController` is a Kafka service responsible for:

- [topic deletion](#)
- ...FIXME

Quoting [Kafka Controller Internals](#):

In a Kafka cluster, one of the brokers serves as the controller, which is responsible for managing the states of partitions and replicas and for performing administrative tasks like reassigning partitions.

`KafkaController` is [created](#) and immediately [started](#) when `KafkaServer` is requested to [start up](#).

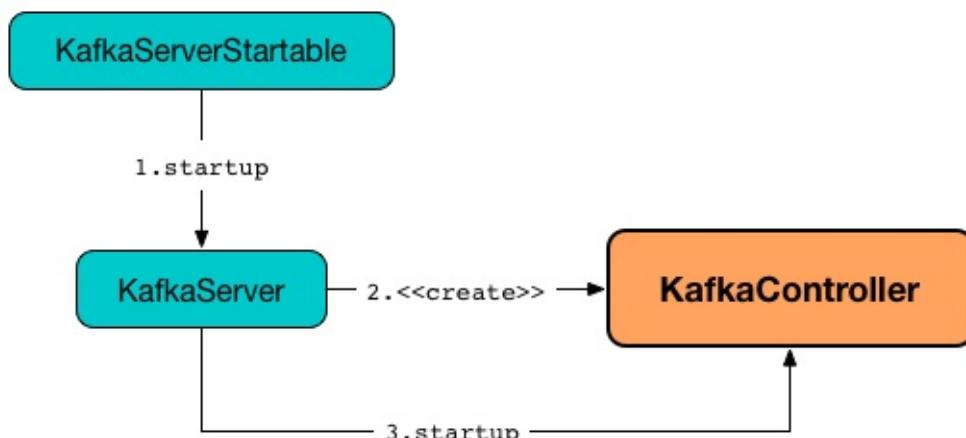


Figure 1. KafkaController

`KafkaController` is in one of the [ControllerStates](#) (that is the [state](#) of the `ControllerEventManager`).

`KafkaController` uses the `KafkaZkClient` for the following:

- Creating the `TopicDeletionManager`, the `ReplicaStateMachine` and the `PartitionStateMachine`
- Registering a `stateChangeHandler` when requested to [startup](#)
- Updating `BrokerInfo`

- `onControllerFailover`
- `onControllerResignation`
- `registerBrokerModificationsHandler`
- `unregisterBrokerModificationsHandler`
- `maybeTriggerPartitionReassignment`
- `incrementControllerEpoch`
- `initializeControllerContext`
- `fetchPendingPreferredReplicaElections`
- `initializePartitionReassignment`
- `fetchTopicDeletionsInProgress`
- `updateLeaderAndIsrCache`
- `areReplicasInIsr`
- `updateAssignedReplicasForPartition`
- `registerPartitionModificationsHandlers`
- `unregisterPartitionModificationsHandlers`
- `unregisterPartitionReassignmentIsrChangeHandlers`
- `readControllerEpochFromZooKeeper`
- `removePartitionsFromReassignedPartitions`
- `removePartitionsFromPreferredReplicaElection`
- `updateLeaderEpoch`
- Processing `Startup`, `BrokerChange`, `BrokerModifications`, `TopicChange`,
`LogDirEventNotification`, `PartitionModifications`, `TopicDeletion`, `PartitionReassignment`,
`PartitionReassignmentIsrChange`, `IsrChangeNotification`,
`PreferredReplicaLeaderElection`, `ControllerChange`, `Reelect` and
`RegisterBrokerAndReelect` controller events
- `triggerControllerMove`
- `elect`

KafkaController uses [listeners](#) as a notification system to monitor znodes in Zookeeper and react accordingly.

KafkaController emulates a state machine using [controller events](#).

Table 1. KafkaController's Controller Events

Event	ControllerState	process Handler
BrokerChange	BrokerChange	
ControllerChange • newControllerId : Int	ControllerChange	<ol style="list-style-type: none"> Assigns the current controller as the input newControllerId (only when the broker is longer an active controller) Resigns as the active controller <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> Note Similar to Reelect with the only difference that it does not trigger an election </div>
ControlledShutdown • ID • controlledShutdownCallback : Try[Set[TopicAndPartition]] ⇒ Unit	ControlledShutdown	
IsrChangeNotification		
LogDirEventNotification		
PartitionReassignment		
PartitionReassignmentIsrChange		
PreferredReplicaLeaderElection		
Reelect	ControllerChange	<ol style="list-style-type: none"> Assigns the current controller as activeControllerId (only when the broker is longer an active controller) Resigns as the active controller elect
		<ol style="list-style-type: none"> registerSessionExpirationHandler

Startup	ControllerChange	2. registerControllerChange 3. elect
TopicChange		
TopicDeletion		

Table 2. KafkaController's ZNodeChangeHandler and ZNodeChildChangeHandlers

Name	Description
brokerChangeHandler	ZNodeChildChangeHandler of /brokers/ids path On handleChildChange , brokerChangeHandler simply sends BrokerChange event to the ControllerEventManager .
isrChangeNotificationHandler	ZNodeChildChangeHandler of /isr_change_notification path On handleChildChange , isrChangeNotificationHandler simply sends IsrChangeNotification event to the ControllerEventManager .
logDirEventNotificationHandler	ZNodeChildChangeHandler of /log_dir_event_notification path On handleChildChange , logDirEventNotificationHandler simply sends LogDirEventNotification event to the ControllerEventManager .
partitionModificationsHandlers	ZNodeChangeHandlers per topic of /brokers/topics/[topic] path On handleDataChange , partitionModificationsHandlers simply send PartitionModifications event to the ControllerEventManager .
partitionReassignmentHandler	ZNodeChangeHandler of /admin/reassign_partitions path On handleCreation , partitionReassignmentHandler simply sends PartitionReassignment event to the ControllerEventManager .
	ZNodeChangeHandler of /admin/preferred_replica_election path

<code>preferredReplicaElectionHandler</code>	<p>On <code>handleCreation</code>, <code>preferredReplicaElectionHandler</code> simply sends <code>PreferredReplicaLeaderElection</code> event to the <code>ControllerEventManager</code>.</p>
<code>topicChangeHandler</code>	<p><code>ZNodeChildChangeHandler</code> of <code>/brokers/topics</code> path</p> <p>On <code>handleChildChange</code>, <code>topicChangeHandler</code> simply sends <code>TopicChange</code> event to the <code>ControllerEventManager</code>.</p>
<code>topicDeletionHandler</code>	<p><code>ZNodeChildChangeHandler</code> of <code>/admin/delete_topics</code> path</p> <p>On <code>handleChildChange</code>, <code>topicDeletionHandler</code> simply sends <code>TopicDeletion</code> event to the <code>ControllerEventManager</code>.</p>

Table 3. KafkaController's Internal Properties (e.g. Registries and Counters)

Name	Description
<code>activeControllerId</code>	<p>The ID of the active <code>KafkaController</code></p> <ul style="list-style-type: none"> Initialized to <code>-1</code>
<code>brokerRequestBatch</code>	<code>ControllerBrokerRequestBatch</code> (with the <code>StateChangeLogger</code>)
<code>controllerChangeHandler</code>	<p>A <code>zNodeChangeHandler</code> (for the <code>KafkaController</code> and the <code>ControllerEventManager</code>) that listens to change events on <code>/controller</code> znode.</p> <p><code>controllerChangeHandler</code> emits controller events as follows:</p> <ul style="list-style-type: none"> <code>ControllerChange</code> when the znode is created or the znode data changed <code>Reelect</code> when the znode is deleted
<code>controllerContext</code>	<code>ControllerContext</code>
	<p><code>ControllerEventManager</code> (with <code>rateAndTimeMetrics</code> of the <code>ControllerContext</code>, the <code>updateMetrics</code> as the <code>eventProcessedListener</code> and the <code>maybeResign</code> as the <code>controllerMovedListener</code>)</p> <p><code>eventManager</code> is used to create other internal components to allow them for emitting controller events at state changes:</p> <ul style="list-style-type: none"> <code>TopicDeletionManager</code>

	<ul style="list-style-type: none"> • ControllerChangeHandler • BrokerChangeHandler • TopicChangeHandler • TopicDeletionHandler • PartitionReassignmentHandler • PreferredReplicaElectionHandler • IsrChangeNotificationHandler • LogDirEventNotificationHandler • BrokerModificationsHandlers • PartitionReassignmentIsrChangeHandlers • PartitionModificationsHandlers • PartitionReassignmentIsrChangeHandlers <p><code>eventManager</code> is started when <code>KafkaController</code> is requested to start.</p> <p><code>eventManager</code> is closed when <code>KafkaController</code> is requested to shutdown.</p>
<code>kafkaScheduler</code>	<code>KafkaScheduler</code> with 1 daemon thread with kafka-scheduler prefix
<code>partitionStateMachine</code>	<code>PartitionStateMachine</code>
<code>replicaStateMachine</code>	<code>ReplicaStateMachine</code>
<code>stateChangeLogger</code>	<code>StateChangeLogger</code> with the <code>broker ID</code> and <code>inControllerContext</code> flag enabled
<code>tokenCleanScheduler</code>	<code>KafkaScheduler</code> with 1 daemon thread with delegation-token-cleaner prefix
<code>topicDeletionManager</code>	<code>TopicDeletionManager</code>

Table 4. KafkaController's Listeners

Listener	Description
brokerChangeListener	BrokerChangeListener for this KafkaController and eventManager
isrChangeNotificationListener	<p>IsrChangeNotificationListener for this KafkaController and eventManager</p> <p>Registered in registerIsrChangeNotificationListener when KafkaController does onControllerFailover.</p> <p>De-registered in deregisterIsrChangeNotificationListener when KafkaController resigns as the active controller.</p>
logDirEventNotificationListener	LogDirEventNotificationListener
partitionModificationsListeners	PartitionModificationsListener by name
partitionReassignmentListener	PartitionReassignmentListener for this KafkaController and ControllerEventManager
preferredReplicaElectionListener	PreferredReplicaElectionListener for this KafkaController and ControllerEventManager
topicDeletionListener	<p>TopicDeletionListener (for this KafkaController and ControllerEventManager)</p> <p>Registered in registerTopicDeletionListener when KafkaController does onControllerFailover.</p> <p>De-registered in deregisterTopicDeletionListener when KafkaController resigns as the active controller.</p>

KafkaController uses [Controller id=[brokerId]] as the logging prefix (aka logIdent).

Enable `WARN`, `INFO` or `DEBUG` logging levels for `kafka.controller.KafkaController` logger to see what happens inside.

Add the following line to `config/log4j.properties`:

```
log4j.logger.kafka.controller.KafkaController=DEBUG
```

Refer to [Logging](#).

Tip Please note that Kafka comes with a preconfigured `kafka.controller` logger in `config/log4j.properties`:

```
log4j.appenders.controllerAppender=org.apache.log4j.DailyRollingFileAppender
log4j.appenders.controllerAppender.DatePattern=''.yyyy-MM-dd-HH
log4j.appenders.controllerAppender.File=${kafka.logs.dir}/controller.log
log4j.appenders.controllerAppender.layout=org.apache.log4j.PatternLayout
log4j.appenders.controllerAppender.layout.ConversionPattern=[%d] %p %m (%c)%n

log4j.logger.kafka.controller=TRACE, controllerAppender
log4j.additivity.kafka.controller=false
```

That means that the logs of `KafkaController` go to `logs/controller.log` file at `TRACE` logging level and are not added to the main logs (per `log4j.additivity` being off).

initiateReassignReplicasForTopicPartition Method

```
initiateReassignReplicasForTopicPartition
```

```
initiateReassignReplicasForTopicPartition ...FIXME
```

Note

`initiateReassignReplicasForTopicPartition` is used when...FIXME

deregisterPartitionReassignmentIsrChangeListeners Method

```
deregisterPartitionReassignmentIsrChangeListeners
```

```
deregisterPartitionReassignmentIsrChangeListeners ...FIXME
```

Note

`deregisterPartitionReassignmentIsrChangeListeners` is used when...FIXME

resetControllerContext Method

```
resetControllerContext
```

```
resetControllerContext ...FIXME
```

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

deregisterBrokerChangeListener Method

```
deregisterBrokerChangeListener
```

```
deregisterBrokerChangeListener ...FIXME
```

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

deregisterTopicChangeListener Method

```
deregisterTopicChangeListener
```

```
deregisterTopicChangeListener ...FIXME
```

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

Resigning As Active Controller — onControllerResignation Method

```
onControllerResignation(): Unit
```

`onControllerResignation` starts by printing out the following DEBUG message to the logs:

```
Resigning
```

`onControllerResignation` unsubscribes from intercepting Zookeeper events for the following znodes in order:

1. Child changes to `/isr_change_notification` znode
2. Data changes to `/admin/reassign_partitions` znode

3. Data changes to /admin/preferred_replica_election znode

4. Child changes to /log_dir_event_notification znode

`onControllerResignation` requests [TopicDeletionManager](#) to [reset](#).

`onControllerResignation` requests [KafkaScheduler](#) to [shutdown](#).

`onControllerResignation` resets the following internal counters:

- `offlinePartitionCount`
- `preferredReplicaBalanceCount`
- `globalTopicCount`
- `globalPartitionCount`

`onControllerResignation` [deregisterPartitionReassignmentIsrChangeListeners](#).

`onControllerResignation` requests [PartitionStateMachine](#) to [shutdown](#).

`onControllerResignation` [deregisterTopicChangeListener](#).

`onControllerResignation` [deregisterPartitionModificationsListener](#) every listener in `partitionModificationsListeners`.

`onControllerResignation` [deregisterTopicDeletionListener](#).

`onControllerResignation` requests [ReplicaStateMachine](#) to [shutdown](#).

`onControllerResignation` [deregisterBrokerChangeListener](#).

`onControllerResignation` [resetControllerContext](#).

In the end, `onControllerResignation` prints out the following DEBUG message to the logs:

Resigned

Note	<p><code>onControllerResignation</code> is used when:</p> <ul style="list-style-type: none"> • <code>ControllerEventThread</code> is requested to process controller events, i.e. ControllerChange and Reelect • triggerControllerMove • <code>KafkaController</code> is requested to shut down
------	--

Unsubscribing from Child Changes to /isr_change_notification ZNode

— deregisterIsrChangeNotificationListener Internal Method

```
deregisterIsrChangeNotificationListener(): Unit
```

`deregisterIsrChangeNotificationListener` prints out the following DEBUG message to the logs:

```
De-registering IsrChangeNotificationListener
```

`deregisterIsrChangeNotificationListener` requests [ZkUtils](#) to unsubscribe from intercepting changes to `/isr_change_notification` znode with [IsrChangeNotificationListener](#).

Note

`deregisterIsrChangeNotificationListener` is used exclusively when KafkaController [resigns as the active controller](#).

Unsubscribing from Child Changes to /log_dir_event_notification ZNode

— deregisterLogDirEventNotificationListener Internal Method

```
deregisterLogDirEventNotificationListener(): Unit
```

`deregisterLogDirEventNotificationListener` prints out the following DEBUG message to the logs:

```
De-registering logDirEventNotificationListener
```

`deregisterLogDirEventNotificationListener` requests [ZkUtils](#) to unsubscribe from intercepting changes to `/log_dir_event_notification` znode with [LogDirEventNotificationListener](#).

Note

`deregisterLogDirEventNotificationListener` is used exclusively when KafkaController [resigns as the active controller](#).

Unsubscribing from Data Changes to /admin/preferred replica election ZNode — deregisterPreferredReplicaElectionListener Method

```
deregisterPreferredReplicaElectionListener(): Unit
```

`deregisterPreferredReplicaElectionListener` requests [ZkUtils](#) to [unsubscribe](#) from intercepting data changes to `/admin/preferred_replica_election` znode with `PreferredReplicaElectionListener`.

Note

`deregisterPreferredReplicaElectionListener` is used exclusively when KafkaController [resigns as the active controller](#).

Unsubscribing from Data Changes to /admin/reassign partitions ZNode — deregisterPartitionReassignmentListener Method

```
deregisterPartitionReassignmentListener(): Unit
```

`deregisterPartitionReassignmentListener` requests [ZkUtils](#) to [unsubscribe](#) from intercepting data changes to `/admin/reassign_partitions` znode with `PartitionReassignmentListener`.

Note

`deregisterPartitionReassignmentListener` is used exclusively when KafkaController [resigns as the active controller](#).

triggerControllerMove Internal Method

```
triggerControllerMove(): Unit
```

`triggerControllerMove` ...FIXME

Note

`triggerControllerMove` is used when:

1. KafkaController [handleIllegalState](#)
2. KafkaController caught an exception while [electing or becoming a controller](#)

handleIllegalState Internal Method

```
handleIllegalState(e: IllegalStateException): Nothing
```

handleIllegalState ...FIXME

Note

`handleIllegalState` is used when `KafkaController` catches an `IllegalStateException` in `updateLeaderEpochAndSendRequest`, `sendUpdateMetadataRequest` and when processing a `ControlledShutdown` event.

sendUpdateMetadataRequest Method

```
sendUpdateMetadataRequest(): Unit
```

`sendUpdateMetadataRequest` requests the `ControllerBrokerRequestBatch` to `newBatch` and `addUpdateMetadataRequestForBrokers`.

In the end, `sendUpdateMetadataRequest` requests the `ControllerBrokerRequestBatch` to `sendRequestsToBrokers` with the current epoch.

In case of `IllegalStateException`, `sendUpdateMetadataRequest` `handleIllegalState` (that triggers controller movement).

Note

`sendUpdateMetadataRequest` is used when:

- `KafkaController` is requested to `onControllerFailover`, `onBrokerStartup`, `onBrokerUpdate`, `onReplicasBecomeOffline`, `onPartitionReassignment`, process a `IsrChangeNotification` controller event
- `TopicDeletionManager` is requested to `onTopicDeletion`

updateLeaderEpochAndSendRequest Internal Method

```
updateLeaderEpochAndSendRequest(
    partition: TopicPartition,
    replicasToReceiveRequest: Seq[Int],
    newAssignedReplicas: Seq[Int]): Unit
```

updateLeaderEpochAndSendRequest ...FIXME

Note

`updateLeaderEpochAndSendRequest` is used when `KafkaController` is requested to `onPartitionReassignment` and `moveReassignedPartitionLeaderIfRequired`.

Shutting Down— `shutdown` Method

```
shutdown(): Unit
```

`shutdown` requests the `ControllerEventManager` to `close` followed by `onControllerResignation`.

Note `shutdown` is used exclusively when `KafkaServer` is requested to `shutdown`.

updateMetrics Internal Method

```
updateMetrics(): Unit
```

`updateMetrics` ...FIXME

Note `updateMetrics` is used exclusively when `KafkaController` is `created` (and creates the `ControllerEventManager`).

onBrokerStartup Method

```
onBrokerStartup(newBrokers: Seq[Int]): Unit
```

`onBrokerStartup` prints out the following INFO message to the logs:

```
New broker startup callback for [newBrokers]
```

`onBrokerStartup` requests the `ControllerContext` for the `replicasOnOfflineDirs` and removes the given broker IDs (in `newBrokers`).

`onBrokerStartup` `sendUpdateMetadataRequest` to the `liveOrShuttingDownBrokerIds` (of the `ControllerContext`).

`onBrokerStartup` requests the `ControllerContext` for the `replicas` on the given `newBrokers` .

`onBrokerStartup` requests the `ReplicaStateMachine` to `handleStateChanges` for the replicas on the new brokers and `OnlineReplica` target state.

`onBrokerStartup` requests the `PartitionStateMachine` to `triggerOnlinePartitionStateChange`.

`onBrokerStartup` requests the [ControllerContext](#) for the `partitionsBeingReassigned` and collects the partitions that have replicas on the new brokers. For every partition with a replica on the new brokers, `onBrokerStartup` [onPartitionReassignment](#).

`onBrokerStartup` collects replicas (on the new brokers) that are scheduled to be deleted by requesting the [TopicDeletionManager](#) to [see whether isTopicQueuedUpForDeletion](#). If there are any, `onBrokerStartup` prints out the following INFO message to the logs and requests the [TopicDeletionManager](#) to [resumeDeletionForTopics](#).

```
Some replicas [replicasForTopicsToDelete] for topics scheduled for deletion [topicsToDelete] are on the newly restarted brokers [newBrokers]. Signaling restart of topic deletion for these topics
```

In the end, `onBrokerStartup` [registerBrokerModificationsHandler](#) for the new brokers.

Note

`onBrokerStartup` is used exclusively when `KafkaController` is requested to process a [BrokerChange](#) controller event.

Controller Election — `elect` Method

```
elect(): Unit
```

`elect` requests the [KafkaZkClient](#) for the [active controller ID](#).

`elect` stops the controller election if there is an active controller ID available and prints out the following DEBUG message to the logs:

```
Broker [activeControllerId] has been elected as the controller, so stopping the election process.
```

Otherwise, `elect` requests the [KafkaZkClient](#) to [create an ephemeral znode](#) at `/controller` path with the znode data in JSON:

```
{"version":1,"brokerid":[brokerId],"timestamp":[timestamp]}
```

Note

`elect` always uses `1` for the version.

Note

`elect` is used when `ControllerEventThread` is requested to process [Startup](#) and [Reelect](#) controller events (while [processing controller events](#)).

Controller Elected

If successful, `elect` prints out the following INFO message to the logs and records the current broker ID as the [activeControllerId](#).

```
[brokerId] successfully elected as the controller
```

In the end, `elect` does [onControllerFailover](#).

Controller Has Already Been Elected (NodeExistsException)

If unsuccessful (and a `NodeExistsException` was reported), `elect` requests the [KafkaZkClient](#) for the [active controller ID](#).

`elect` then prints out the following DEBUG message to the logs:

```
Broker [activeControllerId] was elected as controller instead of broker [brokerId]
```

If however the active controller ID is still unavailable, `elect` prints out the following WARN message to the logs:

```
A controller has been elected but just resigned, this will result in another round of election
```

Other Errors (Throwable)

If unsuccessful (and a `Throwable` was reported), `elect` prints out the following ERROR message to the logs and does [triggerControllerMove](#):

```
Error while electing or becoming controller on broker [brokerId]
```

Is KafkaController The Active Controller? — `isActive` Method

```
isActive: Boolean
```

`isActive` flag says whether the current broker (by the ID) is the active controller (given the [activeControllerId](#)).

Note

`isActive` is on (`true`) after the `KafkaController` of a Kafka broker has been [elected](#).

	<p><code>isActive</code> is used (as a valve to stop processing early) when:</p> <ul style="list-style-type: none"> • <code>ControllerEventThread</code> is requested to process controller events (that should only be processed on the active controller, e.g. <code>AutoPreferredReplicaLeaderElection</code>, <code>UncleanLeaderElectionEnable</code>, <code>ControlledShutdown</code>, <code>LeaderAndIsrResponseReceived</code>, <code>TopicDeletionStopReplicaResponseReceived</code>, <code>BrokerChange</code>, <code>BrokerModifications</code>, <code>TopicChange</code>) • <code>KafkaController</code> is requested to updateMetrics • <code>KafkaApis</code> is requested to handleCreateTopicsRequest, handleCreatePartitionsRequest and handleDeleteTopicsRequest
Note	

registerIsrChangeNotificationListener Internal Method

```
registerIsrChangeNotificationListener(): Option[Seq[String]]
```

```
registerIsrChangeNotificationListener ...FIXME
```

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

deregisterIsrChangeNotificationListener Internal Method

```
deregisterIsrChangeNotificationListener(): Unit
```

```
deregisterIsrChangeNotificationListener ...FIXME
```

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

Creating KafkaController Instance

`KafkaController` takes the following when created:

- [KafkaConfig](#)
- [KafkaZkClient](#)
- `Time`
- [Metrics](#)
- `BrokerInfo`

- `DelegationTokenManager`
- Thread name prefix (default: undefined)

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

Starting Up — `startup` Method

```
startup(): Unit
```

`startup` requests the [KafkaZkClient](#) to [register a StateChangeHandler](#) (under the name **controller-state-change-handler**) that is does the following:

- On `afterInitializingSession`, the `StateChangeHandler` simply puts `RegisterBrokerAndReelect` event on the event queue of the [ControllerEventManager](#)
- On `beforeInitializingSession`, the `StateChangeHandler` simply puts `Expire` event on the event queue of the [ControllerEventManager](#)

`startup` then puts `Startup` event at the end of the event queue of the [ControllerEventManager](#) and immediately requests it to [start](#).

Note	<code>startup</code> is used exclusively when <code>KafkaServer</code> is requested to start .
------	--

Registering SessionExpirationListener To Control Session Recreation — `registerSessionExpirationListener` Internal Method

```
registerSessionExpirationListener(): Unit
```

`registerSessionExpirationListener` requests [ZkUtils](#) to [subscribe to state changes](#) with a `SessionExpirationListener` (with the `KafkaController` and [ControllerEventManager](#)).

Note	<code>SessionExpirationListener</code> puts <code>Reelect</code> event on the event queue of <code>ControllerEventManager</code> every time the Zookeeper session has expired and a new session has been created.
------	---

Note	<code>registerSessionExpirationListener</code> is used exclusively when <code>Startup</code> event is processed (after <code>ControllerEventThread</code> is started).
------	---

Registering ControllerChangeListener for /controller ZNode Changes — registerControllerChangeListener Internal Method

```
registerControllerChangeListener(): Unit
```

`registerControllerChangeListener` requests [ZkUtils](#) to [subscribe to data changes](#) for `/controller` znode with a `ControllerChangeListener` (with the `KafkaController` and `ControllerEventManager`).

- | | |
|------|---|
| Note | <code>ControllerChangeListener</code> emits: <ol style="list-style-type: none"> 1. ControllerChange event with the current controller ID (on the event queue of <code>ControllerEventManager</code>) every time the data of a znode changes 2. Reelect event when the data associated with a znode has been deleted |
|------|---|

- | | |
|------|---|
| Note | <code>registerControllerChangeListener</code> is used exclusively when Startup event is processed (after <code>ControllerEventThread</code> is started). |
|------|---|

registerBrokerChangeListener Internal Method

```
registerBrokerChangeListener(): Option[Seq[String]]
```

`registerBrokerChangeListener` requests [ZkUtils](#) to [subscribeChildChanges](#) for `/brokers/ids` path with [BrokerChangeListener](#).

- | | |
|------|---|
| Note | <code>registerBrokerChangeListener</code> is used exclusively when <code>KafkaController</code> does onControllerFailover . |
|------|---|

Getting Active Controller ID (from JSON under /controller znode) — getControllerID Method

```
getControllerID(): Int
```

`getControllerID` returns the ID of the active Kafka controller that is associated with `/controller` znode in JSON format or `-1` otherwise.

Internally, `getControllerID` requests [ZkUtils](#) for [data associated with](#) `/controller` znode.

If available, `getControllerID` parses the data (being the current controller info in JSON format) to extract `brokerid` field.

```
$ ./bin/zookeeper-shell.sh :2181 get /controller

{"version":1,"brokerid":0,"timestamp":"1543499076007"}
cZxid = 0x60
ctime = Thu Nov 29 14:44:36 CET 2018
mZxid = 0x60
mtime = Thu Nov 29 14:44:36 CET 2018
pZxid = 0x60
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x100073f07ba0003
dataLength = 54
numChildren = 0
```

Otherwise, when no `/controller` znode is available, `getControllerID` returns `-1`.

Note

`getControllerID` is used when:

1. Processing `Reelect controller` event
2. `elect`

Registering TopicDeletionListener for Child Changes to `/admin/delete topics` ZNode

— `registerTopicDeletionListener` Internal Method

```
registerTopicDeletionListener(): Option[Seq[String]]
```

`registerTopicDeletionListener` requests [ZkUtils](#) to [subscribeChildChanges](#) to `/admin/delete_topics` znode with [TopicDeletionListener](#).

Note

`registerTopicDeletionListener` is used exclusively when `kafkaController` does [onControllerFailover](#).

De-Registering TopicDeletionListener for Child Changes to `/admin/delete topics` ZNode

— `deregisterTopicDeletionListener` Internal Method

```
deregisterTopicDeletionListener(): Unit
```

`deregisterTopicDeletionListener` requests [ZkUtils](#) to [unsubscribeChildChanges](#) to `/admin/delete_topics` znode with [TopicDeletionListener](#).

Note

`deregisterTopicDeletionListener` is used exclusively when `KafkaController` resigns as the active controller.

onReplicasBecomeOffline Internal Method

```
onReplicasBecomeOffline(newOfflineReplicas: Set[PartitionAndReplica]): Unit
```

`onReplicasBecomeOffline` ...FIXME

Note

`onReplicasBecomeOffline` is used when...FIXME

onPartitionReassignment Internal Method

```
onPartitionReassignment(
    topicPartition: TopicPartition,
    reassignedPartitionContext: ReassignedPartitionsContext): Unit
```

`onPartitionReassignment` ...FIXME

Note

`onPartitionReassignment` is used when `KafkaController` is requested to [onBrokerStartup](#), [maybeTriggerPartitionReassignment](#) and process a `PartitionReassignmentIsrChange` event.

onBrokerUpdate Internal Method

```
onBrokerUpdate(updatedBrokerId: Int): Unit
```

`onBrokerUpdate` ...FIXME

Note

`onBrokerUpdate` is used when...FIXME

updateBrokerInfo Internal Method

```
updateBrokerInfo(newBrokerInfo: BrokerInfo): Unit
```

```
updateBrokerInfo ...FIXME
```

Note

`updateBrokerInfo` is used exclusively when `DynamicListenerConfig` is requested to [reconfigure](#).

registerBrokerModificationsHandler Internal Method

```
registerBrokerModificationsHandler(brokerIds: Iterable[Int]): Unit
```

```
registerBrokerModificationsHandler ...FIXME
```

Note

`registerBrokerModificationsHandler` is used when `KafkaController` is requested to [onBrokerStartup](#) and [onControllerFailover](#) (indirectly through [initializeControllerContext](#)).

initializeControllerContext Internal Method

```
initializeControllerContext(): Unit
```

```
initializeControllerContext ...FIXME
```

Note

`initializeControllerContext` is used exclusively when `KafkaController` is requested to [onControllerFailover](#).

unregisterBrokerModificationsHandler Internal Method

```
unregisterBrokerModificationsHandler(brokerIds: Iterable[Int]): Unit
```

```
unregisterBrokerModificationsHandler ...FIXME
```

Note

`unregisterBrokerModificationsHandler` is used when `KafkaController` is requested to [onControllerResignation](#) and [onBrokerFailure](#).

onBrokerFailure Internal Method

```
onBrokerFailure(deadBrokers: Seq[Int]): Unit
```

```
onBrokerFailure ...FIXME
```

Note

`onBrokerFailure` is used exclusively when `KafkaController` is requested to handle a `BrokerChange` controller event.

maybeTriggerPartitionReassignment Internal Method

```
maybeTriggerPartitionReassignment(topicPartitions: Set[TopicPartition]): Unit
```

`maybeTriggerPartitionReassignment` ...FIXME

Note

`maybeTriggerPartitionReassignment` is used when `KafkaController` is requested to `onControllerFailover` and process the `PartitionReassignment` controller event.

incrementControllerEpoch Internal Method

```
incrementControllerEpoch(): Unit
```

`incrementControllerEpoch` ...FIXME

Note

`incrementControllerEpoch` is used exclusively when `KafkaController` is requested to `onControllerFailover`.

fetchPendingPreferredReplicaElections Internal Method

```
fetchPendingPreferredReplicaElections(): Set[TopicPartition]
```

`fetchPendingPreferredReplicaElections` ...FIXME

Note

`fetchPendingPreferredReplicaElections` is used exclusively when `KafkaController` is requested to `onControllerFailover`.

initializePartitionReassignment Internal Method

```
initializePartitionReassignment(): Unit
```

`initializePartitionReassignment` ...FIXME

Note	<code>initializePartitionReassignment</code> is used exclusively when <code>KafkaController</code> is requested to initializeControllerContext .
------	--

fetchTopicDeletionsInProgress Internal Method

```
fetchTopicDeletionsInProgress(): (Set[String], Set[String])
```

`fetchTopicDeletionsInProgress` ...FIXME

Note	<code>fetchTopicDeletionsInProgress</code> is used exclusively when <code>KafkaController</code> is requested to onControllerFailover .
------	---

updateLeaderAndIsrCache Internal Method

```
updateLeaderAndIsrCache(partitions: Seq[TopicPartition])
```

Unless given, `updateLeaderAndIsrCache` defaults to [allPartitions](#) of the `ControllerContext` for the partitions.

`updateLeaderAndIsrCache` requests the `KafkaZkClient` to [getTopicPartitionStates](#) for the partitions.

For every pair of a `TopicPartition` and the `LeaderIsrAndControllerEpoch`, `updateLeaderAndIsrCache` adds them to the [partitionLeadershipInfo](#) of the `ControllerContext`.

Note	<code>updateLeaderAndIsrCache</code> is used when <code>KafkaController</code> is requested to initializeControllerContext and process a IsrChangeNotification controller event.
------	--

areReplicasInIsr Internal Method

```
areReplicasInIsr(partition: TopicPartition, replicas: Seq[Int]): Boolean
```

`areReplicasInIsr` ...FIXME

Note	<code>areReplicasInIsr</code> is used exclusively when <code>KafkaController</code> is requested to onPartitionReassignment .
------	---

updateAssignedReplicasForPartition Internal Method

```
updateAssignedReplicasForPartition(
    partition: TopicPartition,
    replicas: Seq[Int]): Unit
```

```
updateAssignedReplicasForPartition ...FIXME
```

Note	<code>updateAssignedReplicasForPartition</code> is used exclusively when <code>KafkaController</code> is requested to onPartitionReassignment .
------	---

registerPartitionModificationsHandlers Internal Method

```
registerPartitionModificationsHandlers(topics: Seq[String]): Unit
```

```
registerPartitionModificationsHandlers ...FIXME
```

Note	<code>registerPartitionModificationsHandlers</code> is used when <code>KafkaController</code> is requested to initializeControllerContext and a <code>TopicChange</code> controller event is processed.
------	---

unregisterPartitionModificationsHandlers Internal Method

```
unregisterPartitionModificationsHandlers(topics: Seq[String]): Unit
```

```
unregisterPartitionModificationsHandlers ...FIXME
```

Note	<code>unregisterPartitionModificationsHandlers</code> is used when: <ul style="list-style-type: none"> • <code>KafkaController</code> is requested to onControllerResignation • <code>TopicDeletionManager</code> is requested to completeDeleteTopic
------	---

unregisterPartitionReassignmentIsrChangeHandlers Internal Method

```
unregisterPartitionReassignmentIsrChangeHandlers(): Unit
```

```
unregisterPartitionReassignmentIsrChangeHandlers ...FIXME
```

Note

`unregisterPartitionReassignmentIsrChangeHandlers` is used exclusively when `KafkaController` is requested to [onControllerResignation](#).

readControllerEpochFromZooKeeper Internal Method

```
readControllerEpochFromZooKeeper(): Unit
```

```
readControllerEpochFromZooKeeper ...FIXME
```

Note

`readControllerEpochFromZooKeeper` is used exclusively when `KafkaController` is requested to [onControllerFailover](#).

removePartitionsFromReassignedPartitions Internal Method

```
removePartitionsFromReassignedPartitions(partitionsToBeRemoved: Set[TopicPartition]): Unit
```

```
removePartitionsFromReassignedPartitions ...FIXME
```

Note

`removePartitionsFromReassignedPartitions` is used when `kafkaController` is requested to [onPartitionReassignment](#) and [maybeTriggerPartitionReassignment](#).

removePartitionsFromPreferredReplicaElection Internal Method

```
removePartitionsFromPreferredReplicaElection(
  partitionsToBeRemoved: Set[TopicPartition],
  isTriggeredByAutoRebalance : Boolean): Unit
```

```
removePartitionsFromPreferredReplicaElection ...FIXME
```

Note

`removePartitionsFromPreferredReplicaElection` is used exclusively when `KafkaController` is requested to [onPreferredReplicaElection](#).

onPreferredReplicaElection Internal Method

```
onPreferredReplicaElection(
    partitions: Set[TopicPartition],
    isTriggeredByAutoRebalance: Boolean = false): Unit
```

onPreferredReplicaElection ...FIXME

Note

`onPreferredReplicaElection` is used when `KafkaController` is requested to [onControllerFailover](#), [checkAndTriggerAutoLeaderRebalance](#) and process a [PreferredReplicaLeaderElection](#) controller event.

updateLeaderEpoch Internal Method

```
updateLeaderEpoch(partition: TopicPartition): Option[LeaderIsrAndControllerEpoch]
```

updateLeaderEpoch ...FIXME

Note

`updateLeaderEpoch` is used exclusively when `KafkaController` is requested to [updateLeaderEpochAndSendRequest](#).

moveReassignedPartitionLeaderIfRequired Internal Method

```
moveReassignedPartitionLeaderIfRequired(
    topicPartition: TopicPartition,
    reassignedPartitionContext: ReassignedPartitionsContext): Unit
```

moveReassignedPartitionLeaderIfRequired ...FIXME

Note

`moveReassignedPartitionLeaderIfRequired` is used exclusively when `KafkaController` is requested to [onPartitionReassignment](#).

onControllerFailover Internal Method

```
onControllerFailover(): Unit
```

`onControllerFailover` prints out the following INFO message to the logs:

```
Registering handlers
```

`onControllerFailover` requests the `KafkaZkClient` to [registerZNodeChildChangeHandlers](#):

- brokerChangeHandler
- topicChangeHandler
- topicDeletionHandler
- logDirEventNotificationHandler
- isrChangeNotificationHandler

`onControllerFailover` requests the [KafkaZkClient](#) to registerZNodeChangeHandlerAndCheckExistence:

- preferredReplicaElectionHandler
- partitionReassignmentHandler

`onControllerFailover` prints out the following INFO message to the logs:

```
Deleting log dir event notifications
```

`onControllerFailover` requests the [KafkaZkClient](#) to deleteLogDirEventNotifications (with the epochZkVersion of the [ControllerContext](#)).

`onControllerFailover` prints out the following INFO message to the logs:

```
Deleting isr change notifications
```

`onControllerFailover` requests the [KafkaZkClient](#) to deletelsrChangeNotifications (with the epochZkVersion of the [ControllerContext](#)).

`onControllerFailover` prints out the following INFO message to the logs:

```
Initializing controller context
```

`onControllerFailover` initializeControllerContext.

`onControllerFailover` prints out the following INFO message to the logs:

```
Fetching topic deletions in progress
```

`onControllerFailover` fetchTopicDeletionsInProgress.

`onControllerFailover` prints out the following INFO message to the logs:

```
Initializing topic deletion manager
```

`onControllerFailover` requests the [TopicDeletionManager](#) to initialize (with the topics to be deleted and ineligible for deletion).

`onControllerFailover` prints out the following INFO message to the logs:

```
Sending update metadata request
```

`onControllerFailover` [sendUpdateMetadataRequest](#) (with the `liveOrShuttingDownBrokerIds` of the [ControllerContext](#)).

`onControllerFailover` requests the [ReplicaStateMachine](#) to start up.

`onControllerFailover` requests the [PartitionStateMachine](#) to start up.

`onControllerFailover` prints out the following INFO message to the logs:

```
Ready to serve as the new controller with epoch [epoch]
```

`onControllerFailover` [maybeTriggerPartitionReassignment](#) (with the `partitionsBeingReassigned` of the [ControllerContext](#)).

`onControllerFailover` requests the [TopicDeletionManager](#) to [tryTopicDeletion](#).

`onControllerFailover` [onPreferredReplicaElection](#) with the [fetchPendingPreferredReplicaElections](#).

`onControllerFailover` prints out the following INFO message to the logs:

```
Starting the controller scheduler
```

`onControllerFailover` requests the [kafkaScheduler KafkaScheduler](#) to startup.

With `auto.leader.rebalance.enable` enabled, `onControllerFailover` [scheduleAutoLeaderRebalanceTask](#) with the delay of 5 seconds.

With `delegation.token.master.key` password set, `onControllerFailover` prints out the following INFO message to the logs:

```
starting the token expiry check scheduler
```

`onControllerFailover` requests the `tokenCleanScheduler KafkaScheduler` to `startup` and requests it to `schedule` the **delete-expired-tokens** task (FIXME).

Note

`onControllerFailover` is used exclusively when `KafkaController` is requested to `elect` (and a broker is successfully elected as the controller).

scheduleAutoLeaderRebalanceTask Internal Method

```
scheduleAutoLeaderRebalanceTask(delay: Long, unit: TimeUnit): Unit
```

`scheduleAutoLeaderRebalanceTask` simply requests the `KafkaScheduler` to `schedule` a one-off task called **auto-leader-rebalance-task** with initial delay of 5 seconds.

The `auto-leader-rebalance-task` simply requests the `ControllerEventManager` to `enqueue` a `AutoPreferredReplicaLeaderElection` event.

Note

`scheduleAutoLeaderRebalanceTask` is used when:

- `KafkaController` is requested to `onControllerFailover`
- `ControllerEventThread` is requested to process a `AutoPreferredReplicaLeaderElection` event (while `processing controller events`).

checkAndTriggerAutoLeaderRebalance Internal Method

```
checkAndTriggerAutoLeaderRebalance(): Unit
```

`checkAndTriggerAutoLeaderRebalance` prints out the following TRACE message to the logs:

```
Checking need to trigger auto leader balancing
```

Note

`checkAndTriggerAutoLeaderRebalance` is used exclusively when `ControllerEventThread` is requested to process a `AutoPreferredReplicaLeaderElection` event (while `processing controller events`).

startChannelManager Internal Method

```
startChannelManager(): Unit
```

```
startChannelManager ...FIXME
```

Note	<code>startChannelManager</code> is used exclusively when <code>KafkaController</code> is requested to initializeControllerContext .
------	--

onNewPartitionCreation Internal Method

```
onNewPartitionCreation(newPartitions: Set[TopicPartition]): Unit
```

```
onNewPartitionCreation ...FIXME
```

Note	<code>onNewPartitionCreation</code> is used when TopicChange and PartitionModifications controller events are processed.
------	--

onBrokerLogDirFailure Internal Method

```
onBrokerLogDirFailure(brokerIds: Seq[Int]): Unit
```

```
onBrokerLogDirFailure ...FIXME
```

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

stopOldReplicasOfReassignedPartition Internal Method

```
stopOldReplicasOfReassignedPartition(  
    topicPartition: TopicPartition,  
    reassignedPartitionContext: ReassignedPartitionsContext,  
    oldReplicas: Set[Int]): Unit
```

```
stopOldReplicasOfReassignedPartition ...FIXME
```

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

startNewReplicasForReassignedPartition Internal Method

```
startNewReplicasForReassignedPartition(  
    topicPartition: TopicPartition,  
    reassignedPartitionContext: ReassignedPartitionsContext,  
    newReplicas: Set[Int]): Unit
```

startNewReplicasForReassignedPartition ...FIXME

Note

startNewReplicasForReassignedPartition is used when...FIXME

enableDefaultUncleanLeaderElection Method

```
enableDefaultUncleanLeaderElection(): Unit
```

enableDefaultUncleanLeaderElection ...FIXME

Note

enableDefaultUncleanLeaderElection is used exclusively when
DynamicLogConfig is requested to [reconfigure](#).

KafkaHealthcheck

`KafkaHealthcheck` registers the broker it runs on with Zookeeper (which in turn makes the broker visible to other brokers that together can form a Kafka cluster).

`KafkaHealthcheck` is created and started when `KafkaServer` is requested to start up.

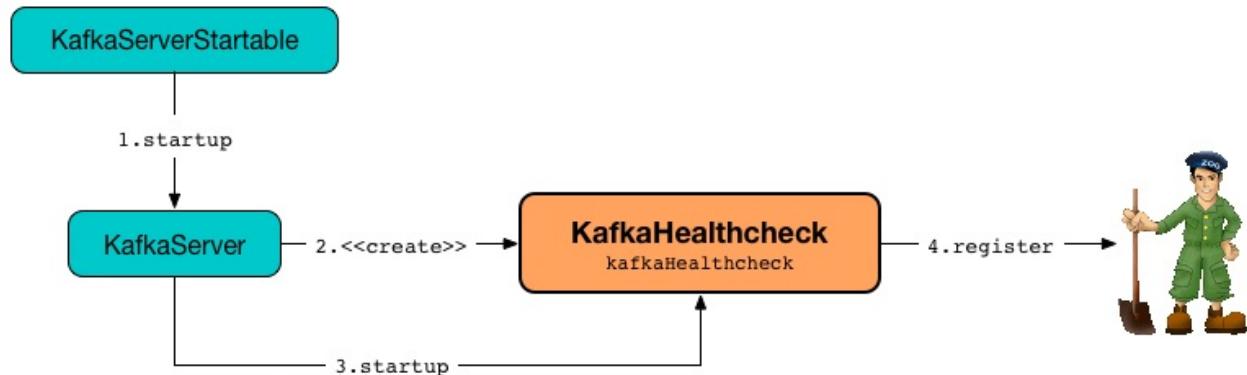


Figure 1. KafkaHealthcheck

Table 1. KafkaHealthcheck's Internal Properties (e.g. Registries and Counters)

Name	Description
sessionExpireListener	SessionExpireListener

Creating KafkaHealthcheck Instance

`KafkaHealthcheck` takes the following when created:

- Broker ID
- Advertised endpoints
- `ZkUtils`
- Optional rack name
- `ApiVersion`

`KafkaHealthcheck` initializes the internal registries and counters.

Starting Up — `startup` Method

`startup`

`startup` requests `ZkUtils` to subscribeStateChanges with `sessionExpireListener`.

In the end, `startup` registers the broker with Zookeeper.

Note	<code>startup</code> is used exclusively when <code>KafkaServer</code> starts up.
------	---

Registering Broker in Zookeeper— `register` Method

<code>register(): Unit</code>

`register` reads `com.sun.management.jmxremote.port` System property or defaults to `-1`.

For every `EndPoint` with no host assigned (in `advertisedEndpoints`), `register` assigns the fully-qualified domain name of the local host.

`register` then finds the first `EndPoint` with `PLAINTEXT` security protocol or creates an empty `EndPoint`.

Tip	Define <code>EndPoint</code> with <code>PLAINTEXT</code> security protocol for older clients to connect.
-----	--

In the end, `register` requests `ZkUtils` to `registerBrokerInZk` for `brokerId`, the host and port of the `PLAINTEXT` endpoint, the updated endpoints, the JMX port, the optional `rack` and `protocol version`.

Note	<code>register</code> makes a broker visible for other brokers to form a Kafka cluster.
------	---

Note	<code>register</code> is used when <code>KafkaHealthcheck</code> starts up and handles a new session.
------	---

handleNewSession Method

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

KafkaServerStartable — Thin Management Layer over KafkaServer

`KafkaServerStartable` is a thin management layer to manage a single `KafkaServer` instance.

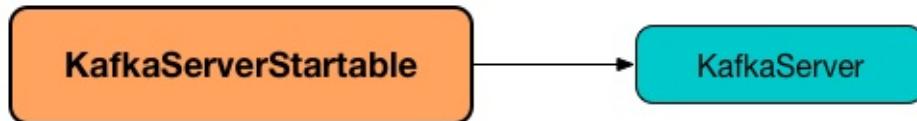


Figure 1. KafkaServerStartable manages KafkaServer

`KafkaServerStartable` allows for the `KafkaServer` instance to be `started`, `shut down` and `waited for until shutdown`.

Table 1. KafkaServerStartable's Internal Properties (e.g. Registries and Counters)

Name	Description
<code>server</code>	<code>KafkaServer</code> instance. Created when <code>KafkaServerStartable</code> is <code>created</code> .

awaitShutdown Method

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

shutdown Method

shutdown(): Unit

Note	shutdown is used exclusively when Kafka standalone application is requested to "shut down" (as part of the <code>kafka-shutdown-hook</code> shutdown hook).
------	---

Creating KafkaServerStartable Instance

`KafkaServerStartable` takes the following when created:

1. `KafkaConfig`
2. Collection of `KafkaMetricsReporters`

`KafkaServerStartable` initializes the internal registries and counters.

Creating KafkaServerStartable From Java Properties

— `fromProps` Method

```
fromProps(serverProps: Properties): KafkaServerStartable
```

`fromProps` creates a `KafkaServerStartable` with a custom `serverProps` properties file.

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

Note	<code>fromProps</code> is used when <code>kafka.Kafka</code> runs as a standalone command-line application
------	--

startup Method

```
startup(): Unit
```

`startup` requests the managed [KafkaServer](#) to [start](#).

In case of any exceptions, `startup` exits the JVM with status `1`. You should see the following FATAL message in the logs if that happens.

```
FATAL Exiting Kafka.
```

Note	<code>startup</code> uses Java's System.exit to terminate a JVM.
------	--

Note	<code>startup</code> is used when <code>kafka.Kafka</code> command-line application (<code>kafka-server-start</code>) is executed .
------	---

KafkaConfig

`KafkaConfig` is the configuration of a Kafka broker and the services.

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

- `DynamicBrokerConfig` is requested to [initialize](#) and [processReconfiguration](#)
- `KafkaConfig` is requested to [fromProps](#) and [apply](#)

Table 1. KafkaConfig's Configuration Values

Value	Kafka Property
advertisedListeners	<p><code>advertised.listeners</code></p> <p>Used when:</p> <ul style="list-style-type: none"> • <code>DynamicListenerConfig</code> is requested to validate reconfiguration • <code>KafkaConfig</code> is requested to validateValues • <code>KafkaServer</code> is requested to createBrokerInfo (when started)
autoCreateTopicsEnable	<p><code>auto.create.topics.enable</code></p> <p>Used when <code>kafkaApis</code> is requested to getTopicMetadata and handleTopicMetadataRequest</p>
autoLeaderRebalanceEnable	<p><code>auto.leader.rebalance.enable</code></p> <p>Used exclusively when <code>KafkaController</code> is requested to onControllerFailover</p>
backgroundThreads	<code>background.threads</code>
brokerId	<code>broker.id</code>
brokerIdGenerationEnable	<p><code>broker.id.generation.enable</code></p> <p>Used exclusively when <code>KafkaServer</code> is requested to getBrokerIdAndOfflineDirs</p>
	<p><code>connections.max.idle.ms</code></p> <p>Used when:</p>

<code>connectionsMaxIdleMs</code>	<ul style="list-style-type: none"> • <code>TransactionMarkerChannelManager</code> is created • <code>SocketServer</code> is requested to create a new network processor thread • <code>ReplicaFetcherBlockingSend</code> is created
<code>deleteTopicEnable</code>	<code>delete.topic.enable</code>
<code>failedAuthenticationDelayMs</code>	<p><code>connection.failed.authentication.delay.ms</code></p> <p>Used exclusively when <code>SocketServer</code> is requested to create a new network processor thread</p>
<code>hostName</code>	<code>host.name</code>
<code>interBrokerListenerName</code>	
<code>interBrokerProtocolVersion</code>	<code>inter.broker.protocol.version</code>
<code>interBrokerSecurityProtocol</code>	
<code>leaderImbalanceCheckIntervalSeconds</code>	<p><code>leader.imbalance.check.interval.seconds</code></p> <p>Used exclusively when <code>ControllerEventThread</code> is requested to process a <code>AutoPreferredReplicaLeaderElection</code> event (while processing controller events)</p>
<code>listeners</code>	<code>listeners</code> (see getListeners)
<code>listenerSecurityProtocolMap</code>	<code>listener.security.protocol.map</code>
<code>logCleanerThreads</code>	<code>log.cleaner.threads</code>
<code>logCleanerDedupeBufferSize</code>	<code>log.cleaner.dedupe.buffer.size</code>
<code>logCleanerDedupeBufferLoadFactor</code>	<code>log.cleaner.io.buffer.load.factor</code>
<code>logCleanerIoBufferSize</code>	<code>log.cleaner.io.buffer.size</code>
<code>logCleanerIoMaxBytesPerSecond</code>	<code>log.cleaner.io.max.bytes.per.second</code>
<code>logCleanerBackoffMS</code>	<code>log.cleaner.backoff.ms</code>

<code>logCleanerEnable</code>	<code>log.cleaner.enable</code>
<code>logDirs</code>	<p><code>log.dirs</code> or <code>log.dir</code></p> <p>At least one log directory must be defined via <code>log.dirs</code> or <code>log.dir</code> properties.</p> <p>Used when:</p> <ul style="list-style-type: none"> • <code>LogManager</code> is created • <code>LogDirFailureChannel</code> is created (when <code>KafkaServer</code> is requested to start up) • <code>KafkaConfig</code> is requested to <code>getNumReplicaAlterLogDirsThreads</code> • <code>KafkaServer</code> is created and is requested to <code>getBrokerIdAndOfflineDirs</code> and <code>checkpointBrokerId</code> • <code>ReplicaManager</code> is requested to <code>describeLogDirs</code>
<code>maxConnectionsPerIp</code>	<code>max.connections.per.ip</code> Used exclusively when <code>SocketServer</code> is requested to start up
<code>maxConnectionsPerIpOverrides</code>	<code>max.connections.per.ip.overrides</code> Used exclusively when <code>SocketServer</code> is requested to start up
<code>maxReservedBrokerId</code>	<code>reserved.broker.max.id</code> Used exclusively when <code>KafkaServer</code> is requested to generateBrokerId
<code>messageMaxBytes</code>	<code>message.max.bytes</code>
<code>numIoThreads</code>	<code>num.io.threads</code>
<code>numNetworkThreads</code>	<code>num.network.threads</code>
<code>numRecoveryThreadsPerDataDir</code>	<code>num.recovery.threads.per.data.dir</code> Used exclusively when <code>LogManager</code> is created
<code>numReplicaFetchers</code>	<code>num.replica.fetchers</code>

port	port
PrincipalBuilderClassProp	principal.builder.class
replicaLagTimeMaxMs	replica.lag.time.max.ms
replicaFetchBackoffMs	replica.fetch.backoff.ms Used as fetchBackOffMs for ReplicaAlterLogDirsThread and ReplicaFetcherThread
SaslMechanismInterBrokerProtocolProp	sasl.mechanism.inter.broker.protocol
SaslJaasConfigProp	sasl.jaas.config
SaslEnabledMechanismsProp	sasl.enabled.mechanisms
SaslKerberosServiceNameProp	sasl.kerberos.service.name
SaslKerberosKinitCmdProp	sasl.kerberos.kinit.cmd
SaslKerberosTicketRenewWindowFactorProp	sasl.kerberos.ticket.renew.window.factor
SaslKerberosTicketRenewJitterProp	sasl.kerberos.ticket.renew.jitter
SaslKerberosMinTimeBeforeReloginProp	sasl.kerberos.min.time.before.relogin
SaslKerberosPrincipalToLocalRulesProp	sasl.kerberos.principal.to.local.rules
SaslLoginRefreshWindowFactorProp	sasl.login.refresh.window.factor
SaslLoginRefreshWindowJitterProp	sasl.login.refresh.window.jitter
SaslLoginRefreshMinPeriodSecondsProp	sasl.login.refresh.min.period.seconds
SaslLoginRefreshBufferSecondsProp	sasl.login.refresh.buffer.seconds
SslProtocolProp	ssl.protocol
SslProviderProp	ssl.provider
SslCipherSuitesProp	ssl.cipher.suites
SslEnabledProtocolsProp	ssl.enabled.protocols

SslKeystoreTypeProp	ssl.keystore.type
SslKeystoreLocationProp	ssl.keystore.location
SslKeystorePasswordProp	ssl.keystore.password
SslKeyPasswordProp	ssl.key.password
SslTruststoreTypeProp	ssl.truststore.type
SslTruststoreLocationProp	ssl.truststore.location
SslTruststorePasswordProp	ssl.truststore.password
SslKeyManagerAlgorithmProp	ssl.keymanager.algorithm
SslTrustManagerAlgorithmProp	ssl.trustmanager.algorithm
SslEndpointIdentificationAlgorithmProp	ssl.endpoint.identification.algorithm
SslSecureRandomImplementationProp	ssl.secure.random.implementation
SslClientAuthProp	ssl.client.auth
socketRequestMaxBytes	<p>socket.request.max.bytes</p> <p>Used when <code>socketServer</code> is created and requested to create a new network processor thread</p>
tokenAuthEnabled	delegation.token.master.key
transactionMaxTimeoutMs	<p>transaction.max.timeout.ms</p> <p>Used exclusively when <code>TransactionCoordinator</code> is created</p>
queuedMaxRequests	<p>queued.max.requests</p> <p>Used exclusively when <code>SocketServer</code> is created</p>
zkConnect	<p>zookeeper.connect</p> <p>Used when:</p> <ul style="list-style-type: none"> • <code>KafkaServer</code> is requested to <code>initZkClient</code>

	<ul style="list-style-type: none"> • <code>SimpleAclAuthorizer</code> is requested to configure
<code>zkConnectionTimeoutMs</code>	<p><code>zookeeper.connection.timeout.ms</code> when set or <code>zookeeper.session.timeout.ms</code></p> <p>Used when:</p> <ul style="list-style-type: none"> • <code>KafkaServer</code> is requested to <code>initZkClient</code> • <code>SimpleAclAuthorizer</code> is requested to configure
<code>zkEnableSecureAcls</code>	<p><code>zookeeper.set.acl</code></p> <p>Used when:</p> <ul style="list-style-type: none"> • <code>KafkaServer</code> is requested to <code>initZkClient</code> • <code>SimpleAclAuthorizer</code> is requested to configure
<code>zkMaxInFlightRequests</code>	<p><code>zookeeper.max.in.flight.requests</code></p> <p>Used when:</p> <ul style="list-style-type: none"> • <code>KafkaServer</code> is requested to <code>initZkClient</code> • <code>SimpleAclAuthorizer</code> is requested to configure
<code>zkSessionTimeoutMs</code>	<p><code>zookeeper.session.timeout.ms</code></p> <p>Used when:</p> <ul style="list-style-type: none"> • <code>KafkaServer</code> is requested to <code>initZkClient</code> • <code>SimpleAclAuthorizer</code> is requested to configure

Table 2. KafkaConfig's Internal Properties (e.g. Registries, Counters and Flags)

Name	Description
dynamicConfig	<p><code>DynamicBrokerConfig</code> (that could be <code>provided</code> or will be <code>created</code> from scratch)</p> <p>Used when:</p> <ul style="list-style-type: none"> • <code>AdminManager</code> is requested to <code>describeConfigs</code> (for <code>BROKER</code> resources), <code>alterConfigs</code> and <code>configSynonyms</code> • <code>BrokerConfigHandler</code> is requested to <code>processConfigChanges</code> • <code>DynamicMetricsReporters</code> is <code>created</code> • <code>DynamicClientQuotaCallback</code> is <code>reconfigure</code> • <code>KafkaConfig</code> is requested to <code>addReconfigurable</code> • <code>KafkaServer</code> is requested to <code>start up</code>

Creating Listeners — `getListeners` Internal Method

```
getListeners: Seq[EndPoint]
```

`getListeners` creates the `EndPoints` if defined using `listeners` Kafka property or defaults to `PLAINTEXT://[hostName]:[port]` (for `hostName` and `port` Kafka properties).

Note

`getListeners` is used when `KafkaConfig` is `created` and for `getAdvertisedListeners`.

`getNumReplicaAlterLogDirsThreads` Method

```
getNumReplicaAlterLogDirsThreads: Int
```

`getNumReplicaAlterLogDirsThreads` ...FIXME

Note

`getNumReplicaAlterLogDirsThreads` is used when...FIXME

Creating KafkaConfig Instance

`KafkaConfig` takes the following when created:

- Key-value properties
- `doLog` flag

- [DynamicBrokerConfig](#)

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

Creating KafkaConfig From Java Properties — `fromProps` Object Method

```
fromProps(props: Properties): KafkaConfig (1)
fromProps(props: Properties, doLog: Boolean): KafkaConfig
fromProps(defaults: Properties, overrides: Properties): KafkaConfig (2)
fromProps(defaults: Properties, overrides: Properties, doLog: Boolean): KafkaConfig (3)
```

1. Seems to be used in tests only
2. Seems to be used in tests only
3. Seems to be used in tests only

`fromProps` ...FIXME

Note	<p><code>fromProps</code> is used when:</p> <ul style="list-style-type: none"> • <code>KafkaServerStartable</code> is requested to create a KafkaServerStartable from Java Properties • <code>SimpleAclAuthorizer</code> is requested to <code>configure</code> itself
------	--

Creating KafkaConfig — `apply` Factory Method

```
apply(props: java.util.Map[_, _]): KafkaConfig
```

`apply` simply creates a [KafkaConfig](#) with the `props` and the `doLog` flag on.

Note	<code>apply</code> seems to be used in tests only.
------	--

`addReconfigurable` Method

```
addReconfigurable(reconfigurable: Reconfigurable): Unit
```

`addReconfigurable` ...FIXME

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

validateValues Internal Method

```
validateValues(): Unit
```

validateValues ...FIXME

Note

validateValues is used when...FIXME

getInterBrokerListenerNameAndSecurityProtocol Internal Method

```
getInterBrokerListenerNameAndSecurityProtocol: (ListenerName, SecurityProtocol)
```

getInterBrokerListenerNameAndSecurityProtocol ...FIXME

Note

getInterBrokerListenerNameAndSecurityProtocol is used when KafkaConfig is requested for [interBrokerListenerName](#) and [interBrokerSecurityProtocol](#).

KafkaMetricsReporter

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

KafkaRequestHandlerPool — Pool of KafkaRequestHandler Daemon Threads

`KafkaRequestHandlerPool` is a pool of KafkaRequestHandler daemon threads.

`KafkaRequestHandlerPool` is created exclusively when `KafkaServer` is requested to start up (and creates the data-plane and control-plane request handler pools).

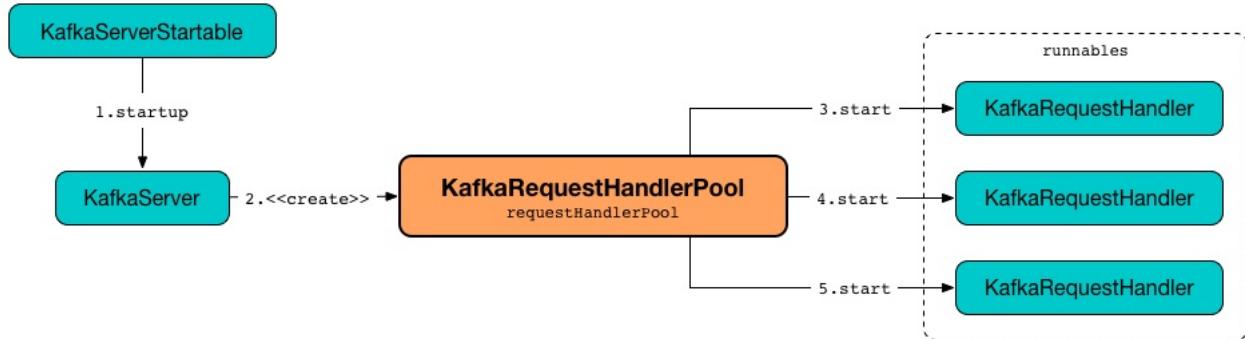


Figure 1. KafkaRequestHandlerPool and KafkaRequestHandler Threads

When created, `KafkaRequestHandlerPool` requests all KafkaRequestHandler daemon threads to start.

Note

The number of **kafka-request-handler** threads is controlled by `num.network.threads` configuration property (default: 3).

`KafkaRequestHandlerPool` uses [Kafka Request Handler on Broker [brokerId]] as the logging prefix (aka `logIdent`).

Tip

Enable ALL logging level for `kafka.server.KafkaRequestHandlerPool` logger to see what happens inside.

Add the following line to `config/log4j.properties` :

```
log4j.logger.kafka.server.KafkaRequestHandlerPool=ALL
```

Refer to [Logging](#).

Creating KafkaRequestHandlerPool Instance

`KafkaRequestHandlerPool` takes the following to be created:

- Broker ID
- RequestChannel

- [KafkaApis](#)
- [Time](#)
- Number of threads (i.e. instances of [KafkaRequestHandlers](#) as defined by `numNetworkThreads` property)
- [requestHandlerAvgIdleMetricName](#)
- [logAndThreadNamePrefix](#)

`KafkaRequestHandlerPool` initializes the [internal properties](#).

`KafkaRequestHandlerPool` starts `numThreads` daemon **kafka-request-handler** threads (as registered in the [runnables](#) internal registry).

Performance Metrics

`KafkaRequestHandlerPool` is a [KafkaMetricsGroup](#) with the following performance metrics.

Table 1. KafkaRequestHandlerPool's Performance Metrics

Metric Name	Description
requestHandlerAvgIdleMetricName	Aggregate idle meter - average free capacity of the request handlers (<i>idle handlers</i>)

The performance metrics are registered in **kafka.server:type=KafkaRequestHandlerPool** group.

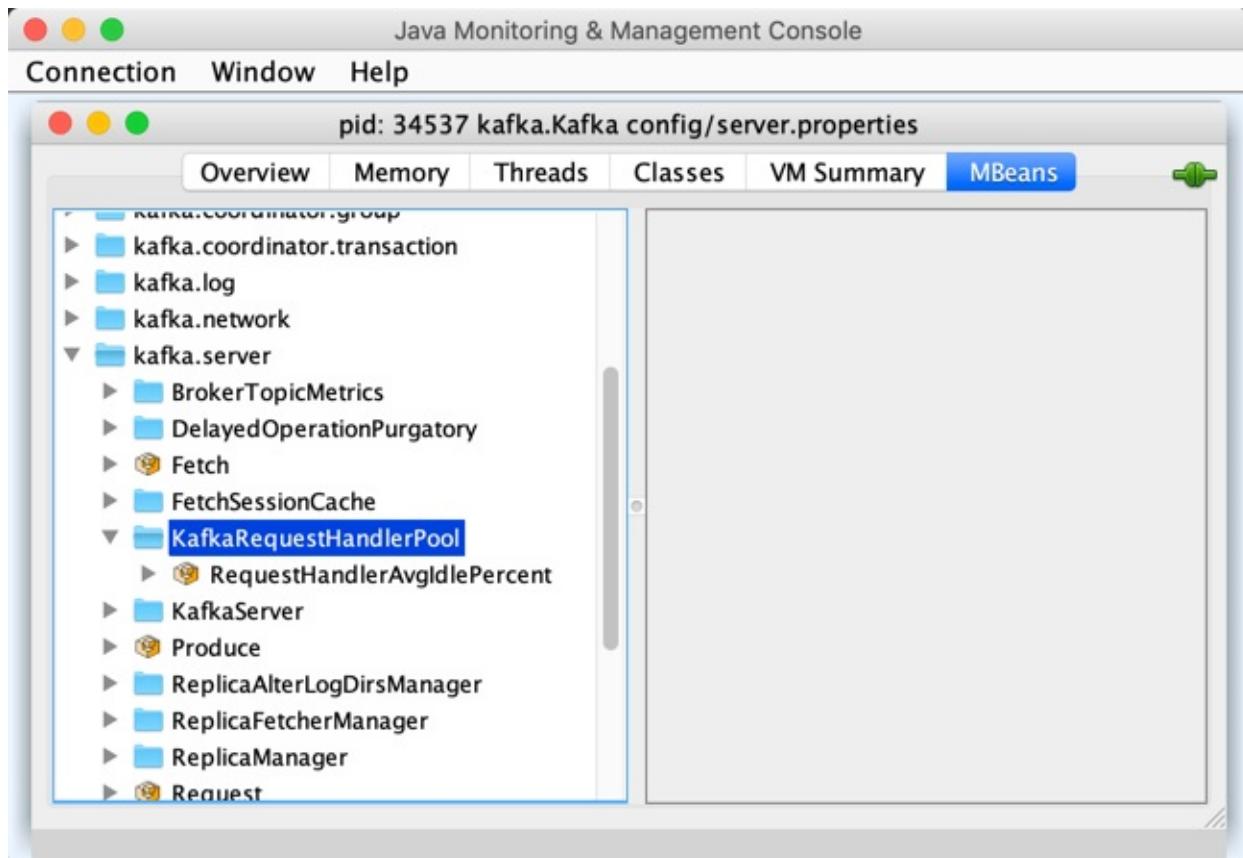


Figure 2. KafkaMetricsGroup in jconsole

Creating and Starting KafkaRequestHandler Daemon Thread — `createHandler` Method

```
createHandler(id: Int): Unit
```

`createHandler` creates a new `KafkaRequestHandler` for the given id (and the `broker ID`, the `aggregate idle meter`, the `threadPoolSize`, the `RequestChannel` and the `KafkaApis`).

`createHandler` adds the `KafkaRequestHandler` to the `runnables` internal registry.

In the end, `createHandler` starts the `KafkaRequestHandler` as a daemon thread with the name with the format:

```
[logAndThreadNamePrefix]-kafka-request-handler-[id]
```

Note	<code>createHandler</code> is used when <code>KafkaRequestHandlerPool</code> is created (to create the <code>numThreads</code> handlers) and requested to resize the thread pool (with new handlers).
------	---

Resizing Thread Pool (of KafkaRequestHandlers)

— `resizeThreadPool` Method

```
resizeThreadPool(newSize: Int): Unit
```

`resizeThreadPool` prints out the following INFO message to the logs:

```
Resizing request handler thread pool size from [currentSize] to [newSize]
```

When the given `newSize` is greater than the current `threadPoolSize`, `resizeThreadPool` creates and starts new KafkaRequestHandler daemon threads.

When the given `newSize` is smaller than the current `threadPoolSize`, `resizeThreadPool` removes `KafkaRequestHandlers` from the `runnables` internal registry and requests them to stop.

In the end, `resizeThreadPool` sets the `threadPoolSize` internal registry to the given `newSize`.

Note	<code>resizeThreadPool</code> is used exclusively when <code>DynamicThreadPool</code> is requested to reconfigure (the <code>num.io.threads</code> configuration property).
------	---

Shutting Down — `shutdown` Method

```
shutdown(): Unit
```

`shutdown` ...FIXME

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

Internal Properties

Name	Description
<code>runnables</code>	Pool of KafkaRequestHandlers daemon threads
<code>threadPoolSize</code>	

KafkaRequestHandler

`KafkaRequestHandler` is a thread of execution (`java.lang.Runnable`) that is responsible for relaying `client requests` (from a `RequestChannel`) to a `KafkaApis` (except `ShutdownRequest` requests that are handled directly).

`KafkaRequestHandler` is `created` exclusively when `KafkaRequestHandlerPool` is `created` (and starts the internal `runnables` threads).

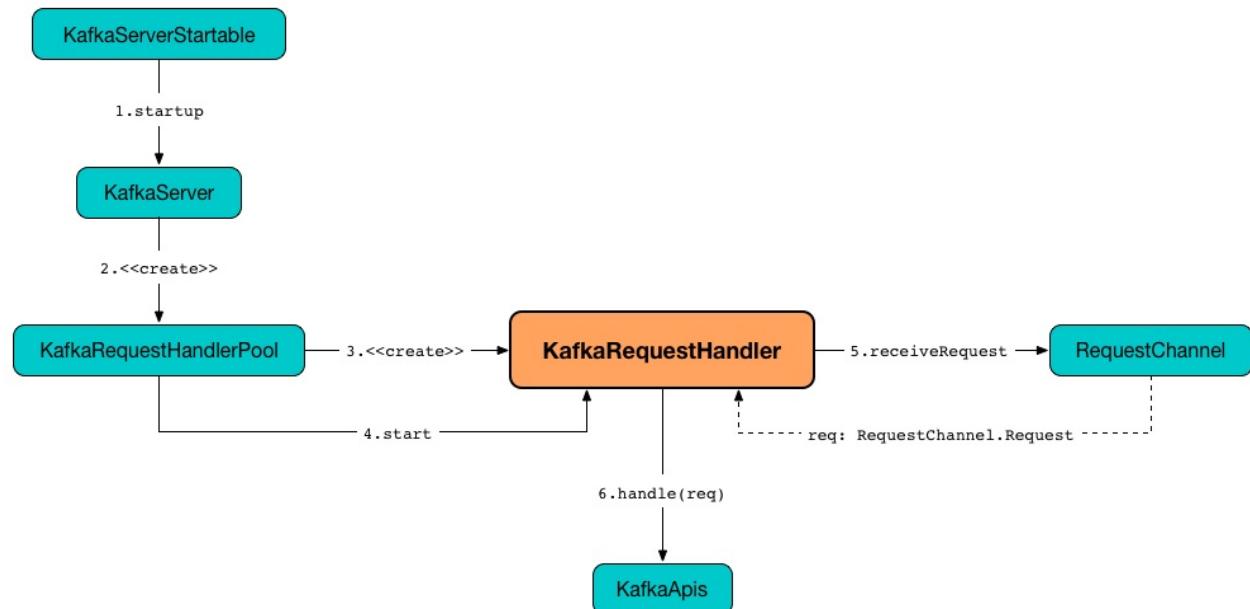


Figure 1. KafkaRequestHandler's Startup and Request Relay

`KafkaRequestHandler` uses **[Kafka Request Handler [id] on Broker [brokerId]]** as the logging prefix (aka `logIdent`).

Tip	<p>Enable <code>ALL</code> logging levels for <code>kafka.server.KafkaRequestHandler</code> logger to see what happens inside.</p> <p>Add the following line to <code>config/log4j.properties</code> :</p> <pre>log4j.logger.kafka.server.KafkaRequestHandler=ALL</pre> <p>Refer to Logging.</p>
------------	--

Creating KafkaRequestHandler Instance

`KafkaRequestHandler` takes the following to be created:

- ID

- Broker ID
- Aggregate Idle Meter
- Total number of handler threads
- RequestChannel
- KafkaApis
- Time

KafkaRequestHandler initializes the internal properties.

Starting KafkaRequestHandler Thread — run Method

```
run(): Unit
```

Note run is part of the `java.lang.Runnable` to start itself as a separately-executing thread.

run runs continuously until KafkaRequestHandler is requested to stop (which turns the stopped internal flag on).

run ...FIXME

Stopping KafkaRequestHandler Thread — stop Method

```
stop(): Unit
```

stop simply turns the stopped internal flag on (true).

Note stop is used exclusively when KafkaRequestHandlerPool is requested to resize the thread pool of KafkaRequestHandlers.

Internal Properties

Name	Description
stopped	<p>Flag to control whether <code>run</code> should stop (<code>true</code>) or not (<code>false</code>) (and hence the <code>KafkaRequestHandler</code>)</p> <p>Default: <code>false</code></p> <p>Turned on (<code>true</code>) when <code>KafkaRequestHandler</code> is requested to <code>stop</code></p>

LogManager

LogManager is created and immediately started when KafkaServer is requested to start up.

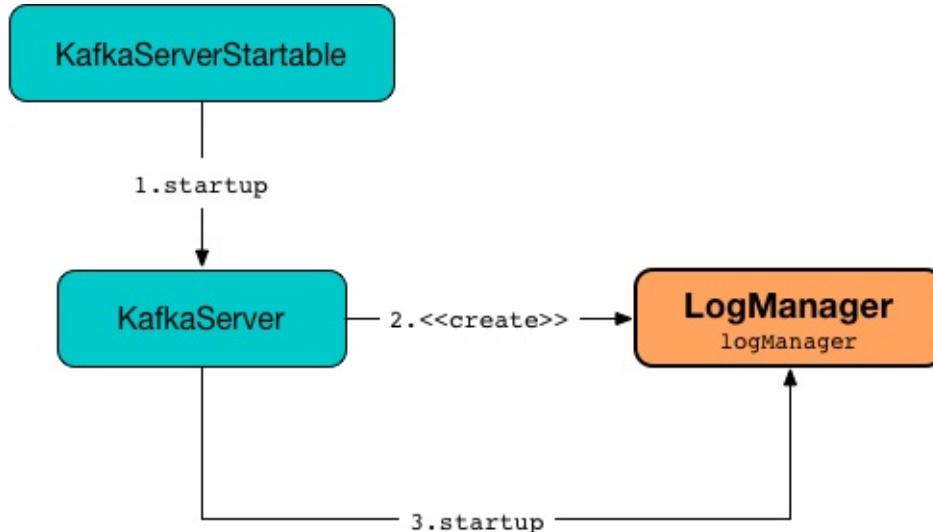


Figure 1. LogManager and KafkaServer

While being created, LogManager is given the log directories that are configured using `log.dirs` or `log.dir` configuration properties (default: `/tmp/kafka-logs`). The log directories are immediately validated and loaded (on a fixed thread pool with `numRecoveryThreadsPerDataDir` recovery threads).

LogManager uses the `num.recovery.threads.per.data.dir` dynamic configuration (default: 1) for the number of threads per log data directory for log recovery at startup and flushing at shutdown.

LogManager is used to create a [ReplicaManager](#), a [DynamicLogConfig](#), a [TopicConfigHandler](#).

LogManager defaults to 30000ms for the **initial task delay**.

Tip

Enable ALL logging level for `kafka.log.LogManager` logger to see what happens inside.

Add the following line to `config/log4j.properties` :

```
log4j.logger.kafka.log.LogManager=ALL
```

Refer to [Logging](#).

Performance Metrics

`LogManager` is a [KafkaMetricsGroup](#) with the following performance metrics.

Table 1. LogManager's Performance Metrics

Metric Name	Description
<code>OfflineLogDirectoryCount</code>	The number of offline log directories
<code>LogDirectoryOffline</code>	<p>Registered for every log directory to indicate whether it is online or offline</p> <p>Possible values:</p> <ul style="list-style-type: none"> • <code>0</code> when a log directory is online • <code>1</code> when a log directory is offline <p>The path of the directory is the tag of the metric.</p>

The performance metrics are registered in `kafka.log:type=LogManager` group.

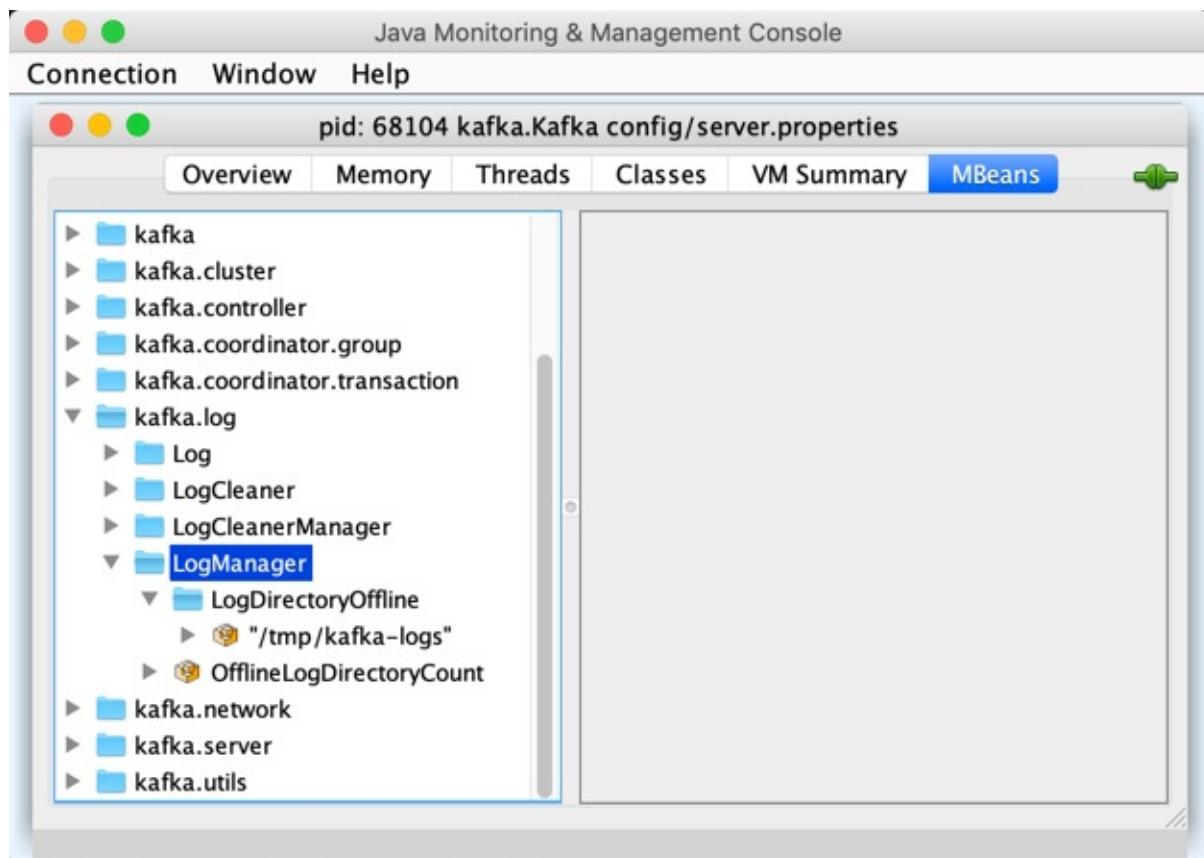


Figure 2. LogManager in jconsole

Creating LogManager — apply Factory Method

```
apply(
  config: KafkaConfig,
  initialOfflineDirs: Seq[String],
  zkClient: KafkaZkClient,
  brokerState: BrokerState,
  kafkaScheduler: KafkaScheduler,
  time: Time,
  brokerTopicStats: BrokerTopicStats,
  logDirFailureChannel: LogDirFailureChannel): LogManager
```

apply ...FIXME

Note

`apply` is used exclusively when `KafkaServer` is requested to [start up](#).

Creating LogManager Instance

`LogManager` takes the following to be created:

- Absolute paths to log directories (`Seq[File]`)
- Initial offline directories (`Seq[File]`)
- Topic configurations - [LogConfigs](#) per topic name (`Map[String, LogConfig]`)
- Initial [LogConfig](#)
- [CleanerConfig](#)
- `recoveryThreadsPerDataDir` (based on the `num.recovery.threads.per.data.dir` dynamic configuration property)
- `flushCheckMs`
- `flushRecoveryOffsetCheckpointMs`
- `flushStartOffsetCheckpointMs`
- `retentionCheckMs`
- `maxPidExpirationMs`
- [Scheduler](#)
- `BrokerState`
- [BrokerTopicStats](#)
- `LogDirFailureChannel`

- Time

LogManager initializes the [internal properties](#).

While being created, LogManager [loads logs](#).

Recovering And Loading Logs In Log Data Directories

— `loadLogs` Internal Method

```
loadLogs(): Unit
```

loadLogs prints out the following INFO message to the logs:

```
Loading logs.
```

For every [live log directory](#), loadLogs first creates a fixed thread pool (with [numRecoveryThreadsPerDataDir](#) threads).

loadLogs then checks whether [.kafka_cleanshutdown](#) file exists in the log directory. If so, loadLogs prints out the following DEBUG message to the logs:

```
Found clean shutdown file. Skipping recovery for all logs in data directory: [dir]
```

loadLogs uses the [recoveryPointCheckpoints](#) to look up the [offsetCheckpointFile](#) for the log directory (**recovery-point-offset-checkpoint** file) and then loads it.

loadLogs uses the [logStartOffsetCheckpoints](#) to look up the [offsetCheckpointFile](#) for the log directory (**recovery-point-offset-checkpoint** file) and then loads it.

For every directory in the log directory, loadLogs creates a new thread to [load the log directory](#) with the recovery points and log start offsets (that have just been loaded).

loadLogs submits the new threads to [load the log directory](#) for execution on the fixed thread pool.

loadLogs then...FIXME (finish me)

In the end, after [having loaded the log directories](#) successfully, loadLogs prints out the following INFO message to the logs:

```
Logs loading complete in [duration] ms.
```

In case `.kafka_cleanshutdown` file does not exist, `loadLogs` transitions the `BrokerState` to `RecoveringFromUncleanShutdown`.

In case of an exception while loading the `offsetCheckpointFile` of a log directory (**recovery-point-offset-checkpoint** file), `loadLogs` simply prints out the following WARN messages to the logs:

```
Error occurred while reading recovery-point-offset-checkpoint file of directory [dir]
Resetting the recovery checkpoint to 0
```

In case of an exception while loading the `offsetCheckpointFile` of a log directory (**log-start-offset-checkpoint** file), `loadLogs` simply prints out the following WARN messages to the logs:

```
Error occurred while reading log-start-offset-checkpoint file of directory [dir]
```

In case of an exception while `load the log directory` or any other task, `loadLogs` adds the log directory to a `offlineDirs` internal registry with the exception and prints out the following ERROR message to the logs:

```
Error while loading log dir [dir]
```

Note	<code>loadLogs</code> is used exclusively when <code>LogManager</code> is <code>created</code> .
-------------	--

Loading Partition Log Directory — `loadLog` Internal Method

```
loadLog(
  logDir: File,
  recoveryPoints: Map[TopicPartition, Long],
  logStartOffsets: Map[TopicPartition, Long]): Unit
```

`loadLog` first prints out the following DEBUG message to the logs:

```
Loading log '[logDir]'
```

`loadLog` then `parses the topic and partition out of the directory name of the log` (by the given `logDir`).

`loadLog` gets the `LogConfig` for the topic (from the `LogConfigs per topic`) or defaults to the `currentDefaultConfig`.

`loadLog` gets `logRecoveryPoint` for the partition (from the given `recoveryPoints`) or defaults to `0`.

`loadLog` gets `logStartOffset` for the partition (from the given `logStartOffsets`) or defaults to `0`.

`loadLog` creates a [Log](#).

In case the name of the given `logDir` ends with `-delete` suffix, `loadLog` [addLogToBeDeleted](#).

Otherwise, `loadLog` adds the `Log` to the [futureLogs](#) or [currentLogs](#) internal registry whether it is [isFuture](#) or not, respectively.

In case there was `Log` already registered (the [futureLogs](#) or [currentLogs](#) internal registry) `loadLog` throws an `IllegalStateException`:

FIXME

Note

`loadLog` is used exclusively when `LogManager` is requested to [recover](#) and [load the logs in log data directories](#).

Starting Up— `startup` Method

`startup(): Unit`

`startup` starts the background threads to flush logs and do log cleanup.

Internally, `startup` prints out the following INFO message to the logs:

Starting log cleanup with a period of [retentionCheckMs] ms.

`startup` requests the [Scheduler](#) to [schedule a task](#) with the name **kafka-log-retention** that [cleanupLogs](#) with the [InitialTaskDelayMs](#) delay and the [retentionCheckMs](#) execution period.

`startup` prints out the following INFO message to the logs:

Starting log flusher with a default period of [flushCheckMs] ms.

`startup` requests the [Scheduler](#) to [schedule a task](#) with the name **kafka-log-flusher** that [flushDirtyLogs](#) with the [InitialTaskDelayMs](#) delay and the [flushCheckMs](#) execution period.

`startup` requests the [Scheduler](#) to schedule a task with the name **kafka-recovery-point-checkpoint** that [checkpointLogRecoveryOffsets](#) with the [InitialTaskDelayMs](#) delay and the [flushRecoveryOffsetCheckpointMs](#) execution period.

`startup` requests the [Scheduler](#) to schedule a task with the name **kafka-log-start-offset-checkpoint** that [checkpointLogStartOffsets](#) with the [InitialTaskDelayMs](#) delay and the [flushStartOffsetCheckpointMs](#) execution period.

`startup` requests the [Scheduler](#) to schedule a task with the name **kafka-delete-logs** that [deleteLogs](#) with the [InitialTaskDelayMs](#) delay.

(only when the [CleanerConfig](#) has the [enableCleaner](#) flag enabled) `startup` requests the [LogCleaner](#) to [start up](#).

Note

`startup` is used exclusively when `KafkaServer` is requested to [start up](#).

BrokerTopicStats

When [created](#), `LogManager` is given a [BrokerTopicStats](#) that is used exclusively to create [Logs](#) when [recovering and loading logs](#) in log data directories and [looking up or creating a Log](#).

cleanupLogs Method

```
cleanupLogs(): Unit
```

`cleanupLogs` ...FIXME

Note

`cleanupLogs` is used when...FIXME

Getting All Partition Logs — allLogs Method

```
allLogs: Iterable[Log]
```

`allLogs` ...FIXME

Note

`allLogs` is used when...FIXME

addLogToDelete Internal Method

```
addLogToBeDeleted(log: Log): Unit
```

`addLogToBeDeleted` ...FIXME

Note

`addLogToBeDeleted` is used when...FIXME

asyncDelete Method

```
asyncDelete(
    topicPartition: TopicPartition,
    isFuture: Boolean = false): Log
```

`asyncDelete` ...FIXME

Note

`asyncDelete` is used when:

- `Partition` is requested to `removeFutureLocalReplica` and `delete`
- `ReplicaManager` is requested to `stopReplica`

Looking Up Or Creating Log — getOrCreateLog Method

```
getOrCreateLog(
    topicPartition: TopicPartition,
    config: LogConfig,
    isNew: Boolean = false,
    isFuture: Boolean = false): Log
```

`getOrCreateLog` ...FIXME

Note

`getOrCreateLog` is used exclusively when `Partition` is requested to `getOrCreateReplica`.

liveLogDirs Method

```
liveLogDirs: Seq[File]
```

`liveLogDirs` ...FIXME

Note

`liveLogDirs` is used when...FIXME

deleteLogs Internal Method

```
deleteLogs(): Unit
```

deleteLogs ...FIXME

Note	deleteLogs is used when...FIXME
------	---------------------------------

flushDirtyLogs Internal Method

```
flushDirtyLogs(): Unit
```

flushDirtyLogs ...FIXME

Note	flushDirtyLogs is used when...FIXME
------	-------------------------------------

checkpointLogRecoveryOffsets Method

```
checkpointLogRecoveryOffsets(): Unit
```

checkpointLogRecoveryOffsets ...FIXME

Note	checkpointLogRecoveryOffsets is used when...FIXME
------	---

checkpointLogStartOffsets Method

```
checkpointLogStartOffsets(): Unit
```

checkpointLogStartOffsets ...FIXME

Note	checkpointLogStartOffsets is used when...FIXME
------	--

isLogDirOnline Method

```
isLogDirOnline(logDir: String): Boolean
```

isLogDirOnline ...FIXME

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

Validating Data Log Directories

— `createAndValidateLogDirs` Internal Method

```
createAndValidateLogDirs(  
    dirs: Seq[File],  
    initialOfflineDirs: Seq[File]): ConcurrentLinkedQueue[File]
```

For every directory in the given `dirs`, `createAndValidateLogDirs` makes sure that the data directory is available (i.e. it is a readable directory) or creates it.

`createAndValidateLogDirs` prints out the following INFO message to the logs when a data directory does not exist:

```
Log directory [dir] not found, creating it.
```

Note	<code>createAndValidateLogDirs</code> is given the <code>logDirs</code> and the <code>initialOfflineDirs</code> that <code>LogManager</code> is created with.
------	---

`createAndValidateLogDirs` throws...FIXME

Note	<code>createAndValidateLogDirs</code> is used exclusively when <code>LogManager</code> is created.
------	--

truncateTo Method

```
truncateTo(  
    partitionOffsets: Map[TopicPartition, Long],  
    isFuture: Boolean): Unit
```

`truncateTo` ...FIXME

Note	<code>truncateTo</code> is used exclusively when <code>Partition</code> is requested to <code>truncateTo</code> .
------	---

truncateFullyAndStartAt Method

```
truncateFullyAndStartAt(  
    topicPartition: TopicPartition,  
    newOffset: Long,  
    isFuture: Boolean): Unit
```

```
truncateFullyAndStartAt ...FIXME
```

Note	<code>truncateFullyAndStartAt</code> is used exclusively when <code>Partition</code> is requested to <code>truncateFullyAndStartAt</code> .
------	---

resizeRecoveryThreadPool Method

```
resizeRecoveryThreadPool(newSize: Int): Unit
```

`resizeRecoveryThreadPool` prints out the following INFO message to the logs and reconfigures the `numRecoveryThreadsPerDataDir` internal registry to be the given `newSize`.

```
Resizing recovery thread pool size for each data dir from [numRecoveryThreadsPerDataDir] to [newSize]
```

Note	<code>resizeRecoveryThreadPool</code> is used exclusively when <code>DynamicThreadPool</code> is requested to <code>reconfigure</code> (with a new value of <code>KafkaConfig.numRecoveryThreadsPerDataDir</code>).
------	--

Shutting Down — shutdown Method

```
shutdown(): Unit
```

`shutdown` prints out the following INFO message to the logs:

```
Shutting down.
```

```
shutdown then...FIXME
```

Note	<code>shutdown</code> is used exclusively when <code>KafkaServer</code> is requested to <code>shutdown</code> .
------	---

replaceCurrentWithFutureLog Method

```
replaceCurrentWithFutureLog(topicPartition: TopicPartition): Unit
```

```
replaceCurrentWithFutureLog ...FIXME
```

Note	<code>replaceCurrentWithFutureLog</code> is used exclusively when <code>Partition</code> is requested to <code>maybeReplaceCurrentWithFutureReplica</code> .
------	--

handleLogDirFailure Method

```
handleLogDirFailure(dir: String): Unit
```

handleLogDirFailure ...FIXME

Note	<code>handleLogDirFailure</code> is used exclusively when <code>ReplicaManager</code> is requested to <code>handleLogDirFailure</code> .
------	--

Internal Properties

Name	Description
<code>_liveLogDirs</code>	Java's <code>ConcurrentLinkedQueue</code> of live log directories (after <code>createAndValidateLogDirs</code> was executed with the <code>logDirs</code> and the <code>initialOfflineDirs</code> directories). Used when...FIXME
<code>cleaner</code>	
<code>currentDefaultConfig</code>	Default <code>LogConfig</code> Used when a custom <code>LogConfig</code> is not available in the <code>topicConfigs</code>
<code>currentLogs</code>	Pool of <code>Logs</code> per <code>TopicPartition</code> (<code>Pool[TopicPartition, Log]</code>)
<code>futureLogs</code>	Pool of <code>Logs</code> per <code>TopicPartition</code> (<code>Pool[TopicPartition, Log]</code>)
<code>logStartOffsetCheckpoints</code>	
<code>numRecoveryThreadsPerDataDir</code>	Number of recovery threads per log data directory Starts as the <code>recoveryThreadsPerDataDir</code> and can then be <code>dynamically changed</code> .
<code>recoveryPointCheckpoints</code>	

Partition Log

`Log` is a **partition log** that is [created](#) when `LogManager` is requested to [loadLog](#) (when `LogManager` is [created](#)) and [getOrCreateLog](#).

`Log` is a [KafkaMetricsGroup](#) and registers [performance metrics](#).

`Log` uses [.kafka_cleanshutdown](#) for...FIXME

`Log` uses [-delete](#) suffix for...FIXME

`Log` is [isFuture](#) when...FIXME

Table 1. Log's Performance Metrics

Metric Name	Description
NumLogSegments	
LogEndOffset	
LogStartOffset	
Size	

The [performance metrics](#) are registered in `kafka.log:type=Log` group.

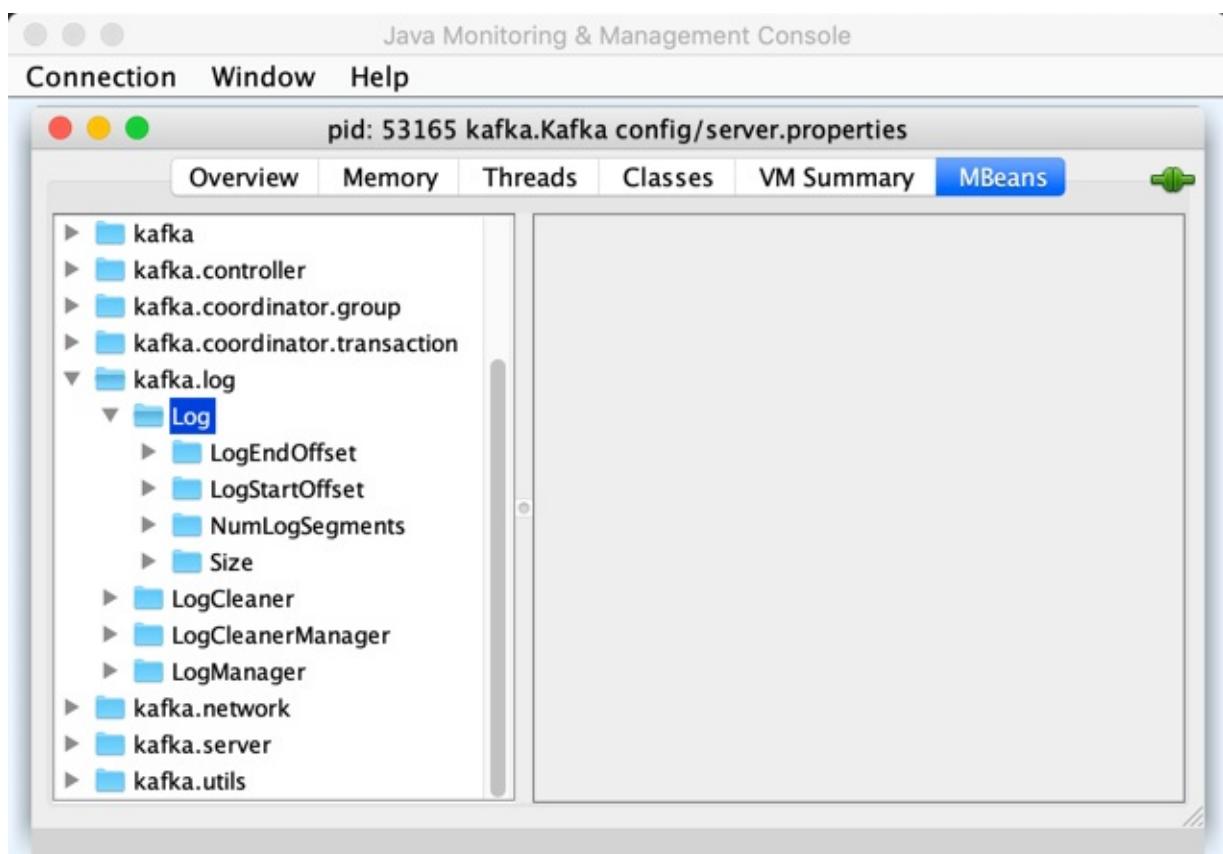


Figure 1. Log in jconsole

`Log` uses a [Scheduler](#) to schedule the [background tasks](#).

Table 2. Log's Background Tasks

Name	Period	
PeriodicProducerExpirationCheck	producerIdExpirationCheckIntervalMs	producerIdE
flush-log	-1 (once)	0L
delete-file	-1 (once)	file.delete.d

`Log` uses **[Log partition=[topicPartition], dir=[parent]]** as the logging prefix (aka `logIdent`).

Table 3. Log's Internal Properties (e.g. Registries, Counters and Flags)

Name	Description
nextOffsetMetadata	<code>LogOffsetMetadata</code> Used when...FIXME
segments	<code>java.util.concurrent.ConcurrentSkipListMap</code> of <code>Longs</code> and their <code>LogSegments</code> Used when...FIXME

Tip	Enable <code>WARN</code> , <code>INFO</code> , <code>DEBUG</code> or <code>TRACE</code> logging levels for <code>kafka.log.Log</code> logger to see what happens inside. Add the following line to <code>log4j.properties</code> : <code>log4j.logger.kafka.log.Log=TRACE</code> Refer to Logging .

Creating Log Instance

`Log` takes the following when created:

- Log directory
- `LogConfig`
- `logStartOffset`
- `recoveryPoint`
- `Scheduler`
- `BrokerTopicStats`
- `Time`
- `maxProducerIdExpirationMs`
- `producerIdExpirationCheckIntervalMs`
- `TopicPartition`
- `ProducerStateManager`
- `LogDirFailureChannel`

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

While being created, `Log` ...FIXME

Creating Log Instance — `apply` Factory Method

```
apply(
    dir: File,
    config: LogConfig,
    logStartOffset: Long,
    recoveryPoint: Long,
    scheduler: Scheduler,
    brokerTopicStats: BrokerTopicStats,
    time: Time = Time.SYSTEM,
    maxProducerIdExpirationMs: Int,
    producerIdExpirationCheckIntervalMs: Int,
    logDirFailureChannel: LogDirFailureChannel): Log
```

`apply` ...FIXME

Note	<code>apply</code> is used when <code>LogManager</code> is requested to <code>loadLog</code> and <code>getOrCreateLog</code> .
------	--

roll Method

```
roll(expectedNextOffset: Long = 0): LogSegment
```

roll ...FIXME

Note	roll is used when...FIXME
------	---------------------------

asyncDeleteSegment Internal Method

```
asyncDeleteSegment(segment: LogSegment): Unit
```

asyncDeleteSegment ...FIXME

Note	asyncDeleteSegment is used when Log is requested to deleteSegment and replaceSegments .
------	---

flush Method

```
flush(offset: Long): Unit
```

flush ...FIXME

Note	flush is used when...FIXME
------	----------------------------

deleteSeg Internal Method

```
deleteSeg(): Unit
```

deleteSeg ...FIXME

Note	deleteSeg is used when...FIXME
------	--------------------------------

appendAsLeader Method

```
appendAsLeader(  
    records: MemoryRecords,  
    leaderEpoch: Int,  
    isFromClient: Boolean = true): LogAppendInfo
```

`appendAsLeader` simply [append](#) with the `assignOffsets` flag on.

Note	<code>appendAsLeader</code> is used exclusively when <code>Partition</code> is requested to appendRecordsToLeader .
------	---

appendAsFollower Method

```
appendAsFollower(records: MemoryRecords): LogAppendInfo
```

`appendAsFollower` simply [append](#) with the `isFromClient` and `assignOffsets` flags off.

Note	<code>appendAsFollower</code> is used exclusively when <code>Partition</code> is requested to doAppendRecordsToFollowerOrFutureReplica .
------	--

append Internal Method

```
append(
    records: MemoryRecords,
    isFromClient: Boolean,
    assignOffsets: Boolean,
    leaderEpoch: Int): LogAppendInfo
```

`append` ...FIXME

Note	<code>append</code> is used when <code>Log</code> is requested to appendAsLeader and appendAsFollower .
------	---

analyzeAndValidateRecords Internal Method

```
analyzeAndValidateRecords(
    records: MemoryRecords,
    isFromClient: Boolean): LogAppendInfo
```

`analyzeAndValidateRecords` ...FIXME

Note	<code>analyzeAndValidateRecords</code> is used exclusively when <code>Log</code> is requested to append .
------	---

deleteSegment Internal Method

```
deleteSegment(segment: LogSegment): Unit
```

```
deleteSegment ...FIXME
```

Note

`deleteSegment` is used when `Log` is requested to `recoverLog`, `deleteSegments`, `roll`, `truncateTo`, and `truncateFullyAndStartAt`.

replaceSegments Internal Method

```
replaceSegments(  
    newSegments: Seq[LogSegment],  
    oldSegments: Seq[LogSegment],  
    isRecoveredSwapFile: Boolean = false): Unit
```

```
replaceSegments ...FIXME
```

Note

`replaceSegments` is used when:

- `Log` is requested to `completeSwapOperations` and `splitOverflowedSegment`
- `Cleaner` is requested to `cleanSegments`

recoverLog Internal Method

```
recoverLog(): Long
```

```
recoverLog ...FIXME
```

Note

`recoverLog` is used exclusively when `Log` is requested to `loadSegments`.

deleteSegments Internal Method

```
deleteSegments(deletable: Iterable[LogSegment]): Int
```

```
deleteSegments ...FIXME
```

Note

`deleteSegments` is used exclusively when `Log` is requested to `deleteOldSegments`.

truncateTo Internal Method

```
truncateTo(targetOffset: Long): Boolean
```

`truncateTo ...FIXME`

Note	<code>truncateTo</code> is used exclusively when <code>LogManager</code> is requested to <code>truncateTo</code> .
------	--

truncateFullyAndStartAt Internal Method

`truncateFullyAndStartAt(newOffset: Long): Unit`

`truncateFullyAndStartAt ...FIXME`

Note	<code>truncateFullyAndStartAt</code> is used when:
------	--

- `Log` is requested to `truncateTo`
- `LogManager` is requested to `truncateFullyAndStartAt`

loadSegments Internal Method

`loadSegments(): Long`

`loadSegments ...FIXME`

Note	<code>loadSegments</code> is used exclusively when <code>Log</code> is <code>created</code> (to create a <code>LogOffsetMetadata</code>).
------	--

deleteOldSegments Method

```
deleteOldSegments(): Long
// Private API
deleteOldSegments(
  predicate: (LogSegment, Option[LogSegment]) => Boolean,
  reason: String): Int
```

`deleteOldSegments ...FIXME`

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

completeSwapOperations Internal Method

`completeSwapOperations(swapFiles: Set[File]): Unit`

```
completeSwapOperations ...FIXME
```

Note	completeSwapOperations is used when...FIXME
------	---

splitOverflowedSegment Internal Method

```
splitOverflowedSegment(segment: LogSegment): List[LogSegment]
```

```
splitOverflowedSegment ...FIXME
```

Note	splitOverflowedSegment is used when...FIXME
------	---

loadProducerState Internal Method

```
loadProducerState(lastOffset: Long, reloadFromCleanShutdown: Boolean): Unit
```

```
loadProducerState ...FIXME
```

Note	loadProducerState is used when Log is created and requested to truncateTo .
------	---

loadSegmentFiles Internal Method

```
loadSegmentFiles(): Unit
```

```
loadSegmentFiles ...FIXME
```

Note	loadSegmentFiles is used exclusively when Log is requested to loadSegments (when created).
------	---

onHighWatermarkIncremented Method

```
onHighWatermarkIncremented(highwatermark: Long): Unit
```

```
onHighWatermarkIncremented ...FIXME
```

Note	onHighWatermarkIncremented is used when...FIXME
------	---

parseTopicPartitionName Object Method

```
parseTopicPartitionName(dir: File): TopicPartition
```

```
parseTopicPartitionName ...FIXME
```

Note

parseTopicPartitionName is used when...FIXME

LogSegment

LogSegment is...FIXME

LogCleanerManager

LogCleanerManager is created exclusively for LogCleaner.

LogCleanerManager is a KafkaMetricsGroup and registers performance metrics.

Table 1. LogCleanerManager's Performance Metrics

Metric Name	Description
max-dirty-percent	
time-since-last-run-ms	
uncleanable-bytes	(for every log directory)
uncleanable-partitions-count	(for every log directory)

The performance metrics are registered in `kafka.log:type=LogCleanerManager` group.

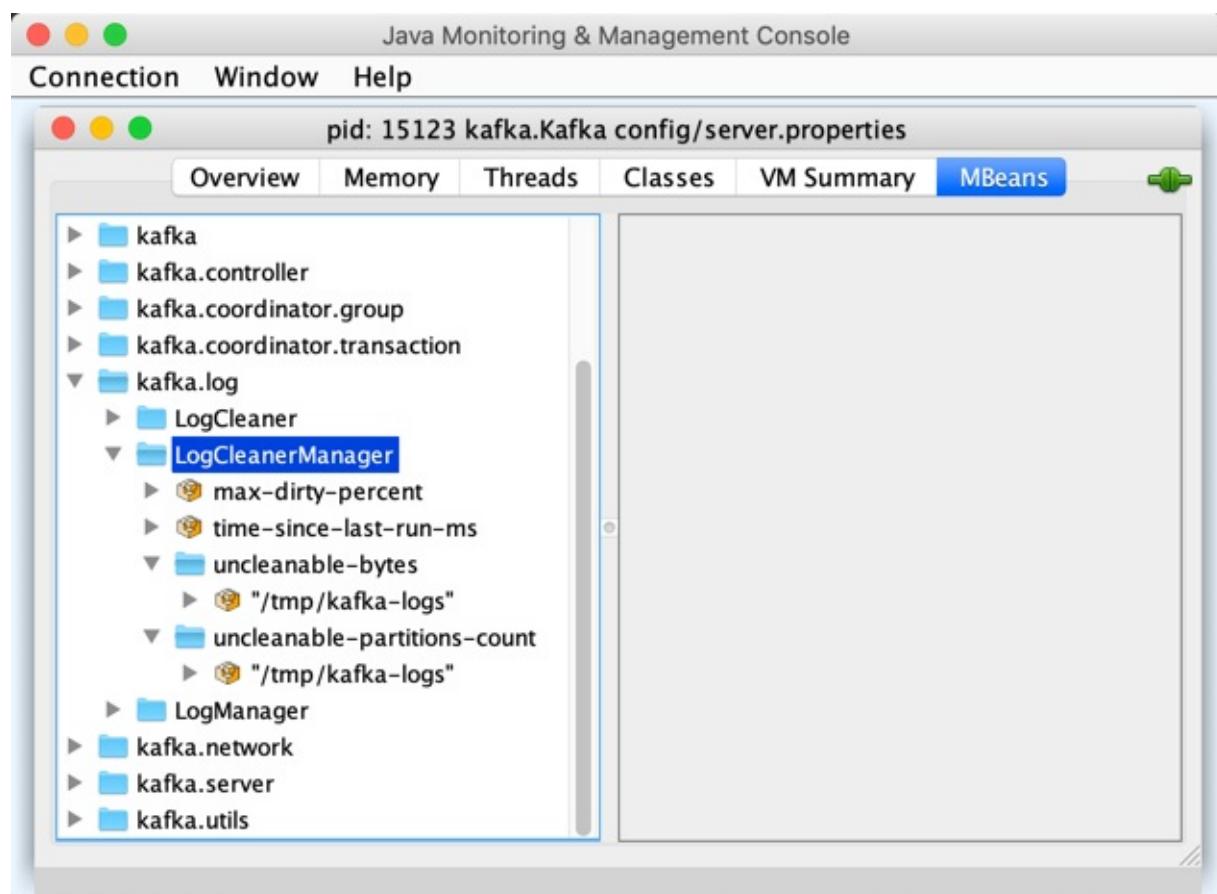


Figure 1. LogCleanerManager in jconsole

LogCleanerManager uses **cleaner-offset-checkpoint** for the name of the offset checkpoint files for every log directory (in checkpoints registry).

Table 2. LogCleanerManager's Internal Properties (e.g. Registries, Counters and Flags)

Name	Description
checkpoints	<p><code>offsetCheckpointFiles</code> per log directory Used in <code>updateCheckpoints</code>, <code>maybeTruncateCheckpoint</code>, and <code>alterCheckpointDir</code> to look up the <code>offsetCheckpointFile</code> per log data directory Used in <code>allCleanerCheckpoints</code> to access all <code>offsetCheckpointFiles</code> Used in <code>handleLogDirFailure</code> to exclude a failed log directory from cleaning</p>

`LogCleanerManager` uses `kafka.log.LogCleaner` logger.

Tip	Enable <code>INFO</code> logging level for <code>kafka.log.LogCleaner</code> logger to see what happens inside.
-----	---

Creating LogCleanerManager Instance

`LogCleanerManager` takes the following to be created:

- Log directories
- Logs per Kafka TopicPartition
- LogDirFailureChannel

`LogCleanerManager` initializes the internal registries and counters.

isCompactAndDelete Predicate

```
isCompactAndDelete(log: Log): Boolean
```

`isCompactAndDelete` ...FIXME

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

cleanableOffsets Method

```
cleanableOffsets(  
    log: Log,  
    topicPartition: TopicPartition,  
    lastClean: immutable.Map[TopicPartition, Long],  
    now: Long): (Long, Long)
```

cleanableOffsets ...FIXME

Note

cleanableOffsets is used when...FIXME

grabFilthiestCompactedLog Method

```
grabFilthiestCompactedLog(time: Time): Option[LogToClean]
```

grabFilthiestCompactedLog ...FIXME

Note

grabFilthiestCompactedLog is used when...FIXME

deletableLogs Method

```
deletableLogs(): Iterable[(TopicPartition, Log)]
```

deletableLogs ...FIXME

Note

deletableLogs is used when...FIXME

doneDeleting Method

```
doneDeleting(topicPartitions: Iterable[TopicPartition]): Unit
```

doneDeleting ...FIXME

Note

doneDeleting is used when...FIXME

doneCleaning Method

```
doneCleaning(
    topicPartition: TopicPartition,
    dataDir: File,
    endOffset: Long): Unit
```

doneCleaning ...FIXME

Note

`doneCleaning` is used exclusively when `cleanerThread` is requested to [cleanLog](#).

allCleanerCheckpoints Method

```
allCleanerCheckpoints: Map[TopicPartition, Long]
```

allCleanerCheckpoints ...FIXME

Note

`allCleanerCheckpoints` is used when:

- `LogCleanerManager` is requested to [grabFilthiestCompactedLog](#) and for the [uncleanable-bytes](#) performance metric
- `LogCleaner` is requested to [awaitCleaned](#) (for tests only)

alterCheckpointDir Method

```
alterCheckpointDir(
    topicPartition: TopicPartition,
    sourceLogDir: File,
    destLogDir: File): Unit
```

alterCheckpointDir ...FIXME

Note

`alterCheckpointDir` is used exclusively when `LogCleaner` is requested to [alterCheckpointDir](#).

handleLogDirFailure Method

```
handleLogDirFailure(dir: String): Unit
```

handleLogDirFailure ...FIXME

Note

`handleLogDirFailure` is used exclusively when `LogCleaner` is requested to [handleLogDirFailure](#).

updateCheckpoints Method

```
updateCheckpoints(  
    dataDir: File,  
    update: Option[(TopicPartition, Long)]): Unit
```

`updateCheckpoints` ...FIXME

Note

`updateCheckpoints` is used when:

- `LogCleaner` is requested to [updateCheckpoints](#)
- `LogCleanerManager` is requested to [alterCheckpointDir](#) and [doneCleaning](#)

maybeTruncateCheckpoint Method

```
maybeTruncateCheckpoint(dataDir: File, topicPartition: TopicPartition, offset: Long):  
Unit
```

`maybeTruncateCheckpoint` ...FIXME

Note

`maybeTruncateCheckpoint` is used exclusively when `LogCleaner` is requested to [maybeTruncateCheckpoint](#).

LogCleaner

`LogCleaner` is created exclusively when `LogManager` is created (with `enableCleaner` flag enabled which is the default).

When created, `LogCleaner` is given a `CleanerConfig` that `LogManager` builds when created (when `KafkaServer` is requested to start up).

`LogCleaner` is a [KafkaMetricsGroup](#) and registers performance metrics.

Table 1. LogCleaner's Performance Metrics

Metric Name	Description
<code>max-buffer-utilization-percent</code>	
<code>cleaner-recopy-percent</code>	
<code>max-clean-time-secs</code>	

The performance metrics are registered in `kafka.log:type=LogCleaner` group.

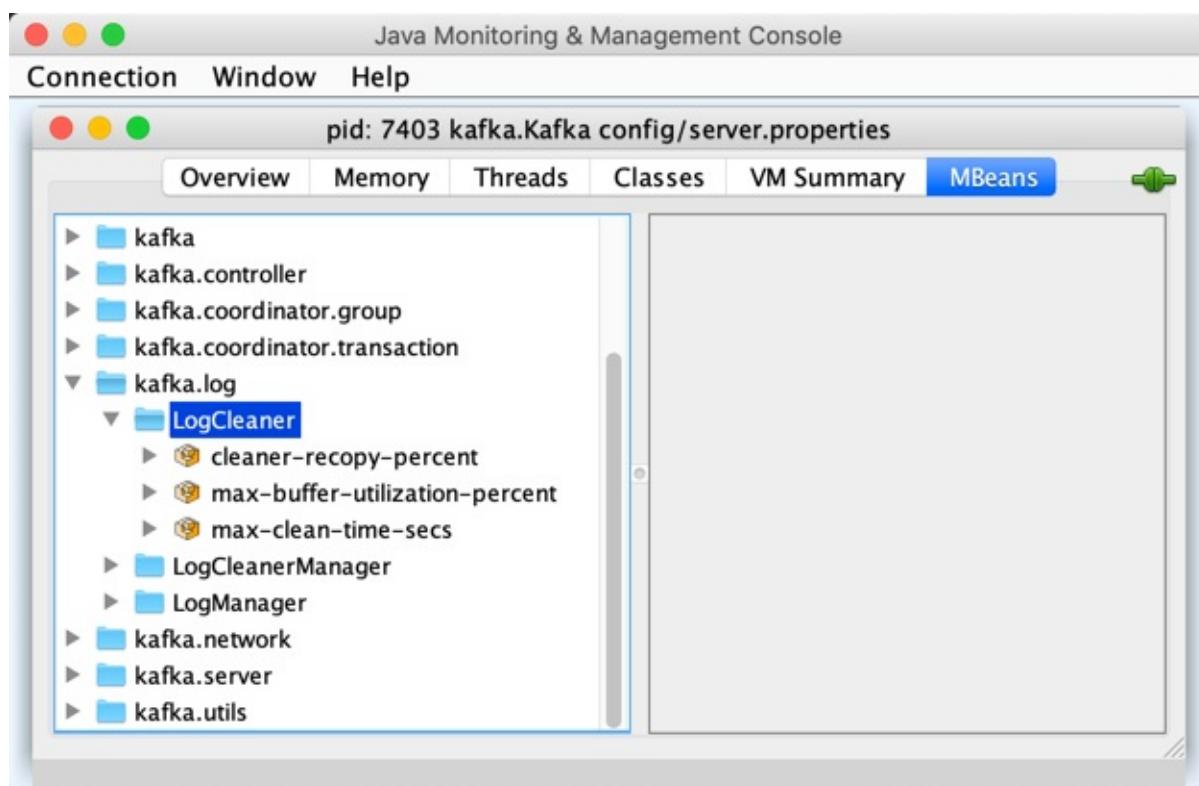


Figure 1. LogCleaner in jconsole

`LogCleaner` is a [BrokerReconfigurable](#) for the following dynamic configurations:

- `log.cleaner.threads`

- `log.cleaner.dedupe.buffer.size`
- `log.cleaner.io.buffer.load.factor`
- `log.cleaner.io.buffer.size`
- `log.cleaner.io.max.bytes.per.second`
- `log.cleaner.backoff.ms`
- `message.max.bytes`

Table 2. LogCleaner's Internal Properties (e.g. Registries, Counters and Flags)

Name	Description
<code>cleaners</code>	<code>CleanerThreads</code> Used when...FIXME
<code>config</code>	<code>CleanerConfig</code> Initialized with the given <code>CleanerConfig</code> Changed in <code>reconfigure</code>
<code>cleanerManager</code>	<code>LogCleanerManager</code>

Enable `WARN` or `INFO` logging levels for `kafka.log.LogCleaner` logger to see what happens inside.

Add the following line to `config/log4j.properties`:

```
log4j.logger.kafka.log.LogCleaner=INFO
```

Refer to [Logging](#).

Tip

Please note that Kafka comes with a preconfigured `kafka.log.LogCleaner` logger in `config/log4j.properties`:

```
log4j.appenders.cleanerAppender=org.apache.log4j.DailyRollingFileAppender
log4j.appenders.cleanerAppender.DatePattern='.' 'yyyy-MM-dd-HH
log4j.appenders.cleanerAppender.File=${kafka.logs.dir}/log-cleaner.log
log4j.appenders.cleanerAppender.layout=org.apache.log4j.PatternLayout
log4j.appenders.cleanerAppender.layout.ConversionPattern=[%d] %p %m (%c)%n

log4j.logger.kafka.log.LogCleaner=INFO, cleanerAppender
log4j.additivity.kafka.log.LogCleaner=false
```

That means that the logs of `LogCleaner` go to `logs/log-cleaner.log` file at `INFO` logging level and are not added to the main logs (per `log4j.additivity` being off).

createNewCleanedSegment Method

```
createNewCleanedSegment(log: Log, baseOffset: Long): LogSegment
```

`createNewCleanedSegment` ...FIXME

Note

`createNewCleanedSegment` is used when...FIXME

cleanSegments Internal Method

```
cleanSegments(
  log: Log,
  segments: Seq[LogSegment],
  map: OffsetMap,
  deleteHorizonMs: Long,
  stats: CleanerStats): Unit
```

`cleanSegments` ...FIXME

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

buildOffsetMap Internal Method

```
buildOffsetMap(
  log: Log,
  start: Long,
  end: Long,
  map: OffsetMap,
  stats: CleanerStats): Unit
```

`buildOffsetMap` ...FIXME

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

Reconfiguring — `reconfigure` Method

```
reconfigure(oldConfig: KafkaConfig, newConfig: KafkaConfig): Unit
```

Note	<code>reconfigure</code> is part of the BrokerReconfigurable Contract to change (<code>reconfigure</code>) the value of a Kafka dynamic configuration.
------	--

`reconfigure` ...FIXME

Creating LogCleaner Instance

`LogCleaner` takes the following to be created:

- [CleanerConfig](#)
- Log directories
- [Logs](#) per Kafka `TopicPartition`
- `LogDirFailureChannel`
- `Time`

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

Starting Up — `startup` Method

```
startup(): Unit
```

`startup` prints out the following INFO message to the logs:

```
Starting the log cleaner
```

`startup` creates new [CleanerThreads](#) and [starts](#) them all immediately.

`startup` adds the cleaner threads in [cleaners](#) internal registry.

Note

The number of `CleanerThreads` is controlled by [log.cleaner.threads](#) dynamic configuration (default: `1`).

Note

`startup` is used when:

- `LogManager` is requested to [start up](#) (with `enableCleaner` enabled which is the default)
- `LogCleaner` is requested to [reconfigure](#)

Building CleanerConfig From KafkaConfig — `cleanerConfig` Method

```
cleanerConfig(config: KafkaConfig): CleanerConfig
```

`cleanerConfig` simply creates a [CleanerConfig](#) from the given [KafkaConfig](#).

Note

`cleanerConfig` is used when:

- `LogCleaner` is requested to [validateReconfiguration](#) and [reconfigure](#)
- `LogManager` is [created](#)

CleanerConfig

`CleanerConfig` represents a set of [dynamic configurations](#) of a [LogCleaner](#):

- `log.cleaner.threads` (default: `1`)
- `dedupeBufferSize` (default: `4*1024*1024L`)
- `dedupeBufferLoadFactor` (default: `0.9d`)
- `ioBufferSize` (default: `1024*1024`)

- `maxMessageSize` (default: `32*1024*1024`)
- `maxIoBytesPerSecond` (default: `Double.MaxValue`)
- `backOffMs` (default: `15 * 1000`)
- `enableCleaner` flag (default: `true`)
- `hashAlgorithm` (default: `MD5`)

`CleanerConfig` is created exclusively when `LogCleaner` is requested to [build a CleanerConfig from a KafkaConfig](#).

awaitCleaned Method

```
awaitCleaned(
    topicPartition: TopicPartition,
    offset: Long,
    maxWaitMs: Long = 60000L): Boolean
```

`awaitCleaned` ...FIXME

Note	<code>awaitCleaned</code> seems to be used exclusively in tests.
------	--

alterCheckpointDir Method

```
alterCheckpointDir(
    topicPartition: TopicPartition,
    sourceLogDir: File,
    destLogDir: File): Unit
```

`alterCheckpointDir` ...FIXME

Note	<code>alterCheckpointDir</code> is used exclusively when <code>LogManager</code> is requested to replaceCurrentWithFutureLog .
------	--

handleLogDirFailure Method

```
handleLogDirFailure(dir: String): Unit
```

`handleLogDirFailure` ...FIXME

Note

`handleLogDirFailure` is used exclusively when `LogManager` is requested to [handleLogDirFailure](#).

updateCheckpoints Method

```
updateCheckpoints(dataDir: File): Unit
```

`updateCheckpoints` ...FIXME

Note

`updateCheckpoints` is used exclusively when `LogManager` is requested to [asyncDelete](#).

maybeTruncateCheckpoint Method

```
maybeTruncateCheckpoint(  
    dataDir: File,  
    topicPartition: TopicPartition,  
    offset: Long): Unit
```

`maybeTruncateCheckpoint` ...FIXME

Note

`maybeTruncateCheckpoint` is used when `LogManager` is requested to [truncateTo](#) and [truncateFullyAndStartAt](#).

CleanerThread

`CleanerThread` is a [ShutdownableThread](#) with the name of the format **kafka-log-cleaner-thread-** followed by the [threadId](#).

`CleanerThread` is [created](#) and immediately [started](#) exclusively when `LogCleaner` is requested to [start up](#).

Note

The number of `CleanerThreads` (that `LogCleaner` uses) is controlled by [log.cleaner.threads](#) dynamic configuration (default: `1`).

`CleanerThread` takes a **threadId** to be created.

`CleanerThread` uses a `Cleaner`.

`CleanerThread` uses [kafka.log.LogCleaner](#) logger.

Tip

Enable `INFO` logging level for `kafka.log.LogCleaner` logger to see what happens inside.

Note

`CleanerThread` is a Java private inner class of [LogCleaner](#). It can only be used by `LogCleaner` and have a direct access to the internals of the `LogCleaner`.

checkDone Internal Method

```
checkDone(topicPartition: TopicPartition): Unit
```

`checkDone` ...FIXME

Note

`checkDone` is used exclusively when `CleanerThread` is [created](#) (to create the `Cleaner`).

dowork Method

```
dowork(): Unit
```

Note

`dowork` is part of the [ShutdownableThread Contract](#) to...FIXME.

`dowork` ...FIXME

cleanFilthiestLog Internal Method

```
cleanFilthiestLog(): Boolean
```

`cleanFilthiestLog` requests the [LogCleanerManager](#) to [grabFilthiestCompactedLog](#).

If there is any filthiest log to work on, `cleanFilthiestLog` [cleanLog](#) and turns the `cleaned` flag on.

With no logs to work on, `cleanFilthiestLog` simply turns the `cleaned` flag off.

Note	The value of <code>cleaned</code> flag is what <code>cleanFilthiestLog</code> returns.
------	--

`cleanFilthiestLog` requests the [LogCleanerManager](#) for the [deletable logs](#) and then requests every `Log` to [deleteOldSegments](#).

In the end, `cleanFilthiestLog` requests the [LogCleanerManager](#) to [doneDeleting](#) (with the `TopicPartitions` of the deletable logs).

In case of an exception, `cleanFilthiestLog` ...FIXME

Note	<code>cleanFilthiestLog</code> is used exclusively when <code>CleanerThread</code> is requested to doWork .
------	---

cleanLog Internal Method

```
cleanLog(cleanable: LogToClean): Unit
```

`cleanLog` requests the [Cleaner](#) to `clean` the given `LogToClean` and [recordStats](#).

In the end, `cleanLog` requests the [LogCleanerManager](#) to [doneCleaning](#).

Note	<code>cleanLog</code> is used exclusively when <code>CleanerThread</code> is requested to cleanFilthiestLog .
------	---

recordStats Internal Method

```
recordStats(
  id: Int,
  name: String,
  from: Long,
  to: Long,
  stats: CleanerStats): Unit
```

`recordStats ...FIXME`

Note

`recordStats` is used exclusively when `CleanerThread` is requested to [cleanLog](#).

ProducerStateManager

ProducerStateManager is...FIXME

removeExpiredProducers Method

```
removeExpiredProducers(currentTimeMs: Long): Unit
```

removeExpiredProducers ...FIXME

Note

removeExpiredProducers is used when...FIXME

LogConfig

`LogConfig` is a Apache Kafka [AbstractConfig](#) for the [configuration properties](#) of a...FIXME

Table 1. LogConfig's Configuration Properties

Name / Default Value / Property	Description
FileDeleteDelayMsProp Default: 60000 Property: <code>file.delete.delay.ms</code>	The time (in millis) to wait before deleting a file from the filesystem Must be at least 0

MetadataCache

`MetadataCache` is...FIXME

`MetadataCache` is created exclusively for [KafkaServer](#) (when started).

`MetadataCache` takes a single broker ID when created.

`MetadataCache` maintains the [metadataSnapshot](#) and is used to...FIXME

`MetadataCache` tracks the `controllerId` ...FIXME

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

Name	Description
<code>metadataSnapshot</code>	<code>MetadataSnapshot</code> <code>metadataSnapshot</code> is updated when <code>MetadataCache</code> is requested to updateMetadata
<code>partitionMetadataLock</code>	Java's ReentrantReadWriteLock
<code>stateChangeLogger</code>	<code>StateChangeLogger</code>

`MetadataCache` uses **[MetadataCache brokerId=[brokerId]]** as the logging prefix (aka `logIdent`).

Tip	<p>Enable <code>DEBUG</code> or <code>ERROR</code> logging level for <code>kafka.server.MetadataCache</code> logger to see what happens inside.</p> <p>Add the following line to <code>config/log4j.properties</code> :</p> <pre>log4j.logger.kafka.server.MetadataCache=DEBUG</pre> <p>Refer to Logging.</p>
------------	---

getAllTopics Method

`getAllTopics(): Set[String]`

`getAllTopics` ...FIXME

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

getTopicMetadata Method

```
getTopicMetadata(  
    topics: Set[String],  
    listenerName: ListenerName,  
    errorUnavailableEndpoints: Boolean = false,  
    errorUnavailableListeners: Boolean = false): Seq[MetadataResponse.TopicMetadata]
```

getTopicMetadata ...FIXME

Note	getTopicMetadata is used when...FIXME
------	---------------------------------------

getAliveBrokers Method

```
getAliveBrokers: Seq[Broker]
```

getAliveBrokers ...FIXME

Note	getAliveBrokers is used when...FIXME
------	--------------------------------------

Updating Metadata (and Returning Deleted Partitions) — updateMetadata Method

```
updateMetadata(  
    correlationId: Int,  
    updateMetadataRequest: UpdateMetadataRequest): Seq[TopicPartition]
```

updateMetadata ...FIXME

Note	updateMetadata is used exclusively when <code>ReplicaManager</code> is requested to <code>maybeUpdateMetadataCache</code> .
------	---

getClusterMetadata Method

```
getClusterMetadata(clusterId: String, listenerName: ListenerName): Cluster
```

getClusterMetadata ...FIXME

Note	getClusterMetadata is used exclusively when <code>KafkaApis</code> is requested to handle an <code>UpdateMetadata</code> request.
------	---

OffsetConfig

OffsetConfig is...FIXME

Table 1. OffsetConfig's Properties (in alphabetical order)

Property	Default Value	Description
offsetsTopicNumPartitions	50	Number of partitions for the __consumer_offsets consumer group metadata topic

ReplicaManager

`ReplicaManager` manages log replicas using the [LogManager](#).

`ReplicaManager` is [created](#) and immediately [started](#) when `KafkaServer` is requested to start up.

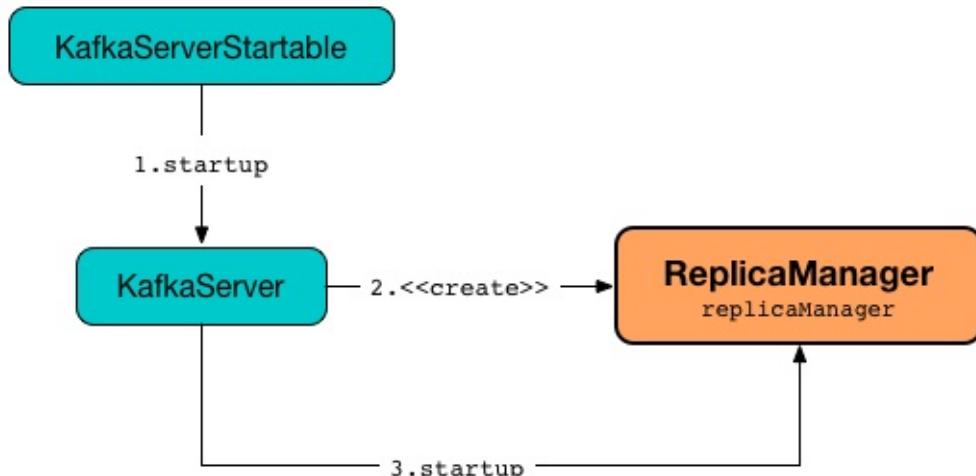


Figure 1. ReplicaManager and KafkaServer

`ReplicaManager` is given all the required services (e.g. [Metrics](#), [KafkaZkClient](#), [Scheduler](#), [LogManager](#), [QuotaManagers](#), [MetadataCache](#)) from the `KafkaServer`.

Note

`KafkaServer` creates a `ReplicaManager` for the only purpose of creating [KafkaApis](#), [GroupCoordinator](#), and [TransactionCoordinator](#).

When [started](#), `ReplicaManager` schedules [isr-expiration](#) and [isr-change-propagation](#) recurring tasks (every half of `replica.lag.time.max.ms` property and 2500 ms, respectively).

`ReplicaManager` uses the [MetadataCache](#) for the following:

- ...FIXME

Enable `TRACE` logging level for `kafka.server.ReplicaManager` logger to see what happens inside.

Add the following line to `log4j.properties`:

```
log4j.logger.kafka.server.ReplicaManager=TRACE
```

Refer to [Logging](#).

Tip

Please note that `ReplicaManager` also uses a preconfigured `state.change.logger` logger in `config/log4j.properties`:

```
log4j.appenders.stateChangeAppender=org.apache.log4j.DailyRollingFileAppender
log4j.appenders.stateChangeAppender.DatePattern='.' 'yyyy-MM-dd-HH
log4j.appenders.stateChangeAppender.File=${kafka.logs.dir}/state-change.log
log4j.appenders.stateChangeAppender.layout=org.apache.log4j.PatternLayout
log4j.appenders.stateChangeAppender.layout.ConversionPattern=[%d] %p %m (%c)%n

log4j.logger.state.change.logger=TRACE, stateChangeAppender
log4j.additivity.state.change.logger=false
```

That means that the logs of `ReplicaManager` go to `logs/state-change.log` file at `TRACE` logging level and are not added to the main logs (per `log4j.additivity` being off).

Performance Metrics

`ReplicaManager` is a [KafkaMetricsGroup](#) with the following performance metrics.

Table 1. ReplicaManager's Performance Metrics

Metric Name	Description
<code>FailedIsrUpdatesPerSec</code>	
<code>IsrExpandsPerSec</code>	
<code>IsrShrinksPerSec</code>	
<code>LeaderCount</code>	
<code>OfflineReplicaCount</code>	
<code>PartitionCount</code>	Number of allPartitions
<code>UnderMinIsrPartitionCount</code>	
<code>UnderReplicatedPartitions</code>	underReplicatedPartitionCount

The performance metrics are registered in **kafka.server:type=ReplicaManager** group.

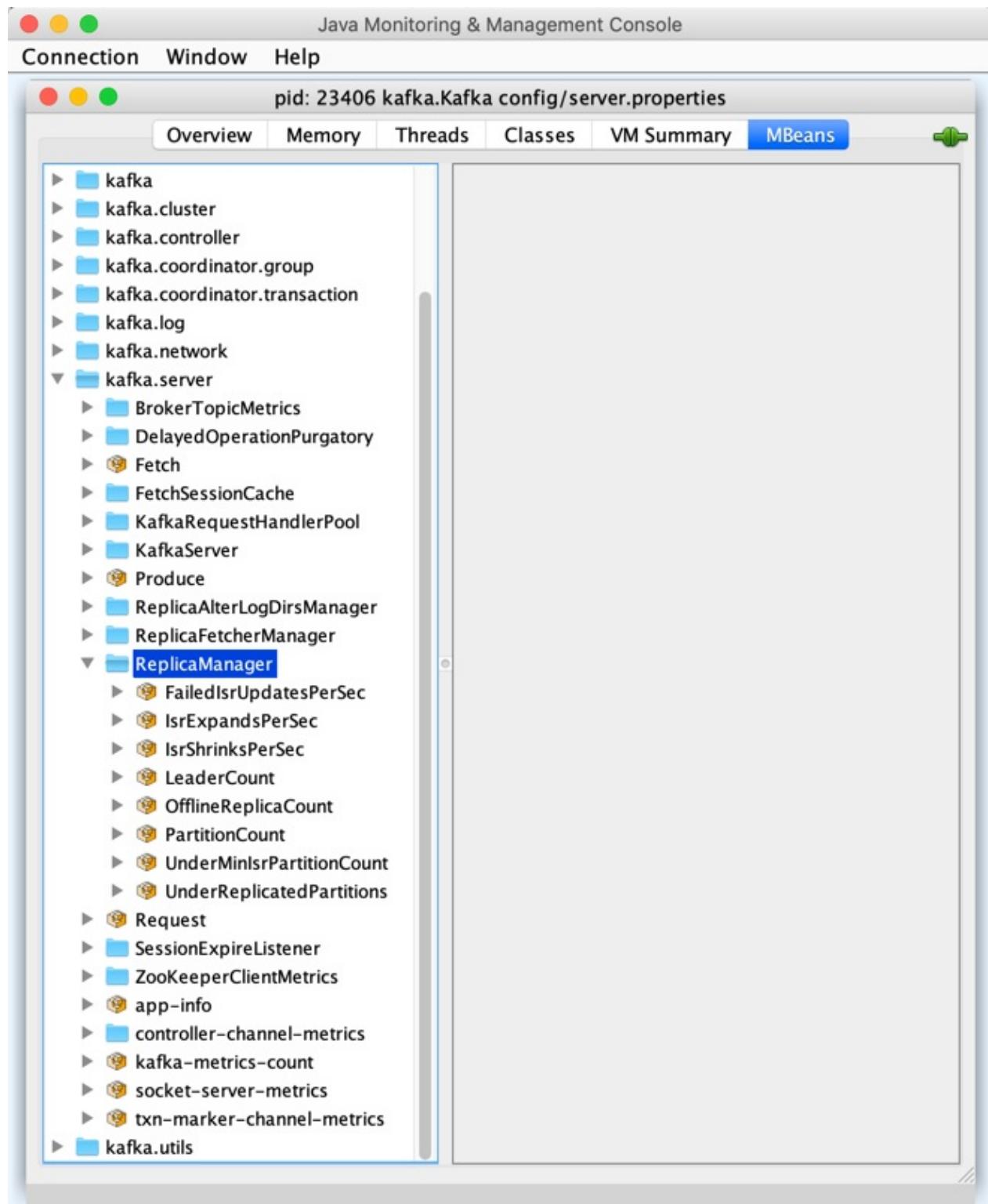


Figure 2. ReplicaManager in jconsole

Creating ReplicaFetcherManager

— `createReplicaFetcherManager` Internal Factory Method

```
createReplicaFetcherManager(  
    metrics: Metrics  
    time: Time  
    threadNamePrefix: Option[String]  
    quotaManager: ReplicationQuotaManager): ReplicaFetcherManager
```

`createReplicaFetcherManager` simply creates a [ReplicaFetcherManager](#).

Note	<code>createReplicaFetcherManager</code> is used exclusively when <code>ReplicaManager</code> is created.
------	---

shutdown Method

```
shutdown(checkpointHW: Boolean = true): Unit
```

`shutdown` ...FIXME

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

alterReplicaLogDirs Method

```
alterReplicaLogDirs(partitionDirs: Map[TopicPartition, String]): Map[TopicPartition, E  
rrors]
```

`alterReplicaLogDirs` ...FIXME

Note	<code>alterReplicaLogDirs</code> is used exclusively when <code>KafkaApis</code> is requested to handle an AlterReplicaLogDirs request.
------	---

becomeLeaderOrFollower Method

```
becomeLeaderOrFollower(  
    correlationId: Int,  
    leaderAndIsrRequest: LeaderAndIsrRequest,  
    onLeadershipChange: (Iterable[Partition], Iterable[Partition]) => Unit): LeaderAndIs  
rResponse
```

`becomeLeaderOrFollower` ...FIXME

Note	<code>becomeLeaderOrFollower</code> is used exclusively when <code>KafkaApis</code> is requested to handle a LeaderAndIsr request.
------	--

makeFollowers Internal Method

```
makeFollowers(
    controllerId: Int,
    epoch: Int,
    partitionState: Map[Partition, LeaderAndIsrRequest.PartitionState],
    correlationId: Int,
    responseMap: mutable.Map[TopicPartition, Errors]): Set[Partition]
```

`makeFollowers` ...FIXME

Note	<code>makeFollowers</code> is used exclusively when <code>ReplicaManager</code> is requested to becomeLeaderOrFollower .
------	--

recordIsrChange Method

```
recordIsrChange(topicPartition: TopicPartition): Unit
```

`recordIsrChange` adds the input `topicPartition` to `isrChangeSet` internal registry and sets `lastIsrChangeMs` to the current time.

Note	<code>recordIsrChange</code> is used exclusively when <code>Partition</code> does updatelsr
------	---

updateFollowerLogReadResults Internal Method

```
updateFollowerLogReadResults(
    replicaId: Int,
    readResults: Seq[(TopicPartition, LogReadResult)]): Seq[(TopicPartition, LogReadResult)]
```

`updateFollowerLogReadResults` ...FIXME

Note	<code>updateFollowerLogReadResults</code> is used exclusively when <code>ReplicaManager</code> is requested to fetch messages from the leader replica .
------	---

fetchMessages Method

```
fetchMessages(
    timeout: Long,
    replicaId: Int,
    fetchMinBytes: Int,
    fetchMaxBytes: Int,
    hardMaxBytesLimit: Boolean,
    fetchInfos: Seq[(TopicPartition, FetchRequest.PartitionData)],
    quota: ReplicaQuota = UnboundedQuota,
    responseCallback: Seq[(TopicPartition, FetchPartitionData)] => Unit,
    isolationLevel: IsolationLevel): Unit
```

`fetchMessages` ...FIXME

Note	<code>fetchMessages</code> is used exclusively when <code>KafkaApis</code> is requested to handle a Fetch request .
------	---

maybePropagateIsrChanges Method

```
maybePropagateIsrChanges(): Unit
```

`maybePropagateIsrChanges` ...FIXME

Note	<code>maybePropagateIsrChanges</code> is used exclusively when isr-change-propagation task is executed (every 2500 milliseconds).
------	---

Creating ReplicaManager Instance

`ReplicaManager` takes the following when created:

- [KafkaConfig](#)
- [Metrics](#)
- [Time](#)
- [KafkaZkClient](#)
- [Scheduler](#)
- [LogManager](#)
- [isShuttingDown flag](#)
- [QuotaManagers](#)
- [BrokerTopicStats](#)

- `MetadataCache`
- `LogDirFailureChannel`
- `DelayedOperationPurgatory[DelayedProduce]`
- `DelayedOperationPurgatory[DelayedFetch]`
- `DelayedOperationPurgatory[DelayedDeleteRecords]`
- Optional thread name prefix

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

Starting ReplicaManager (and Scheduling ISR-Related Tasks) — `startup` Method

```
startup(): Unit
```

`startup` requests [Scheduler](#) to [schedule the ISR-related tasks](#):

1. [isr-expiration](#)
2. [isr-change-propagation](#)

`startup` then creates a [LogDirFailureHandler](#) and requests it to [start](#).

Note	<code>startup</code> uses Scheduler that was specified when <code>ReplicaManager</code> was created.
------	--

Note	<code>startup</code> is used exclusively when <code>KafkaServer</code> starts up.
------	---

`maybeShrinkIsr` Internal Method

```
maybeShrinkIsr(): Unit
```

`maybeShrinkIsr` prints out the following TRACE message to the logs:

TRACE Evaluating ISR list of partitions to see which replicas can be removed from the ISR

`maybeShrinkIsr` requests the partitions (from [allPartitions](#) pool that are not [offline partitions](#)) to [maybeShrinkIsr](#) (with [replicaLagTimeMaxMs](#) property).

Note

`maybeShrinkIsr` is used exclusively to schedule `isr-expiration` recurring task when `ReplicaManager` starts up.

makeLeaders Internal Method

```
makeLeaders(
    controllerId: Int,
    epoch: Int,
    partitionState: Map[Partition, LeaderAndIsrRequest.PartitionState],
    correlationId: Int,
    responseMap: mutable.Map[TopicPartition, Errors]): Set[Partition]
```

`makeLeaders` ...FIXME

Note

`makeLeaders` is used exclusively when `ReplicaManager` is requested to `becomeLeaderOrFollower`.

describeLogDirs Method

```
describeLogDirs(partitions: Set[TopicPartition]): Map[String, LogDirInfo]
```

`describeLogDirs` ...FIXME

Note

`describeLogDirs` is used exclusively when `KafkaApis` is requested to handle a `DescribeLogDirs` request.

Finding Log For TopicPartition — getLog Method

```
getLog(topicPartition: TopicPartition): Option[Log]
```

`getLog` ...FIXME

Note

`getLog` is used when:

- `GroupMetadataManager` is requested to `doLoadGroupsAndOffsets`
- `TransactionStateManager` is requested to `loadTransactionMetadata`

startHighWaterMarksCheckPointThread Method

```
startHighWaterMarksCheckPointThread(): Unit
```

`startHighWaterMarksCheckPointThread` ...FIXME

Note

`startHighWaterMarksCheckPointThread` is used when...FIXME

checkpointHighWatermarks Method

```
checkpointHighWatermarks(): Unit
```

`checkpointHighWatermarks` ...FIXME

Note

`checkpointHighWatermarks` is used when...FIXME

shutdownIdleReplicaAlterLogDirsThread Method

```
shutdownIdleReplicaAlterLogDirsThread(): Unit
```

`shutdownIdleReplicaAlterLogDirsThread` ...FIXME

Note

`shutdownIdleReplicaAlterLogDirsThread` is used when...FIXME

handleLogDirFailure Method

```
handleLogDirFailure(  
  dir: String,  
  sendZkNotification: Boolean = true): Unit
```

`handleLogDirFailure` ...FIXME

Note

`handleLogDirFailure` is used exclusively when `LogDirFailureHandler` is requested to [do the work](#).

maybeUpdateMetadataCache Method

```
maybeUpdateMetadataCache(  
  correlationId: Int,  
  updateMetadataRequest: UpdateMetadataRequest) : Seq[TopicPartition]
```

`maybeUpdateMetadataCache ...FIXME`

Note

`maybeUpdateMetadataCache` is used exclusively when `KafkaApis` is requested to handle an `UpdateMetadata` request.

Appending Records — `appendRecords` Method

```
appendRecords(
    timeout: Long,
    requiredAcks: Short,
    internalTopicsAllowed: Boolean,
    isFromClient: Boolean,
    entriesPerPartition: Map[TopicPartition, MemoryRecords],
    responseCallback: Map[TopicPartition, PartitionResponse] => Unit,
    delayedProduceLock: Option[Lock] = None,
    recordConversionStatsCallback: Map[TopicPartition, RecordConversionStats] => Unit =
  _ => (): Unit
```

`appendRecords ...FIXME`

Note

`appendRecords` is used when:

- `GroupMetadataManager` is requested to request the `ReplicaManager` to append records
- `TransactionStateManager` is requested to `enableTransactionalIdExpiration` and `appendTransactionToLog`
- `KafkaApis` is requested to handle `Produce` and `WriteTxnMarkers` requests

Validating requiredAcks — `isValidRequiredAcks` Internal Method

```
isValidRequiredAcks(requiredAcks: Short): Boolean
```

`isValidRequiredAcks` is positive (`true`) when the given `requiredAcks` is one of the following:

- `-1`
- `1`
- `0`

Otherwise, `isValidRequiredAcks` is negative (`false`).

Note

`isValidRequiredAcks` is used exclusively when `ReplicaManager` is requested to [appendRecords](#).

appendToLocalLog Internal Method

```
appendToLocalLog(
    internalTopicsAllowed: Boolean,
    isFromClient: Boolean,
    entriesPerPartition: Map[TopicPartition, MemoryRecords],
    requiredAcks: Short): Map[TopicPartition, LogAppendResult]
```

`appendToLocalLog` processes (*maps over*) the given `Map[TopicPartition, MemoryRecords]` (`entriesPerPartition`), so that the leader partition (of every `TopicPartition`) is requested to [appendRecordsToLeader](#).

Internally, `appendToLocalLog` prints out the following TRACE message to the logs:

```
Append [[entriesPerPartition]] to local log
```

For every tuple in the given `entriesPerPartition` (`Map[TopicPartition, MemoryRecords]`), `appendToLocalLog` does the following steps:

1. Requests the [BrokerTopicStats](#) to mark the occurrence of an event for the `totalProduceRequestRate` for the topic (of the `TopicPartition`) in the [topicStats](#) and for all topics
2. Gets the partition (or throws an exception) (with `expectLeader` flag enabled)
3. Requests the `Partition` to [appendRecordsToLeader](#) (with the `MemoryRecords`, the `isFromClient` flag, and the `requiredAcks` bit map)
4. Requests the [BrokerTopicStats](#) to mark the `sizeInBytes` of the `MemoryRecords` for the `bytesInRate` for the topic (of the `TopicPartition`) in the [bytesInRate](#) and for all topics
5. Requests the [BrokerTopicStats](#) to mark the number of messages appended for the `messagesInRate` for the topic (of the `TopicPartition`) in the [bytesInRate](#) and for all topics
6. Prints out the following TRACE message to the logs:

```
[sizeInBytes] written to log [topicPartition] beginning at offset [firstOffset] and ending at offset [lastOffset]
```

In case `Topic.isInternal(topicPartition.topic) && !internalTopicsAllowed`,
`appendToLocalLog ...FIXME`

In case of exceptions, `appendToLocalLog ...FIXME`

Note

`appendToLocalLog` is used exclusively when `ReplicaManager` is requested to [append records](#).

Getting Partition Or Throwing Exception

— `getPartitionOrException` Method

```
getPartitionOrException(  
    topicPartition: TopicPartition,  
    expectLeader: Boolean): Partition
```

`getPartitionOrException` [gets the partition](#) if available or throws one of the following exceptions:

- `KafkaStorageException` when the partition is offline

`Partition [topicPartition] is in an offline log directory`

- `NotLeaderForPartitionException`

`Broker [localBrokerId] is not a replica of [topicPartition]`

- `ReplicaNotAvailableException`

`Partition [topicPartition] is not available`

- `UnknownTopicOrPartitionException`

`Partition [topicPartition] doesn't exist`

Note

`getPartitionOrException` is used when...FIXME

Getting Partition by TopicPartition (If Available)

— `getPartition` Method

```
getPartition(topicPartition: TopicPartition): Option[Partition]
```

`getPartition` gets the [partition](#) for the given `TopicPartition`.

Note

`getPartition` is used when:

- `DelayedDeleteRecords` is requested to `tryComplete`
- `DelayedProduce` is requested to `tryComplete`
- `ReplicaAlterLogDirsThread` is requested to `processPartitionData`
- `ReplicaFetcherThread` is requested to `processPartitionData`, `truncate`, and `truncateFullyAndStartAt`
- `ReplicaManager` is requested to `nonOfflinePartition`, `getPartitionOrException`, `alterReplicaLogDirs`, `appendToLocalLog`, `becomeLeaderOrFollower`, and `lastOffsetForLeaderEpoch`

stopReplica Method

```
stopReplica(  
    topicPartition: TopicPartition,  
    deletePartition: Boolean): Unit
```

`stopReplica` ...FIXME

Note

`stopReplica` is used exclusively when `ReplicaManager` is requested to [stopReplicas](#).

underReplicatedPartitionCount Method

```
underReplicatedPartitionCount: Int
```

`underReplicatedPartitionCount` ...FIXME

Note

`underReplicatedPartitionCount` is used exclusively for the [UnderReplicatedPartitions](#) performance metric.

leaderPartitionsIterator Internal Method

```
leaderPartitionsIterator: Iterator[Partition]
```

`leaderPartitionsIterator` ...FIXME

Note

`leaderPartitionsIterator` is used exclusively for the performance metrics: [LeaderCount](#), [UnderMinIsrPartitionCount](#), and [UnderReplicatedPartitions](#) (indirectly using [underReplicatedPartitionCount](#)).

nonOfflinePartitionsIterator Internal Method

```
nonOfflinePartitionsIterator: Iterator[Partition]
```

nonOfflinePartitionsIterator ...FIXME

Note

`nonOfflinePartitionsIterator` is used when `ReplicaManager` is requested to [leaderPartitionsIterator](#), [checkpointHighWatermarks](#), and [handleLogDirFailure](#).

getOrCreatePartition Method

```
getOrCreatePartition(topicPartition: TopicPartition): Partition
```

getOrCreatePartition ...FIXME

Note

`getOrCreatePartition` is used exclusively when `ReplicaManager` is requested to [becomeLeaderOrFollower](#).

offlinePartitionsIterator Internal Method

```
offlinePartitionsIterator: Iterator[Partition]
```

offlinePartitionsIterator ...FIXME

Note

`offlinePartitionsIterator` is used when...FIXME

markPartitionOffline Method

```
markPartitionOffline(tp: TopicPartition): Unit
```

markPartitionOffline ...FIXME

Note

`markPartitionOffline` is used when...FIXME

lastOffsetForLeaderEpoch Method

```
lastOffsetForLeaderEpoch(  
    requestedEpochInfo: Map[TopicPartition, OffsetsForLeaderEpochRequest.PartitionData]  
) : Map[TopicPartition, EpochEndOffset]
```

lastOffsetForLeaderEpoch ...FIXME

Note	lastOffsetForLeaderEpoch is used when...FIXME
------	---

nonOfflinePartition Method

```
nonOfflinePartition(topicPartition: TopicPartition): Option[Partition]
```

nonOfflinePartition ...FIXME

Note	nonOfflinePartition is used when...FIXME
------	--

deleteRecords Method

```
deleteRecords(  
    timeout: Long,  
    offsetPerPartition: Map[TopicPartition, Long],  
    responseCallback: Map[TopicPartition, DeleteRecordsResponse.PartitionResponse] => Unit  
)
```

deleteRecords ...FIXME

Note	deleteRecords is used when...FIXME
------	------------------------------------

fetchOffsetForTimestamp Method

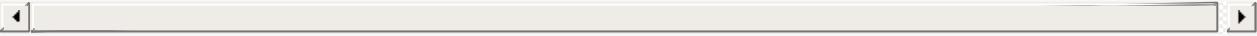
```
fetchOffsetForTimestamp(  
    topicPartition: TopicPartition,  
    timestamp: Long,  
    isolationLevel: Option[IsolationLevel],  
    currentLeaderEpoch: Optional[Integer],  
    fetchOnlyFromLeader: Boolean): TimestampOffset
```

fetchOffsetForTimestamp ...FIXME

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

stopReplicas Method

```
stopReplicas(  
    stopReplicaRequest: StopReplicaRequest): (mutable.Map[TopicPartition, Errors], Errors  
)
```



`stopReplicas` ...FIXME

Note	<code>stopReplicas</code> is used exclusively when <code>KafkaApis</code> is requested to handle a StopReplica request .
------	--

getMagic Method

```
getMagic(topicPartition: TopicPartition): Option[Byte]
```

`getMagic` ...FIXME

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

localReplicaOrException Method

```
localReplicaOrException(topicPartition: TopicPartition): Replica
```

`localReplicaOrException` finds the partition (or throws an exception) for the given `TopicPartition` (and `expectLeader` flag off) and requests the `Partition` to get the local partition replica (or throw an exception).

Note	A partition replica is local when the replica ID is exactly the local broker ID.
------	--

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

shouldLeaderThrottle Method

```
shouldLeaderThrottle(  
    quota: ReplicaQuota,  
    topicPartition: TopicPartition,  
    replicaId: Int): Boolean
```

shouldLeaderThrottle ...FIXME

Note	shouldLeaderThrottle is used when...FIXME
------	---

readFromLocalLog Method

```
readFromLocalLog(  
    replicaId: Int,  
    fetchOnlyFromLeader: Boolean,  
    fetchIsolation: FetchIsolation,  
    fetchMaxBytes: Int,  
    hardMaxBytesLimit: Boolean,  
    readPartitionInfo: Seq[(TopicPartition, PartitionData)],  
    quota: ReplicaQuota): Seq[(TopicPartition, LogReadResult)]
```

readFromLocalLog ...FIXME

Note	readFromLocalLog is used when: <ul style="list-style-type: none">DelayedFetch is requested to onCompleteReplicaManager is requested to fetchMessages (when KafkaApis is requested to handle a Fetch request)
------	---

Internal Properties

Name	Description
replicaFetcherManager	<p>ReplicaFetcherManager</p> <p>Created immediately with <code>ReplicaManager</code></p> <p>Used when:</p> <ul style="list-style-type: none"> • <code>ReplicaManager</code> is requested to stopReplicas, becomeLeaderOrFollower, makeLeaders, makeFollowers, handleLogDirFailure, shutdown • <code>DynamicThreadPool</code> is requested to reconfigure (resize) the thread pool • <code>KafkaApis</code> is requested to handle a StopReplica request
allPartitions	<p>Partitions by TopicPartition</p> <p>Available as PartitionCount performance metric</p> <ul style="list-style-type: none"> • <code>TopicPartition</code> removed in stopReplica (and marked as the OfflinePartition) • <code>TopicPartition</code> added in getOrCreatePartition and markPartitionOffline <p>Used in getPartition, nonOfflinePartitionsIterator, offlinePartitionsIterator, becomeLeaderOrFollower, maybeShrinkIsr, handleLogDirFailure</p>
highWatermarkCheckpoints	<code>OffsetCheckpointFiles</code> per live log data directory
stateChangeLogger	<code>StateChangeLogger</code> with the broker ID and <code>inControllerContext</code> flag off (that sets <code>Broker</code> log prefix)
isrChangeSet	Collection of <code>TopicPartition</code> that...FIXME
lastIsrChangeMs	Time when isrChangeSet has a new <code>TopicPartition</code> added.
logDirFailureHandler	LogDirFailureHandler
offlinePartition	

ReplicaFetcherManager

`ReplicaFetcherManager` is a `AbstractFetcherManager` of `ReplicaFetcherThreads` that...
FIXME (describe properties)

`ReplicaFetcherManager` is `created` exclusively when `ReplicaManager` is requested to `create one` (which is when `ReplicaManager` is `created`).

createFetcherThread Method

```
createFetcherThread(  
    fetcherId: Int,  
    sourceBroker: BrokerEndPoint): ReplicaFetcherThread
```

Note	<code>createFetcherThread</code> is part of the <code>AbstractFetcherManager Contract</code> to... FIXME.
------	--

`createFetcherThread` ...FIXME

Creating ReplicaFetcherManager Instance

`ReplicaFetcherManager` takes the following when created:

- `KafkaConfig`
- `ReplicaManager`
- `Metrics`
- `Time`
- Optional thread name prefix (undefined by default)
- `ReplicationQuotaManager`

`ReplicaFetcherManager` initializes the internal registries and counters.

ReplicaFetcherThread

ReplicaFetcherThread is an uninterruptible AbstractFetcherThread that...FIXME

ReplicaFetcherThread is used (*managed*) exclusively by ReplicaFetcherManager.

ReplicaFetcherThread uses replica.fetch.backoff.ms configuration property for the fetchBackOffMs.

ReplicaFetcherThread is created exclusively when ReplicaFetcherManager is requested to create a fetcher thread (when...FIXME).

Creating ReplicaFetcherThread Instance

ReplicaFetcherThread takes the following when created:

- Name
- Fetcher ID
- Source BrokerEndPoint
- KafkaConfig
- ReplicaManager
- Metrics
- Time
- ReplicaQuota
- Optional BlockingSend (default: None)

ReplicaFetcherThread initializes the internal registries and counters.

earliestOrLatestOffset Internal Method

```
earliestOrLatestOffset(topicPartition: TopicPartition, earliestOrLatest: Long): Long
```

earliestOrLatestOffset ...FIXME

Note	earliestOrLatestOffset is used when...FIXME
------	---

fetchEpochsFromLeader Method

```
fetchEpochsFromLeader(partitions: Map[TopicPartition, Int]): Map[TopicPartition, EpochEndOffset]
```

Note

`fetchEpochsFromLeader` is a part of [AbstractFetcherThread Contract](#).

`fetchEpochsFromLeader` ...FIXME

processPartitionData Method

```
processPartitionData(  
    topicPartition: TopicPartition,  
    fetchOffset: Long,  
    partitionData: FetchData): Option[LogAppendInfo]
```

Note

`processPartitionData` is part of the [AbstractFetcherThread Contract](#) to...
FIXME.

`processPartitionData` ...FIXME

truncate Method

```
truncate(tp: TopicPartition, offsetTruncationState: OffsetTruncationState): Unit
```

Note

`truncate` is a part of [AbstractFetcherThread Contract](#).

`truncate` ...FIXME

truncateFullyAndStartAt Method

```
truncateFullyAndStartAt(topicPartition: TopicPartition, offset: Long): Unit
```

Note

`truncateFullyAndStartAt` is a part of [AbstractFetcherThread Contract](#).

`truncateFullyAndStartAt` ...FIXME

latestEpoch Method

```
latestEpoch(topicPartition: TopicPartition): Option[Int]
```

Note

`latestEpoch` is a part of [AbstractFetcherThread Contract](#) to get the latest (current) leader epoch for the given `TopicPartition`.

`latestEpoch` requests the [ReplicaManager](#) for the local partition replica (or throw an exception) for the given `TopicPartition` and requests the replica for the latest (current) leader epoch.

isOffsetForLeaderEpochSupported Method

```
isOffsetForLeaderEpochSupported: Boolean
```

Note

`isOffsetForLeaderEpochSupported` is part of the [AbstractFetcherThread Contract](#) to control whether a `OffsetsForLeaderEpochRequest` is supported (that was added in Kafka 0.11.0.0).

`isOffsetForLeaderEpochSupported` simply returns the [brokerSupportsLeaderEpochRequest](#) flag.

buildFetch Method

```
buildFetch(  
    partitionMap: Map[TopicPartition, PartitionFetchState]  
) : ResultWithPartitions[Option[FetchRequest.Builder]]
```

Note

`buildFetch` is part of the [AbstractFetcherThread Contract](#) to...FIXME

`buildFetch` ...FIXME

Internal Properties

Name	Description
<code>brokerSupportsLeaderEpochRequest</code>	<p>Flag that says whether the <code>inter.broker.protocol.version</code> is <code>0.11.0-IV2</code> or above</p> <p>Used exclusively in <code>isOffsetForLeaderEpochSupported</code></p>

ReplicaAlterLogDirsManager

ReplicaAlterLogDirsManager is...FIXME

createFetcherThread Method

```
createFetcherThread(  
    fetcherId: Int,  
    sourceBroker: BrokerEndPoint): ReplicaAlterLogDirsThread
```

Note

createFetcherThread is part of the [AbstractFetcherManager Contract](#) to...
FIXME.

createFetcherThread ...FIXME

ReplicaAlterLogDirsThread

`ReplicaAlterLogDirsThread` is an uninterruptible `AbstractFetcherThread` that...FIXME

`ReplicaAlterLogDirsThread` uses `replica.fetch.backoff.ms` configuration property for the `fetchBackOffMs`.

`ReplicaAlterLogDirsThread` is `created` exclusively when `ReplicaAlterLogDirsManager` is requested to `create a fetcher thread` (when...FIXME).

`ReplicaAlterLogDirsThread` has `isOffsetForLeaderEpochSupported` flag always enabled (`true`).

fetchFromLeader Method

```
fetchFromLeader(  
    fetchRequest: FetchRequest.Builder): Seq[(TopicPartition, FetchData)]
```

Note	<code>fetchFromLeader</code> is part of the <code>AbstractFetcherThread Contract</code> to...FIXME.
------	---

`fetchFromLeader` ...FIXME

processPartitionData Method

```
processPartitionData(  
    topicPartition: TopicPartition,  
    fetchOffset: Long,  
    partitionData: PartitionData[Records]): Option[LogAppendInfo]
```

Note	<code>processPartitionData</code> is part of the <code>AbstractFetcherThread Contract</code> to... FIXME.
------	--

`processPartitionData` ...FIXME

truncateFullyAndStartAt Method

```
truncateFullyAndStartAt(  
    topicPartition: TopicPartition,  
    offset: Long): Unit
```

Note

`truncateFullyAndStartAt` is part of the [AbstractFetcherThread Contract](#) to...
FIXME.

`truncateFullyAndStartAt` ...FIXME

truncate Method

```
truncate(  
    topicPartition: TopicPartition,  
    truncationState: OffsetTruncationState): Unit
```

Note

`truncate` is part of the [AbstractFetcherThread Contract](#) to...FIXME.

`truncate` ...FIXME

buildFetch Method

```
buildFetch(  
    partitionMap: Map[TopicPartition, PartitionFetchState]  
) : ResultWithPartitions[Option[FetchRequest.Builder]]
```

Note

`buildFetch` is part of the [AbstractFetcherThread Contract](#) to...FIXME

`buildFetch` ...FIXME

Creating ReplicaAlterLogDirsThread Instance

`ReplicaAlterLogDirsThread` takes the following to be created:

- Thread Name
- Source `BrokerEndPoint`
- [KafkaConfig](#)
- [ReplicaManager](#)
- [ReplicationQuotaManager](#)
- [BrokerTopicStats](#)

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

AbstractFetcherManager

`AbstractFetcherManager` is the [abstraction](#) of fetcher thread managers that can [create a fetcher thread](#) (for [ReplicaManager](#)).

Note

`AbstractFetcherManager` is a Java abstract class and cannot be [created](#) directly. It is created indirectly for the [concrete AbstractFetcherManagers](#).

Table 1. AbstractFetcherManager Contract (Abstract Methods)

Method	Description
<code>createFetcherThread</code>	<pre>createFetcherThread(fetcherId: Int, sourceBroker: BrokerEndPoint)</pre> <p>Used when <code>AbstractFetcherManager</code> is requested to addFetcherForPartitions</p>

Table 2. AbstractFetcherManagers

AbstractFetcherManager	Description
ReplicaAlterLogDirsManager	
ReplicaFetcherManager	

Table 3. AbstractFetcherManager's Internal Properties (e.g. Registries, Counters and Flags)

Name	Description
<code>numFetchersPerBroker</code>	<p>Number of fetcher threads per broker</p> <p>Initialized with the given numFetchers</p> <p>Resized in resizeThreadPool</p> <p>Used exclusively in getFetcherId</p>

addFetcherForPartitions Method

```
addFetcherForPartitions(  
    partitionAndOffsets: Map[TopicPartition, InitialFetchState]): Unit
```

`addFetcherForPartitions` ...FIXME

Note

- `addFetcherForPartitions` is used when:
- `AbstractFetcherManager` is requested to `resizeThreadPool` (when `DynamicThreadPool` is requested to `reconfigure`)
 - `ReplicaManager` is requested to `alterReplicaLogDirs`, `becomeLeaderOrFollower` and `makeFollowers`

resizeThreadPool Method

```
resizeThreadPool(newSize: Int): Unit
```

`resizeThreadPool` prints out the following INFO message to the logs:

```
Resizing fetcher thread pool size from [currentSize] to [newSize]
```

`resizeThreadPool ...FIXME`

Note

`resizeThreadPool` is used exclusively when `DynamicThreadPool` is requested to `reconfigure`.

Hashing Topic and Partition — getFetcherId Method

```
getFetcherId(topicPartition: TopicPartition): Int
```

`getFetcherId` calculates a hash of the topic and partition (from the given `TopicPartition`) modulo (%) the `numFetchersPerBroker`.

Note

`getFetcherId` is used when `AbstractFetcherManager` is requested to `getFetcher`, `markPartitionsForTruncation` and `addFetcherForPartitions`.

getFetcher Method

```
getFetcher(topicPartition: TopicPartition): Option[T]
```

`getFetcher ...FIXME`

Note

`getFetcher` is used when...FIXME

markPartitionsForTruncation Method

```
markPartitionsForTruncation(  
    brokerId: Int,  
    topicPartition: TopicPartition,  
    truncationOffset: Long): Unit
```

markPartitionsForTruncation ...FIXME

Note

markPartitionsForTruncation is used when...FIXME

removeFetcherForPartitions Method

```
removeFetcherForPartitions(partitions: Set[TopicPartition]): Unit
```

removeFetcherForPartitions ...FIXME

Note

removeFetcherForPartitions is used when:

- AbstractFetcherManager is requested to [resizeThreadPool](#)
- ReplicaManager is requested to [stopReplicas](#), [alterReplicaLogDirs](#), [makeLeaders](#), [makeFollowers](#), and [handleLogDirFailure](#)

AbstractFetcherThread

`AbstractFetcherThread` is the extension of the `ShutdownableThread` contract for fetcher threads that `maybeTruncate` followed by `maybeFetch` repeatedly (while doing the work) until shut down.

Note

`AbstractFetcherThread` is a Java abstract class and cannot be created directly. It is created indirectly for the concrete `AbstractFetcherThreads`.

Table 1. AbstractFetcherThread Contract (Abstract Methods)

Method	Description
<code>buildFetch</code>	<pre>buildFetch(partitionMap: Map[TopicPartition, PartitionFetch]): ResultWithPartitions[Option[FetchRequest.Builder]]</pre> <p>Used exclusively when <code>AbstractFetcherThread</code> is req to <code>maybeFetch</code>.</p>
<code>endOffsetForEpoch</code>	<pre>endOffsetForEpoch(topicPartition: TopicPartition, epoch: Int): Option[OffsetAndEpoch]</pre> <p>Used exclusively when <code>AbstractFetcherThread</code> is req to <code>getOffsetTruncationState</code>.</p>
<code>fetchEarliestOffsetFromLeader</code>	<pre>fetchEarliestOffsetFromLeader(topicPartition: TopicPartition, currentLeaderEpoch: Int): Long</pre> <p>Used exclusively when <code>AbstractFetcherThread</code> is req to <code>fetchOffsetAndTruncate</code>.</p>
<code>fetchEpochEndOffsets</code>	<pre>fetchEpochEndOffsets(partitions: Map[TopicPartition, EpochData]): Map[TopicPartition, EpochEndOffset]</pre> <p>Used exclusively when <code>AbstractFetcherThread</code> is req to <code>truncateToEpochEndOffsets</code>.</p>
<code>fetchFromLeader</code>	<pre>fetchFromLeader(fetchRequest: FetchRequest.Builder): Seq[(TopicPartition, FetchData)]</pre>

	<p>Used exclusively when <code>AbstractFetcherThread</code> is requested to processFetchRequest.</p>
<code>fetchLatestOffsetFromLeader</code>	<pre>fetchLatestOffsetFromLeader(topicPartition: TopicPartition, currentLeaderEpoch: Int): Long</pre> <p>Used exclusively when <code>AbstractFetcherThread</code> is requested to fetchOffsetAndTruncate.</p>
<code>isOffsetForLeaderEpochSupported</code>	<pre>isOffsetForLeaderEpochSupported: Boolean</pre> <p>Controls whether an <code>offsetsForLeaderEpochRequest</code> is supported (that was added in Kafka 0.11.0.0)</p> <p>Used exclusively when <code>AbstractFetcherThread</code> is requested to fetchTruncatingPartitions.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <p>Tip Consult KIP-101: Alter Replication Protocol to use Leader Epoch rather than High Watermark for Truncation.</p> </div>
<code>latestEpoch</code>	<pre>latestEpoch(topicPartition: TopicPartition): Option[Long]</pre> <p>Gets the latest (current) leader epoch for the given <code>TopicPartition</code></p> <p>Used exclusively when <code>AbstractFetcherThread</code> is requested to fetchTruncatingPartitions.</p>
<code>logEndOffset</code>	<pre>logEndOffset(topicPartition: TopicPartition): Long</pre> <p>Used when <code>AbstractFetcherThread</code> is requested to getOffsetTruncationState and fetchOffsetAndTruncate.</p>
<code>processPartitionData</code>	<pre>processPartitionData(topicPartition: TopicPartition, fetchOffset: Long, partitionData: FetchData): Option[LogAppendInfo]</pre> <p>Used exclusively when <code>AbstractFetcherThread</code> is requested to processFetchRequest.</p>
	<pre>truncate(topicPartition: TopicPartition, truncationState: OffsetTruncationState): Unit</pre>

<code>truncate</code>	Used when <code>AbstractFetcherThread</code> is requested to <code>truncateToHighWatermark</code> , <code>maybeTruncateToEpochEndOffsets</code> , and <code>fetchOffsetAndTruncate</code> .
<code>truncateFullyAndStartAt</code>	<pre>truncateFullyAndStartAt(topicPartition: TopicPartition, offset: Long): Unit</pre> <p>Used exclusively when <code>AbstractFetcherThread</code> is requested to <code>fetchOffsetAndTruncate</code>.</p>

Table 2. AbstractFetcherThreads

AbstractFetcherThread	Description
<code>ReplicaAlterLogDirsThread</code>	
<code>ReplicaFetcherThread</code>	

`AbstractFetcherThread` registers [performance metrics](#).

Table 3. AbstractFetcherThread's Performance Metrics

Metric Name	Description
<code>BytesPerSec</code>	
<code>RequestsPerSec</code>	

The [performance metrics](#) are registered in `kafka.server:type=FetcherStats` group.

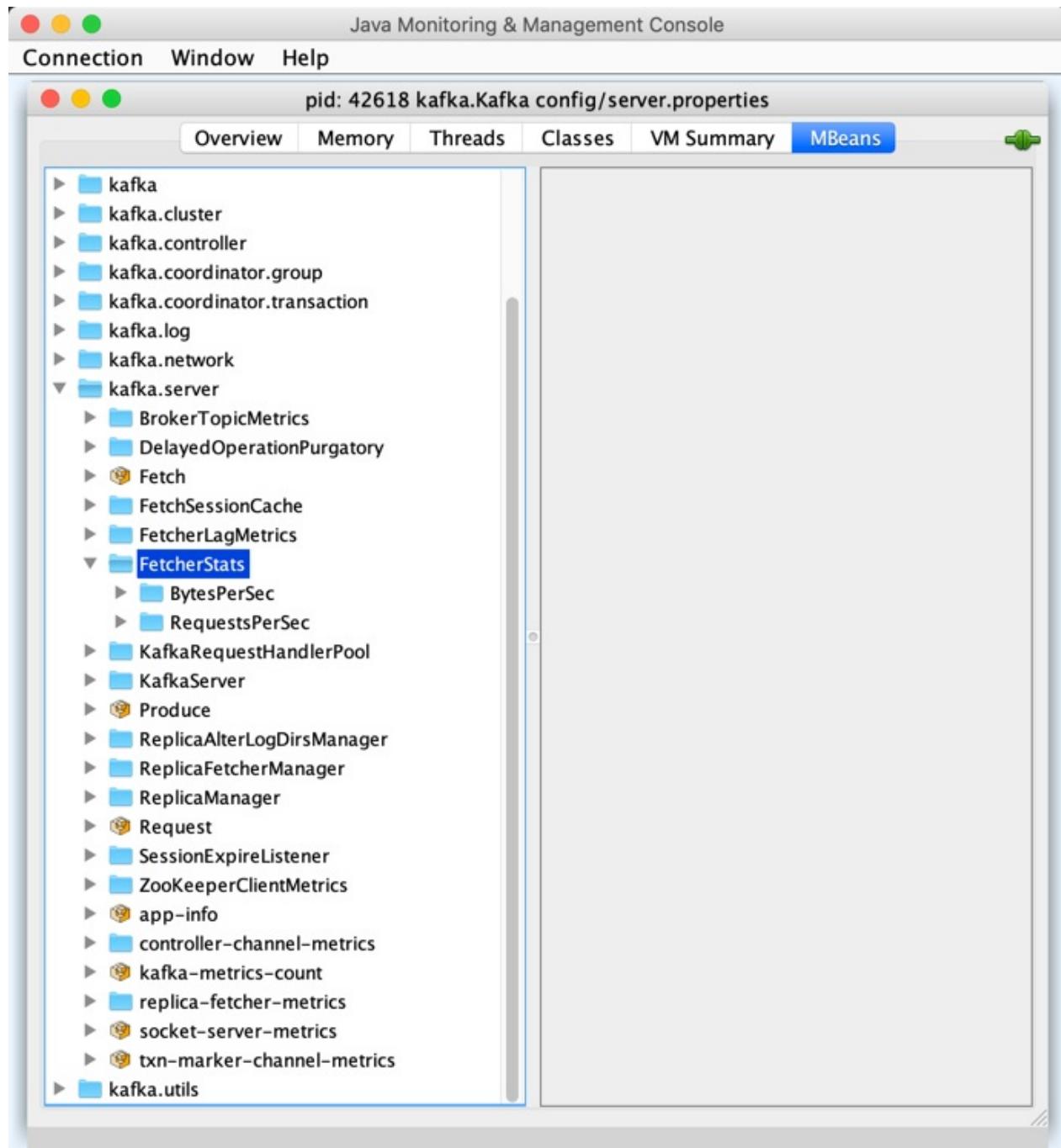


Figure 1. AbstractFetcherThread in jconsole

Table 4. AbstractFetcherThread's Internal Properties (e.g. Registries, Counters and Flags)

Name	Description
fetcherStats	KafkaMetricsGroup (with two performance metrics)
partitionStates	<p>TopicPartitions with their PartitionFetchState (PartitionStates[PartitionFetchState])</p> <ul style="list-style-type: none"> • A TopicPartition is removed in onPartitionFenced and removePartitions • PartitionFetchState updated in processFetchRequest, markPartitionsForTruncation, addPartitions, updateFetchOffsetAndMaybeMarkTruncationComplete, handleOutOfRangeError, delayPartitions <p>Used when...FIXME</p>

Creating AbstractFetcherThread Instance

`AbstractFetcherThread` takes the following to be created:

- Thread Name
- Client ID
- `BrokerEndPoint`
- `fetchBackOffMs` (default: `0`)
- `isInterruptible` flag (default: `true`)

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

dowork Method

```
dowork(): Unit
```

Note	<code>dowork</code> is part of the ShutdownableThread Contract to do the work.
------	--

`dowork` simply [maybeTruncate](#) followed by [maybeFetch](#).

maybeTruncate Internal Method

```
maybeTruncate(): Unit
```

`maybeTruncate` [fetchTruncatingPartitions](#) to find the partitions with and without epochs.

For partitions with epochs, `maybeTruncate` [truncateToEpochEndOffsets](#).

For partitions without epochs, `maybeTruncate` [truncateToHighWatermark](#).

Note

`maybeTruncate` is used exclusively when `AbstractFetcherThread` is requested to do the work.

maybeFetch Internal Method

`maybeFetch(): Unit`

`maybeFetch` [buildFetch](#) (with the [partitionStates](#)).

Note

`buildFetch` is implementation-specific.

`maybeFetch` then [handlePartitionsWithErrors](#) with partitions that [buildFetch](#) could not handle.

In the end, `maybeFetch` [processes](#) the `FetchRequest` (if created).

`maybeFetch` blocks the thread (*waits*) until the [fetchBackOffMs](#) elapses when the `FetchRequest` was not created (when [buildFetch](#)). `maybeFetch` prints out the following TRACE message to the logs:

```
There are no active partitions. Back off for [fetchBackOffMs] ms before sending a fetch request
```

Note

`maybeFetch` is used exclusively when `AbstractFetcherThread` is requested to do the work.

fetchTruncatingPartitions Internal Method

`fetchTruncatingPartitions(): (Map[TopicPartition, EpochData], Set[TopicPartition])`

`fetchTruncatingPartitions` finds the `TopicPartitions` (in the [partitionStates](#) registry) that are in `Truncating` state (and are not delayed) and splits them into two groups: with and without epochs.

Internally, for every truncating `TopicPartition`, `fetchTruncatingPartitions` gets the latest epoch.

If the latest epoch is available and `isOffsetForLeaderEpochSupported`, `fetchTruncatingPartitions` registers the `TopicPartition` as "with epoch" whereas the others as "without epoch".

Note	Both actions (getting the latest epoch and <code>isOffsetForLeaderEpochSupported</code>) are implementation-specific.
------	--

Note	<code>fetchTruncatingPartitions</code> is used exclusively when <code>AbstractFetcherThread</code> is requested to <code>maybeTruncate</code> .
------	---

onPartitionFenced Internal Method

```
onPartitionFenced(tp: TopicPartition): Unit
```

```
onPartitionFenced ...FIXME
```

Note	<code>onPartitionFenced</code> is used when <code>AbstractFetcherThread</code> is requested to <code>maybeTruncateToEpochEndOffsets</code> , <code>processFetchRequest</code> , and <code>handleOutOfRangeError</code> .
------	--

maybeTruncateToEpochEndOffsets Internal Method

```
maybeTruncateToEpochEndOffsets(
  fetchedEpochs: Map[TopicPartition, EpochEndOffset]
): ResultWithPartitions[Map[TopicPartition, OffsetTruncationState]]
```

```
maybeTruncateToEpochEndOffsets ...FIXME
```

Note	<code>maybeTruncateToEpochEndOffsets</code> is used exclusively when <code>AbstractFetcherThread</code> is requested to <code>truncateToEpochEndOffsets</code> .
------	--

processFetchRequest Internal Method

```
processFetchRequest(
  fetchStates: Map[TopicPartition, PartitionFetchState],
  fetchRequest: FetchRequest.Builder): Unit
```

`processFetchRequest` prints out the following TRACE message to the logs:

```
Sending fetch request [fetchRequest]
```

`processFetchRequest` then [fetchFromLeader](#).

Note	fetchFromLeader is implementation-specific.
------	---

`processFetchRequest` requests the [FetcherStats](#) to...FIXME

Note	<code>processFetchRequest</code> is used exclusively when <code>AbstractFetcherThread</code> is requested to maybeFetch .
------	---

handleOutOfRangeError Internal Method

```
handleOutOfRangeError(  
    topicPartition: TopicPartition,  
    fetchState: PartitionFetchState): Boolean
```

`handleOutOfRangeError` ...FIXME

Note	<code>handleOutOfRangeError</code> is used exclusively when <code>AbstractFetcherThread</code> is requested to processFetchRequest .
------	--

markPartitionsForTruncation Method

```
markPartitionsForTruncation(  
    topicPartition: TopicPartition,  
    truncationOffset: Long): Unit  
markPartitionsForTruncation(  
    brokerId: Int,  
    topicPartition: TopicPartition,  
    truncationOffset: Long): Unit
```

`markPartitionsForTruncation` ...FIXME

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

addPartitions Method

```
addPartitions(initialFetchStates: Map[TopicPartition, OffsetAndEpoch])
```

`addPartitions` ...FIXME

Note	<code>addPartitions</code> is used exclusively when <code>AbstractFetcherManager</code> is requested to addFetcherForPartitions .
------	---

updateFetchOffsetAndMaybeMarkTruncationComplete Internal Method

```
updateFetchOffsetAndMaybeMarkTruncationComplete(  
    fetchOffsets: Map[TopicPartition, OffsetTruncationState]): Unit
```

updateFetchOffsetAndMaybeMarkTruncationComplete ...FIXME

Note	updateFetchOffsetAndMaybeMarkTruncationComplete is used when AbstractFetcherThread is requested to truncateToEpochEndOffsets and truncateToHighWatermark .
------	--

truncateToEpochEndOffsets Internal Method

```
truncateToEpochEndOffsets(  
    latestEpochsForPartitions: Map[TopicPartition, EpochData]): Unit
```

truncateToEpochEndOffsets ...FIXME

Note	truncateToEpochEndOffsets is used exclusively when AbstractFetcherThread is requested to maybeTruncate .
------	--

truncateToHighWatermark Internal Method

```
truncateToHighWatermark(partitions: Set[TopicPartition]): Unit
```

truncateToHighWatermark ...FIXME

Note	truncateToHighWatermark is used exclusively when AbstractFetcherThread is requested to maybeTruncate .
------	--

partitionsAndOffsets Method

```
partitionsAndOffsets: Map[TopicPartition, InitialFetchState]
```

partitionsAndOffsets ...FIXME

Note	partitionsAndOffsets is used exclusively when AbstractFetcherManager is requested to resizeThreadPool .
------	---

getOffsetTruncationState Internal Method

```
getOffsetTruncationState(  
    tp: TopicPartition,  
    leaderEpochOffset: EpochEndOffset): OffsetTruncationState
```

getOffsetTruncationState ...FIXME

Note `getOffsetTruncationState` is used exclusively when `AbstractFetcherThread` is requested to [maybeTruncateToEpochEndOffsets](#).

fetchOffsetAndTruncate Method

```
fetchOffsetAndTruncate(  
    topicPartition: TopicPartition,  
    currentLeaderEpoch: Int): Long
```

fetchOffsetAndTruncate ...FIXME

Note `fetchOffsetAndTruncate` is used exclusively when `AbstractFetcherThread` is requested to [addPartitions](#) and [handleOutOfRangeError](#).

handlePartitionsWithErrors Method

```
handlePartitionsWithErrors(partitions: Iterable[TopicPartition]): Unit
```

handlePartitionsWithErrors ...FIXME

Note `handlePartitionsWithErrors` is used when `AbstractFetcherThread` is requested to [maybeFetch](#), [truncateToEpochEndOffsets](#), [truncateToHighWatermark](#), and [processFetchRequest](#).

ReplicaFetcherBlockingSend

`ReplicaFetcherBlockingSend` is...FIXME

`ReplicaFetcherBlockingSend` is [created](#) exclusively when `ReplicaFetcherThread` is [created](#).

`ReplicaFetcherBlockingSend` uses [NetworkClient](#) to [send requests](#) for...FIXME

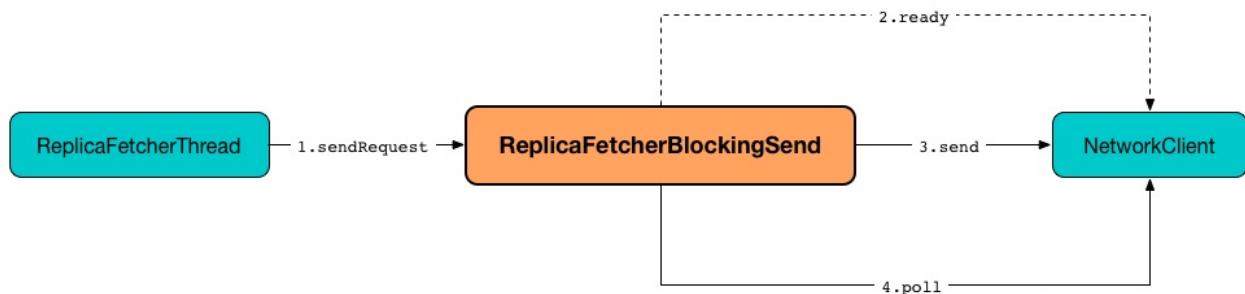


Figure 1. `ReplicaFetcherBlockingSend`'s Sending Client Request and Waiting for Response

`ReplicaFetcherBlockingSend` uses [replica.socket.timeout.ms](#) Kafka property for...FIXME

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

Name	Description
<code>networkClient</code>	NetworkClient Used when...FIXME
<code>sourceNode</code>	Node Used when...FIXME

Creating `ReplicaFetcherBlockingSend` Instance

`ReplicaFetcherBlockingSend` takes the following when created:

- `BrokerEndPoint`
- [KafkaConfig](#)
- `Metrics`
- `Time`
- Fetcher ID
- Client ID

- `LogContext`

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

Sending Client Request and Waiting for Response — `sendRequest` Method

```
sendRequest(requestBuilder: Builder[_ <: AbstractRequest]): ClientResponse
```

`sendRequest` requests `NetworkClientutils` to [wait until the connection is ready](#) to the [source broker node](#) (in `replica.socket.timeout.ms`).

`sendRequest` requests `NetworkClient` to [create a new client request](#) to the [source broker](#).

`sendRequest` requests `NetworkClientutils` to [send the client request and wait for a response](#).

Note	<code>sendRequest</code> is a blocking operation (i.e. blocks the current thread) and polls for responses until the one arrives or a disconnection or a version mismatch happens.
------	---

In case `NetworkClientutils` found the broker node unavailable, `sendRequest` reports a `SocketTimeoutException`:

```
Failed to connect to [sourceNode] within [socketTimeout] ms
```

Note	<code>sendRequest</code> is used when <code>ReplicaFetcherThread</code> <code>earliestOrLatestOffset</code> and <code>fetchEpochsFromLeader</code> .
------	--

close Method

```
close(): Unit
```

`close` ...FIXME

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

ReplicationQuotaManager

ReplicationQuotaManager is...FIXME

LogDirFailureHandler

`LogDirFailureHandler` is...FIXME

start Method

Caution

FIXME

dowork Method

```
def dowork(): Unit
```

Note

`dowork` is a part of [ShutdownableThread Contract](#).

`dowork` ...FIXME

Selector — Selectable on Socket Channels (from Java's New IO API)

`Selector` is the one and only [Selectable](#) that uses Java's selectable channels for stream-oriented connecting sockets (i.e. Java's [java.nio.channels.SocketChannel](#)).

`Selector` is used by Kafka services to [create](#) a `NetworkClient`.

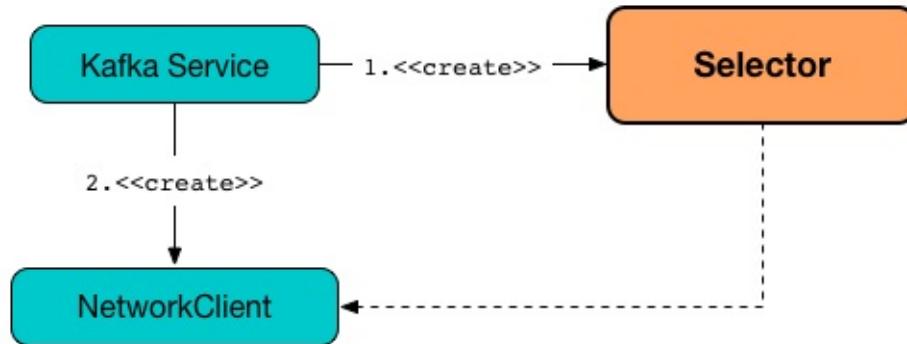


Figure 1. Selector is Created for Kafka Services For NetworkClient

`Selector` is created when:

- `KafkaAdminClient` is created (using `createInternal`)
- `KafkaConsumer` is created
- `KafkaProducer` is created
- `AdminClient` is created
- `ControllerChannelManager` is requested to [addNewBroker](#)
- `TransactionMarkerChannelManager` is created
- `Processor` is created
- `KafkaServer` does `doControlledShutdown`
- `ReplicaFetcherBlockingSend` is created

connect Method

```

void connect(
    String id,
    InetSocketAddress address,
    int sendBufferSize,
    int receiveBufferSize) throws IOException
  
```

Note

`connect` is a part of [Selectable Contract](#) that `NetworkClient` uses when requested to establish a connection to a broker.

`connect` ...FIXME

addToCompletedReceives Internal Method

```
void addToCompletedReceives()
void addToCompletedReceives(
    KafkaChannel channel,
    Deque<NetworkReceive> stagedDeque)
```

`addToCompletedReceives` ...FIXME

Note

`addToCompletedReceives` is used exclusively when `Selector` is requested to `poll`.

poll Method

```
void poll(long timeout)
```

Note

`poll` is part of the [Selectable Contract](#) to...FIXME.

`poll` ...FIXME

Selectable

`Selectable` is the [contract](#) for asynchronous, multi-channel network I/O.

Note

`Selector` is the one and only `Selectable`.

```
package org.apache.kafka.common.network;

public interface Selectable {
    void connect(String id, InetSocketAddress address, int sendBufferSize, int receiveBu
fferSize) throws IOException;
    void close();
    void close(String id);

    void send(Send send);
    void poll(long timeout) throws IOException;
    void wakeup();

    List<Send> completedSends();
    List<NetworkReceive> completedReceives();

    Map<String, ChannelState> disconnected();
    List<String> connected();

    void mute(String id);
    void unmute(String id);
    void muteAll();
    void unmuteAll();

    boolean isChannelReady(String id);
}
```

Table 1. Selectable Contract (in alphabetical order)

Method	Description
<code>connect</code>	Used exclusively when <code>NetworkClient</code> is requested to establish a connection to a broker
<code>poll</code>	

ShutdownableThread

`ShutdownableThread` is the [contract](#) for [non-daemon threads of execution](#).

`ShutdownableThread` contract expects that the objects implement [doWork](#) method.

```
def dowork(): Unit
```

Table 1. ShutdownableThread's Internal Properties (e.g. Registries and Counters)

Name	Description
<code>isRunning</code>	Flag that controls how long to execute run method.
<code>shutdownLatch</code>	Java's java.util.concurrent.CountDownLatch with the number of passes being 1

run Method

```
run(): Unit
```

Note	<code>run</code> is a part of java.lang.Runnable that is executed when the thread is started.
------	---

`run` first prints out the following INFO message to the logs:

```
Starting
```

`run` then executes [doWork](#) method until [isRunning](#) flag is disabled.

In the end, `run` decrements the count of [shutdownLatch](#) and prints out the following INFO message to the logs:

```
Stopped
```

SocketServer

`SocketServer` is a NIO socket server for a [KafkaServer](#).

`SocketServer` is [created](#) and then [started up](#) exclusively when `KafkaServer` is requested to [start up](#).

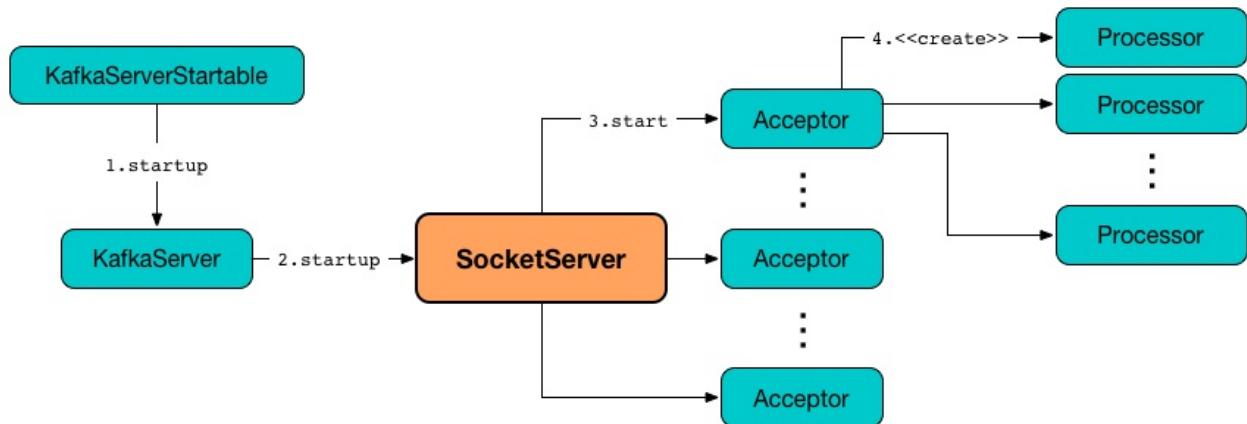


Figure 1. `SocketServer`'s Startup

`SocketServer` uses [queued.max.requests](#) configuration property for...FIXME

Table 1. `SocketServer`'s Metrics (in `kafka.network` group)

Name	Description
<code>NetworkProcessorAvgIdlePercent</code>	
<code>MemoryPoolAvailable</code>	
<code>MemoryPoolUsed</code>	

`SocketServer` uses an empty metric prefix for...FIXME

`SocketServer` uses **ControlPlane** metric prefix for...FIXME

Creating `SocketServer` Instance

`SocketServer` takes the following to be created:

- [KafkaConfig](#)
- [Metrics](#)
- [Time](#)
- [CredentialProvider](#)

`SocketServer` initializes the [internal properties](#).

Creating Network Processor Thread— `newProcessor` Internal Method

```
newProcessor(
    id: Int,
    connectionQuotas: ConnectionQuotas,
    listenerName: ListenerName,
    securityProtocol: SecurityProtocol,
    memoryPool: MemoryPool): Processor
```

`newProcessor` simply creates a new [Processor](#) with the [RequestChannel](#) and the following configuration properties:

- `socket.request.max.bytes`
- `connections.max.idle.ms`
- `connection.failed.authentication.delay.ms`

Note

`newProcessor` is used exclusively when `SocketServer` is requested to [addProcessors](#).

Starting Up (and Auxiliary Services)— `startup` Method

```
startup(startupProcessors: Boolean = true): Unit
```

Internally, `startup` creates the [ConnectionQuotas](#) (with [maxConnectionsPerIp](#) and [maxConnectionsPerIpOverrides](#)).

For every endpoint (in [endpoints](#) registry) `startup` does the following:

1. Creates up to [numProcessorThreads](#) number of [Processors](#) (for [ConnectionQuotas](#) and [MemoryPool](#))
2. Creates a `Acceptor` for the endpoint and processors
3. Records the `Acceptor` in [acceptors](#) internal registry
4. Starts a non-daemon thread for the `Acceptor` with the name as `kafka-socket-acceptor-[listenerName]-[securityProtocol]-[port]` (e.g. `kafka-socket-acceptor-ListenerName(PLAINTEXT)-PLAINTEXT-9092`) and waits until it has started fully

`startup` then registers [metrics](#).

In the end, `startup` prints out the following INFO message to the logs:

```
INFO [SocketServer brokerId=[brokerID]] Started [number] acceptor threads
```

Note	<code>startup</code> is used exclusively when <code>KafkaServer</code> is requested to start up .
------	---

addProcessors Internal Method

```
addProcessors(  
    acceptor: Acceptor,  
    endpoint: EndPoint,  
    newProcessorsPerListener: Int): Unit
```

`addProcessors` ...FIXME

Note	<code>addProcessors</code> is used when <code>SocketServer</code> is requested to createAcceptorAndProcessors and resizeThreadPool .
------	--

createAcceptorAndProcessors Internal Method

```
createAcceptorAndProcessors(  
    processorsPerListener: Int,  
    endpoints: Seq[EndPoint]): Unit
```

`createAcceptorAndProcessors` ...FIXME

Note	<code>createAcceptorAndProcessors</code> is used when <code>SocketServer</code> is requested to startup and addListeners .
------	--

resizeThreadPool Method

```
resizeThreadPool(  
    oldNumNetworkThreads: Int,  
    newNumNetworkThreads: Int): Unit
```

`resizeThreadPool` ...FIXME

Note	<code>resizeThreadPool</code> is used exclusively when <code>DynamicThreadPool</code> is requested to reconfigure (the number of network threads).
------	--

addListeners Method

```
addListeners(listenersAdded: Seq[EndPoint]): Unit
```

```
addListeners ...FIXME
```

Note

`addListeners` is used exclusively when `DynamicListenerConfig` is requested to reconfigure.

Stopping Request Processors — `stopProcessingRequests` Method

```
stopProcessingRequests(): Unit
```

```
stopProcessingRequests ...FIXME
```

Note

`stopProcessingRequests` is used when:

- `SocketServer` is requested to `shutdown`
- `KafkaServer` is requested to `shutdown`

Shutting Down — `shutdown` Method

```
shutdown(): Unit
```

```
shutdown ...FIXME
```

Note

`shutdown` is used when...FIXME

`updateMaxConnectionsPerIpOverride` Method

```
updateMaxConnectionsPerIpOverride(  
  maxConnectionsPerIpOverrides: Map[String, Int]): Unit
```

```
updateMaxConnectionsPerIpOverride ...FIXME
```

Note

`updateMaxConnectionsPerIpOverride` is used when...FIXME

`updateMaxConnectionsPerIp` Method

```
updateMaxConnectionsPerIp(maxConnectionsPerIp: Int): Unit
```

updateMaxConnectionsPerIp ...FIXME

Note

updateMaxConnectionsPerIp is used when...FIXME

removeListeners Method

```
removeListeners(listenersRemoved: Seq[EndPoint]): Unit
```

removeListeners ...FIXME

Note

removeListeners is used when...FIXME

addDataPlaneProcessors Internal Method

```
addDataPlaneProcessors(  
    acceptor: Acceptor,  
    endpoint: EndPoint,  
    newProcessorsPerListener: Int): Unit
```

addDataPlaneProcessors ...FIXME

Note

addDataPlaneProcessors is used when SocketServer is requested to [createDataPlaneAcceptorsAndProcessors](#) and [resizeThreadPool](#).

createDataPlaneAcceptorsAndProcessors Internal Method

```
createDataPlaneAcceptorsAndProcessors(  
    dataProcessorsPerListener: Int,  
    endpoints: Seq[EndPoint]): Unit
```

createDataPlaneAcceptorsAndProcessors ...FIXME

Note

createDataPlaneAcceptorsAndProcessors is used when SocketServer is requested to [start up](#) and [addListeners](#).

Internal Properties

Name	Description
acceptors	Acceptor threads per EndPoint
connectionQuotas	ConnectionQuotas
controlPlaneRequestChannelOpt	Optional RequestChannel (with the queue size of 20 and the ControlPlaneMetricPrefix metric name prefix)
dataPlaneRequestChannel	RequestChannel (with the queue size of maxQueuedRequests and the DataPlaneMetricPrefix metric name prefix)
	Initialized when SocketServer is requested to addDataPlaneProcessors
	Used to create the dataPlaneRequestProcessor and dataPlaneRequestHandlerPool for KafkaServer
endpoints	EndPoints (aka listeners) per name (as configured using listeners Kafka property)
maxConnectionsPerIp	
maxConnectionsPerIpOverrides	
memoryPool	MemoryPool
numProcessorThreads	The number of processors per endpoint (as configured using num.network.threads Kafka property)
processors	Network processor threads per ID (initially totalProcessorThreads)
	New processor threads are added in addProcessors
	Used in stopProcessingRequests (to shut down the network processor threads)
requestChannel	A RequestChannel (with queued.max.requests queue size)
	Used when: <ul style="list-style-type: none"> SocketServer is requested to create a network processor thread, addProcessors, stopProcessingRequests, resizeThreadPool, and shutdown

	<ul style="list-style-type: none">• <code>KafkaServer</code> is requested to start up (and creates the KafkaApis and the KafkaRequestHandlerPool)
<code>totalProcessorThreads</code>	Total number of processors, i.e. <code>numProcessorThreads</code> for every endpoint

Network Processor Thread (Socket Server Request Processor)

`Processor` is a [java.lang.Runnable](#) (as a `AbstractServerThread`) that is executed as a separately-executing thread.

`Processor` is also a [KafkaMetricsGroup](#) that...FIXME

`Processor` is [created](#) exclusively when `SocketServer` is requested to [create a new processor](#).

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

Name	Description
<code>inflightResponses</code>	<code>Map[String, RequestChannel.Response]</code> Used when...FIXME
<code>newConnections</code>	<code>ConcurrentLinkedQueue[SocketChannel]</code> Used when...FIXME
<code>responseQueue</code>	<code>LinkedBlockingDeque[RequestChannel.Response]</code> Used when...FIXME
<code>selector</code>	Selector Used when...FIXME

Creating Processor Instance

`Processor` takes the following to be created:

- `ID`
- `Time`
- `maxRequestSize`
- [RequestChannel](#)
- `ConnectionQuotas`
- `connectionsMaxIdleMs`
- `failedAuthenticationDelayMs`

- Listener Name
- SecurityProtocol
- KafkaConfig
- Metrics
- CredentialProvider
- MemoryPool
- LogContext

Processor initializes the internal registries and counters.

processCompletedReceives Internal Method

```
processCompletedReceives()
```

processCompletedReceives ...FIXME

Note	processCompletedReceives is used exclusively when Processor is requested to run.
------	--

run Method

```
run(): Unit
```

Note	run is part of the java.lang.Runnable to start itself as a separately-executing thread.
------	---

run ...FIXME

RequestChannel

`RequestChannel` is...FIXME

`RequestChannel` is also a [KafkaMetricsGroup](#) that...FIXME

`RequestChannel` is [created](#) exclusively when `SocketServer` is created (and is then used to create a [Processor](#), [KafkaApis](#), [KafkaRequestHandler](#) and [KafkaRequestHandlerPool](#)).

`RequestChannel` takes the size of the queue to be created (that is controlled by `queued.max.requests` configuration property).

RequestChannel.Request

`RequestChannel.Request` is created exclusively when `Processor` is requested to `processCompletedReceives`.

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

Name	Description
<code>bodyAndSize</code>	<code>RequestAndSize</code> that is created by requesting the <code>RequestContext</code> to parse a request (from the given <code>ByteBuffer</code>) Used when...FIXME

Creating RequestChannel.Request Instance

`Request` takes the following to be created:

- Processor ID
- `RequestContext`
- `startTimeNanos`
- `MemoryPool`
- `java.nio.ByteBuffer`
- `RequestChannel.Metrics`

`Request` initializes the internal registries and counters.

TransactionCoordinator

TransactionCoordinator is...FIXME

TransactionCoordinator is [created](#) when...FIXME

startup Method

startup

startup ...FIXME

Note `startup` is used exclusively when `KafkaServer` is requested to [start up](#).

Creating TransactionCoordinator Instance

TransactionCoordinator takes the following when created:

- Broker ID
- `TransactionConfig`
- [Scheduler](#)
- `ProducerIdManager`
- [TransactionStateManager](#)
- `TransactionMarkerChannelManager`
- `Time`
- `LogContext`

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

Creating TransactionCoordinator — [apply Factory Method](#)

```
apply(  
  config: KafkaConfig,  
  replicaManager: ReplicaManager,  
  scheduler: Scheduler,  
  zkClient: KafkaZkClient,  
  metrics: Metrics,  
  metadataCache: MetadataCache,  
  time: Time): TransactionCoordinator
```

apply ...FIXME

Note

apply is used exclusively when KafkaServer is requested to start up (to create a TransactionCoordinator).

shutdown Method

```
shutdown(): Unit
```

shutdown ...FIXME

Note

shutdown is used when...FIXME

abortTimedOutTransactions Internal Method

```
abortTimedOutTransactions(): Unit
```

abortTimedOutTransactions ...FIXME

Note

abortTimedOutTransactions is used when...FIXME

partitionFor Method

```
partitionFor(transactionalId: String): Int
```

partitionFor ...FIXME

Note

partitionFor is used when...FIXME

TransactionMarkerChannelManager

`TransactionMarkerChannelManager` is a [InterBrokerSendThread](#) that...FIXME

`TransactionMarkerChannelManager` uses the name **TxnMarkerSenderThread-[brokerId]**.

`TransactionMarkerChannelManager` uses **[Transaction Marker Channel Manager [brokerId]]** as the logging prefix (aka `logIdent`).

`TransactionMarkerChannelManager` is [created](#) (using [apply](#) factory method) exclusively when `TransactionCoordinator` is [created](#) (for a [KafkaServer](#)).

Creating TransactionMarkerChannelManager Instance

`TransactionMarkerChannelManager` takes the following to be created:

- [KafkaConfig](#)
- [MetadataCache](#)
- [NetworkClient](#)
- [TransactionStateManager](#)
- `DelayedOperationPurgatory[DelayedTxnMarker]`
- `Time`

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

Creating TransactionMarkerChannelManager — [apply](#) Factory Method

```
apply(  
    config: KafkaConfig,  
    metrics: Metrics,  
    metadataCache: MetadataCache,  
    txnStateManager: TransactionStateManager,  
    txnMarkerPurgatory: DelayedOperationPurgatory[DelayedTxnMarker],  
    time: Time,  
    logContext: LogContext): TransactionMarkerChannelManager
```

`apply` ...FIXME

Note	apply is used exclusively when TransactionCoordinator is created (when KafkaServer is requested to start up).
-------------	---

InterBrokerSendThread

`InterBrokerSendThread` is the extension of the [ShutdownableThread contract](#) for inter-broker send threads with a non-blocking [NetworkClient](#) that `doWork`.

Note

`TransactionMarkerChannelManager` is the only available implementation of the [InterBrokerSendThread Contract](#) in Apache Kafka.

dowork Method

`dowork(): Unit`

Note

`dowork` is a part of [ShutdownableThread Contract](#).

`dowork ...FIXME`

sendRequests Internal Method

`sendRequests(now: Long): Long`

`sendRequests ...FIXME`

Note

`sendRequests` is used exclusively when `InterBrokerSendThread` is requested to [doWork](#).

completeWithDisconnect Method

```
completeWithDisconnect(
  request: ClientRequest,
  now: Long,
  authenticationException: AuthenticationException): Unit
```

`completeWithDisconnect ...FIXME`

Note

`completeWithDisconnect` is used when `InterBrokerSendThread` is requested to [checkDisconnects](#) and [failExpiredRequests](#) (when `InterBrokerSendThread` is requested to [doWork](#)).

checkDisconnects Internal Method

```
checkDisconnects(now: Long): Unit
```

```
checkDisconnects ...FIXME
```

Note

`checkDisconnects` is used exclusively when `InterBrokerSendThread` is requested to `doWork`.

failExpiredRequests Internal Method

```
failExpiredRequests(now: Long): Unit
```

```
failExpiredRequests ...FIXME
```

Note

`failExpiredRequests` is used exclusively when `InterBrokerSendThread` is requested to `doWork`.

Creating InterBrokerSendThread Instance

`InterBrokerSendThread` takes the following to be created:

- Name
- [NetworkClient](#)
- `Time`
- `isInterruptible` flag (default: `true`)

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

TransactionStateManager

TransactionStateManager is...FIXME

getTransactionTopicPartitionCount Method

getTransactionTopicPartitionCount

getTransactionTopicPartitionCount ...FIXME

Note getTransactionTopicPartitionCount is used when...FIXME

enableTransactionalIdExpiration Method

enableTransactionalIdExpiration(): Unit

enableTransactionalIdExpiration ...FIXME

Note enableTransactionalIdExpiration is used when...FIXME

loadTransactionsForTxnTopicPartition Method

```
loadTransactionsForTxnTopicPartition(  
    partitionId: Int,  
    coordinatorEpoch: Int,  
    sendTxnMarkers: SendTxnMarkersCallback): Unit
```

loadTransactionsForTxnTopicPartition ...FIXME

Note loadTransactionsForTxnTopicPartition is used when...FIXME

removeTransactionsForTxnTopicPartition Method

```
removeTransactionsForTxnTopicPartition(partitionId: Int, coordinatorEpoch: Int): Unit
```

removeTransactionsForTxnTopicPartition ...FIXME

Note removeTransactionsForTxnTopicPartition is used when...FIXME

loadTransactions Method

```
loadTransactions(): Unit
```

loadTransactions ...FIXME

Note	loadTransactions is used when...FIXME
------	---------------------------------------

removeTransactions Method

```
removeTransactions(): Unit
```

removeTransactions ...FIXME

Note	removeTransactions is used when...FIXME
------	---

appendTransactionToLog Method

```
appendTransactionToLog(  
    transactionalId: String,  
    coordinatorEpoch: Int,  
    newMetadata: TxnTransitMetadata,  
    responseCallback: Errors => Unit,  
    retryOnError: Errors => Boolean = _ => false): Unit
```

appendTransactionToLog ...FIXME

Note	appendTransactionToLog is used when...FIXME
------	---

loadTransactionMetadata Internal Method

```
loadTransactionMetadata(  
    topicPartition: TopicPartition,  
    coordinatorEpoch: Int): Pool[String, TransactionMetadata]
```

loadTransactionMetadata ...FIXME

Note	loadTransactionMetadata is used when...FIXME
------	--

QuotaManagers

QuotaManagers is...FIXME

ZkUtils

`zkutils` is...FIXME

`zkutils` is created when...FIXME

	<p>Quoting Nodes and ephemeral nodes from the Zookeeper documentation:</p> <p>Unlike standard file systems, each node in a ZooKeeper namespace can have data associated with it as well as children. It is like having a file-system that allows a file to also be a directory. (ZooKeeper was designed to store coordination data: status information, configuration, location information, etc., so the data stored at each node is usually small, in the byte to kilobyte range.) We use the term znode to make it clear that we are talking about ZooKeeper data nodes.</p>
Note	<p>Znodes maintain a stat structure that includes version numbers for data changes, ACL changes, and timestamps, to allow cache validations and coordinated updates. Each time a znode's data changes, the version number increases. For instance, whenever a client retrieves data it also receives the version of the data.</p> <p>The data stored at each znode in a namespace is read and written atomically. Reads get all the data bytes associated with a znode and a write replaces all the data. Each node has an Access Control List (ACL) that restricts who can do what.</p>

Table 1. ZkUtils's ZNodes in Zookeeper

ZNode	Description
ControllerPath	/controller

Table 2. ZkUtils's Internal Properties (e.g. Registries and Counters) (in alphabetical order)

Name	Description
persistentZkPaths	
zkPath	

getCluster Method

`getCluster(): Cluster`

`getcluster` gets the children znodes of `/brokers/ids` znode and reads their data (as a JSON blob).

`getCluster` then adds `creates` a `Broker` from the znode id and the JSON blob (with a host, a port and endpoints).

Note

`getCluster` is used exclusively when `ZKRebalancerListener` does `syncedRebalance` (that happens for the currently-deprecated `ZookeeperConsumerConnector`).

deletePathRecursive Method**Caution**

FIXME

deletePath Method**Caution**

FIXME

Creating ZkUtils Instance — apply Factory Method

```
apply(  
    zkUrl: String,  
    sessionTimeout: Int,  
    connectionTimeout: Int,  
    isZkSecurityEnabled: Boolean): ZkUtils
```

`apply` ...FIXME

Note

`apply` is used when:

1. `KafkaServer` connects to Zookeeper
2. FIXME

**Registering Listener for State Changes
— subscribeStateChanges Method**

```
subscribeStateChanges(listener: IZkStateListener): Unit
```

`subscribeStateChanges` requests `ZkClient` to `subscribeStateChanges` with the `listener`.

Note	<code>subscribeStateChanges</code> is used when:
	1. <code>KafkaController</code> is requested to register a SessionExpirationListener
	2. <code>FIXME</code>

Registering Listener for Child Changes

— `subscribeChildChanges` Method

```
subscribeChildChanges(path: String, listener: IZkChildListener): Option[Seq[String]]
```

`subscribeChildChanges` ...`FIXME`

Note	<code>subscribeChildChanges</code> is used... <code>FIXME</code>
------	--

De-Registering Listener for Child Changes

— `unsubscribeChildChanges` Method

```
unsubscribeChildChanges(path: String, childListener: IZkChildListener): Unit
```

`unsubscribeChildChanges` requests [ZkClient](#) to `unsubscribeChildChanges` for the input `path` and `childListener`.

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

De-Registering Listener for Data Changes

— `unsubscribeDataChanges` Method

```
unsubscribeDataChanges(path: String, dataListener: IZkDataListener): Unit
```

`unsubscribeDataChanges` requests [ZkClient](#) to `unsubscribeDataChanges` for the input `path` and `dataListener`.

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

registerBrokerInZk Method

```
registerBrokerInZk(
  id: Int,
  host: String,
  port: Int,
  advertisedEndpoints: Seq[EndPoint],
  jmxPort: Int,
  rack: Option[String],
  apiVersion: ApiVersion): Unit
```

registerBrokerInZk ...FIXME

Note	registerBrokerInZk is used exclusively when KafkaHealthcheck is requested to register.
------	--

getTopicPartitionCount Method

```
getTopicPartitionCount(topic: String): Option[Int]
```

getTopicPartitionCount ...FIXME

Note	<p>getTopicPartitionCount is used when:</p> <ol style="list-style-type: none"> GroupMetadataManager is requested for getGroupMetadataTopicPartitionCount of __consumer_offsets topic TransactionStateManager is requested for getTransactionTopicPartitionCount of __transaction_state topic
------	--

Creating JSON with Broker ID— controllerZkData Method

```
controllerZkData(brokerId: Int, timestamp: Long): String
```

controllerZkData creates a JSON with the following fields:

- "version":1
- "brokerid": [brokerId]
- "timestamp": [timestamp]

```
import kafka.utils._
scala> ZkUtils.controllerZkData(1, System.currentTimeMillis())
res0: String = {"version":1,"brokerid":1,"timestamp":"1506161225262"}
```

Note

`controllerZkData` is used exclusively when `KafkaController` is requested for [elect](#).

Creating ZkUtils Instance

`zkutils` takes the following when created:

- `ZkClient`
- `ZkConnection`
- `isSecure` flag

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

Reading Data Associated with ZNode — `readDataMaybeNull` Method

```
readDataMaybeNull(path: String): (Option[String], Stat)
```

`readDataMaybeNull` requests `ZkClient` to `readData` from `path` `znode`.

`readDataMaybeNull` returns `None` (for `Option[String]`) when `path` `znode` is not available.

ZKRebalancerListener

`ZKRebalancerListener` is...FIXME

syncedRebalance Method

`syncedRebalance(): Unit`

`syncedRebalance` ...FIXME

Note

`syncedRebalance` is used when:

- `ZKSessionExpireListener` does `handleNewSession`
- That happens exclusively in (deprecated) `ZookeeperConsumerConnector`
- `ZKRebalancerListener` is created (and creates `watcherExecutorThread`)
- That happens exclusively in (deprecated) `ZookeeperConsumerConnector`
- (deprecated) `ZookeeperConsumerConnector` does `reinitializeConsumer`

ControllerContext

`ControllerContext` is the context of a [KafkaController](#) (and is created exclusively when `KafkaController` is [created](#)).

Table 1. ControllerContext's Registries

Name	Description
allTopics	
controllerChannelManager	ControllerChannelManager
epoch	
epochZkVersion	
liveBrokers	
liveBrokerIds	
liveBrokersUnderlying	
liveBrokerIdsUnderlying	
liveOrShuttingDownBrokerIds	
liveOrShuttingDownBrokers	
partitionsBeingReassigned	
partitionLeadershipInfo	partitionLeadershipInfo: mutable.Map[TopicParti
partitionReplicaAssignmentUnderlying	partitionReplicaAssignmentUnderlying: mutable
replicasOnOfflineDirs	partitionReplicaAssignmentUnderlying is update requested to updatePartitionReplicaAssignment
shuttingDownBrokerIds	
stats	<p>ControllerStats with UncleanLeaderElectionsPer every ControllerState (except Idle state)</p> <p>stats is used exclusively to create the Controller is then used to collect the times (metrics) of proc ShutdownEventThread)</p> <ul style="list-style-type: none"> • Every ControllerState has the RateAndTir <p>The timer metric name pattern is kafka.controller.stats</p>

ControllerContext takes no input arguments when created.

allPartitions Method

```
allPartitions: Set[TopicPartition]
```

`allPartitions` converts the `partitionReplicaAssignmentUnderlying` into `TopicPartitions`, i.e. `allPartitions` takes the partitions for the topics and simply creates new `TopicPartitions`.

Note

`allPartitions` is used when:

- `KafkaController` is requested to `updateLeaderAndIsrCache`, `checkAndTriggerAutoLeaderRebalance`, and `updateMetrics`
- `PartitionStateMachine` is requested to `initializePartitionState`
- `ReplicaStateMachine` is requested to `initializeReplicaState`

updatePartitionReplicaAssignment Method

```
updatePartitionReplicaAssignment(topicPartition: TopicPartition, newReplicas: Seq[Int]): Unit
```

`updatePartitionReplicaAssignment` simply updates the `partitionReplicaAssignmentUnderlying` registry with `newReplicas` for the topic and the partition (of a given `TopicPartition`).

Note

`updatePartitionReplicaAssignment` is used when:

- `KafkaController` is requested to `initializeControllerContext`, `moveReassignedPartitionLeaderIfRequired`, `updateAssignedReplicasForPartition`, and at `TopicChange` and `PartitionModifications` controller events
- `ReplicaStateMachine` is requested to `doHandleStateChanges`

partitionsOnBroker Method

```
partitionsOnBroker(brokerId: Int): Set[TopicPartition]
```

`partitionsOnBroker` ...FIXME

Note

`partitionsOnBroker` is used when...FIXME

replicasOnBrokers Method

```
replicasOnBrokers(brokerIds: Set[Int]): Set[PartitionAndReplica]
```

replicasOnBrokers ...FIXME

Note	replicasOnBrokers is used when...FIXME
------	--

replicasForTopic Method

```
replicasForTopic(topic: String): Set[PartitionAndReplica]
```

replicasForTopic ...FIXME

Note	replicasForTopic is used when...FIXME
------	---------------------------------------

partitionsForTopic Method

```
partitionsForTopic(topic: String): collection.Set[TopicPartition]
```

partitionsForTopic ...FIXME

Note	partitionsForTopic is used when...FIXME
------	---

removeTopic Method

```
removeTopic(topic: String): Unit
```

removeTopic ...FIXME

Note	removeTopic is used when...FIXME
------	----------------------------------

isReplicaOnline Method

```
isReplicaOnline(  
  brokerId: Int,  
  topicPartition: TopicPartition,  
  includeShuttingDownBrokers: Boolean = false): Boolean
```

`isReplicaOnline ...FIXME`

Note

`isReplicaOnline` is used when...FIXME

allLiveReplicas Method

`allLiveReplicas(): Set[PartitionAndReplica]`

`allLiveReplicas ...FIXME`

Note

`allLiveReplicas` is used when...FIXME

ControllerEventManager

`ControllerEventManager` is responsible for processing `ControllerEvents` (off the `controller event queue`).

`ControllerEventManager` allows for emitting `ControllerEvents` (with optional clearing the `event queue first`).

`ControllerEventManager` is created exclusively for the `KafkaController` (when `KafkaServer` is requested to start up).

When started, `ControllerEventManager` simply requests the `ControllerEventThread` to start processing `ControllerEvents`.

`ControllerEventManager` is started when `KafkaController` is started up.

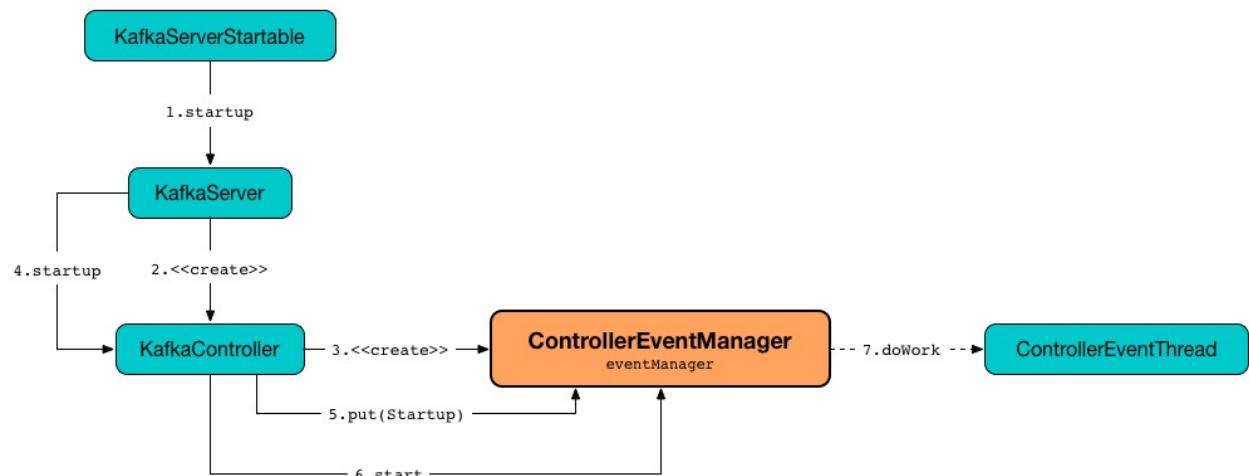


Figure 1. `ControllerEventManager` is Created and Started With `KafkaController`

`ControllerEventManager` is in one of the possible `ControllerStates`.

Note

The state of `ControllerEventManager` is exactly the state of the owning `KafkaController`.

Table 1. ControllerEventManager's Internal Properties (e.g. Registries and Counters)

Name	Description
queue	Controller event queue queue is a Java java.util.concurrent.LinkedBlockingQueue (i.e. an optionally-bounded blocking queue based on linked nodes that orders elements in first-in-first-out fashion) of ControllerEvents .
_state	ControllerState : <ul style="list-style-type: none"> • Idle when ControllerEventManager is created and right after ControllerEventThread has finished processing a controller event • State transitions happen per the requested state of the ControllerEvent being processed (while ControllerEventThread is processing controller events)
thread	ControllerEventThread with controller-event-thread thread name

Creating ControllerEventManager Instance

ControllerEventManager takes the following when created:

- Controller ID (i.e. the [broker.id](#) of the broker)
- `rateAndTimeMetrics` collection (`Map[ControllerState, KafkaTimer]`)
- `eventProcessedListener` (`ControllerEvent → Unit`)
- `controllerMovedListener` (`() → Unit`)

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

Emitting (Enqueuing) Controller Event — `put` Method

```
put(event: ControllerEvent): Unit
```

put simply inserts the ControllerEvent at the end of the controller event queue.

Note

- `put` is used when:
- `ControllerEventManager` is requested to `clearAndPut`
 - `KafkaController` is requested to do its operation and emits various events

Starting ControllerEventManager (and ControllerEventThread) — `start` Method

```
start(): Unit
```

`start` simply requests the `ControllerEventThread` to start processing `ControllerEvents`.

Note

`ControllerEventThread` is a `ShutdownableThread` that triggers `dowork()` method when started.

Note

`start` is used exclusively when `KafkaController` is requested to start up (when `KafkaServer` is requested to `start`).

clearAndPut Method

```
clearAndPut(event: ControllerEvent): Unit
```

`clearAndPut` simply clears the controller event queue and enqueues the controller event.

Note

- `clearAndPut` is used when:
- `ControllerEventManager` is requested to `close`
 - `KafkaController` is requested to `startup` (and registers a `StateChangeHandler` that emits a `Expire` event when `beforeInitializingSession`)

Closing Up — `close` Method

```
close(): Unit
```

`close` simply clears the event queue and emits a `ShutdownEventThread` event.

In the end, `close` waits until the shutdown is complete. You should see the following INFO message in the logs:

Shutdown completed

Note	<code>close</code> is used exclusively when <code>KafkaController</code> is requested to shutdown .
------	---

ControllerEventThread

`ControllerEventThread` is a `ShutdownableThread` that is `started` when `ControllerEventManager` is `started`

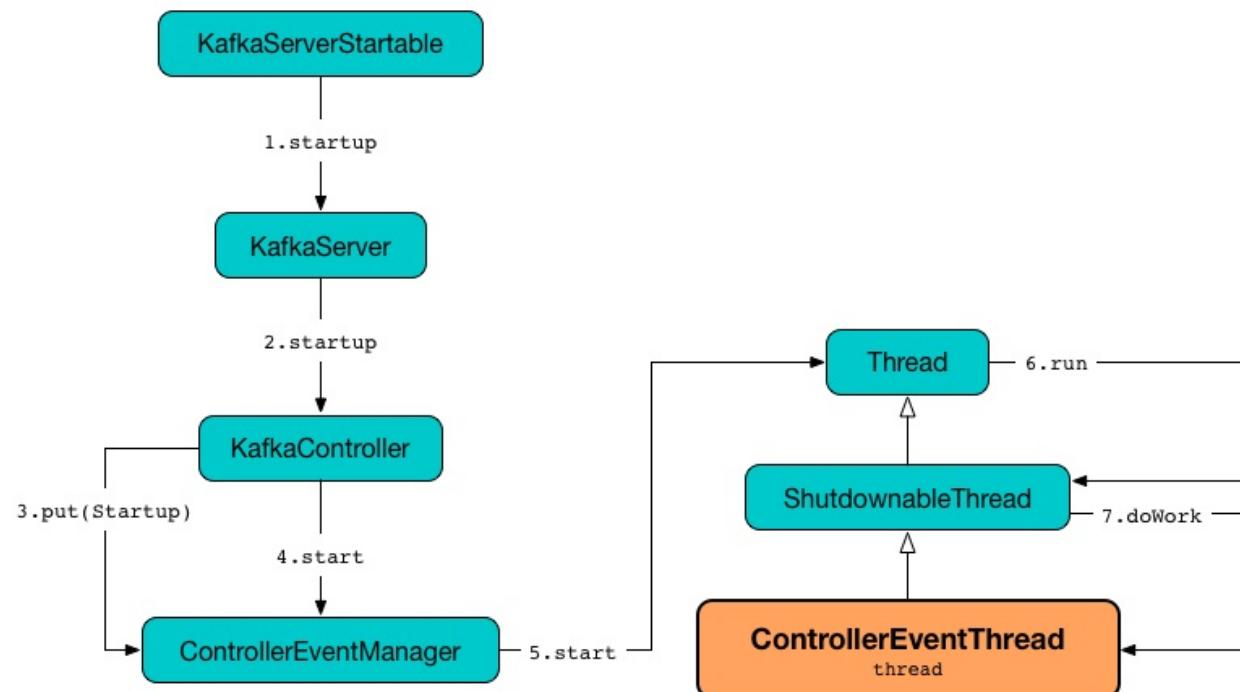


Figure 1. `ControllerEventThread` is Started Alongside `ControllerEventManager`. When created, `ControllerEventThread` takes the name of the thread (which `ControllerEventManager` sets as **controller-event-thread**).

```

"controller-event-thread" #44 prio=5 os_prio=31 tid=0x00007fac45730800 nid=0xad03 waiting on condition [0x0000000178b30000]
  java.lang.Thread.State: WAITING (parking)
    at sun.misc.Unsafe.park(Native Method)
      - parking to wait for  <0x00000007bcb03938> (a java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject)
        at java.util.concurrent.locks.LockSupport.park(LockSupport.java:175)
        at java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.await(AbstractQueuedSynchronizer.java:2039)
        at java.util.concurrent.LinkedBlockingQueue.take(LinkedBlockingQueue.java:442)
        at kafka.controller.ControllerEventManager$ControllerEventThread.dowork(ControllerEventManager.scala:48)
        at kafka.utils.ShutdownableThread.run(ShutdownableThread.scala:64)
  
```

Processing Controller Events — `dowork` Method

```
dowork(): Unit
```

`dowork` takes and removes the head of `event queue` (waiting if necessary until an element becomes available).

Note

The very first event in the event queue is `startup` that `KafkaController` puts when it is `started`.

`dowork` sets `_state` (of `ControllerEventManager`) as the state of the event.

`dowork` finds the `KafkaTimer` for the state in `rateAndTimeMetrics` lookup table (of `ControllerEventManager`).

`dowork` processes the event (i.e. calls `controllerEvent.process` method).

In the end, `dowork` passes the event to `eventProcessedListener` (of `ControllerEventManager`) and sets `_state` (of `ControllerEventManager`) as `Idle`.

ControllerEvent

`ControllerEvent` is the [contract](#) of the [events](#) in the lifecycle of [KafkaController](#) (state machine) that trigger [state](#) change with the corresponding [process](#) action.

`ControllerEvent` events are managed by [ControllerEventManager](#).

Table 1. ControllerEvent Contract

Method	Description
<code>state</code>	<code>state: ControllerState</code> The requested ControllerState of the ControllerEventManager (that <code>ControllerEventThread</code> sets while processing the controller event)
<code>process</code>	<code>process(): Unit</code> The action to execute when <code>ControllerEventThread</code> is requested to process the controller event

Table 2. ControllerEvents

ControllerEvent	ControllerState	
AutoPreferredReplicaLeaderElection	AutoLeaderBalance	
BrokerChange	BrokerChange	
BrokerModifications	BrokerChange	
ControlledShutdown	ControlledShutdown	
ControllerChange	ControllerChange	
Expire	ControllerChange	
IsrChangeNotification	IsrChange	E ZI be ch
LeaderAndIsrResponseReceived	LeaderAndIsrResponseReceived	

LogDirEventNotification	LogDirChange	
PartitionModifications	TopicChange	EI ZI nc
PartitionReassignment	PartitionReassignment	
PartitionReassignmentIsrChange	PartitionReassignment	
PreferredReplicaLeaderElection	ManualLeaderBalance	
Reelect	ControllerChange	
RegisterBrokerAndReelect	ControllerChange	
ShutdownEventThread	ControllerShutdown	
Startup	ControllerChange	
TopicChange	TopicChange	
TopicDeletion	TopicDeletion	
TopicDeletionStopReplicaResponseReceived	TopicDeletion	
TopicUncleanLeaderElectionEnable	TopicUncleanLeaderElectionEnable	
UncleanLeaderElectionEnable	UncleanLeaderElectionEnable	EI re er

Note	<code>ControllerEvent</code> is a Scala sealed trait and so all the possible controller events are in a single compilation unit (i.e. a file).
------	--

AutoPreferredReplicaLeaderElection Controller Event

`AutoPreferredReplicaLeaderElection` is a [controller event](#) that transitions the `KafkaController` to `AutoLeaderBalance` state.

`AutoPreferredReplicaLeaderElection` event is ignored (skipped) when processed on an [inactive controller](#).

When [processed](#) on an [active controller](#), `AutoPreferredReplicaLeaderElection` event [checkAndTriggerAutoLeaderRebalance](#) and in the end [scheduleAutoLeaderRebalanceTask](#) with the delay as configured using the `leader.imbalance.check.interval.seconds` property.

Startup Controller Event

Startup is a controller event that transition the KafkaController to ControllerChange state.

When processed, startup event requests the KafkaZkClient (of the parent KafkaController) to registerZNodeChangeHandlerAndCheckExistence with the ControllerChangeHandler followed by requesting the parent KafkaController to elect.

Reelect

Reelect is...FIXME

TopicDeletion Controller Event

`TopicDeletion` is a [ControllerEvent](#) that is [executed](#) on the active `KafkaController` (and does nothing otherwise).

Note

`TopicDeletion` uses `delete.topic.enable` Kafka property.

Note

Topics to be deleted are created in `/admin/delete_topics` path in Zookeeper.

`state` is `TopicDeletion`.

process Method

```
process(): Unit
```

Note

`process` is a part of [ControllerEvent Contract](#).

Note

`process` is executed on the [active controller](#) only (and does nothing otherwise).

`process` prints out the following DEBUG message to the logs:

```
Delete topics listener fired for topics [topicsToDelete] to be deleted
```

`process` requests [ControllerContext](#) for `allTopics` and finds topics that are supposed to be deleted, but are not available in the Kafka cluster.

If there are any non-existent topics, `process` prints out the following WARN message to the logs and requests [ZkUtils](#) to [deletePathRecursive](#) `/admin/delete_topics/[topicName]` znode for every topic in the list.

```
Ignoring request to delete non-existing topics [nonExistentTopics]
```

`process` branches off per `delete.topic.enable` Kafka property.

process with `delete.topic.enable` Enabled

With `delete.topic.enable` enabled (i.e. `true`), `process` prints out the following INFO message to the logs:

```
Starting topic deletion for topics [topicsToDelete]
```

`process` requests [TopicDeletionManager](#) to `markTopicIneligibleForDeletion` for topics to be deleted with partitions in `controllerContext.partitionsBeingReassigned` list.

`process` requests [TopicDeletionManager](#) to `enqueueTopicsForDeletion`.

process with `delete.topic.enable` Disabled

With `delete.topic.enable` disabled (i.e. `false`), `process` prints out the following INFO message to the logs (for every topic):

```
Removing /admin/delete_topics/[topicName] since delete topic is disabled
```

`process` requests [ZkUtils](#) to `deletePath` `/admin/delete_topics/[topicName]` znode (for every topic).

ControllerState Contract

`ControllerState` is the [contract](#) of the [states](#) that [KafkaController](#) can be in.

Every `ControllerEvent` has an associated [state](#). When a `ControllerEvent` is processed, it triggers a state transition to the requested state. These state transitions are used for [ControllerEventManager](#) and the owning [KafkaController](#).

`ControllerState` has the [rateAndTimeMetricName](#) (except [Idle](#) state). [ControllerContext](#) uses them to build a registry of `KafkaTimers` for every `ControllerState`. The timer metric name pattern is **kafka.controller:type=ControllerStats,name=**.

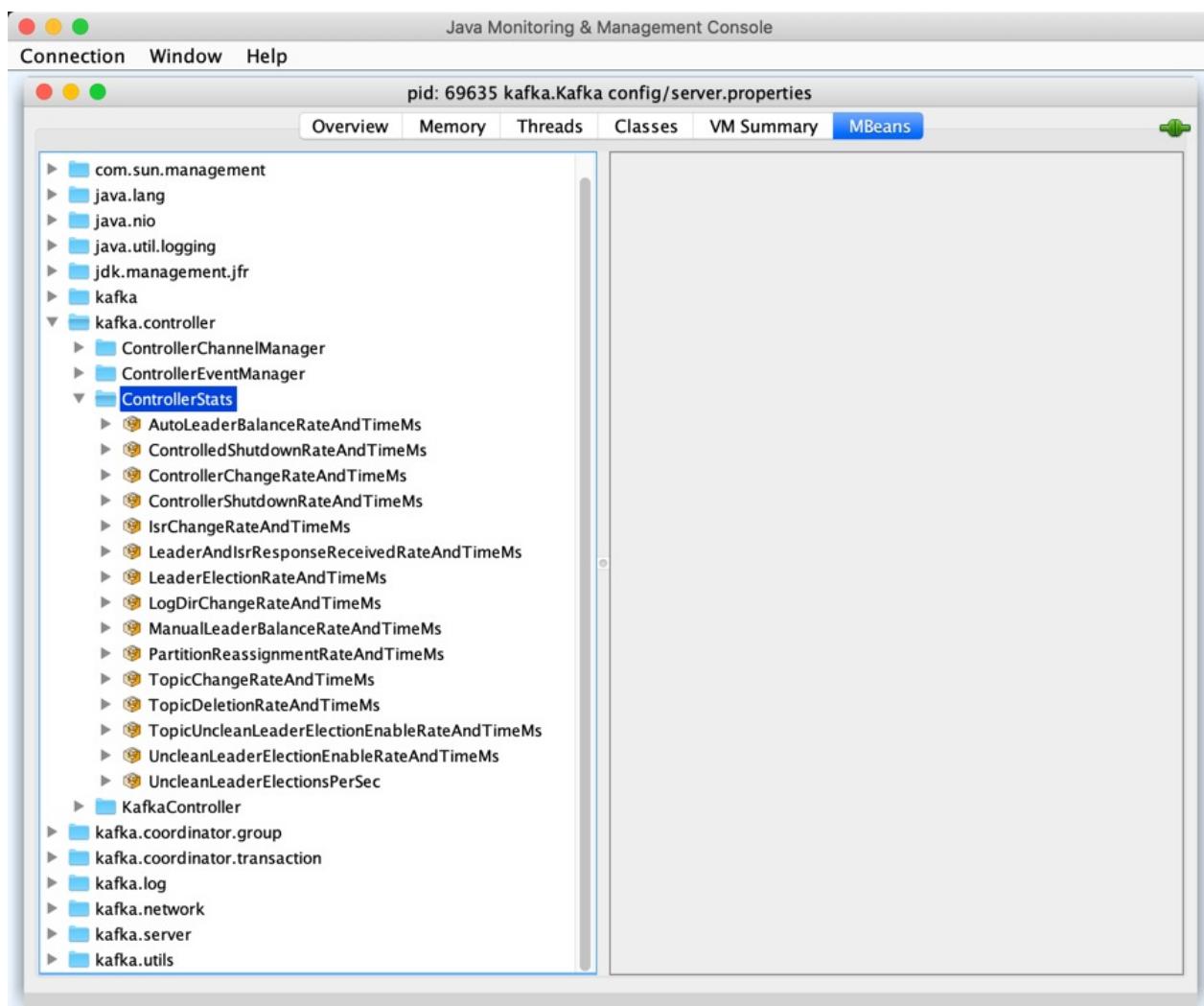


Figure 1. ControllerStates and RateAndTimeMs Timer Metrics in JConsole

Table 1. ControllerState Contract

Method	Description
<code>value</code>	<p><code>value: Byte</code></p> <p>Internal identifier</p>
<code>rateAndTimeMetricName</code>	<p><code>rateAndTimeMetricName: Option[String]</code></p> <p><code>rateAndTimeMetricName</code> depends on the <code>hasRateAndTimeMetric</code> flag:</p> <ul style="list-style-type: none"> • [ControllerState_name]RateAndTimeMs when enabled • Undefined when disabled
<code>hasRateAndTimeMetric</code>	<p><code>hasRateAndTimeMetric: Boolean</code></p> <p>Controls whether the <code>ControllerState</code> has a <code>RateAndTime metric</code> (<code>true</code>) or not (<code>false</code>)</p> <p>Default: <code>true</code></p>

Table 2. ControllerStates (and Triggering ControllerEvents)

ControllerState	Description
AutoLeaderBalance	
BrokerChange	<p><code>rateAndTimeMetricName</code> is <code>LeaderElectionRateAndTimeMs</code></p> <p>ControllerEvents: BrokerChange, BrokerModifications</p>
ControlledShutdown	
ControllerChange	ControllerEvents: RegisterBrokerAndReelect
ControllerShutdown	
Idle	<p>The initial state of <code>ControllerEventManager</code> (and indirectly <code>KafkaController</code>)</p> <p><code>Idle</code> has no <code>rateAndTime</code> metric.</p>
IsrChange	ControllerEvents: <code>IsrChangeNotification</code>
LeaderAndIsrResponseReceived	
LogDirChange	ControllerEvents: LogDirEventNotification
ManualLeaderBalance	ControllerEvents: PreferredReplicaLeaderElection
PartitionReassignment	ControllerEvents: PartitionReassignment, PartitionReassignmentIsrChange
TopicChange	ControllerEvents: TopicChange, PartitionModifications
TopicDeletion	ControllerEvents: TopicDeletion
TopicUncleanLeaderElectionEnable	
UncleanLeaderElectionEnable	

Note

`ControllerState` is a Scala sealed abstract class and so all the possible controller states are in a single compilation unit (i.e. a file).

ControllerChannelManager

`ControllerChannelManager` is...FIXME

`ControllerChannelManager` is created exclusively when `KafkaController` is requested to startChannelManager.

Table 1. ControllerChannelManager's Metrics

Name	Description
<code>TotalQueueSize</code>	

Table 2. ControllerChannelManager's Internal Properties

Name	Description
<code>brokerStateInfo</code>	Used when...FIXME
<code>brokerLock</code>	Used when...FIXME

`ControllerChannelManager` uses [Channel manager on controller [brokerId]] as the logging prefix (aka `logIdent`).

addNewBroker Internal Method

`addNewBroker(broker: Broker): Unit`

`addNewBroker` ...FIXME

Note	<code>addNewBroker</code> is used when <code>ControllerChannelManager</code> is created (to connect to live brokers) and is requested to addBroker.
------	---

addBroker Method

`addBroker(broker: Broker): Unit`

`addBroker` ...FIXME

Note	<code>addBroker</code> is used exclusively when <code>BrokerChange</code> controller event is processed.
------	--

Creating ControllerChannelManager Instance

`ControllerChannelManager` takes the following when created:

- `ControllerContext`
- `KafkaConfig`
- `Time`
- `Metrics`
- `StateChangeLogger`
- Optional thread name prefix (default: `None`)

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

In the end, `ControllerChannelManager` requests the `ControllerContext` for the `liveBrokers` that it [addNewBroker](#) each.

ControllerBrokerRequestBatch

`ControllerBrokerRequestBatch` is created exclusively for a [KafkaController](#) that uses it for the following:

- [addLeaderAndIsrRequestForBrokers](#)
- [addUpdateMetadataRequestForBrokers](#)
- [addStopReplicaRequestForBrokers](#)

Every time `ControllerBrokerRequestBatch` is used it is first [newBatch](#) that is then [sendRequestsToBrokers](#).

`ControllerBrokerRequestBatch` takes the following to be created:

- [KafkaController](#)
- [StateChangeLogger](#)

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

Name	Description
<code>leaderAndIsrRequestMap</code>	<p><code>Map[Int, mutable.Map[TopicPartition, LeaderAndIsrRequest.PartitionState]]</code></p> <ul style="list-style-type: none"> • A new entry added in addLeaderAndIsrRequestForBrokers • Cleared in clear and sendRequestsToBrokers <p>Used when <code>ControllerBrokerRequestBatch</code> is requested to sendRequestsToBrokers</p>
<code>stopReplicaRequestMap</code>	<code>Map[Int, Seq[StopReplicaRequestInfo]]</code>
<code>updateMetadataRequestBrokerSet</code>	<p>Broker IDs to send UPDATE_METADATA requests to (and to be removed right after a successful sending)</p> <p>New broker IDs added in addUpdateMetadataRequestForBrokers</p>
<code>updateMetadataRequestPartitionInfoMap</code>	TopicPartitions with their PartitionState

sendRequestsToBrokers Method

```
sendRequestsToBrokers(controllerEpoch: Int): Unit
```

`sendRequestsToBrokers` ...FIXME

Note

`sendRequestsToBrokers` is used when:

- `KafkaController` is requested to `updateLeaderEpochAndSendRequest`, `sendUpdateMetadataRequest` and at `ControlledShutdown` controller event
- `PartitionStateMachine` is requested to `handleStateChanges` and `triggerOnlinePartitionStateChange`
- `ReplicaStateMachine` is requested to `handleStateChanges`

newBatch Method

```
newBatch(): Unit
```

`newBatch` simply throws a `IllegalStateException` if one of the `leaderAndIsrRequestMap`, `stopReplicaRequestMap` and `updateMetadataRequestBrokerSet` internal registries are not empty.

Controller to broker state change requests batch is not empty while creating a new one
. Some LeaderAndIsr state changes [`leaderAndIsrRequestMap`] might be lost

Controller to broker state change requests batch is not empty while creating a new one
. Some StopReplica state changes [`stopReplicaRequestMap`] might be lost

Controller to broker state change requests batch is not empty while creating a new one
. Some UpdateMetadata state changes to brokers [`updateMetadataRequestBrokerSet`] with partition info [`updateMetadataRequestPartitionInfoMap`] might be lost

Note

`newBatch` is used when:

- `KafkaController` is requested to `updateLeaderEpochAndSendRequest`, `sendUpdateMetadataRequest`, and process a `ControlledShutdown` controller event
- `PartitionStateMachine` is requested to `handleStateChanges`
- `PartitionStateMachine` is requested to `handleStateChanges`

addLeaderAndIsrRequestForBrokers Method

```
addLeaderAndIsrRequestForBrokers(
    brokerIds: Seq[Int],
    topicPartition: TopicPartition,
    leaderIsrAndControllerEpoch: LeaderIsrAndControllerEpoch,
    replicas: Seq[Int],
    isNew: Boolean): Unit
```

addLeaderAndIsrRequestForBrokers ...FIXME

Note

- `addLeaderAndIsrRequestForBrokers` is used when:
- `KafkaController` is requested to [updateLeaderEpochAndSendRequest](#)
 - `PartitionStateMachine` is requested to [initializeLeaderAndIsrForPartitions](#) and [doElectLeaderForPartitions](#)
 - `ReplicaStateMachine` is requested to [doHandleStateChanges](#)

addUpdateMetadataRequestForBrokers Method

```
addUpdateMetadataRequestForBrokers(
    brokerIds: Seq[Int],
    partitions: collection.Set[TopicPartition]): Unit
```

addUpdateMetadataRequestForBrokers ...FIXME

Note

- `addUpdateMetadataRequestForBrokers` is used when:
- `ControllerBrokerRequestBatch` is requested to [addLeaderAndIsrRequestForBrokers](#)
 - `KafkaController` is requested to [sendUpdateMetadataRequest](#)

addStopReplicaRequestForBrokers Method

```
addStopReplicaRequestForBrokers(
    brokerIds: Seq[Int],
    topicPartition: TopicPartition,
    deletePartition: Boolean,
    callback: (AbstractResponse, Int) => Unit): Unit
```

addStopReplicaRequestForBrokers ...FIXME

	<p><code>addStopReplicaRequestForBrokers</code> is used when:</p> <ul style="list-style-type: none">• <code>KafkaController</code> is requested to process a <code>ControlledShutdown</code> controller event• <code>ReplicaStateMachine</code> is requested to <code>doHandleStateChanges</code>
--	--

clear Method

```
clear(): Unit
```

`clear` simply removes all bindings in the `leaderAndIsrRequestMap`, `stopReplicaRequestMap` and `updateMetadataRequestBrokerSet` internal registries.

	<p><code>clear</code> is used exclusively when <code>KafkaController</code> is requested to <code>handleIllegalState</code>.</p>
--	--

TopicDeletionManager

`TopicDeletionManager` is...FIXME

`TopicDeletionManager` is created exclusively when `KafkaController` is created.

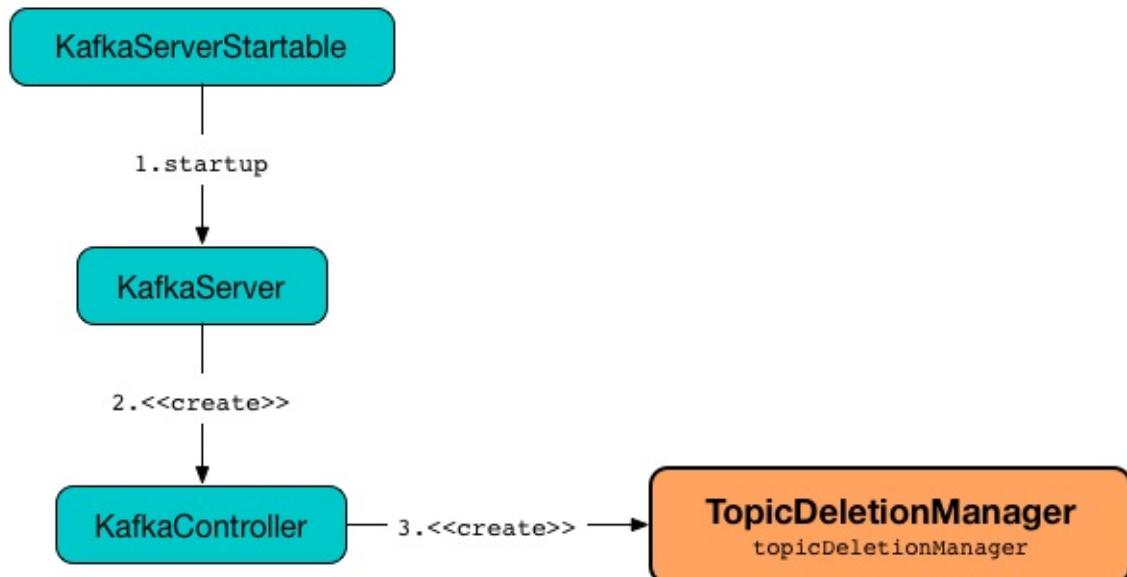


Figure 1. `TopicDeletionManager` is Created Alongside `KafkaController`

`TopicDeletionManager` is controlled by `delete.topic.enable` Kafka property and does nothing when it is turned off (i.e. `false`).

Table 1. `TopicDeletionManager`'s Internal Properties (e.g. Registries and Counters)

Name	Description
<code>partitionsToDelete</code>	<code>TopicAndPartitions</code> to be deleted
<code>topicsToDelete</code>	The names of the topics to be deleted
<code>topicsIneligibleForDeletion</code>	The names of the topics that must not be deleted (i.e. are ineligible for deletion)

`TopicDeletionManager` uses **[Topic Deletion Manager [brokerId]]** as the logging prefix (aka `logIdent`).

Enable `INFO` logging level for `kafka.controller.TopicDeletionManager` logger to see what happens inside.

Add the following line to `config/log4j.properties`:

Tip

```
log4j.logger.kafka.controller.TopicDeletionManager=INFO
```

Refer to [Logging](#).

startReplicaDeletion Internal Method

```
startReplicaDeletion(replicasForTopicsToDelete: Set[PartitionAndReplica]): Unit
```

```
startReplicaDeletion ...FIXME
```

Note

`startReplicaDeletion` is used when...FIXME

enqueueTopicsForDeletion Method

Caution

FIXME

failReplicaDeletion Method

```
failReplicaDeletion(replicas: Set[PartitionAndReplica]): Unit
```

```
failReplicaDeletion ...FIXME
```

Note

`failReplicaDeletion` is used when...FIXME

Creating TopicDeletionManager Instance

`TopicDeletionManager` takes the following when created:

- [KafkaController](#)
- [ControllerEventManager](#)

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

markTopicIneligibleForDeletion Method

```
markTopicIneligibleForDeletion(topics: Set[String]): Unit
```

(only with `delete.topic.enable` Kafka property enabled) `markTopicIneligibleForDeletion` computes the intersection between `topicsToBeDeleted` and the input `topics` sets and adds the intersection to `topicsIneligibleForDeletion` set.

If there are any topics in the intersection, `markTopicIneligibleForDeletion` prints out the following INFO message to the logs:

```
Halted deletion of topics [newTopicsToHaltDeletion]
```

Note

- `markTopicIneligibleForDeletion` is used when:
 - `KafkaController` `initiateReassignReplicasForTopicPartition`
 - `TopicDeletion` controller event is `processed` (for topics to be deleted with partitions in `controllerContext.partitionsBeingReassigned` list)
 - `TopicDeletionManager` does `startReplicaDeletion` and `failReplicaDeletion`

Resetting — `reset` Method

```
reset(): Unit
```

(only with `delete.topic.enable` Kafka property enabled) `reset` removes all elements from the following internal registries:

- `topicsToBeDeleted`
- `partitionsToBeDeleted`
- `topicsIneligibleForDeletion`

Note

`reset` does nothing when `delete.topic.enable` Kafka property is `false`.

Note

`reset` is used exclusively when `KafkaController` resigns as the active controller.

onTopicDeletion Internal Method

```
onTopicDeletion(topics: Set[String]): Unit
```

```
onTopicDeletion ...FIXME
```

Note	onTopicDeletion is used when...FIXME
------	--------------------------------------

completeDeleteTopic Internal Method

```
completeDeleteTopic(topic: String): Unit
```

```
completeDeleteTopic ...FIXME
```

Note	completeDeleteTopic is used when...FIXME
------	--

Initializing — init Method

```
init(  
  initialTopicsToDelete: Set[String],  
  initialTopicsIneligibleForDeletion: Set[String]): Unit
```

```
init ...FIXME
```

Note	init is used when...FIXME
------	---------------------------

tryTopicDeletion Method

```
tryTopicDeletion(): Unit
```

```
tryTopicDeletion ...FIXME
```

Note	tryTopicDeletion is used when...FIXME
------	---------------------------------------

isTopicQueuedUpForDeletion Method

```
isTopicQueuedUpForDeletion(topic: String): Boolean
```

```
isTopicQueuedUpForDeletion ...FIXME
```

Note	isTopicQueuedUpForDeletion is used when...FIXME
------	---

resumeDeletionForTopics Method

```
resumeDeletionForTopics(topics: Set[String] = Set.empty): Unit
```

resumeDeletionForTopics ...FIXME

Note

resumeDeletionForTopics is used when...FIXME

completeReplicaDeletion Method

```
completeReplicaDeletion(replicas: Set[PartitionAndReplica]): Unit
```

completeReplicaDeletion ...FIXME

Note

completeReplicaDeletion is used when...FIXME

markTopicForDeletionRetry Internal Method

```
markTopicForDeletionRetry(topic: String): Unit
```

markTopicForDeletionRetry ...FIXME

Note

markTopicForDeletionRetry is used when...FIXME

ReplicaStateMachine

`ReplicaStateMachine` is [created](#) exclusively for a `KafkaController` that uses it for the following:

- `FIXME`

`ReplicaStateMachine` is [started](#) when `KafkaController` is requested to [onControllerFailover](#) (when `KafkaController` is requested for [controller election](#) and a broker is successfully elected as the controller).

`ReplicaStateMachine` is [shut down](#) when `KafkaController` is requested to [resign as the active controller](#).

`ReplicaStateMachine` uses **[ReplicaStateMachine controllerId=[brokerID]]** as the logging prefix (aka `logIdent`).

Enable `INFO` or `ERROR` logging levels for `kafka.controller.ReplicaStateMachine` logger to see what happens inside.

Add the following line to `log4j.properties` :

```
log4j.logger.kafka.controller.ReplicaStateMachine=INFO
```

Refer to [Logging](#).

Tip Please note that Kafka comes with a preconfigured `kafka.controller` logger in `config/log4j.properties` :

```
log4j.appenders.controllerAppender=org.apache.log4j.DailyRollingFileAppender
log4j.appenders.controllerAppender.DatePattern=''.yyyy-MM-dd-HH
log4j.appenders.controllerAppender.File=${kafka.logs.dir}/controller.log
log4j.appenders.controllerAppender.layout=org.apache.log4j.PatternLayout
log4j.appenders.controllerAppender.layout.ConversionPattern=[%d] %p %m (%c)%n

log4j.logger.kafka.controller=TRACE, controllerAppender
log4j.additivity.kafka.controller=false
```

That means that the logs of `ReplicaStateMachine` go to `logs/controller.log` file at `TRACE` logging level and are not added to the main logs (per `log4j.additivity` being off).

handleStateChanges Method

```
handleStateChanges(
  replicas: Seq[PartitionAndReplica],
  targetState: ReplicaState,
  callbacks: Callbacks = new Callbacks()): Unit
```

handleStateChanges ...FIXME

Note

- `handleStateChanges` is used when:
- `KafkaController` is requested to [onBrokerLogDirFailure](#), [onBrokerStartup](#), [onReplicasBecomeOffline](#), [onNewPartitionCreation](#), [onPartitionReassignment](#), [stopOldReplicasOfReassignedPartition](#), [startNewReplicasForReassignedPartition](#), and process a [ControlledShutdown](#) controller event
 - `ReplicaStateMachine` is requested to [start up](#)
 - `TopicDeletionManager` is requested to [failReplicaDeletion](#), [completeReplicaDeletion](#), [markTopicForDeletionRetry](#), [completeDeleteTopic](#), and [startReplicaDeletion](#)

shutdown Method

`shutdown(): Unit`

shutdown ...FIXME

Note

`shutdown` is used when...FIXME

startup Method

`startup(): Unit`

`startup` prints out the following INFO message to the logs:

Initializing replica state

`startup` [initializeReplicaState](#).

`startup` prints out the following INFO message to the logs:

Triggering online replica state changes

`startup` `handleStateChanges` with the `live replicas` (from the `ControllerContext`) and `OnlineReplica` target state.

In the end, `startup` prints out the following INFO message to the logs:

```
Started replica state machine with initial state -> [replicaState]
```

Note

`startup` is used exclusively when `KafkaController` is requested to `onControllerFailover` (when a `KafkaController` is requested to `elect` and a broker is successfully elected as the controller).

getTopicPartitionStatesFromZk Internal Method

```
getTopicPartitionStatesFromZk(partitions: Seq[TopicPartition])
: (Map[TopicPartition, LeaderAndIsr], Seq[TopicPartition], Map[TopicPartition, Exception])
```

`getTopicPartitionStatesFromZk` ...FIXME

Note

`getTopicPartitionStatesFromZk` is used when...FIXME

initializeReplicaState Internal Method

```
initializeReplicaState(): Unit
```

`initializeReplicaState` ...FIXME

Note

`initializeReplicaState` is used exclusively when `ReplicaStateMachine` is requested to `start up`.

doHandleStateChanges Internal Method

```
doHandleStateChanges(
  replicaId: Int,
  partitions: Seq[TopicPartition],
  targetState: ReplicaState,
  callbacks: Callbacks): Unit
```

`doHandleStateChanges` ...FIXME

Note

`doHandleStateChanges` is used exclusively when `ReplicaStateMachine` is requested to `handleStateChanges`.

Creating ReplicaStateMachine Instance

`ReplicaStateMachine` takes the following to be created:

- `KafkaConfig`
- `StateChangeLogger`
- `ControllerContext`
- `TopicDeletionManager`
- `KafkaZkClient`
- `PartitionAndReplica` With `ReplicaState` (`mutable.Map[PartitionAndReplica, ReplicaState]`)
- `ControllerBrokerRequestBatch`

PartitionStateMachine

`PartitionStateMachine` is created exclusively for `KafkaController` that uses it for the following:

- `FIXME`

Immediately after having been created, `KafkaController` associates it with a `TopicDeletionManager`.

`PartitionStateMachine` is started when `KafkaController` is requested to `onControllerFailover` (when `KafkaController` is requested for controller election and the broker is successfully elected as the active controller).

`PartitionStateMachine` is shut down when `KafkaController` is requested to resign as the active controller.

`PartitionStateMachine` is requested to `triggerOnlinePartitionStateChange` when `KafkaController` is `onBrokerStartup` (when requested to process a `BrokerChange` controller event).

`PartitionStateMachine` uses a `PartitionLeaderElectionStrategy` when `handleStateChanges` (with `doHandleStateChanges`) and `electLeaderForPartitions` (with `doElectLeaderForPartitions`):

- `ControlledShutdownPartitionLeaderElectionStrategy`
- `OfflinePartitionLeaderElectionStrategy`
- `PreferredReplicaPartitionLeaderElectionStrategy`
- `ReassignPartitionLeaderElectionStrategy`

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

Name	Description
<code>controllerId</code>	
<code>topicDeletionManager</code>	<code>TopicDeletionManager</code>
<code>offlinePartitionCount</code>	

`PartitionStateMachine` uses `[PartitionStateMachine controllerId=[brokerID]]` as the logging prefix (aka `logIdent`).

Enable `INFO` or `ERROR` logging levels for `kafka.controller.PartitionStateMachine` logger to see what happens inside.

Add the following line to `log4j.properties`:

```
log4j.logger.kafka.controller.PartitionStateMachine=INFO
```

Refer to [Logging](#).

Tip Please note that Kafka comes with a preconfigured `kafka.controller` logger in `config/log4j.properties`:

```
log4j.appenders.controllerAppender=org.apache.log4j.DailyRollingFileAppender
log4j.appenders.controllerAppender.DatePattern='.' 'yyyy-MM-dd-HH
log4j.appenders.controllerAppender.File=${kafka.logs.dir}/controller.log
log4j.appenders.controllerAppender.layout=org.apache.log4j.PatternLayout
log4j.appenders.controllerAppender.layout.ConversionPattern=[%d] %p %m (%c)%n

log4j.logger.kafka.controller=TRACE, controllerAppender
log4j.additivity.kafka.controller=false
```

That means that the logs of `PartitionStateMachine` go to `logs/controller.log` file at `TRACE` logging level and are not added to the main logs (per `log4j.additivity` being off).

triggerOnlinePartitionStateChange Method

```
triggerOnlinePartitionStateChange(): Unit (1)
triggerOnlinePartitionStateChange(topic: String): Unit (2)
triggerOnlinePartitionStateChange(partitionState: Map[TopicPartition, PartitionState]): Unit
```

1. Uses the `partitionState` for the input `partitionState`
2. Takes `TopicPartitions` from the `partitionState` that match the input `topic` for the input `partitionState`

`triggerOnlinePartitionStateChange` collects the `TopicPartitions` (from the given `PartitionState` per `TopicPartition` collection) that meet the following requirements:

1. `TopicDeletionManager` is not going to [delete their topics](#)
2. `PartitionState` is neither `offlinePartition` nor `NewPartition`

`triggerOnlinePartitionStateChange` [handleStateChanges](#) for the partitions with `OnlinePartition` target state and the `OfflinePartitionLeaderElectionStrategy`.

Note

- `triggerOnlinePartitionStateChange` is used when:
- `KafkaController` is requested to `onBrokerStartup`, `onReplicasBecomeOffline`, and process `UncleanLeaderElectionEnable` and `TopicUncleanLeaderElectionEnable` controller events
 - `PartitionStateMachine` is requested to `start up`

handleStateChanges Method

```
handleStateChanges(
    partitions: Set[TopicAndPartition],
    targetState: PartitionState,
    leaderSelector: PartitionLeaderSelector = noOpPartitionLeaderS...,
    callbacks: Callbacks): Unit
```

`handleStateChanges` ...FIXME

Note

- `handleStateChanges` is used when:
- `KafkaController` is requested to `onReplicasBecomeOffline`, `onNewPartitionCreation`, `onPreferredReplicaElection`, `moveReassignedPartitionLeaderIfRequired` and at `ControlledShutdown` controller event
 - `PartitionStateMachine` is requested to `triggerOnlinePartitionStateChange`
 - `TopicDeletionManager` is requested to `onTopicDeletion`

shutdown Method

```
shutdown(): Unit
```

`shutdown` ...FIXME

Note

`shutdown` is used when...FIXME

Starting Up (On Active Controller) — `startup` Method

```
startup(): Unit
```

`startup` prints out the following INFO message to the logs:

```
Initializing partition state
```

`startup` [initializePartitionState](#).

`startup` prints out the following INFO message to the logs:

```
Triggering online partition state changes
```

`startup` [triggerOnlinePartitionStateChange](#).

In the end, `startup` prints out the following INFO message to the logs:

```
Started partition state machine with initial state -> [partitionState]
```

Note	<code>startup</code> is used exclusively when <code>KafkaController</code> is requested to onControllerFailover (when <code>KafkaController</code> is requested to elect and a broker is successfully elected as the controller).
------	---

doElectLeaderForPartitions Internal Method

```
doElectLeaderForPartitions(  
    partitions: Seq[TopicPartition],  
    partitionLeaderElectionStrategy: PartitionLeaderElectionStrategy)  
: (Seq[TopicPartition], Seq[TopicPartition], Map[TopicPartition, Exception])
```

`doElectLeaderForPartitions ...FIXME`

Note	<code>doElectLeaderForPartitions</code> is used exclusively when <code>PartitionStateMachine</code> is requested to electLeaderForPartitions .
------	--

Creating PartitionStateMachine Instance

`PartitionStateMachine` takes the following when created:

- [KafkaConfig](#)
- [StateChangeLogger](#)
- [ControllerContext](#)
- [KafkaZkClient](#)

- `TopicPartitions` and their `PartitionState` (`mutable.Map[TopicPartition, PartitionState]`)
- `ControllerBrokerRequestBatch`

`PartitionStateMachine` initializes the internal registries and counters.

electLeaderForPartitions Internal Method

```
electLeaderForPartitions(
    partitions: Seq[TopicPartition],
    partitionLeaderElectionStrategy: PartitionLeaderElectionStrategy): Seq[TopicPartition]
```

`electLeaderForPartitions` ...FIXME

Note	<code>electLeaderForPartitions</code> is used exclusively when <code>PartitionStateMachine</code> is requested to <code>doHandleStateChanges</code> .
------	---

doHandleStateChanges Internal Method

```
doHandleStateChanges(
    partitions: Seq[TopicPartition],
    targetState: PartitionState,
    partitionLeaderElectionStrategyOpt: Option[PartitionLeaderElectionStrategy]): Unit
```

`doHandleStateChanges` ...FIXME

Note	<code>doHandleStateChanges</code> is used exclusively when <code>PartitionStateMachine</code> is requested to <code>handleStateChanges</code> .
------	---

initializePartitionState Internal Method

```
initializePartitionState(): Unit
```

`initializePartitionState` requests the `ControllerContext` for all partitions (across all the brokers in the Kafka cluster).

For every `TopicPartition`, `initializePartitionState` requests the `ControllerContext` for the `LeaderIsrAndControllerEpoch` metadata (using the `partitionLeadershipInfo` internal registry).

`initializePartitionState` `changeStateTo` of a `TopicPartition` as follows:

- `OnlinePartition` when the `ControllerContext` says that the `replica` is `online` (for the leader ISR and the `TopicPartition`)
- `OfflinePartition` when the `ControllerContext` says that the `replica` is `not online` (for the leader ISR and the `TopicPartition`)
- `NewPartition` when the `ControllerContext` has no metadata about the `TopicPartition`

Note

`initializePartitionState` is used exclusively when `PartitionStateMachine` is requested to [start up on the active Controller](#).

changeStateTo Internal Method

```
changeStateTo(
    partition: TopicPartition,
    currentState: PartitionState,
    targetState: PartitionState): Unit
```

`changeStateTo ...FIXME`

Note

`changeStateTo` is used when...FIXME

initializeLeaderAndIsrForPartitions Internal Method

```
initializeLeaderAndIsrForPartitions(partitions: Seq[TopicPartition]): Seq[TopicPartition]
```

`initializeLeaderAndIsrForPartitions ...FIXME`

Note

`initializeLeaderAndIsrForPartitions` is used exclusively when `PartitionStateMachine` is requested to [doHandleStateChanges](#).

Partition

A Kafka topic is spread across a Kafka cluster as a virtual group of one or more **partitions**.

A single partition of a topic (**topic partition**) can be replicated across a Kafka cluster to one or more Kafka brokers.

A topic partition has one partition leader node and zero or more replicas.

Kafka producers publish messages to topic leaders as do Kafka consumers consume them from.

In-Sync Replicas (ISR) are brokers that...FIXME

Offline Replicas are...FIXME

`Partition` is an internal data structure that `ReplicaManager` uses to represent a `TopicPartition` (and manage [all partitions](#)).

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

- `ReplicaManager` is requested to [add a TopicPartition to allPartitions](#) (indirectly using [apply](#))
- `ReplicaManager` is requested for the [OfflinePartition](#)

`Partition` is a [KafkaMetricsGroup](#) and registers [performance metrics](#).

Table 1. Partition's Performance Metrics

Metric Name	Description
InSyncReplicasCount	<p>One of the two possible values:</p> <ul style="list-style-type: none"> • Number of <code>inSyncReplicas</code> when <code>isLeaderReplicaLocal</code> • <code>0</code> otherwise
LastStableOffsetLag	
ReplicasCount	<p>One of the two possible values:</p> <ul style="list-style-type: none"> • Number of <code>assignedReplicas</code> when <code>isLeaderReplicaLocal</code> • <code>0</code> otherwise
UnderMinIsr	<p>One of the two possible values:</p> <ul style="list-style-type: none"> • <code>1</code> when <code>isUnderMinIsr</code> • <code>0</code> otherwise
UnderReplicated	<p>One of the two possible values:</p> <ul style="list-style-type: none"> • <code>1</code> when <code>isUnderReplicated</code> • <code>0</code> otherwise

The [performance metrics](#) are registered in **kafka.cluster:type=Partition** group (only when the partition is not [offline](#)).

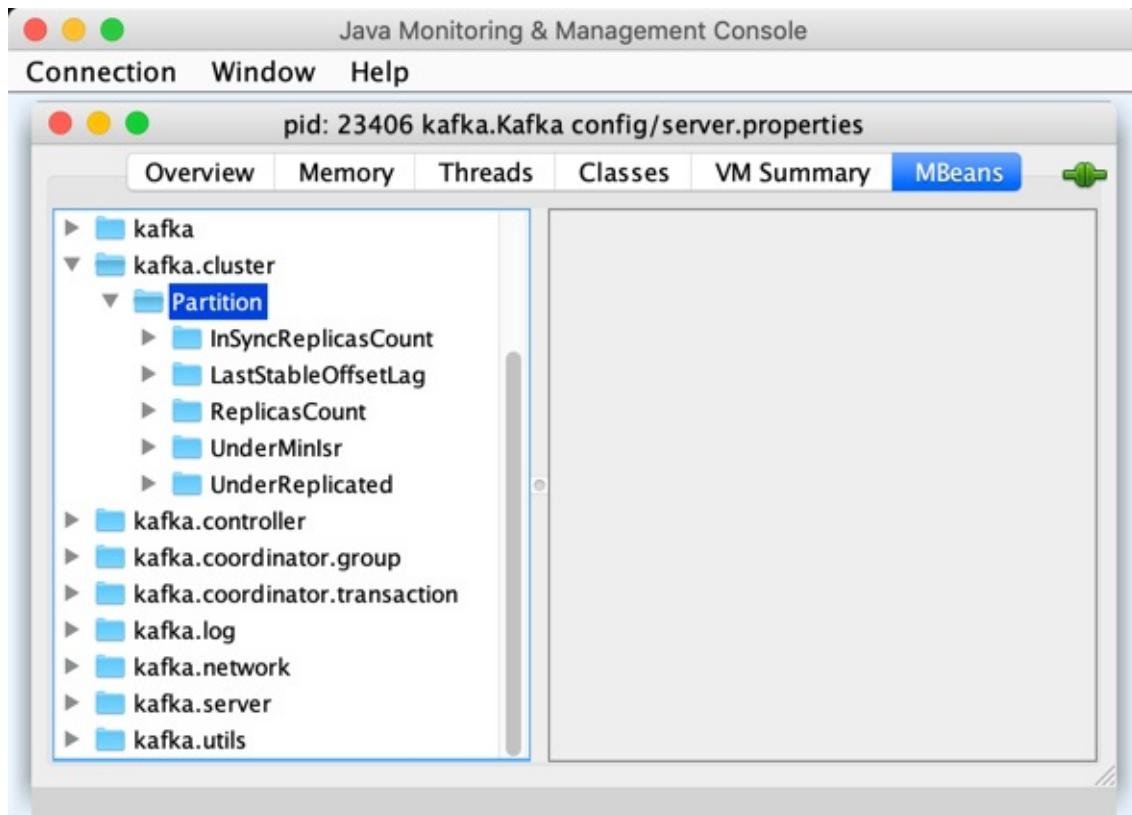


Figure 1. Partition in jconsole

`Partition` is given the `localBrokerId` that is one of the following:

- `broker.id` configuration property (from the `ReplicaManager`) if not `offline`
- `-1` if `offline`

`Partition` uses **[Partition [topicPartition] broker=[localBrokerId]]** as the logging prefix (aka `logIdent`).

Table 2. Partition's Internal Properties (e.g. Registries, Counters and Flags)

Name	Description
inSyncReplicas	In-sync Replicas
leaderReplicaIdOpt	<p>Broker ID of the broker that manages the leader replica Default: <code>None</code> / <code>undefined</code></p> <ul style="list-style-type: none"> Assigned a broker ID when makeLeader and makeFollower <code>None</code> when delete
allReplicasMap	<p>Replicas by ID</p> <ul style="list-style-type: none"> A new entry added in getOrCreateReplica, addReplicaIfNotExists An entry removed in removeReplica, removeFutureLocalReplica, maybeReplaceCurrentWithFutureReplica, and delete <p>Used in getReplica, getLocalReplica, assignedReplicas, allReplicas, toString</p>

maybeExpandIsr Method

```
maybeExpandIsr(replicaId: Int, logReadResult: LogReadResult): Boolean
```

maybeExpandIsr ...FIXME

Note

`maybeExpandIsr` is used exclusively when `Partition` is requested to [updateReplicaLogReadResult](#).

maybeShrinkIsr Method

```
maybeShrinkIsr(replicaMaxLagTimeMs: Long): Unit
```

maybeShrinkIsr ...FIXME

Note

`maybeShrinkIsr` is used exclusively when `ReplicaManager` [maybeShrinkIsr](#).

updateReplicaLogReadResult Method

```
updateReplicaLogReadResult(replica: Replica, logReadResult: LogReadResult): Boolean
```

```
updateReplicaLogReadResult ...FIXME
```

Note

`updateReplicaLogReadResult` is used exclusively when `ReplicaManager` [updateFollowerLogReadResults](#).

updateIsr Internal Method

```
updateIsr(newIsr: Set[Replica]): Unit
```

```
updateIsr ...FIXME
```

Note

`updateIsr` is used when `Partition` is requested to [expand](#) or [shrink](#) the ISR.

makeFollower Method

```
makeFollower(  
    controllerId: Int,  
    partitionStateInfo: LeaderAndIsrRequest.PartitionState,  
    correlationId: Int): Boolean
```

```
makeFollower ...FIXME
```

Note

`makeFollower` is used exclusively when `ReplicaManager` is requested to [makeFollowers](#).

leaderReplicaIfLocal Method

```
leaderReplicaIfLocal: Option[Replica]
```

`leaderReplicaIfLocal` returns a `Replica` when the `leaderReplicaOpt` is the `localBrokerId`. Otherwise, `leaderReplicaIfLocal` returns `None` (i.e. undefined).

Note

`leaderReplicaIfLocal` is used...FIXME

maybeShrinkIsr Method

```
maybeShrinkIsr(replicaMaxLagTimeMs: Long): Unit
```

```
maybeShrinkIsr ...FIXME
```

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

Creating Partition Instance

`Partition` takes the following when created:

- `TopicPartition`
- `isOffline` flag
- `replicaLagTimeMaxMs`
- Local broker ID
- `Time`
- `ReplicaManager`
- `LogManager`
- `KafkaZkClient`

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

isUnderReplicated Predicate

isUnderReplicated: Boolean

`isUnderReplicated` is `true` only if the partition `isLeaderReplicaLocal` and the number of [in-sync replicas](#) is below the `assignedReplicas`.

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

isUnderMinIsr Predicate

isUnderMinIsr: Boolean

`isUnderMinIsr` is `true` only if the partition `isLeaderReplicaLocal` and the number of [in-sync replicas](#) is below the `min.insync.replicas` configuration property (as configured for the [Log](#) of the [leader replica](#)).

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

checkEnoughReplicasReachOffset Method

```
checkEnoughReplicasReachOffset(requiredOffset: Long): (Boolean, Errors)
```

checkEnoughReplicasReachOffset ...FIXME

Note

`checkEnoughReplicasReachOffset` is used when...FIXME

makeLeader Method

```
makeLeader(
    controllerId: Int,
    partitionStateInfo: LeaderAndIsrRequest.PartitionState,
    correlationId: Int): Boolean
```

makeLeader ...FIXME

Note

`makeLeader` is used exclusively when `ReplicaManager` is requested to [makeLeaders](#).

getOrCreateReplica Method

```
getOrCreateReplica(
    replicaId: Int = localBrokerId,
    isNew: Boolean = false): Replica
```

getOrCreateReplica ...FIXME

Note

`getOrCreateReplica` is used when:

- `Partition` is requested to [maybeCreateFutureReplica](#), [makeLeader](#), and [makeFollower](#)
- `ReplicaManager` is requested to [becomeLeaderOrFollower](#) and [makeFollowers](#)

maybeCreateFutureReplica Method

```
maybeCreateFutureReplica(logDir: String): Boolean
```

maybeCreateFutureReplica ...FIXME

Note

`maybeCreateFutureReplica` is used exclusively when `ReplicaManager` is requested to [alterReplicaLogDirs](#).

appendRecordsToLeader Method

```
appendRecordsToLeader(
    records: MemoryRecords,
    isFromClient: Boolean,
    requiredAcks: Int = 0): LogAppendInfo
```

`appendRecordsToLeader` basically requests the `Log` (of the leader `Replica`) to [appendAsLeader](#).

Internally, `appendRecordsToLeader` ...FIXME

Note

`appendRecordsToLeader` is used when:

- `GroupMetadataManager` is requested to [cleanupGroupMetadata](#)
- `ReplicaManager` is requested to [appendToLocalLog](#)

doAppendRecordsToFollowerOrFutureReplica Internal Method

```
doAppendRecordsToFollowerOrFutureReplica(
    records: MemoryRecords,
    isFuture: Boolean): Option[LogAppendInfo]
```

`doAppendRecordsToFollowerOrFutureReplica` ...FIXME

Note

`doAppendRecordsToFollowerOrFutureReplica` is used exclusively when `Partition` is requested to [appendRecordsToFollowerOrFutureReplica](#).

appendRecordsToFollowerOrFutureReplica Method

```
appendRecordsToFollowerOrFutureReplica(
    records: MemoryRecords,
    isFuture: Boolean): Option[LogAppendInfo]
```

`appendRecordsToFollowerOrFutureReplica` ...FIXME

Note

- `appendRecordsToFollowerOrFutureReplica` is used when:
- `ReplicaAlterLogDirsThread` is requested to [processPartitionData](#)
 - `ReplicaFetcherThread` is requested to [processPartitionData](#)

truncateTo Method

```
truncateTo(offset: Long, isFuture: Boolean): Unit
```

`truncateTo` ...FIXME

Note

- `truncateTo` is used when:
- `ReplicaAlterLogDirsThread` is requested to [truncate](#)
 - `ReplicaFetcherThread` is requested to [truncate](#)

truncateFullyAndStartAt Method

```
truncateFullyAndStartAt(newOffset: Long, isFuture: Boolean): Unit
```

`truncateFullyAndStartAt` ...FIXME

Note

- `truncateFullyAndStartAt` is used when:
- `Partition` is requested to [appendRecordsToFollowerOrFutureReplica](#)
 - `ReplicaAlterLogDirsThread` is requested to [truncateFullyAndStartAt](#)
 - `ReplicaFetcherThread` is requested to [truncateFullyAndStartAt](#)

maybeReplaceCurrentWithFutureReplica Method

```
maybeReplaceCurrentWithFutureReplica(): Boolean
```

`maybeReplaceCurrentWithFutureReplica` ...FIXME

Note

`maybeReplaceCurrentWithFutureReplica` is used exclusively when `ReplicaAlterLogDirsThread` is requested to [processPartitionData](#).

delete Method

```
delete(): Unit
```

`delete` ...FIXME

Note	<code>delete</code> is used exclusively when <code>ReplicaManager</code> is requested to stopReplica .
------	--

removeFutureLocalReplica Method

```
removeFutureLocalReplica(deleteFromLogDir: Boolean = true): Unit
```

`removeFutureLocalReplica` ...FIXME

Note	<code>removeFutureLocalReplica</code> is used when <code>ReplicaManager</code> is requested to alterReplicaLogDirs and handleLogDirFailure .
------	--

isLeaderReplicaLocal Internal Method

```
isLeaderReplicaLocal: Boolean
```

`isLeaderReplicaLocal` is positive (`true`) when the [optional Replica](#) is defined. Otherwise, `false`.

Note	<code>isLeaderReplicaLocal</code> is used when <code>ReplicaManager</code> is requested for the performance metrics (InSyncReplicasCount and ReplicasCount), isUnderReplicated , and lowWatermarkIfLeader .
------	---

Creating Partition — apply Factory Method

```
apply(
  topicPartition: TopicPartition,
  time: Time,
  replicaManager: ReplicaManager): Partition
```

`apply` simply [creates a Partition](#) with the given `TopicPartition`, `Time`, and `ReplicaManager` and the following:

- `isOffline` is negative (`false`)
- `replicaLagTimeMaxMs` is `replica.lag.time.max.ms` configuration property
- `localBrokerId` as `broker.id` configuration property

- `LogManager` from the given `ReplicaManager`
- `KafkaZkClient` from the given `ReplicaManager`

Note

`apply` is used exclusively when `ReplicaManager` is requested to add a `TopicPartition` to allPartitions.

localReplicaOrException Method

```
localReplicaOrException: Replica
```

`localReplicaOrException` `localReplica` and returns the local replica if available. Otherwise, `localReplicaOrException` throws a `ReplicaNotAvailableException`:

```
Replica for partition [topicPartition] is not available on broker [localBrokerId]
```

Note

`localReplicaOrException` is used when:

- `Partition` is requested to `maybeCreateFutureReplica`, `maybeReplaceCurrentWithFutureReplica`, `makeLeader`, `doAppendRecordsToFollowerOrFutureReplica`, `appendRecordsToFollowerOrFutureReplica`
- `ReplicaManager` is requested to `localReplicaOrException`, `alterReplicaLogDirs`, `makeFollowers`

localReplica Method

```
localReplica: Option[Replica]
```

`localReplica` simply gets the partition replica for the local broker ID.

Note

`localReplica` is used when:

- `Partition` is requested to `localReplicaOrException` and `leaderReplicaIfLocal`
- `ReplicaManager` is requested to `localReplica`, `becomeLeaderOrFollower`, `checkpointHighWatermarks`, and `handleLogDirFailure`

getReplica Method

```
getReplica(replicaId: Int): Option[Replica]
```

`getReplica` returns the `replica` by the given `replicaId` (in the `allReplicasMap` registry) or `None`.

Note

`getReplica` is used when:

- `Partition` is requested to `localReplica`, `futureLocalReplica`, and `maybeExpandIsr`
- `ReplicaManager` is requested to `shouldLeaderThrottle` and `updateFollowerLogReadResults`

getLocalReplica Internal Method

```
getLocalReplica(  
    replicaId: Int,  
    currentLeaderEpoch: Optional[Integer],  
    requireLeader: Boolean): Either[Replica, Errors]
```

`getLocalReplica` ...FIXME

Note

`getLocalReplica` is used when...FIXME

addReplicaIfNotExists Method

```
addReplicaIfNotExists(replica: Replica): Replica
```

`addReplicaIfNotExists` ...FIXME

Note

`addReplicaIfNotExists` is used when...FIXME

assignedReplicas Method

```
assignedReplicas: Set[Replica]
```

`assignedReplicas` ...FIXME

Note

`assignedReplicas` is used when...FIXME

allReplicas Method

```
allReplicas: Set[Replica]
```

allReplicas ...FIXME

Note	allReplicas is used when...FIXME
------	----------------------------------

removeReplica Internal Method

```
removeReplica(replicaId: Int): Unit
```

removeReplica ...FIXME

Note	removeReplica is used when...FIXME
------	------------------------------------

removeFutureLocalReplica Method

```
removeFutureLocalReplica(deleteFromLogDir: Boolean = true): Unit
```

removeFutureLocalReplica ...FIXME

Note	removeFutureLocalReplica is used when...FIXME
------	---

maybeReplaceCurrentWithFutureReplica Method

```
maybeReplaceCurrentWithFutureReplica(): Boolean
```

maybeReplaceCurrentWithFutureReplica ...FIXME

Note	maybeReplaceCurrentWithFutureReplica is used when...FIXME
------	---

delete Method

```
delete(): Unit
```

delete ...FIXME

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

toString Method

```
toString(): String
```

Note	<code>toString</code> is part of the java.lang.Object Contract for a string representation of the object.
------	---

```
toString ...FIXME
```

Replica

`Replica` (aka **partition replica**) is...FIXME

`Replica` is a **local partition replica** when the replica ID is the **local broker ID**.

`Replica` is a **future local replica** when the replica ID is `-3`.

`Replica` is **created** exclusively when `Partition` is requested to `getOrCreateReplica`.

`Replica` is **local** when the `Log` was defined (at the time the replica was **created**).

Tip	<p>Enable <code>INFO</code> or <code>TRACE</code> logging level for <code>kafka.cluster.Replica</code> logger to see what happens inside.</p> <p>Add the following line to <code>config/tools-log4j.properties</code> :</p> <pre>log4j.logger.kafka.cluster.Replica=TRACE</pre> <p>Refer to Logging.</p>
------------	--

latestEpoch Method

`latestEpoch: Option[Int]`

`latestEpoch` ...FIXME

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

Creating Replica Instance

`Replica` takes the following to be created:

- Broker ID
- `TopicPartition`
- `Time` (default: `SYSTEM`)
- `initialHighWatermarkValue` (default: `0L`)
- Optional `Log` (default: `None`)

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

While being created, `Replica` prints out the following INFO message to the logs and requests the `Log` to `onHighWatermarkIncremented` with the given `initialHighWatermarkValue`.

```
Replica loaded for partition [topicPartition] with initial high watermark [initialHigh  
WatermarkValue]
```

ReplicationUtils

ReplicationUtils ...FIXME

propagateIsrChanges Method

```
propagateIsrChanges(zkUtils: ZkUtils, isrChangeSet: Set[TopicPartition]): Unit
```

propagateIsrChanges ...FIXME

Note	propagateIsrChanges is used exclusively when maybePropagatesIsrChanges .
------	--

KafkaMetricsGroup

`KafkaMetricsGroup` is the abstraction of metrics groups that can register **performance metrics**, e.g. [gauges](#), [histograms](#), [meters](#) and [timers](#).

```
package kafka.metrics

trait KafkaMetricsGroup {
    // No required properties (vals and methods) that have no implementation
}
```

Table 1. KafkaMetricsGroups (Direct Implementations)

KafkaMetricsGroup	Description
AbstractFetcherManager	
Acceptor	
AdminManager	
AppInfo	
BrokerTopicMetrics	Topic metrics
ControllerChannelManager	
ControllerEventManager	
ControllerStats	
DelayedDeleteRecordsMetrics	
DelayedFetchMetrics	
DelayedOperationPurgatory	
DelayedProduceMetrics	
DelegationTokenManager	
FetcherLagMetrics	
FetcherStats	

FetchSessionCache	
GroupMetadataManager	Consumer group metrics
KafkaController	
KafkaRequestHandlerPool	
KafkaServer	
KafkaZkClient	
Log	
LogCleaner	
LogCleanerManager	
LogFlushStats	
LogManager	
Partition	
Processor	
ReplicaManager	
RequestChannel	
RequestMetrics	
SocketServer	
Throttler	
TransactionMarkerChannelManager	
ZooKeeperClient	

newGauge Method

```
newGauge[T](
  name: String,
  metric: Gauge[T],
  tags: scala.collection.Map[String, String] = Map.empty): Gauge[T]
```

`newGauge ...FIXME`

Note

`newGauge` is used when...FIXME

newMeter Method

```
newMeter(
  name: String,
  eventType: String,
  timeUnit: TimeUnit,
  tags: scala.collection.Map[String, String] = Map.empty): Meter
```

`newMeter ...FIXME`

Note

`newMeter` is used when...FIXME

newHistogram Method

```
newHistogram(
  name: String,
  biased: Boolean = true,
  tags: scala.collection.Map[String, String] = Map.empty): Histogram
```

`newHistogram ...FIXME`

Note

`newHistogram` is used when...FIXME

newTimer Method

```
newTimer(
  name: String,
  durationUnit: TimeUnit,
  rateUnit: TimeUnit,
  tags: scala.collection.Map[String, String] = Map.empty): Timer
```

`newTimer ...FIXME`

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

metricName Method

```
metricName(name: String, tags: scala.collection.Map[String, String]): MetricName
```

`metricName` ...FIXME

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

BrokerTopicStats

`BrokerTopicStats` holds a set of topic-related metrics:

- `BrokerTopicMetrics` stats per topic (by topic name)
- Accumulated `BrokerTopicMetrics` for all topics

`BrokerTopicStats` is created exclusively when `KafkaServer` is requested to `start up` (and is used to create the `KafkaApis`, the `ReplicaManager`, the `LogManager`).

BrokerTopicStats, BrokerTopicMetrics and KafkaMetricsGroup

`BrokerTopicStats` holds [one or more BrokerTopicMetrics](#).

`BrokerTopicMetrics` is a `KafkaMetricsGroup` that registers topic-related metrics in `kafka.server:type=BrokerTopicMetrics` group.

Every time `BrokerTopicStats` is updated so are the `BrokerTopicMetrics` of topics. In other words, every update of `BrokerTopicStats` changes `kafka.server:type=BrokerTopicMetrics` group.

Looking Up or Creating New BrokerTopicMetrics — `topicStats` Method

```
topicStats(topic: String): BrokerTopicMetrics
```

`topicStats` looks up or creates a new `BrokerTopicMetrics` for the given topic name.

Note	<p><code>topicStats</code> is used when:</p> <ul style="list-style-type: none"> • <code>Log</code> is requested to <code>append</code> and <code>analyzeAndValidateRecords</code> • <code>KafkaApis</code> is requested to <code>updateRecordConversionStats</code> (for <code>Produce</code> and <code>Fetch</code> requests) • <code>BrokerTopicStats</code> is requested to <code>updateBytesOut</code> • <code>ReplicaManager</code> is requested to <code>appendToLocalLog</code> and <code>readFromLocalLog</code>
-------------	--

updateBytesOut Method

```
updateBytesOut(
    topic: String,
    isFollower: Boolean,
    value: Long): Unit
```

updateBytesOut ...FIXME

Note

`updateBytesOut` is used exclusively when `KafkaApis` is requested to [handle a Fetch request](#).

updateReplicationBytesOut Internal Method

```
updateReplicationBytesOut(value: Long): Unit
```

updateReplicationBytesOut ...FIXME

Note

`updateReplicationBytesOut` is used exclusively when `BrokerTopicStats` is requested to [updateBytesOut](#).

updateReplicationBytesIn Method

```
updateReplicationBytesIn(value: Long): Unit
```

updateReplicationBytesIn ...FIXME

Note

`updateReplicationBytesIn` is used exclusively when `ReplicaFetcherThread` is requested to [processPartitionData](#).

removeMetrics Method

```
removeMetrics(topic: String): Unit
```

removeMetrics ...FIXME

Note

`removeMetrics` is used when `ReplicaManager` is requested to [stopReplica](#) and [handleLogDirFailure](#).

close Method

```
close(): Unit
```

`close` ...FIXME

Note

`close` is used exclusively when `KafkaServer` is requested to [shut down](#).

BrokerTopicMetrics

`BrokerTopicMetrics` is a [KafkaMetricsGroup](#) with [performance metrics](#) of topics.

Table 1. BrokerTopicMetrics's Performance Metrics

Metric Name	Description
<code>BytesInPerSec</code>	
<code>BytesOutPerSec</code>	
<code>BytesRejectedPerSec</code>	
<code>FailedFetchRequestsPerSec</code>	
<code>FailedProduceRequestsPerSec</code>	
<code>FetchMessageConversionsPerSec</code>	
<code>MessagesInPerSec</code>	
<code>ProduceMessageConversionsPerSec</code>	
<code>ReplicationBytesInPerSec</code>	(only when topic name is undefined)
<code>ReplicationBytesOutPerSec</code>	(only when topic name is undefined)
<code>TotalFetchRequestsPerSec</code>	
<code>TotalProduceRequestsPerSec</code>	

The [performance metrics](#) are registered in **kafka.server:type=BrokerTopicMetrics** group.

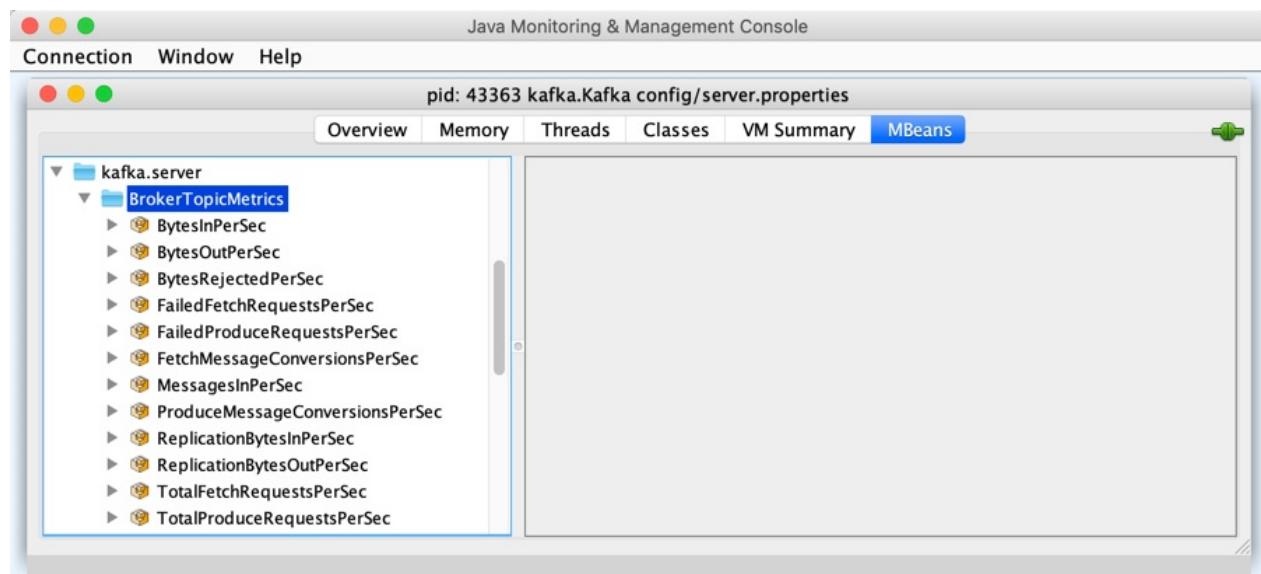


Figure 1. BrokerTopicMetrics in jconsole

`BrokerTopicMetrics` is [created](#) for `BrokerTopicStats` when requested to [look up or create a new BrokerTopicMetrics](#) for a [topic](#) or accumulated metrics for all topics.

`BrokerTopicMetrics` takes an optional **topic name** to be created. When not specified, `BrokerTopicMetrics` is assumed to be used for [all topics](#).

`BrokerTopicMetrics` uses metric tags of `topic` and the [topic name](#) (if defined).

ConfigCommand Standalone Application

`kafka.admin.ConfigCommand` is a [standalone application](#) that...FIXME

`ConfigCommand` can be executed using `kafka-configs` shell script (i.e. `bin/kafka-configs.sh` OR `bin\windows\kafka-configs.bat`).

```
// Triggers alterBrokerConfig
./bin/kafka-configs.sh \
--bootstrap-server :9092 \
--alter \
--entity-type brokers \
--entity-name 0 \
--add-config advertised.listeners=plaintext://:9092
```

```
// Triggers processBrokerConfig
./bin/kafka-configs.sh \
--bootstrap-server :9092 \
--describe \
--entity-type brokers \
--entity-name 0
```

alterBrokerConfig Method

```
alterBrokerConfig(
    adminClient: JAdminClient,
    opts: ConfigCommandOptions,
    entityName: String)
```

alterBrokerConfig ...FIXME

Note	<code>alterBrokerConfig</code> is used exclusively when <code>ConfigCommand</code> is requested to processBrokerConfig (with <code>alter</code> action).
------	--

processBrokerConfig Internal Method

```
processBrokerConfig(opts: ConfigCommandOptions): Unit
```

processBrokerConfig ...FIXME

Note

`processBrokerConfig` is used exclusively when `ConfigCommand` standalone application is [executed](#) (without `zookeeper` command-line option).

Executing Standalone Application — `main` Method

```
main(args: Array[String]): Unit
```

`main` is the entry point of the [ConfigCommand](#) standalone application when launched on command line (e.g. from `bin/kafka-configs.sh`).

```
main ...FIXME
```

`processCommandWithZk` Internal Method

```
processCommandWithZk(zkConnectionString: String, opts: ConfigCommandOptions): Unit
```

```
processCommandWithZk ...FIXME
```

Note

`processCommandWithZk` is used when...FIXME

`alterConfig` Internal Method

```
alterConfig(
  zkClient: KafkaZkClient,
  opts: ConfigCommandOptions,
  adminZkClient: AdminZkClient): Unit
```

```
alterConfig ...FIXME
```

Note

`alterConfig` is used when...FIXME

ReassignPartitionsCommand — Partition Reassignment on Command Line

`kafka.admin.ReassignPartitionsCommand` is a command-line application that allows for generating, executing and verifying a custom partition (re)assignment configuration (as specified using a reassignment JSON file).

Table 1. ReassignPartitionsCommand's Actions

Action	Description
--execute	Executes the reassignment as specified by the reassignment-json-file option
--generate	Generates a candidate partition reassignment configuration <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> Note This only generates a candidate assignment, it does not execute it. </div>
--verify	Verifies if the reassignment completed as specified by the reassignment-json-file option. If there is a throttle engaged for the replicas specified, and the rebalance has completed, the throttle will be removed

ReassignPartitionsCommand can be executed using `kafka-reassign-partitions` shell script (i.e. `bin/kafka-reassign-partitions.sh` or `bin\windows\kafka-reassign-partitions.bat`).

```
$ ./bin/kafka-reassign-partitions.sh
This command moves topic partitions between replicas.
```

Table 2. ReassignPartitionsCommand's Options

Option	Description
	A JSON file with a custom partition (re)assignment configuration The format to use is as follows:

`reassignment-json-file`

```
{
  "partitions": [
    {
      "topic": "foo",
      "partition": 1,
      "replicas": [
        1,
        2,
        3
      ],
      "log_dirs": [
        "dir1",
        "dir2",
        "dir3"
      ]
    }
  ],
  "version": 1
}
```

Note that "log_dirs" is optional. When specified, its length must equal the length of the replicas list. The value in this list can be either "any" or the absolute path of the log directory on the broker.

If absolute log directory path is specified, it is currently required that the replica has not already been created on that broker. The replica will then be created in the specified log directory on the broker later.

`replica-alter-log-dirs-throttle`

The movement of replicas between log directories on the same broker will be throttled to this value (bytes/sec).

Default: -1

Rerunning with this option, whilst a rebalance is in progress, will alter the throttle value. The throttle rate should be at least 1 KB/s.

`throttle`

The movement of partitions between brokers will be throttled to this value (bytes/sec).

Default: -1

Rerunning with this option, whilst a rebalance is in progress, will alter the throttle value. The throttle rate should be at least 1 KB/s.

`timeout`

The maximum time (in ms) allowed to wait for partition reassignment execution to be successfully initiated

Default: 10000

```
$ ./bin/kafka-topics.sh --list --zookeeper :2181
my-topic

$ cat reassign-partitions.json
{
  "partitions": [
    {
      "topic": "my-topic",
      "partition": 1,
      "replicas": [
        1
      ]
    }
  ],
  "version": 1
}

$ ./bin/kafka-reassign-partitions.sh \
--generate \
--zookeeper :2181 \
--topics-to-move-json-file reassign-partitions.json \
--broker-list 0

$ ./bin/kafka-reassign-partitions.sh \
--verify \
--zookeeper :2181 \
--reassignment-json-file reassign-partitions.json
```

`ReassignPartitionsCommand` is [created](#) exclusively when `ReassignPartitionsCommand` is requested to [executeAssignment](#).

Executing Standalone Application — `main` Method

```
main(args: Array[String]): Unit
```

```
main ...FIXME
```

`executeAssignment` Method

```
executeAssignment(
    zkClient: KafkaZkClient,
    adminClientOpt: Option[JAdminClient],
    opts: ReassignPartitionsCommandOptions): Unit (1)
executeAssignment(
    zkClient: KafkaZkClient,
    adminClientOpt: Option[JAdminClient],
    reassignmentJsonString: String,
    throttle: Throttle,
    timeoutMs: Long = 10000L): Unit
```

1. Uses `options` for `reassignmentJsonString`, `throttle` and `timeoutMs` inputs

`executeAssignment ...FIXME`

Note

`executeAssignment` is used exclusively when `ReassignPartitionsCommand` is [executed](#) (with `execute` action).

reassignPartitions Method

```
reassignPartitions(throttle: Throttle = NoThrottle, timeoutMs: Long = 10000L): Boolean
```

`reassignPartitions ...FIXME`

Note

`reassignPartitions` is used exclusively when `ReassignPartitionsCommand` [executeAssignment](#).

alterReplicaLogDirsIgnoreReplicaNotAvailable Internal Method

```
alterReplicaLogDirsIgnoreReplicaNotAvailable(
    replicaAssignment: Map[TopicPartitionReplica, String],
    adminClient: JAdminClient,
    timeoutMs: Long): Set[TopicPartitionReplica]
```

`alterReplicaLogDirsIgnoreReplicaNotAvailable ...FIXME`

Note

`alterReplicaLogDirsIgnoreReplicaNotAvailable` is used exclusively when `ReassignPartitionsCommand` [reassignPartitions](#)

generateAssignment Method

```
generateAssignment(
  zkClient: KafkaZkClient,
  brokerListToReassign: Seq[Int],
  topicsToMoveJsonString: String,
  disableRackAware: Boolean)
: (Map[TopicPartition, Seq[Int]], Map[TopicPartition, Seq[Int]])
```

generateAssignment ...FIXME

Note

generateAssignment is used when...FIXME

verifyAssignment Method

```
verifyAssignment(
  zkClient: KafkaZkClient,
  adminClientOpt: Option[JAdminClient],
  jsonString: String): Unit
```

verifyAssignment ...FIXME

Note

verifyAssignment is used when...FIXME

parseAndValidate Method

```
parseAndValidate(
  zkClient: KafkaZkClient,
  reassignmentJsonString: String)
: (Seq[(TopicPartition, Seq[Int])], Map[TopicPartitionReplica, String])
```

parseAndValidate ...FIXME

Note

parseAndValidate is used when...FIXME

removeThrottle Method

```
removeThrottle(
  zkClient: KafkaZkClient,
  reassignedPartitionsStatus: Map[TopicPartition, ReassignmentStatus],
  replicasReassignmentStatus: Map[TopicPartitionReplica, ReassignmentStatus],
  adminZkClient: AdminZkClient): Unit
```

removeThrottle ...FIXME

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

existingAssignment Method

```
existingAssignment(): Map[TopicPartition, Seq[Int]]
```

`existingAssignment` takes the topics (from the keys) from the [proposedPartitionAssignment](#) and requests the [KafkaZkClient](#) to [getReplicaAssignmentForTopics](#).

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

Creating ReassignPartitionsCommand Instance

`ReassignPartitionsCommand` takes the following when created:

- [KafkaZkClient](#)
- Optional [AdminClient](#)
- Proposed partition assignment (`Map[TopicPartition, Seq[Int]]`)
- Proposed replica assignment (`Map[TopicPartitionReplica, String]`)
- [AdminZkClient](#)

TopicCommand — Topic Management on Command Line

`kafka.admin.TopicCommand` is a command-line application that can [alter](#), [create](#), [delete](#), [describe](#) and [list](#) topics in a Kafka cluster.

Table 1. TopicCommand's Actions

Action	Description
--alter	Alters the number of partitions, replica assignment, and/or configuration of a topic or topics
--create	Creates a new topic
--delete	Deletes a topic or topics
--describe	Describes a topic or topics
--list	Lists available topics

TopicCommand can be executed using `kafka-topics` shell script (i.e. `bin/kafka-topics.sh` or `bin\windows\kafka-topics.bat`).

```
$ ./bin/kafka-topics.sh  
Create, delete, describe, or change a topic.
```

Table 2. TopicCommand's Options

Option	Description			
config	A topic configuration override for the topic being worked on (as <code>name=value</code>)			
disable-rack-aware	Disable rack aware replica assignment			
if-not-exists	When creating a topic(s), the action will only execute if the topic does not already exist			
partitions	<p>The number of partitions of the topic being created or altered</p> <table border="1"> <tr> <td style="text-align: center;">Warning</td> <td>If partitions are increased for a topic that has a key, the partition logic or ordering of the messages will be affected.</td> </tr> </table>		Warning	If partitions are increased for a topic that has a key, the partition logic or ordering of the messages will be affected.
Warning	If partitions are increased for a topic that has a key, the partition logic or ordering of the messages will be affected.			
replica-assignment	A list of manual partition-to-broker assignments for the topic being created or altered			
replication-factor	The replication factor for each partition of the topic being created			
topic	A topic or topics to be altered , created , deleted , or listed and described (through <code>getTopics</code>)			

```
$ ./bin/kafka-topics.sh \
--create \
--zookeeper :2181 \
--replication-factor 1 \
--partitions 1 \
--topic my-topic

$ ./bin/kafka-topics.sh --list --zookeeper :2181
my-topic

$ ./bin/kafka-topics.sh \
--zookeeper :2181 \
--describe \
--topic my-topic
Topic:my-topic PartitionCount:1 ReplicationFactor:1 Configs:
Topic: my-topic Partition: 0 Leader: 0 Replicas: 0 Isr: 0
```

alterTopic Method

```
alterTopic(zkClient: KafkaZkClient, opts: TopicCommandOptions): Unit
```

`alterTopic` ...FIXME

Note

`alterTopic` is used when...FIXME

Creating Topic— `createTopic` Method

```
createTopic(zkClient: KafkaZkClient, opts: TopicCommandOptions): Unit
```

`createTopic` takes the values of [topic](#), [config](#) and [if-not-exists](#) options.

`createTopic` creates a new [AdminZkClient](#) (with the input [KafkaZkClient](#)).

`createTopic` branches off per [replica-assignment](#) option's existence:

- With [replica-assignment](#), `createTopic` requests the [AdminZkClient](#) to [createOrUpdateTopicPartitionAssignmentPathInZK](#) (with the [update](#) flag off)
- With no [replica-assignment](#), `createTopic` checks the required arguments, i.e. [partitions](#) and [replication-factor](#), and together with [disable-rack-aware](#) flag requests the [AdminZkClient](#) to [create the topic](#)

In the end, `createTopic` prints out the following:

```
Created topic "[topic]"
```

Note

`createTopic` is used exclusively when `TopicCommand` is executed with [--create](#) action.

deleteTopic Method

```
deleteTopic(zkClient: KafkaZkClient, opts: TopicCommandOptions): Unit
```

`deleteTopic` ...FIXME

Note

`deleteTopic` is used when...FIXME

describeTopic Method

```
describeTopic(zkClient: KafkaZkClient, opts: TopicCommandOptions): Unit
```

`describeTopic` ...FIXME

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

listTopics Method

```
listTopics(zkClient: KafkaZkClient, opts: TopicCommandOptions): Unit
```

`listTopics` ...FIXME

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

Executing Standalone Application — main Method

```
main(args: Array[String]): Unit
```

`main` is the entry point of the [TopicCommand](#) standalone application when launched on command line (e.g. from `bin/kafka-topics.sh`).

`main` creates a [TopicCommandOptions](#).

`main` prints out the following message and exits when no command-line arguments were given:

```
Create, delete, describe, or change a topic.
```

`main` prints out the following message and exits when no [action](#) was specified:

```
Command must include exactly one action: --list, --describe, --create, --alter or --delete
```

`main` makes sure the action and the options were all valid.

`main` creates a new [KafkaZkClient](#) (with the `zookeeper` option).

`main` branches off per the [action](#).

In case of any exception, `main` prints out the following and exits with `1` exit code:

```
Error while executing topic command : [message]
```

getTopics Internal Method

```
getTopics(zkClient: KafkaZkClient, opts: TopicCommandOptions): Seq[String]
```

```
getTopics ...FIXME
```

Note

`getTopics` is used when `TopicCommand` is requested to [alterTopic](#), [listTopics](#), [deleteTopic](#), [describeTopic](#).

kafka-consumer-groups.sh Shell Script

`kafka-consumer-groups.sh` is a shell script that...FIXME

ConsumerGroupCommand

`ConsumerGroupCommand` is a [standalone command-line application](#) that is used for the following actions:

- Listing all consumer groups (`--list` option)
- Describing a consumer group (`--describe` option)
- Resetting consumer group offsets (`--reset-offsets` option)
- (only for the old Zookeeper-based consumer API) Deleting consumer group info (`--delete` option)

`ConsumerGroupCommand` can be [executed as `kafka-consumer-groups.sh` shell script](#).

Quoting Kafka 0.9.0.0:

Note

The `kafka-consumer-offset-checker.sh` (`kafka.tools.ConsumerOffsetChecker`) has been deprecated. Going forward, please use `kafka-consumer-groups.sh` (`kafka.admin.ConsumerGroupCommand`) for this functionality.

main Method

```
main(args: Array[String]): Unit
```

`main` parses and checks the command-line arguments (using `ConsumerGroupCommandOptions`).

`main` creates a [KafkaConsumerGroupService](#).

Note

`main` creates the old `zkConsumerGroupService` when `--zookeeper` option for the old client API is used.

`main` branches per option used.

- `--list` option
- Describing a consumer group (`--describe` option)
- Resetting consumer group offsets (`--reset-offsets` option)

Caution

FIXME

list Option

`main` simply [request groups](#) from the consumer group service (e.g. `KafkaConsumerGroupService` for the new consumer API) and prints them out to the console.

KafkaConsumerGroupService

`KafkaConsumerGroupService` is a [ConsumerGroupService](#) that [ConsumerGroupCommand](#) uses for [listing](#), [describing](#) and [resetting offsets](#) of consumer groups.

`KafkaConsumerGroupService` is [created](#) exclusively when `ConsumerGroupCommand` is [executed](#) (as a standalone command-line application, i.e. using `kafka-consumer-groups.sh` shell script).

`KafkaConsumerGroupService` is used by [ConsumerGroupCommand](#) for consumer groups that use the new Java consumer API (and hence do not use Zookeeper to store information).

`KafkaConsumerGroupService` uses [AdminClient](#) for the actions.

Listing All Consumer Groups — `listGroups` Method

```
listGroups(): List[String]
```

Note	<code>listGroups</code> is a part of ConsumerGroupService Contract .
------	--

`listGroups` requests [AdminClient](#) for [all consumer groups](#) and takes their group ids.

Note	<code>listGroups</code> is used exclusively when <code>ConsumerGroupCommand</code> is requested for all consumer groups using <code>--list</code> option.
------	---

`describeGroup` Method

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

`resetOffsets` Method

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

Creating KafkaConsumerGroupService Instance

`KafkaConsumerGroupService` takes the following when created:

- `ConsumerGroupCommandOptions`

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

prepareOffsetsToReset Method

FIXME

prepareOffsetsToReset ...FIXME

Note	prepareOffsetsToReset is used when...FIXME
------	--

getPartitionsToReset Method

FIXME

getPartitionsToReset ...FIXME

Note	getPartitionsToReset is used when...FIXME
------	---

collectGroupAssignment Method

FIXME

collectGroupAssignment ...FIXME

Note	collectGroupAssignment is used when...FIXME
------	---

ConsumerGroupService

ConsumerGroupService is...FIXME

KafkaAdminClient

`KafkaAdminClient` is a [AdminClient](#) that...FIXME

`KafkaAdminClient` is created when...FIXME

describeTopics Method

```
DescribeTopicsResult describeTopics(  
    final Collection<String> topicNames,  
    DescribeTopicsOptions options)
```

`describeTopics` ...FIXME

Note

`describeTopics` is used when...FIXME

alterReplicaLogDirs Method

```
AlterReplicaLogDirsResult alterReplicaLogDirs(  
    Map<TopicPartitionReplica, String> replicaAssignment,  
    final AlterReplicaLogDirsOptions options)
```

`alterReplicaLogDirs` ...FIXME

Note

`alterReplicaLogDirs` is used when...FIXME

Creating KafkaAdminClient Instance

`KafkaAdminClient` takes the following when created:

- `AdminClientConfig`
- client ID
- sanitized client ID
- `Time`
- [Metadata](#)
- [Metrics](#)
- [KafkaClient](#)

- `TimeoutProcessorFactory`
- `LogContext`

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

alterConfigs Method

```
AlterConfigsResult alterConfigs(
    Map<ConfigResource, Config> configs,
    final AlterConfigsOptions options)
// Private API
Map<ConfigResource, KafkaFutureImpl<Void>> alterConfigs(
    Map<ConfigResource, Config> configs,
    final AlterConfigsOptions options,
    Collection<ConfigResource> resources,
    NodeProvider nodeProvider)
```

Note	<code>alterConfigs</code> is part of the AdminClient Contract to...FIXME.
------	---

`alterConfigs` ...FIXME

describeCluster Method

```
DescribeClusterResult describeCluster(DescribeClusterOptions options)
```

Note	<code>describeCluster</code> is part of the AdminClient Contract to...FIXME.
------	--

`describeCluster` ...FIXME

deleteRecords Method

```
DeleteRecordsResult deleteRecords(
    final Map<TopicPartition, RecordsToDelete> recordsToDelete,
    final DeleteRecordsOptions options)
```

Note	<code>deleteRecords</code> is part of the AdminClient Contract to...FIXME.
------	--

`deleteRecords` ...FIXME

listConsumerGroups Method

```
ListConsumerGroupsResult listConsumerGroups(ListConsumerGroupsOptions options)
```

Note	listConsumerGroups is part of the AdminClient Contract to...FIXME.
------	--

listConsumerGroups ...FIXME

listTopics Method

```
ListTopicsResult listTopics(final ListTopicsOptions options)
```

Note	listTopics is part of the AdminClient Contract to...FIXME.
------	--

listTopics ...FIXME

describeConsumerGroups Method

```
DescribeConsumerGroupsResult describeConsumerGroups(  
    final Collection<String> groupIds,  
    final DescribeConsumerGroupsOptions options)
```

Note	describeConsumerGroups is part of the AdminClient Contract to...FIXME.
------	--

describeConsumerGroups ...FIXME

listConsumerGroupOffsets Method

```
ListConsumerGroupOffsetsResult listConsumerGroupOffsets(  
    final String groupId,  
    final ListConsumerGroupOffsetsOptions options)
```

Note	listConsumerGroupOffsets is part of the AdminClient Contract to...FIXME.
------	--

listConsumerGroupOffsets ...FIXME

deleteConsumerGroups Method

```
DeleteConsumerGroupsResult deleteConsumerGroups(  
    Collection<String> groupIds,  
    DeleteConsumerGroupsOptions options)
```

Note	<code>deleteConsumerGroups</code> is part of the AdminClient Contract to...FIXME.
------	---

`deleteConsumerGroups` ...FIXME

createTopics Method

```
CreateTopicsResult createTopics(  
    final Collection<NewTopic> newTopics,  
    final CreateTopicsOptions options)
```

Note	<code>createTopics</code> is part of the AdminClient Contract to...FIXME.
------	---

`createTopics` ...FIXME

deleteTopics Method

```
DeleteTopicsResult deleteTopics(  
    Collection<String> topicNames,  
    DeleteTopicsOptions options)
```

Note	<code>deleteTopics</code> is part of the AdminClient Contract to...FIXME.
------	---

`deleteTopics` ...FIXME

createPartitions Method

```
CreatePartitionsResult createPartitions(  
    Map<String, NewPartitions> newPartitions,  
    final CreatePartitionsOptions options)
```

Note	<code>createPartitions</code> is part of the AdminClient Contract to...FIXME.
------	---

`createPartitions` ...FIXME

AdminClient

`AdminClient ...FIXME`

`AdminClient` uses the [admin-client-network-thread](#) to poll continuously (using [ConsumerNetworkClient](#)).

Table 1. ConsumerNetworkClient's Internal Properties (e.g. Registries and Counters)

Name	Description
<code>networkThread</code>	admin-client-network-thread

Tip	Enable <code>ALL</code> logging level for <code>kafka.admin.AdminClient</code> logger to see what happens inside. Add the following line to <code>config/tools-log4j.properties</code> : <code>log4j.logger.kafka.admin.AdminClient=ALL</code> Refer to Logging .

Creating AdminClient— `create` Method

`create(config: AdminConfig): AdminClient`

`create ...FIXME`

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

Sending Request (Using ConsumerNetworkClient) — `send` Internal Method

```
send(
  target: Node,
  api: ApiKeys,
  request: AbstractRequest.Builder[_ <: AbstractRequest]): AbstractResponse
```

`send` requests [ConsumerNetworkClient](#) to `send` the input `request` to the `target`.



Figure 1. AdminClient Sends Requests Using ConsumerNetworkClient
`send` records the future result in `pendingFutures` registry.

`send` waits until the future result has come after which it is removed from `pendingFutures` registry.

Caution

`FIXME Why is the future result registered in pendingFutures ?`

When the future result has completed, `send` takes the response body for a successful result or reports a `RuntimeException`.

Note

`send` is used when `AdminClient` does `sendAnyNode`, `listGroups`, `getApiVersions`, `listGroupOffsets` and `describeConsumerGroupHandler`.

findAllBrokers Method

`findAllBrokers(): List[Node]`

`findAllBrokers` creates a `Metadata` API request and sends it to one of the bootstrap brokers.

`findAllBrokers` returns the nodes from the `cluster metadata` of the `MetadataResponse`.

Note

`findAllBrokers` is used when `AdminClient` does `awaitBrokers`, `lists all groups per broker` and `listAllBrokerVersionInfo`.

Sending API Request to Bootstrap Broker — sendAnyNode Internal Method

`sendAnyNode(api: ApiKeys, request: AbstractRequest.Builder[_ <: AbstractRequest]): AbstractResponse`

`sendAnyNode` walks over `bootstrapBrokers` and sends the input `request`.

`sendAnyNode` exits in case of `AuthenticationException`.

In case of any other `Exceptions` (but `AuthenticationException`) `sendAnyNode` prints DEBUG message to the logs and tries the remaining brokers.

```
Request [api] failed against node [broker]
```

When no brokers succeeded, `sendAnyNode` reports a `RuntimeException` with the following message:

```
Request [api] failed on brokers [bootstrapBrokers]
```

Note

`sendAnyNode` is used when `AdminClient` is requested to [find the coordinator](#), [finds all brokers](#) and [deleteRecordsBefore](#).

findCoordinator Method

```
FIXME
```

```
findCoordinator ...FIXME
```

Note

`findCoordinator` is used when `KafkaConsumerGroupService` ...`FIXME`

deleteRecordsBefore Method

```
FIXME
```

```
deleteRecordsBefore ...FIXME
```

Note

`deleteRecordsBefore` is used when `KafkaConsumerGroupService` ...`FIXME`

listGroups Method

```
FIXME
```

```
listGroups ...FIXME
```

Note

`listGroups` is used when `KafkaConsumerGroupService` ...`FIXME`

listAllBrokerVersionInfo Method

```
FIXME
```

```
listAllBrokerVersionInfo ...FIXME
```

Note	listAllBrokerVersionInfo is used when KafkaConsumerGroupService ...FIXME
------	--

awaitBrokers Method

```
FIXME
```

```
awaitBrokers ...FIXME
```

Note	awaitBrokers is used when KafkaConsumerGroupService ...FIXME
------	--

listAllConsumerGroups Method

```
FIXME
```

```
listAllConsumerGroups ...FIXME
```

Note	listAllConsumerGroups is used when KafkaConsumerGroupService ...FIXME
------	---

Listing All Groups per Broker — listAllGroups Method

```
listAllGroups(): Map[Node, List[GroupOverview]]
```

listAllGroups finds all brokers (in a cluster) and collects their groups.

Note	listAllGroups is used when AdminClient does listAllConsumerGroups and listAllGroupsFlattened .
------	--

listAllGroupsFlattened Method

```
listAllGroupsFlattened(): List[GroupOverview]
```

listAllGroupsFlattened simply takes all groups (across all brokers in a cluster).

Note	listAllGroupsFlattened is used exclusively when AdminClient lists all consumer groups.
------	--

Listing All Consumer Groups

— `listAllConsumerGroupsFlattened` Method

```
listAllConsumerGroupsFlattened(): List[GroupOverview]
```

`listAllConsumerGroupsFlattened` takes [all groups](#) with [consumer](#) protocol type.

Note

`listAllConsumerGroupsFlattened` is used exclusively when `KafkaConsumerGroupService` is requested for [all consumer groups](#).

listGroupOffsets Method

```
FIXME
```

`listGroupOffsets` ...[FIXME](#)

Note

`listGroupOffsets` is used when `KafkaConsumerGroupService` ...[FIXME](#)

AdminMetadataManager

AdminMetadataManager is...FIXME

clearController Method

```
void clearController()
```

clearController ...FIXME

Note	clearController is used when KafkaAdminClient is requested to createTopics , deleteTopics , and createPartitions (and Errors.NOT_CONTROLLER error is received).
------	---

AdminClientRunnable

AdminClientRunnable is...FIXME

sendEligibleCalls Internal Method

```
long sendEligibleCalls(long now)
```

sendEligibleCalls ...FIXME

Note	sendEligibleCalls is used exclusively when AdminClientRunnable is requested to run.
------	---

Starting Thread of Execution — run Method (of Java's Runnable)

```
void run()
```

Note	run is a part of java.lang.Runnable that is executed when the thread is started.
------	--

run ...FIXME

timeoutCallsInFlight Internal Method

```
void timeoutCallsInFlight(TimeoutProcessor processor)
```

timeoutCallsInFlight ...FIXME

Note	timeoutCallsInFlight is used when...FIXME
------	---

handleResponses Internal Method

```
void handleResponses(long now, List<ClientResponse> responses)
```

handleResponses ...FIXME

Note	handleResponses is used when...FIXME
------	--------------------------------------

AdminUtils Helper Object

`AdminUtils` is an utility (a Scala object) with [methods](#) for...FIXME

Table 1. AdminUtils API

Name	Description
<code>assignReplicasToBrokers</code>	<code>assignReplicasToBrokers(brokerMetadatas: Seq[BrokerMetadata], nPartitions: Int, replicationFactor: Int, fixedStartIndex: Int = -1, startPartitionId: Int = -1): Map[Int, Seq[Int]]</code>

`AdminUtils` uses `__admin_client` for...FIXME

assignReplicasToBrokers Method

```
assignReplicasToBrokers(  
  brokerMetadatas: Seq[BrokerMetadata],  
  nPartitions: Int,  
  replicationFactor: Int,  
  fixedStartIndex: Int = -1,  
  startPartitionId: Int = -1): Map[Int, Seq[Int]]
```

`assignReplicasToBrokers` branches off per whether all the brokers have rack information or not:

- `assignReplicasToBrokers assignReplicasToBrokersRackUnaware` when all the brokers (in `brokerMetadatas`) have no rack information available.
- `assignReplicasToBrokers assignReplicasToBrokersRackAware` when all the brokers (in `brokerMetadatas`) have rack information available.

`assignReplicasToBrokers` throws a `InvalidPartitionsException` if the given `nPartitions` is `0` or less.

Number of partitions must be larger than 0.

`assignReplicasToBrokers` throws a `InvalidReplicationFactorException` if the given `replicationFactor` is `0` or less.

Replication factor must be larger than 0.

`assignReplicasToBrokers` throws a `InvalidReplicationFactorException` if the given `replicationFactor` is greater than the number of all brokers in the cluster.

Replication factor: [replicationFactor] larger than available brokers: [brokerMetadata.s.size].

`assignReplicasToBrokers` throws an `AdminOperationException` if there is at least one broker (in `brokerMetadatas`) with no rack information (when it is assumed either all brokers have it or none).

Not all brokers have rack information for replica rack aware assignment.

Note

`assignReplicasToBrokers` is used when:

- `ReassignPartitionsCommand` is requested to [generateAssignment](#)
- `AdminManager` is requested to [createTopics](#)
- `AdminZkClient` is requested to [create a topic](#) and [addPartitions](#)

assignReplicasToBrokersRackUnaware Internal Method

```
assignReplicasToBrokersRackUnaware(
    nPartitions: Int,
    replicationFactor: Int,
    brokerList: Seq[Int],
    fixedStartIndex: Int,
    startPartitionId: Int): Map[Int, Seq[Int]]
```

`assignReplicasToBrokersRackUnaware` performs the replicas to brokers assignment in a fairly random manner (i.e. includes two random numbers). No additional information is used to make the decision (except the input parameters).

```

import kafka.admin.{AdminUtils, BrokerMetadata}
// assignReplicasToBrokersRackUnaware is a private method
// Using assignReplicasToBrokers instead as the entry point
val brokerMetadatas = Seq(
  BrokerMetadata(0, None),
  BrokerMetadata(1, None),
  BrokerMetadata(2, None))
val assignment = AdminUtils.assignReplicasToBrokers(
  brokerMetadatas,
  nPartitions = 3,
  replicationFactor = 2)
val output = assignment.toSeq.sortBy(_._1).map { case (brokerId, replicas) => s"$broke
rId => $replicas" }
scala> output.foreach(println)
0 => ArrayBuffer(2, 1)
1 => ArrayBuffer(0, 2)
2 => ArrayBuffer(1, 0)

```

assignReplicasToBrokersRackUnaware ...FIXME

Note

`assignReplicasToBrokersRackUnaware` is used exclusively when `AdminUtils` is requested to `assignReplicasToBrokers` (when all the brokers in a cluster have no rack information assigned).

assignReplicasToBrokersRackAware Internal Method

```

assignReplicasToBrokersRackAware(
  nPartitions: Int,
  replicationFactor: Int,
  brokerMetadatas: Seq[BrokerMetadata],
  fixedStartIndex: Int,
  startPartitionId: Int): Map[Int, Seq[Int]]

```

assignReplicasToBrokersRackAware ...FIXME

Note

`assignReplicasToBrokersRackAware` is used exclusively when `AdminUtils` is requested to `assignReplicasToBrokers` (when all the brokers in a cluster have rack information assigned).

Kafka Controller Election

A Kafka broker can be elected as the controller in the process known as **Kafka Controller Election**.

Kafka Controller Election process relies heavily on the features of Apache ZooKeeper that acts as the source of truth and guarantees that only one broker can ever be elected (due to how **ephemeral nodes** work).

From [Nodes and ephemeral nodes](#) in the official documentation of Apache ZooKeeper:

ZooKeeper also has the notion of ephemeral nodes. These znodes exists as long as the session that created the znode is active. When the session ends the znode is deleted.

From [Leader Election](#) in the official documentation of Apache ZooKeeper:

A simple way of doing leader election with ZooKeeper is to use the SEQUENCE|EPHEMERAL flags when creating znodes that represent "proposals" of clients. The idea is to have a znode, say "/election"

When `ControllerEventThread` is requested to process [Startup](#) and [Reelect](#) controller events, `KafkaController` (instance that runs on every Kafka broker) is requested to [elect](#).

Tip	Consult Demo: Kafka Controller Election to learn about the process.
-----	---

Given that all the state is in ZooKeeper use `zookeeper-shell` script to know which broker is the active controller.

```
$ ./bin/zookeeper-shell.sh :2181 get /controller
Connecting to :2181

{"version":1,"brokerid":0,"timestamp":"1543491973573"}
cZxid = 0x48
ctime = Thu Nov 29 12:46:13 CET 2018
mZxid = 0x48
mtime = Thu Nov 29 12:46:13 CET 2018
pZxid = 0x48
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x100073f07ba0001
dataLength = 54
numChildren = 0
```

If you receive `Node does not exist: /controller` error message, that means that no Kafka broker has been elected as the active controller yet (or that you use an incorrect ZooKeeper server to talk to).

You could also use `nc` to talk to ZooKeeper in a more direct way (that allows for `dump` command).

```
$ nc localhost 2181
dump
SessionTracker dump:
Session Sets (2):
0 expire at Fri Jan 02 10:57:03 CET 1970:
1 expire at Fri Jan 02 10:57:06 CET 1970:
    0x100073f07ba0001
ephemeral nodes dump:
Sessions with Ephemerals (1):
0x100073f07ba0001:
    /controller
    /brokers/ids/0
```

From [Notable changes in 0.10.1.0](#) in the official documentation of Apache Kafka:

The recommended way to detect if a given broker is the controller is via the `kafka.controller:type=KafkaController, name=ActiveControllerCount` metric.

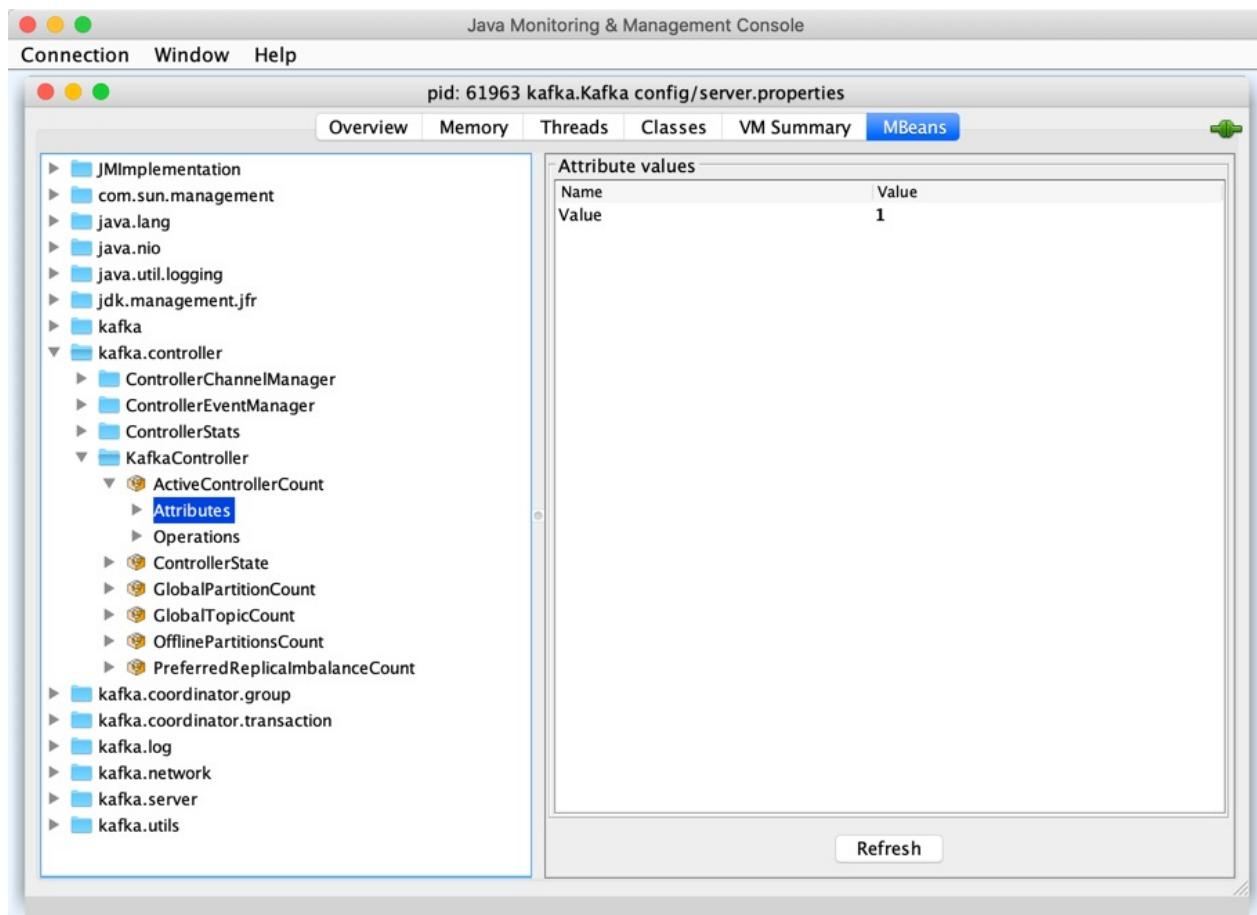


Figure 1. Active KafkaController in jconsole

Controller ID Registered (in ZooKeeper)

The election process stops when [there is a controller ID registered in Zookeeper](#) (using `KafkaZkClient` that [gets the ID of the active controller](#) and the ID is any number but `-1`).

No Controller ID Registered (in ZooKeeper)

If there is no controller ID registered, every `KafkaController` instance tries to [register itself as the controller \(in Zookeeper\)](#) and [increment the controller epoch](#) (using `KafkaZkClient`).

In the end, the active `KafkaController` is requested to [onControllerFailover](#).

Topic Replication

Topic Replication is the process to offer fail-over capability for a topic.

Replication factor defines the number of copies of a topic in a Kafka cluster.

Replication factor can be defined at topic level.

```
./bin/kafka-topics.sh --create \
--topic my-topic \
--replication-factor 1 \ // <-- define replication factor
--partitions 1 \
--zookeeper localhost:2181
```

Replicas are distributed evenly among Kafka brokers in a cluster.

Leader replica is...FIXME

Follower replica is...FIXME

Producers always send requests to the broker that is the current leader replica for a topic partition.

Data from producers is first saved to a commit log before consumers can find out that it is available. It will only be visible to consumers when the followers acknowledge that they have got the data and stored in their local logs.

Topic Deletion

Topic Deletion is a feature of Kafka that allows for deleting topics.

[TopicDeletionManager](#) is responsible for topic deletion.

Topic deletion is controlled by `delete.topic.enable` Kafka property that turns it on when `true`.

Start a Kafka broker with broker ID `100`.

```
$ ./bin/kafka-server-start.sh config/server.properties \
--override delete.topic.enable=true \
--override broker.id=100 \
--override log.dirs=/tmp/kafka-logs-100 \
--override port=9192
```

Create **remove-me** topic.

```
$ ./bin/kafka-topics.sh --zookeeper localhost:2181 \
--create \
--topic remove-me \
--partitions 1 \
--replication-factor 1
Created topic "remove-me".
```

Use `kafka-topics.sh --list` to list available topics.

```
$ ./bin/kafka-topics.sh --zookeeper localhost:2181 --list
__consumer_offsets
remove-me
```

Use `kafka-topics.sh --describe` to list details for `remove-me` topic.

```
$ ./bin/kafka-topics.sh --zookeeper localhost:2181 --describe --topic remove-me
Topic:remove-me PartitionCount:1      ReplicationFactor:1      Configs:
      Topic: remove-me          Partition: 0      Leader: 100      Replicas: 100    Isr: 1
00
```

Note that the broker `100` is the leader for `remove-me` topic.

Stop the broker `100` and start another with broker ID `200`.

```
$ ./bin/kafka-server-start.sh config/server.properties \
--override delete.topic.enable=true \
--override broker.id=200 \
--override log.dirs=/tmp/kafka-logs-200 \
--override port=9292
```

Use `kafka-topics.sh --delete` to delete `remove-me` topic.

```
$ ./bin/kafka-topics.sh --zookeeper localhost:2181 --delete --topic remove-me
Topic remove-me is marked for deletion.
Note: This will have no impact if delete.topic.enable is not set to true.
```

List the topics.

```
$ ./bin/kafka-topics.sh --zookeeper localhost:2181 --list
__consumer_offsets
remove-me - marked for deletion
```

As you may have noticed, `kafka-topics.sh --delete` will only delete a topic if the topic's leader broker is available (and can acknowledge the removal). Since the broker 100 is down and currently unavailable the topic deletion has only been recorded in Zookeeper.

```
$ ./bin/zkCli.sh -server localhost:2181
[zk: localhost:2181(CONNECTED) 0] ls /admin/delete_topics
[remove-me]
```

As long as the leader broker `100` is not available, the topic to be deleted remains marked for deletion.

Start the broker `100`.

```
$ ./bin/kafka-server-start.sh config/server.properties \
--override delete.topic.enable=true \
--override broker.id=100 \
--override log.dirs=/tmp/kafka-logs-100 \
--override port=9192
```

With [kafka.controller.KafkaController](#) logger at `DEBUG` level, you should see the following messages in the logs:

```
DEBUG [Controller id=100] Delete topics listener fired for topics remove-me to be deleted (kafka.controller.KafkaController)
INFO [Controller id=100] Starting topic deletion for topics remove-me (kafka.controller.KafkaController)
INFO [GroupMetadataManager brokerId=100] Removed 0 expired offsets in 0 milliseconds. (kafka.coordinator.group.GroupMetadataManager)
DEBUG [Controller id=100] Removing replica 100 from ISR 100 for partition remove-me-0. (kafka.controller.KafkaController)
INFO [Controller id=100] Retaining last ISR 100 of partition remove-me-0 since unclean leader election is disabled (kafka.controller.KafkaController)
INFO [Controller id=100] New leader and ISR for partition remove-me-0 is {"leader": -1, "leader_epoch": 1, "isr": [100]} (kafka.controller.KafkaController)
INFO [ReplicaFetcherManager on broker 100] Removed fetcher for partitions remove-me-0 (kafka.server.ReplicaFetcherManager)
INFO [ReplicaFetcherManager on broker 100] Removed fetcher for partitions (kafka.server.ReplicaFetcherManager)
INFO [ReplicaFetcherManager on broker 100] Removed fetcher for partitions remove-me-0 (kafka.server.ReplicaFetcherManager)
INFO Log for partition remove-me-0 is renamed to /tmp/kafka-logs-100/remove-me-0.fe6d039ff884498b9d6113fb22a75264-delete and is scheduled for deletion (kafka.log.LogManager)
)
DEBUG [Controller id=100] Delete topic callback invoked for org.apache.kafka.common.requests.StopReplicaResponse@8c0f4f0 (kafka.controller.KafkaController)
INFO [Controller id=100] New topics: [Set()], deleted topics: [Set()], new partition replication assignment [Map()] (kafka.controller.KafkaController)
DEBUG [Controller id=100] Delete topics listener fired for topics to be deleted (kafka.controller.KafkaController)
```

The topic is now deleted. Use Zookeeper CLI tool to confirm it.

```
$ ./bin/zkCli.sh -server localhost:2181
[zk: localhost:2181(CONNECTED) 1] ls /admin/delete_topics
[]
```

Transactional Producer

Transactional Producer is a Kafka [Producer](#) that uses a [TransactionManager](#) for...FIXME

Idempotent Producer

Idempotent Producer is a Kafka [Producer](#) that...FIXME

Consumer Contract — Kafka Clients for Consuming Records

`consumer` is the abstraction of Kafka clients that can subscribe to topics or assign partitions to consume records from.

`Consumer` is also a Java `java.io.Closeable` and so can be considered as a "a source of data that can be closed". A `consumer` instance should always be `closed` to release system resources.

Table 1. Consumer Contract

Method	Description
<code>assign</code>	<code>void assign(Collection<TopicPartition> partitions)</code>
<code>assignment</code>	<code>Set<TopicPartition> assignment()</code>
<code>beginningOffsets</code>	<code>Map<TopicPartition, Long> beginningOffsets(Collection<TopicPartition> partitions)</code> <code>Map<TopicPartition, Long> beginningOffsets(Collection<TopicPartition> partitions, Duration timeout)</code>
<code>close</code>	<code>void close()</code> <code>void close(Duration timeout)</code> Closes the <code>Consumer</code> to release system resources used
<code>commitAsync</code>	<code>void commitAsync()</code> <code>void commitAsync(Map<TopicPartition, OffsetAndMetadata> offsets, OffsetCommitCallback callback)</code> <code>void commitAsync(OffsetCommitCallback callback)</code>
<code>commitSync</code>	<code>void commitSync()</code> <code>void commitSync(Duration timeout)</code> <code>void commitSync(Map<TopicPartition, OffsetAndMetadata> offsets)</code> <code>void commitSync(Map<TopicPartition, OffsetAndMetadata> offsets, Duration timeout)</code>

committed	<pre>OffsetAndMetadata committed(TopicPartition partition) OffsetAndMetadata committed(TopicPartition partition, Duration timeout)</pre>
endOffsets	<pre>Map<TopicPartition, Long> endOffsets(Collection<TopicPartition> partitions) Map<TopicPartition, Long> endOffsets(Collection<TopicPartition> partitions, Duration timeout)</pre>
listTopics	<pre>Map<String, List<PartitionInfo>> listTopics() Map<String, List<PartitionInfo>> listTopics(Duration timeout)</pre>
metrics	<pre>Map<MetricName, ? extends Metric> metrics()</pre>
offsetsForTimes	<pre>Map<TopicPartition, OffsetAndTimestamp> offsetsForTimes(Map<TopicPartition, Long> timestampsToSearch) Map<TopicPartition, OffsetAndTimestamp> offsetsForTimes(Map<TopicPartition, Long> timestampsToSearch, Duration timeout)</pre>
partitionsFor	<pre>List<PartitionInfo> partitionsFor(String topic) List<PartitionInfo> partitionsFor(String topic, Duration timeout)</pre>
pause	<pre>void pause(Collection<TopicPartition> partitions)</pre>
paused	<pre>Set<TopicPartition> paused()</pre>
poll	<pre>ConsumerRecords<K, V> poll(Duration timeout)</pre>

position	<pre>long position(TopicPartition partition) long position(TopicPartition partition, Duration timeout)</pre>
resume	<pre>void resume(Collection<TopicPartition> partitions)</pre>
seek	<pre>void seek(TopicPartition partition, long offset)</pre>
seekToBeginning	<pre>void seekToBeginning(Collection<TopicPartition> partitions)</pre>
seekToEnd	<pre>void seekToEnd(Collection<TopicPartition> partitions)</pre>
subscribe	<pre>void subscribe(Collection<String> topics) void subscribe(Collection<String> topics, ConsumerRebalanceListener callback) void subscribe(Pattern pattern) void subscribe(Pattern pattern, ConsumerRebalanceListener callback)</pre>
subscription	<pre>Set<String> subscription()</pre>
unsubscribe	<pre>void unsubscribe()</pre>
wakeup	<pre>void wakeup()</pre>

Table 2. Consumers

Consumer	Description
KafkaConsumer	Primary Kafka client
MockConsumer	Used for unit testing

KafkaConsumer — Primary Kafka Client

`KafkaConsumer` is a concrete Kafka client to consume records from Kafka topics for Kafka developers to write Kafka consumers.

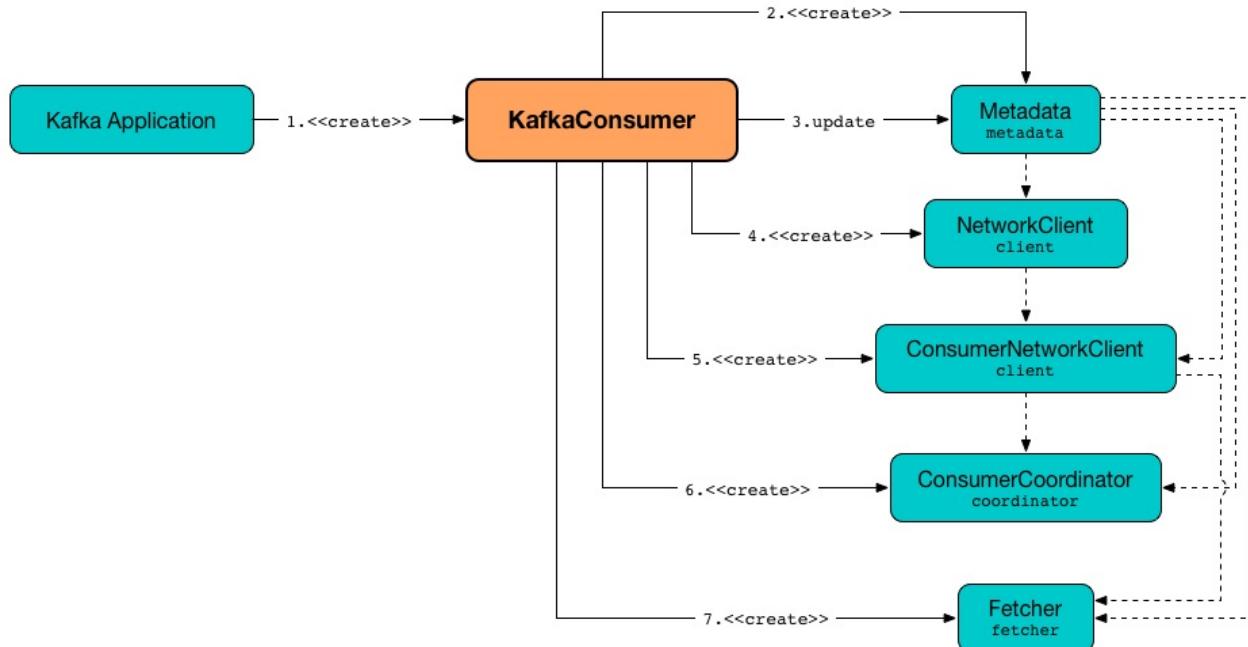


Figure 1. KafkaConsumer

`KafkaConsumer` is created with properties and (key and value) deserializers.

Note	<p><code>bootstrap.servers</code> and <code>group.id</code> properties are mandatory.</p> <p>Use <code>ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG</code> and <code>ConsumerConfig.GROUP_ID_CONFIG</code> values in your source code, respectively.</p>
------	---

```

val bootstrapServers = ":9092,localhost:9092"
val groupId = "kafka-sandbox"
import org.apache.kafka.clients.consumer.ConsumerConfig
val requiredConfigsOnly = Map[String, Object](
  ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG -> bootstrapServers,
  ConsumerConfig.GROUP_ID_CONFIG -> groupId
)
import org.apache.kafka.common.serialization.StringDeserializer
val keyDeserializer = new StringDeserializer
val valueDeserializer = new StringDeserializer

import scala.collection.JavaConverters.-
import org.apache.kafka.clients.consumer.KafkaConsumer
val consumer = new KafkaConsumer[String, String](
  requiredConfigsOnly.asJava,
  keyDeserializer,
  valueDeserializer)

```

Once `created`, `KafkaConsumer` is supposed to subscribe to topics or assign partitions.

```

scala> :type consumer
org.apache.kafka.clients.consumer.KafkaConsumer[String, String]

import scala.collection.JavaConverters.-
val topics = Seq("input").asJava
consumer.subscribe(topics)

```

`KafkaConsumer` is then requested to poll for records (in a loop).

```

scala> :type consumer
org.apache.kafka.clients.consumer.KafkaConsumer[String, String]

import java.time.Duration.ZERO
val records = consumer.poll(ZERO)
records.asList.foreach(println)

```

`KafkaConsumer` registers itself in JMX with **kafka.consumer** prefix.

Important

`KafkaConsumer` is not thread-safe, i.e. you should not use the same single instance of `KafkaConsumer` from multiple threads. You should use only one thread per `KafkaConsumer` instance.

`KafkaConsumer` uses light locks protecting itself from multi-threaded access and reports `ConcurrentModificationException` when it happens.

`KafkaConsumer` is not safe for multi-threaded access

Table 1. KafkaConsumer's Internal Properties (e.g. Registries and Counters)

Name	Description
assignors	<p>Zero or more PartitionAssignors</p> <p>Configured using ConsumerConfig.PARTITION_ASSIGNMENT_STRATEGY_CONFIG (aka <code>partition.assignment.strategy</code>) configuration property</p> <p>Used exclusively to create the ConsumerCoordinator</p>
client	<p>ConsumerNetworkClient</p> <p>Used mainly (?) to create the Fetcher and ConsumerCoordinator</p> <p>Used also in poll, pollOnce and wakeup (but I think the usage should be limited to create Fetcher and ConsumerCoordinator)</p>
clientId	
coordinator	<p>ConsumerCoordinator</p> <p>Used in subscribe, unsubscribe, assign, updateAssignmentMetadataIfNeeded, pollForFetches, commitSync, commitAsync, committed, close, and updateFetchPositions</p>
fetcher	<p>Fetcher</p> <p>Created right when <code>KafkaConsumer</code> is created.</p> <p>Used when...FIXME</p>
interceptors	<p><code>ConsumerInterceptors</code> that holds ConsumerInterceptor instances (defined using interceptor.classes setting).</p> <p>Used when...FIXME</p>
metadata	<p>Metadata</p> <p>Created for a KafkaConsumer with the following:</p> <ul style="list-style-type: none"> • retryBackoffMs configuration property • ConsumerConfig.METADATA_MAX_AGE_CONFIG configuration property • <code>allowAutoTopicCreation</code> flag on • <code>topicExpiryEnabled</code> flag off <p><code>metadata</code> is requested to update itself immediately</p> <p>Used to create a NetworkClient (for the ConsumerNetworkClient), the ConsumerNetworkClient itself, the ConsumerCoordinator and Fetcher</p>

	Used in...FIXME
metrics	Metrics
retryBackoffMs	<code>retry.backoff.ms</code> property or a user-defined value
requestTimeoutMs	Corresponds to <code>request.timeout.ms</code> property KafkaConsumer reports <code>ConfigException</code> when smaller or equal than <code>session.timeout.ms</code> and <code>fetch.max.wait.ms</code> properties.
subscriptions	<code>SubscriptionState</code> for <code>auto.offset.reset</code> setting. Created when <code>KafkaConsumer</code> is created
Tip	<p>Enable <code>DEBUG</code> or <code>TRACE</code> logging levels for <code>org.apache.kafka.clients.consumer.KafkaConsumer</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.clients.consumer.KafkaConsumer=TRACE</pre> <p>Refer to Logging.</p>

(Manually) Assigning Partitions — `assign` Method

```
void assign(Collection<TopicPartition> partitions)
```

Note	<code>assign</code> is part of Consumer Contract to...FIXME.
------	--

`assign` ...FIXME

unsubscribe Method

```
void unsubscribe()
```

Note	<code>unsubscribe</code> is part of Consumer Contract to...FIXME.
------	---

`unsubscribe` ...FIXME

Subscribing to Topics — `subscribe` Method

```
void subscribe(Collection<String> topics) (1)
void subscribe(Collection<String> topics, ConsumerRebalanceListener listener)
void subscribe(Pattern pattern, ConsumerRebalanceListener listener)
```

1. A short-hand for the other subscribe with `NoOpConsumerRebalanceListener` as `ConsumerRebalanceListener`

`subscribe` subscribes `KafkaConsumer` to the given topics.

Note

`subscribe` is a part of [Consumer Contract](#) to...FIXME

```
val topics = Seq("topic1")
println(s"Subscribing to ${topics.mkString(", ")}")

import scala.collection.JavaConverters._
consumer.subscribe(topics.asJava)
```

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

```
DEBUG Subscribed to topic(s): [comma-separated topics]
```

`subscribe` then requests [SubscriptionState](#) to `subscribe` for the topics and listener .

In the end, `subscribe` requests [SubscriptionState](#) for `groupSubscription` that it then passes along to [Metadata](#) to set the topics to track.

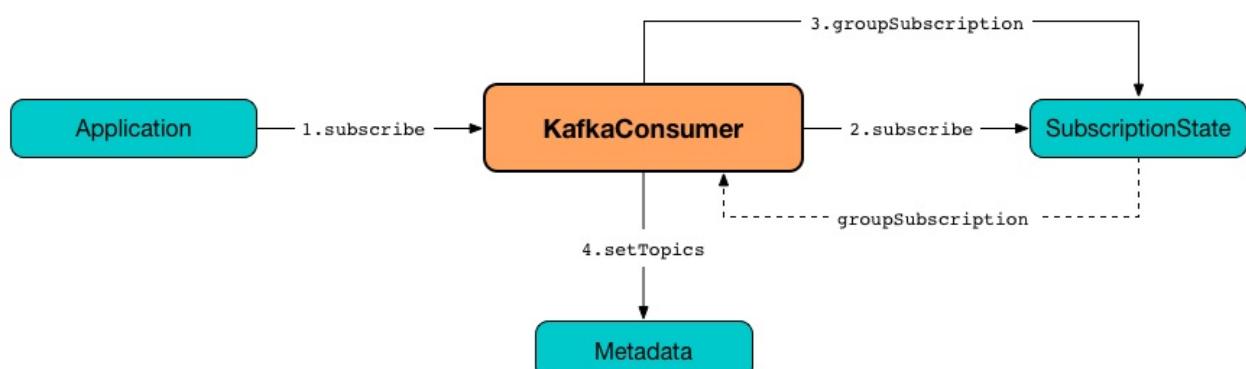


Figure 2. KafkaConsumer subscribes to topics

Poll For ConsumerRecords (per TopicPartitions) — `poll` Method

```
ConsumerRecords<K, V> poll(final Duration timeout)
```

Note `poll` is part of the [Consumer Contract](#) to poll for [ConsumerRecords](#).

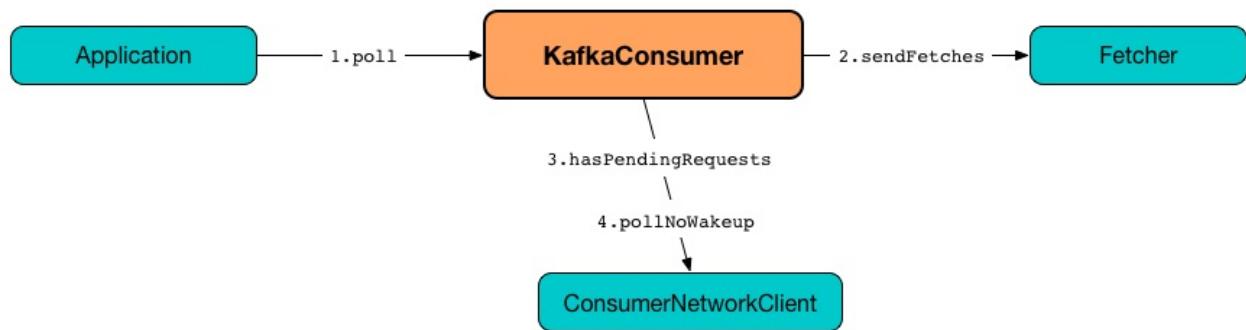


Figure 3. KafkaConsumer polls topics

`poll` polls for new records until `timeout` expires.

```

scala> :type consumer
org.apache.kafka.clients.consumer.KafkaConsumer[String,String]

import java.time.Duration.ZERO
while (true) {
  println(s"Polling for records for $ZERO secs")
  val records = consumer.poll(ZERO)
  // do something with the records
  // e.g. print them out to the console
  records.asScala.foreach(println)
}
  
```

Note `KafkaConsumer` has to be subscribed to topics or assigned partitions before calling `poll`.

```

scala> :type consumer
org.apache.kafka.clients.consumer.KafkaConsumer[String,String]

import java.time.Duration
scala> val records = consumer.poll(Duration.ZERO)
java.lang.IllegalStateException: Consumer is not subscribed to any topics or assigned
any partitions
  at org.apache.kafka.clients.consumer.KafkaConsumer.poll(KafkaConsumer.java:1171)
  at org.apache.kafka.clients.consumer.KafkaConsumer.poll(KafkaConsumer.java:1164)
  ... 36 elided
  
```

Internally, `poll` simply calls the internal `poll` method with the `Time` that expires after the given `timeout` and the `includeMetadataInTimeout` flag on.

poll Internal Method

```
ConsumerRecords<K, V> poll(
    final Timer timer,
    final boolean includeMetadataInTimeout)
```

`poll` first [acquireAndEnsureOpen](#).

`poll` requests the [ConsumerNetworkClient](#) to [maybeTriggerWakeup](#).

`poll` ...FIXME

commitSync Method

```
void commitSync()
```

Note

`commitSync` is part of [Consumer Contract](#) to...FIXME.

`commitSync` ...FIXME

seek Method

```
void seek(TopicPartition partition, long offset)
```

Note

`seek` is part of [Consumer Contract](#) to...FIXME.

`seek` ...FIXME

Getting Partitions For Topic — partitionsFor Method

Caution

FIXME

endOffsets Method

Caution

FIXME

offsetsForTimes Method

Caution

FIXME

updateFetchPositions Internal Method

```
boolean updateFetchPositions(final Timer timer)
```

`updateFetchPositions ...FIXME`

Note

`updateFetchPositions` is used when...FIXME

Polling One-Off for ConsumerRecords per TopicPartition

— `pollOnce` Internal Method

```
Map<TopicPartition, List<ConsumerRecord<K, V>>> pollOnce(long timeout)
```

`pollOnce ...FIXME`

Note

`pollOnce` is used exclusively when `KafkaConsumer` is requested to `poll`

Requesting Metadata for All Topics (From Brokers)

— `listTopics` Method

```
Map<String, List<PartitionInfo>> listTopics()
```

Internally, `listTopics` simply requests `Fetcher` for metadata for all topics and returns it.

```
consumer.listTopics().asScala.foreach { case (name, partitions) =>
  println(s"topic: $name (partitions: ${partitions.size()})")
}
```

Note

`listTopics` uses `requestTimeoutMs` that corresponds to `request.timeout.ms` property.

beginningOffsets Method

```
Map<TopicPartition, Long> beginningOffsets(Collection<TopicPartition> partitions)
```

`beginningOffsets` requests `Fetcher` for `beginningOffsets` and returns it.

Creating KafkaConsumer Instance

`KafkaConsumer` takes the following when created:

- Consumer configuration (that is converted internally to [ConsumerConfig](#))
- [Deserializer](#) for keys
- [Deserializer](#) for values

`KafkaConsumer` initializes the internal registries and counters.

Note

`KafkaConsumer` API offers other constructors that in the end use the public 3-argument constructor that in turn passes the call on to the private internal constructor.

KafkaConsumer Public Constructor

```
// Public API
KafkaConsumer(
    Map<String, Object> configs,
    Deserializer<K> keyDeserializer,
    Deserializer<V> valueDeserializer)
```

When created, `KafkaConsumer` adds the `keyDeserializer` and `valueDeserializer` to `configs` (as `key.deserializer` and `value.deserializer` properties respectively) and creates a `ConsumerConfig`.

`KafkaConsumer` passes the call on to the internal constructor.

KafkaConsumer Internal Constructor

```
KafkaConsumer(
    ConsumerConfig config,
    Deserializer<K> keyDeserializer,
    Deserializer<V> valueDeserializer)
```

When called, the internal `KafkaConsumer` constructor prints out the following DEBUG message to the logs:

```
DEBUG Starting the Kafka consumer
```

`KafkaConsumer` sets the internal `requestTimeoutMs` to `request.timeout.ms` property.

`KafkaConsumer` sets the internal `clientId` to `client.id` or generates one with prefix `consumer-` (starting from 1) if not set.

`KafkaConsumer` sets the internal `Metrics` (and `JmxReporter` with `kafka.consumer` prefix).

KafkaConsumer sets the internal `retryBackoffMs` to `retry.backoff.ms` property.

Caution	FIXME Finish me!
---------	------------------

KafkaConsumer creates the internal [Metadata](#) with the following arguments:

1. `retryBackoffMs`
2. `metadata.max.age.ms`
3. `allowAutoTopicCreation` enabled
4. `topicExpiryEnabled` disabled
5. `ClusterResourceListeners` with user-defined list of `ConsumerInterceptors` in `interceptor.classes` property

KafkaConsumer updates metadata with `bootstrap.servers`.

Caution	FIXME Finish me!
---------	------------------

KafkaConsumer creates a [NetworkClient](#) with...FIXME

Caution	FIXME Finish me!
---------	------------------

KafkaConsumer creates [Fetcher](#) with the following properties:

- `fetch.min.bytes`
- `fetch.max.bytes`
- `fetch.max.wait.ms`
- `max.partition.fetch.bytes`
- `max.poll.records`
- `check.crcs`

In the end, KafkaConsumer prints out the following DEBUG message to the logs:

```
DEBUG Kafka consumer created
```

Any issues while creating a KafkaConsumer are reported as `kafkaException`.

```
org.apache.kafka.common.KafkaException: Failed to construct kafka consumer
```

wakeup Method

```
void wakeup()
```

Note

wakeup is a part of Consumer Contract.

wakeup simply requests ConsumerNetworkClient to wakeup.

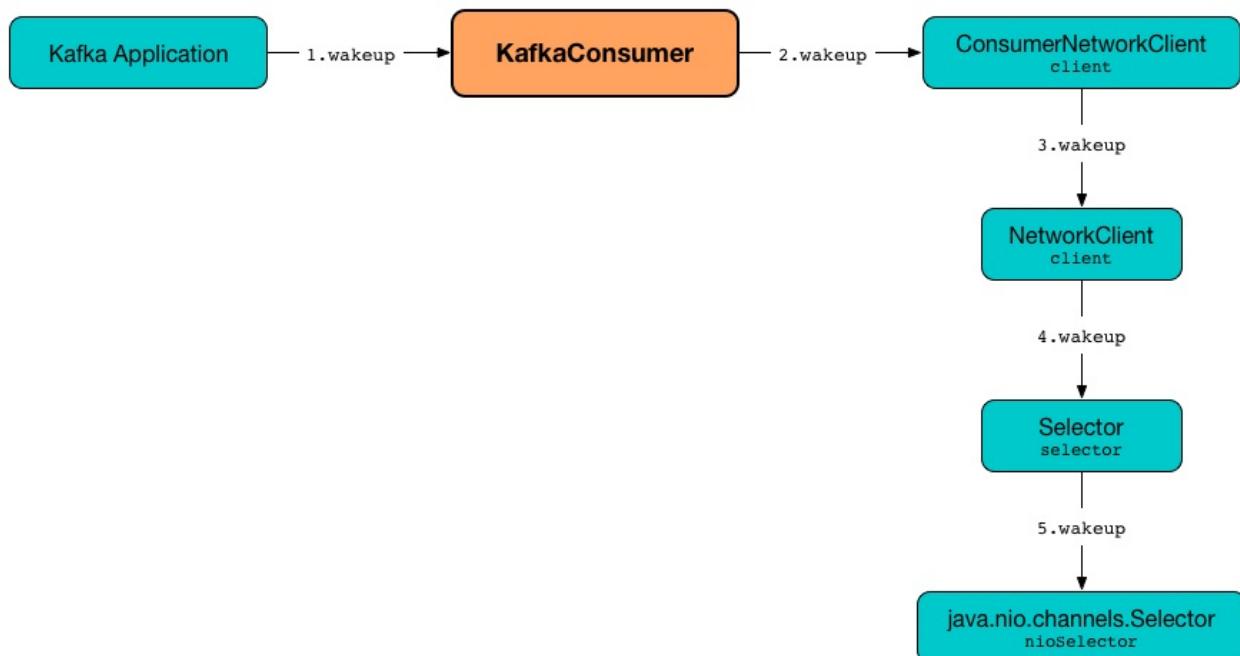


Figure 4. KafkaConsumer's wakeup Method

Note

Quoting `wakeup` of Java's `java.nio.channels.Selector` given `wakeup` simply passes through the intermediaries and in the end triggers it.

Causes the first selection operation that has not yet returned to return immediately.

Read about Selection in [java.nio.channels.Selector's javadoc](#).

Note

wakeup is used when...FIXME

Configuring ClusterResourceListeners — `configureClusterResourceListeners` Internal Method

```
ClusterResourceListeners configureClusterResourceListeners(
    Deserializer<K> keyDeserializer,
    Deserializer<V> valueDeserializer,
    List<?>... candidateLists)
```

`configureClusterResourceListeners` creates a [ClusterResourceListeners](#) and registers `ClusterResourceListener` instances from the input `candidateLists`, `keyDeserializer` and `valueDeserializer`.

Note

`configureClusterResourceListeners` is used exclusively when `KafkaConsumer` is created (to create the [Metadata](#)) with the following input arguments:

- `key` and `value` deserializers (defined when `KafkaConsumer` is created)
- `ConsumerInterceptors` from `interceptor.classes` Kafka property
- `MetricsReporters` from `metric.reporters` Kafka property

throwIfNoAssignorsConfigured Internal Method

```
void throwIfNoAssignorsConfigured()
```

`throwIfNoAssignorsConfigured` ...FIXME

Note

`throwIfNoAssignorsConfigured` is used exclusively when `KafkaConsumer` is requested to [subscribe to topics](#).

updateAssignmentMetadataIfNeeded Internal Method

```
boolean updateAssignmentMetadataIfNeeded(final Timer timer)
```

`updateAssignmentMetadataIfNeeded` requests the [ConsumerCoordinator](#) to [poll](#) until the `Timer` expires.

`updateAssignmentMetadataIfNeeded` returns `false` if the poll was unsuccessful, i.e. FIXME

If the poll was successful, `updateAssignmentMetadataIfNeeded` [updateFetchPositions](#).

Note

`updateAssignmentMetadataIfNeeded` is used exclusively when `KafkaConsumer` is requested to [poll for records](#).

pollForFetches Internal Method

```
Map<TopicPartition, List<ConsumerRecord<K, V>>> pollForFetches(Timer timer)
```

`pollForFetches` ...FIXME

Note

`pollForFetches` is used exclusively when `KafkaConsumer` is requested to [poll for records](#).

commitAsync Method

```
void commitAsync()
void commitAsync(
    OffsetCommitCallback callback)
void commitAsync(
    final Map<TopicPartition, OffsetAndMetadata> offsets,
    OffsetCommitCallback callback)
```

Note

`commitAsync` is part of the [Consumer Contract](#) to...FIXME.

`commitAsync` ...FIXME

committed Method

```
OffsetAndMetadata committed(
    TopicPartition partition)
OffsetAndMetadata committed(
    TopicPartition partition,
    final Duration timeout)
```

Note

`committed` is part of the [Consumer Contract](#) to...FIXME.

`committed` ...FIXME

close Method

```
void close()
void close(Duration timeout)
```

Note

`close` is part of the [Consumer Contract](#) to...FIXME.

`close` ...FIXME

close Internal Method

```
void close(long timeoutMs, boolean swallowException)
```

close ...FIXME

MockConsumer

`MockConsumer` is a concrete [Consumer](#) to be used for unit testing.

Note

`MockConsumer` is used in Kafka Streams' `TopologyTestDriver` to write tests to verify the behavior of topologies.

`MockConsumer` takes a single `OffsetResetStrategy` to be created.

```
import org.apache.kafka.clients.consumer._  
val consumer = new MockConsumer[String, String](OffsetResetStrategy.EARLIEST)
```

While being created, `MockConsumer` initializes the [internal registries and counters](#).

Beside [Consumer API](#), `MockConsumer` defines its own [testing-specific API](#).

Table 1. MockConsumer API

Method	Description
addEndOffsets	<code>void addEndOffsets(Map<TopicPartition, Long> newOffsets)</code>
addRecord	<code>void addRecord(ConsumerRecord<K, V> record)</code>
closed	<code>boolean closed()</code>
rebalance	<code>void rebalance(Collection<TopicPartition> newAssignment)</code>
scheduleNopPollTask	<code>void scheduleNopPollTask()</code>
schedulePollTask	<code>void schedulePollTask(Runnable task)</code>
setException	<code>void setException(KafkaException exception)</code>
updateBeginningOffsets	<code>void updateBeginningOffsets(Map<TopicPartition, Long> newOffsets)</code>
updateEndOffsets	<code>void updateEndOffsets(Map<TopicPartition, Long> newOffsets)</code>
updatePartitions	<code>void updatePartitions(String topic, List<PartitionInfo> partitions)</code>

Table 2. MockConsumer's Internal Properties (e.g. Registries, Counters and Flags)

Name	Description
subscriptions	<code>SubscriptionState</code> Used when...FIXME

partitions	<code>Map<String, List<PartitionInfo>> partitions</code> Used when...FIXME
records	<code>Map<TopicPartition, List<ConsumerRecord<K, V>>> records</code> Used when...FIXME
paused	<code>Kafka's TopicPartition paused</code> Used when...FIXME
closed	<code>Flag that says whether...FIXME Defaults to false</code> Used when...FIXME
beginningOffsets	<code>Map<TopicPartition, Long> beginningOffsets</code> Used when...FIXME
endOffsets	<code>Map<TopicPartition, List<Long>> endOffsets</code> Used when...FIXME
pollTasks	<code>Queue<Runnable></code> Used when...FIXME
exception	<code>A KafkaException if thrown</code> Used when...FIXME
wakeup	<code>Flag that says whether...FIXME (a Java AtomicBoolean)</code> Used when...FIXME
committed	<code>Map<TopicPartition, OffsetAndMetadata> committed</code> Used when...FIXME

ConsumerRecord

ConsumerRecord is...FIXME

OffsetAndMetadata

OffsetAndMetadata is...FIXME

OffsetAndTimestamp

OffsetAndTimestamp is...FIXME

OffsetCommitCallback

`OffsetCommitCallback` is...FIXME

ConsumerRebalanceListener

`ConsumerRebalanceListener` is a [callback interface](#) to be notified when the set of partitions assigned to the consumer (aka *assignment*) changes.

`ConsumerRebalanceListener` is notified about the newly-assigned partitions through `onPartitionsAssigned` callback that happens when `ConsumerCoordinator` is requested to `onJoinComplete`.

`ConsumerRebalanceListener` is notified about the partitions revoked through `onPartitionsRevoked` callback that happens when `ConsumerCoordinator` is requested to `onJoinPrepare`.

Caution

FIXME Picture when the notifications are triggered

```
package org.apache.kafka.clients.consumer;

interface ConsumerRebalanceListener {
    void onPartitionsAssigned(Collection<TopicPartition> partitions);
    void onPartitionsRevoked(Collection<TopicPartition> partitions);
}
```

`ConsumerRebalanceListener` is used when a Kafka `Consumer` is requested to [subscribe to topics](#).

```
void subscribe(Collection<String> topics, ConsumerRebalanceListener callback);
void subscribe(Pattern pattern, ConsumerRebalanceListener callback);
```

Table 1. ConsumerRebalanceListener Contract

Method	Description
<code>onPartitionsAssigned</code>	Used exclusively when <code>ConsumerCoordinator</code> is requested to <code>onJoinComplete</code>
<code>onPartitionsRevoked</code>	Used exclusively when <code>ConsumerCoordinator</code> is requested to <code>onJoinPrepare</code>

ConsumerConfig — Configuration Properties for KafkaConsumer

`ConsumerConfig` is a Apache Kafka [AbstractConfig](#) for the [configuration properties](#) of a [KafkaConsumer](#).

```
import org.apache.kafka.clients.consumer.ConsumerConfig

val conf = new java.util.Properties()
conf.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, ":9092,localhost:9192")
import org.apache.kafka.common.serialization.Serdes
conf.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, Serdes.String.deserializer.getClass)
conf.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, Serdes.String.deserializer.getClass)

import org.apache.kafka.clients.consumer.KafkaConsumer
val consumer = new KafkaConsumer[String, String](conf)
```

Table 1. ConsumerConfig's Configuration Properties

Name, Property, Default Value	Description				
BOOTSTRAP_SERVERS_CONFIG					
GROUP_ID_CONFIG					
METADATA_MAX_AGE_CONFIG metadata.max.age.ms Default: 5 mins	The period of time (in milliseconds) after which we force a refresh of metadata even if we haven't seen any partition leadership changes. That is to proactively discover any new brokers or partitions.				
PARTITION_ASSIGNMENT_STRATEGY_CONFIG partition.assignment.strategy Default: RangeAssignor	<p>The class name of the partition assignment strategy that the client will use to distribute partition ownership amongst consumer instances when group management is used</p> <table border="1"> <tr> <td>Tip</td><td>Read up Kafka Client-side Assignment Proposal on the group management in Apache Kafka's Consumer API.</td></tr> </table> <p>Used when:</p> <ul style="list-style-type: none"> KafkaConsumer is created Kafka Streams' StreamsConfig is requested to getMainConsumerConfigs <table border="1"> <tr> <td>Note</td><td>Required for KafkaConsumer to subscribe to topics</td></tr> </table>	Tip	Read up Kafka Client-side Assignment Proposal on the group management in Apache Kafka's Consumer API.	Note	Required for KafkaConsumer to subscribe to topics
Tip	Read up Kafka Client-side Assignment Proposal on the group management in Apache Kafka's Consumer API.				
Note	Required for KafkaConsumer to subscribe to topics				
REQUEST_TIMEOUT_MS_CONFIG request.timeout.ms Default: 30000 ms	<p>How long (in millis) a Kafka client waits for a response to a request.</p> <p>If the response is not received before the timeout elapses the client will resend the request if necessary or fail the request if retries are exhausted.</p> <p>Must be at least 0</p>				

CommitFailedException

CommitFailedException is...FIXME

InvalidOffsetException

InvalidOffsetException is...FIXME

NoOffsetForPartitionException

NoOffsetForPartitionException is...FIXME

OffsetOutOfRangeException

OffsetOutOfRangeException is...FIXME

RetriableCommitFailedException

RetriableCommitFailedException is...FIXME

ConsumerInterceptor

Example

```
package pl.jaceklaskowski.kafka

import java.util

import org.apache.kafka.clients.consumer.{ConsumerInterceptor, ConsumerRecords, OffsetAndMetadata}
import org.apache.kafka.common.TopicPartition

class KafkaInterceptor extends ConsumerInterceptor[String, String] {
  override def onConsume(records: ConsumerRecords[String, String]): ConsumerRecords[String, String] = {
    ConsumerRecords[String, String] = {
      println(s"KafkaInterceptor.onConsume")
      import scala.collection.JavaConverters._
      records.asScala.foreach { r =>
        println(s"=> $r")
      }
      records
    }
  }

  override def close(): Unit = {
    println("KafkaInterceptor.close")
  }

  override def onCommit(offsets: util.Map[TopicPartition, OffsetAndMetadata]): Unit =
  {
    println("KafkaInterceptor.onCommit")
    println(s"$offsets")
  }

  override def configure(configs: util.Map[String, _]): Unit = {
    println(s"KafkaInterceptor.configure($configs)")
  }
}
```

onConsume Method

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

PartitionAssignor Contract

`PartitionAssignor` is the abstraction of [partition assignors](#) that are identified by `name` and can perform [partition assignment](#).

Table 1. PartitionAssignor Contract

Method	Description
<code>assign</code>	<pre>Map<String, Assignment> assign(Cluster metadata, Map<String, Subscription> subscriptions)</pre> <p>Assigns partitions to the members of a consumer group Used exclusively when <code>ConsumerCoordinator</code> is requested to perform partition assignment</p>
<code>name</code>	<pre>String name()</pre> <p>Used when <code>ConsumerCoordinator</code> is requested to metadata, lookupAssignor, and performAssignment (only for logging purposes)</p>
<code>onAssignment</code>	<pre>void onAssignment(Assignment assignment)</pre> <p>Used exclusively when <code>ConsumerCoordinator</code> is requested to onJoinComplete</p>
<code>subscription</code>	<pre>Subscription subscription(Set<String> topics)</pre> <p>Used exclusively when <code>ConsumerCoordinator</code> is requested to metadata</p>

Table 2. PartitionAssignors (Direct Implementations)

PartitionAssignor	Description
<code>AbstractPartitionAssignor</code>	Extension for collecting partition counts
<code>StreamsPartitionAssignor</code>	Default <code>PartitionAssignor</code> in Kafka Streams

RangeAssignor

RangeAssignor is...FIXME

RoundRobinAssignor

RoundRobinAssignor is...FIXME

StickyAssignor

StickyAssignor is...FIXME

AbstractPartitionAssignor

`AbstractPartitionAssignor` is the [extension](#) of the [PartitionAssignor contract](#) for [partition assignors](#) that can [perform group assignment given partition counts and member subscriptions](#).

Note	<code>AbstractPartitionAssignor</code> is a Java abstract class and cannot be created directly. It is created indirectly for the concrete PartitionAssignors .
------	--

Table 1. AbstractPartitionAssignor Contract

Method	Description
<code>assign</code>	<pre>Map<String, List<TopicPartition>> assign(Map<String, Integer> partitionsPerTopic, Map<String, Subscription> subscriptions)</pre> <p>Performs group assignment given partition counts and member subscriptions</p> <p>Used exclusively when requested to assign (given cluster metadata and member subscriptions)</p>

Table 2. AbstractPartitionAssignors

AbstractPartitionAssignor	Description
RangeAssignor	
RoundRobinAssignor	
StickyAssignor	

assign Method

```
Map<String, Assignment> assign(
    Cluster metadata,
    Map<String, Subscription> subscriptions)
```

Note	<code>assign</code> is part of the PartitionAssignor Contract to perform group assignment given cluster metadata and member subscriptions.
------	--

`assign ...FIXME`

ConsumerCoordinator

`ConsumerCoordinator` is a concrete `AbstractCoordinator` that...FIXME

`ConsumerCoordinator` is created exclusively when `KafkaConsumer` is created.

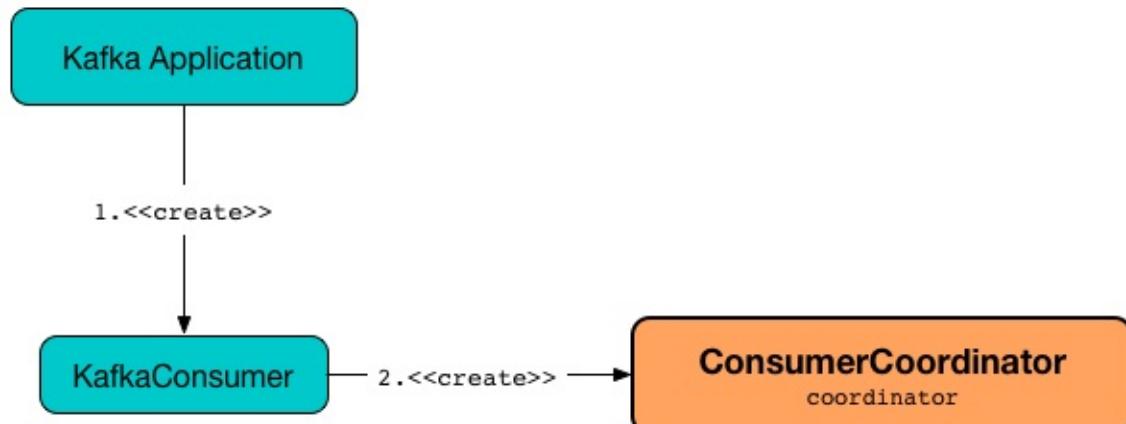


Figure 1. ConsumerCoordinator and KafkaConsumer

`KafkaConsumer` uses the following [Consumer configuration properties](#) to create a `ConsumerCoordinator`.

Table 1. ConsumerCoordinator's Consumer Configuration Properties

Configuration Property	ConsumerCoordinator's Property
<code>max.poll.interval.ms</code>	<code>rebalanceTimeoutMs</code>
<code>session.timeout.ms</code>	<code>sessionTimeoutMs</code>
<code>heartbeat.interval.ms</code>	<code>heartbeatIntervalMs</code>
<code>retry.backoff.ms</code>	<code>retryBackoffMs</code>
<code>enable.auto.commit</code>	<code>autoCommitEnabled</code>
<code>auto.commit.interval.ms</code>	<code>autoCommitIntervalMs</code>
<code>exclude.internal.topics</code>	<code>excludeInternalTopics</code>
<code>internal.leave.group.on.close</code>	<code>leaveGroupOnClose</code>

With `autoCommitEnabled` enabled (i.e. `enable.auto.commit` is `true`), `ConsumerCoordinator` does:

1. Sending asynchronous auto-commit of offsets

2. Sending synchronous auto-commit of offsets

`ConsumerCoordinator` uses zero or more [PartitionAssignors](#) for the following:

- [metadata](#)
- Looking up the [PartitionAssignor](#) by name (when `onJoinComplete` and `performAssignment`)

Table 2. ConsumerCoordinator's Internal Properties (e.g. Registries, Counters and Flags)

Name	Description
<code>assignmentSnapshot</code>	<p><code>MetadataSnapshot</code></p> <p>Used in <code>rejoinNeededOrPending</code></p> <ul style="list-style-type: none"> • Set to the <code>metadataSnapshot</code> in <code>performAssignment</code> • Reset (to <code>null</code>) in <code>onJoinComplete</code> (when <code>isLeader</code> flag is off)
<code>completedOffsetCommits</code>	<p>Java's <code>java.util.concurrent.ConcurrentLinkedQueue</code> of <code>OffsetCommitCompletions</code> with user-defined <code>OffsetCommitCallbacks</code></p> <p><code>OffsetCommitCallbacks</code> are executed in <code>invokeCompletedOffsetCommitCallbacks</code></p> <p><code>OffsetCommitCompletion</code> are added in <code>commitOffsetsAsync</code> (and also directly in <code>doCommitOffsetsAsync</code>)</p>
<code>isLeader</code>	<p>Flag that says whether the <code>ConsumerCoordinator</code> is a leader (<code>true</code>) or not (<code>false</code>).</p> <p>Default: <code>false</code></p> <p>Turned on in <code>performAssignment</code> and off in <code>onJoinPrepare</code></p> <p>Used exclusively for <code>onJoinComplete</code> (to reset the <code>assignmentSnapshot</code> when off)</p>
<code>metadataSnapshot</code>	<p><code>MetadataSnapshot metadataSnapshot</code></p> <p><code>MetadataSnapshot</code> (for the <code>SubscriptionState</code> and the <code>cluster metadata</code> from the <code>Metadata</code>)</p> <ul style="list-style-type: none"> • Updated in the <code>Metadata.Listener</code> from <code>addMetadataListener</code> <p>Used in <code>rejoinNeededOrPending</code> and <code>performAssignment</code></p>

Enable `ERROR`, `WARN`, `INFO`, `DEBUG`, or `TRACE` logging level for `org.apache.kafka.clients.consumer.internals.ConsumerCoordinator` logger to see what happens inside.

Add the following line to `log4j.properties`:

Tip

```
log4j.logger.org.apache.kafka.clients.consumer.internals.ConsumerCoordinator=TRA
```

Refer to [Logging](#).

Sending Asynchronous Auto-Commit of Offsets — `maybeAutoCommitOffsetsAsync` Method

```
void maybeAutoCommitOffsetsAsync(long now)
```

`maybeAutoCommitOffsetsAsync` ...FIXME

Note

`maybeAutoCommitOffsetsAsync` is used when:

- `KafkaConsumer` is requested to [manually assign partitions](#)
- `ConsumerCoordinator` is requested to [poll](#)

Sending Synchronous Auto-Commit of Offsets — `maybeAutoCommitOffsetsSync` Internal Method

```
void maybeAutoCommitOffsetsSync(long timeoutMs)
```

`maybeAutoCommitOffsetsSync` ...FIXME

Note

`maybeAutoCommitOffsetsSync` is used when `ConsumerCoordinator` is requested to [close](#) or [onJoinPrepare](#)

`doAutoCommitOffsetsAsync` Internal Method

```
void doAutoCommitOffsetsAsync()
```

`doAutoCommitOffsetsAsync` ...FIXME

Note

`doAutoCommitOffsetsAsync` is used exclusively when `ConsumerCoordinator` is requested to `maybeAutoCommitOffsetsAsync`.

close Method

```
void close(long timeoutMs)
```

`close ...FIXME`

Note

`close` is used when...FIXME

commitOffsetsAsync Method

```
void commitOffsetsAsync(
    final Map<TopicPartition, OffsetAndMetadata> offsets,
    final offsetCommitCallback callback)
```

`commitOffsetsAsync ...FIXME`

Note

`commitOffsetsAsync` is used when:

- `KafkaConsumer` is requested to `commitAsync`
- `ConsumerCoordinator` is requested to `doAutoCommitOffsetsAsync`

commitOffsetsSync Method

```
boolean commitOffsetsSync(
    Map<TopicPartition,
    OffsetAndMetadata> offsets,
    long timeoutMs)
```

`commitOffsetsSync ...FIXME`

Note

`commitOffsetsSync` is used when...FIXME

refreshCommittedOffsetsIfNeeded Method

```
void refreshCommittedOffsetsIfNeeded()
```

```
refreshCommittedOffsetsIfNeeded ...FIXME
```

Note

`refreshCommittedOffsetsIfNeeded` is used when...FIXME

onJoinComplete Callback

```
void onJoinComplete(
    int generation,
    String memberId,
    String assignmentStrategy,
    ByteBuffer assignmentBuffer)
```

Note

`onJoinComplete` is part of [AbstractCoordinator Contract](#) to...FIXME.

`onJoinComplete` ...FIXME

onJoinPrepare Method

```
void onJoinPrepare(int generation, String memberId)
```

Note

`onJoinPrepare` is part of [AbstractCoordinator Contract](#) to...FIXME.

`onJoinPrepare` ...FIXME

Performing Partition Assignment (using PartitionAssignor) — performAssignment Method

```
Map<String, ByteBuffer> performAssignment(
    String leaderId,
    String assignmentStrategy,
    Map<String, ByteBuffer> allSubscriptions)
```

Note

`performAssignment` is part of [AbstractCoordinator Contract](#) to perform partition assignment (i.e. assign partitions to the members of a consumer group).

`performAssignment` tries to find the [PartitionAssignor](#) by the given `assignmentStrategy`.

`performAssignment` deserializes `Subscriptions` (from the given `allSubscriptions`) for every entry (and creates a `Map<String, Subscription>`).

`performAssignment` requests the [SubscriptionState](#) to add the topics to group subscription (i.e. add all the topics that the consumer group is subscribed to to receive topic metadata updates).

`performAssignment` requests the [Metadata](#) to `setTopics` to be the `groupSubscription` of the [SubscriptionState](#).

`performAssignment` requests the [ConsumerNetworkClient](#) to update the cluster metadata.

`performAssignment` turns the `isLeader` internal flag on.

`performAssignment` prints out the following DEBUG message to the logs:

```
Performing assignment using strategy [assignor] with subscriptions [subscriptions]
```

`performAssignment` requests the [PartitionAssignor](#) to assign partitions to the members of the consumer group (with the current [cluster metadata](#) and subscriptions) and gets the partition assignment back (as `Map<String, Assignment>`).

Note	Here <code>performAssignment</code> does some customizations that a user-customized assignor could request. It is not a very interesting code to spend your time on.
------	--

`performAssignment` sets the [assignmentSnapshot](#) to be [metadataSnapshot](#).

`performAssignment` prints out the following DEBUG message to the logs:

```
Finished assignment for group: [assignment]
```

In the end, `performAssignment` takes the partition assignment, encodes the `Assignment` per group member to create a `groupAssignment` (i.e. the `Map<String, ByteBuffer>` that is returned).

`performAssignment` throws an `IllegalStateException` when no [PartitionAssignor](#) could be found:

```
Coordinator selected invalid assignment protocol: [assignmentStrategy]
```

maybeLeaveGroup Method

```
void maybeLeaveGroup()
```

`maybeLeaveGroup` ...FIXME

Note`maybeLeaveGroup` is used when...FIXME

updatePatternSubscription Method

```
void updatePatternSubscription(Cluster cluster)
```

`updatePatternSubscription` ...FIXME

Note`updatePatternSubscription` is used when...FIXME

needRejoin Method

```
boolean needRejoin()
```

Note`needRejoin` is part of the [AbstractCoordinator Contract](#) to...FIXME.

`needRejoin` ...FIXME

timeToNextPoll Method

```
long timeToNextPoll(long now)
```

`timeToNextPoll` ...FIXME

Note`timeToNextPoll` is used when...FIXME

Polling for Group Coordinator Events — poll Method

```
boolean poll(Timer timer)
```

`poll` first [invokeCompletedOffsetCommitCallbacks](#).

`poll` branches off per whether the [SubscriptionState](#) is [partitionsAutoAssigned](#) or not.

Caution`FIXME` What does `partitionsAutoAssigned` mean exactly?

In [partitionsAutoAssigned](#), `poll` [pollHeartbeat](#).

`poll` returns `false` if [coordinatorUnknown](#) and [ensureCoordinatorReady](#) failed (`false`).

`poll ...FIXME`

Note	<code>poll</code> is used exclusively when <code>KafkaConsumer</code> is requested to <code>updateAssignmentMetadataIfNeeded</code> .
------	---

Registering Metadata.Listener — `addMetadataListener` Internal Method

`void addMetadataListener()`

`addMetadataListener` requests the `Metadata` to add a new `Metadata Update Listener` that intercepts `onMetadataUpdate` events and does the following:

- `FIXME`

`addMetadataListener` throws a `TopicAuthorizationException` for any unauthorized topics (i.e. when the given `cluster` has at least one topic in `unauthorizedTopics`).

`FIXME`

Note	<code>addMetadataListener</code> is used exclusively when <code>ConsumerCoordinator</code> is <code>created</code> .
------	--

`fetchCommittedOffsets` Method

`Map<TopicPartition, OffsetAndMetadata> fetchCommittedOffsets(Set<TopicPartition> partitions)``fetchCommittedOffsets ...FIXME`

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

Creating ConsumerCoordinator Instance

`ConsumerCoordinator` takes the following when created:

- `LogContext`
- `ConsumerNetworkClient`
- Group ID
- `rebalanceTimeoutMs`

- `sessionTimeoutMs`
- `heartbeatIntervalMs`
- [PartitionAssignors](#)
- [Metadata](#)
- [SubscriptionState](#)
- [Metrics](#)
- Prefix of the metric group
- `Time`
- `retryBackoffMs`
- `autoCommitEnabled` flag
- `autoCommitIntervalMs`
- [ConsumerInterceptors](#)
- `excludeInternalTopics` flag
- `leaveGroupOnClose` flag

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

In the end, `ConsumerCoordinator` requests the [Metadata](#) to [update](#) and [addMetadataListener](#).

rejoinNeededOrPending Method

```
boolean rejoinNeededOrPending()
```

Note	<code>rejoinNeededOrPending</code> is part of the AbstractCoordinator Contract to...FIXME.
------	--

`rejoinNeededOrPending` ...FIXME

Sending OffsetCommitRequest to Group Coordinator (Kafka Broker) — `sendOffsetCommitRequest` Internal Method

```
RequestFuture<Void> sendOffsetCommitRequest(
    final Map<TopicPartition, OffsetAndMetadata> offsets)
```

`sendOffsetCommitRequest` gets the node of the consumer group.

`sendOffsetCommitRequest` creates a `OffsetCommitRequest.Builder` for the group ID and an offset data based on the given offsets.

`sendOffsetCommitRequest` prints out the following TRACE message to the logs:

```
Sending OffsetCommit request with [offsets] to coordinator [coordinator]
```

`sendOffsetCommitRequest` requests the `ConsumerNetworkClient` to send an `OffsetCommitRequest` to the group coordinator and then creates a new `OffsetCommitResponseHandler` to handle a response.

`sendOffsetCommitRequest` returns immediately when there is no offsets to send.

`sendOffsetCommitRequest` returns immediately with a `COORDINATOR_NOT_AVAILABLE` failure when the consumer group coordinator is not available.

Note	<code>sendOffsetCommitRequest</code> is used when <code>ConsumerCoordinator</code> is requested to <code>doCommitOffsetsAsync</code> and <code>commitOffsetsSync</code> .
------	---

sendOffsetFetchRequest Internal Method

```
RequestFuture<Map<TopicPartition, OffsetAndMetadata>> sendOffsetFetchRequest(
    Set<TopicPartition> partitions)
```

`sendOffsetFetchRequest` ...FIXME

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

invokeCompletedOffsetCommitCallbacks Internal Method

```
void invokeCompletedOffsetCommitCallbacks()
```

`invokeCompletedOffsetCommitCallbacks` takes (polls) every `offsetCommitCompletion` from the `completedOffsetCommits` internal registry and requests it to invoke `OffsetCommitCallback.onComplete`.

Note

`invokeCompletedOffsetCommitCallbacks` is used when `ConsumerCoordinator` is requested to `poll`, `close`, `commitOffsetsAsync`, and `commitOffsetsSync`.

Finding PartitionAssignor by Name — `lookupAssignor` Internal Method

```
PartitionAssignor lookupAssignor(String name)
```

`lookupAssignor` tries to find the `PartitionAssignor` by the given `name` in the `assignors`.

If not found, `lookupAssignor` returns `null`.

Note

`lookupAssignor` is used when `ConsumerCoordinator` is requested to `onJoinComplete` and `performAssignment`.

`doCommitOffsetsAsync` Internal Method

```
void doCommitOffsetsAsync(  
    final Map<TopicPartition, OffsetAndMetadata> offsets,  
    final OffsetCommitCallback callback)
```

`doCommitOffsetsAsync` ...FIXME

Note

`doCommitOffsetsAsync` is used exclusively when `ConsumerCoordinator` is requested to `commitOffsetsAsync`.

AbstractCoordinator Contract

`AbstractCoordinator` is the [base](#) of [Coordinators](#) that [FIXME](#).

Note	<code>AbstractCoordinator</code> is a Java abstract class and cannot be created directly. It is created indirectly for the concrete Coordinators .
------	--

Note	<code>ConsumerCoordinator</code> is the one and only known implementation of the AbstractCoordinator Contract in Apache Kafka's Consumer API.
------	---

Table 1. AbstractCoordinator Contract (Abstract Methods Only)

Method	Description
<code>metadata</code>	<pre data-bbox="620 287 1097 316"><code>List<ProtocolMetadata> metadata()</code></pre> <p>Used exclusively when <code>AbstractCoordinator</code> is requested to send a JoinGroupRequest to the group coordinator</p>
<code>onJoinComplete</code>	<pre data-bbox="620 521 1049 664"><code>void onJoinComplete(int generation, String memberId, String protocol, ByteBuffer memberAssignment)</code></pre> <p>Used exclusively when <code>AbstractCoordinator</code> is requested to joinGroupIfNeeded (and the request to initiateJoinGroup succeeded)</p>
<code>onJoinPrepare</code>	<pre data-bbox="620 902 898 983"><code>void onJoinPrepare(int generation, String memberId)</code></pre> <p>Used exclusively when <code>AbstractCoordinator</code> is requested to joinGroupIfNeeded (and the <code>needsJoinPrepare</code> flag is on)</p>
<code>performAssignment</code>	<pre data-bbox="620 1224 1240 1345"><code>Map<String, ByteBuffer> performAssignment(String leaderId, String protocol, Map<String, ByteBuffer> allMemberMetadata)</code></pre> <p>Performs partition assignment (i.e. assigns partitions to the members of a consumer group)</p> <p>Used exclusively when <code>AbstractCoordinator</code> is requested to perform partition assignment and notify the group coordinator</p>
<code>protocolType</code>	<pre data-bbox="620 1666 927 1695"><code>String protocolType()</code></pre> <p>Used exclusively when <code>AbstractCoordinator</code> is requested to send a JoinGroupRequest to the group coordinator</p>

Table 2. AbstractCoordinator's Internal Properties (e.g. Registries, Counters and Flags)

Name	Description
	Group Coordinator (as a Kafka Node)

<code>coordinator</code>	Initialized when <code>FindCoordinatorResponseHandler</code> is requested to <code>onSuccess</code>
<code>findCoordinatorFuture</code>	<p><code>RequestFuture<Void></code> for <code>sendFindCoordinatorRequest</code> when <code>lookupCoordinator</code></p> <p>Reset in <code>clearFindCoordinatorFuture</code></p> <p>Used in <code>lookupCoordinator</code> and when <code>HeartbeatThread</code> is requested to run</p>
<code>generation</code>	Defaults to <code>NO_GENERATION</code>
<code>heartbeatThread</code>	<p><code>HeartbeatThread</code></p> <p>Initialized and immediately started when <code>AbstractCoordinator</code> is requested to start a <code>HeartbeatThread</code></p>
<code>joinFuture</code>	<p><code>RequestFuture<ByteBuffer></code> for sending a <code>JoinGroupRequest</code> to the group coordinator when <code>initiateJoinGroup</code></p> <p>Reset in <code>resetJoinGroupFuture</code></p> <p>Used in <code>rejoinNeededOrPending</code> and <code>initiateJoinGroup</code></p>
<code>needsJoinPrepare</code>	Flag to control whether to execute <code>onJoinPrepare</code> while performing <code>joinGroupIfNeeded</code>
<code>rejoinNeeded</code>	<p><code>boolean rejoinNeeded</code></p> <p>Flag to control whether to <code>rejoinNeededOrPending</code>. Enabled (<code>true</code>) by default.</p> <p>Turned off (<code>false</code>) when the <code>joinFuture</code> completed successfully in <code>initiateJoinGroup</code></p> <p>Turned on (<code>true</code>) in <code>resetGeneration</code> and <code>requestRejoin</code></p>
<code>sensors</code>	<code>GroupCoordinatorMetrics</code>
<code>state</code>	<p>State of the member of a consumer group</p> <p>Possible states:</p> <ul style="list-style-type: none"> • <code>UNJOINED</code> (when not joined in a consumer group when attempting to join a consumer group or <code>resetGeneration</code>) • <code>REBALANCING</code> (when attempting to join a consumer group)

- `STABLE` (when successfully joined a consumer group)

Default: `UNJOINED`

Tip

Enable `WARN`, `INFO`, or `DEBUG` logging level for `org.apache.kafka.clients.consumer.internals.AbstractCoordinator` logger to see what happens inside.

Add the following line to `log4j.properties`:

```
log4j.logger.org.apache.kafka.clients.consumer.internals.AbstractCoordinator=DEBUG
```

Refer to [Logging](#).

Creating AbstractCoordinator Instance

`AbstractCoordinator` takes the following when created:

- `LogContext`
- [ConsumerNetworkClient](#)
- Group ID
- `rebalanceTimeoutMs`
- `sessionTimeoutMs`
- `heartbeatIntervalMs` (or simply a `Heartbeat` instance)
- [Metrics](#)
- Metric group prefix
- `Time`
- `retryBackoffMs`
- `leaveGroupOnClose` flag

Note

`AbstractCoordinator` is a Java abstract class and cannot be [created](#) directly. It is created indirectly for the [concrete Coordinators](#).

joinGroupIfNeeded Method

```
boolean joinGroupIfNeeded(final Timer timer)
```

```
joinGroupIfNeeded ...FIXME
```

Note

`joinGroupIfNeeded` is used exclusively when `AbstractCoordinator` is requested to `ensureActiveGroup`.

Attempting to Join Consumer Group — `initiateJoinGroup` Internal Method

```
RequestFuture<ByteBuffer> initiateJoinGroup()
```

`initiateJoinGroup` disables the `HeartbeatThread`.

`initiateJoinGroup` sets the `state` as `REBALANCING`.

`initiateJoinGroup` sends a `JoinGroupRequest` to the group coordinator (and initializes the `joinFuture`).

`initiateJoinGroup` registers a `RequestFutureListener` to handle a response.

When the response is successful, `initiateJoinGroup` does the following:

1. Prints out the following INFO message to the logs:

```
Successfully joined group with generation [generationId]
```

2. Sets the `state` as `STABLE` (and the `rejoinNeeded` flag off)
3. Requests the `HeartbeatThread` to enable itself.

When the response is a failure, `initiateJoinGroup` simply sets the `state` as `UNJOINED`

`initiateJoinGroup` exits immediately when the `joinFuture` has already been initialized (is not `null`) and gives it back.

Note

`initiateJoinGroup` is used exclusively when `AbstractCoordinator` is requested to `joinGroupIfNeeded`.

ensureActiveGroup Method

```
void ensureActiveGroup() (1)
boolean ensureActiveGroup(final Timer timer)
```

1. Calls the other `ensureActiveGroup` in an infinite `while` loop with an unexpiring timer

```
ensureActiveGroup ...FIXME
```

Note

`ensureActiveGroup` is used exclusively when `ConsumerCoordinator` is requested to poll for coordinator events.

Discovering Current Coordinator for Consumer Group — `lookupCoordinator` Method

```
RequestFuture<Void> lookupCoordinator()
```

`lookupCoordinator` uses the `RequestFuture` internal registry to know whether it was requested earlier but have not finished yet. In other words, the `RequestFuture` is available when looking up a coordinator of a consumer group is in progress.

If the `RequestFuture` internal registry is available (not a `null`), `lookupCoordinator` returns it immediately.

Otherwise, when the `RequestFuture` internal registry is uninitialized (`null`), `lookupCoordinator` requests the `ConsumerNetworkClient` for the least-loaded Kafka broker. `lookupCoordinator` then sends a `FindCoordinatorRequest` to the Kafka broker.

`lookupCoordinator` prints out the following DEBUG message to the logs when no nodes are available and finishes with a `RequestFuture.noBrokersAvailable` failure.

```
No broker available to send FindCoordinator request
```

Note

`lookupCoordinator` is used when:

- `AbstractCoordinator` is requested to `ensureCoordinatorReady`
- `ConsumerCoordinator` is requested to `commitOffsetsAsync`
- `HeartbeatThread` is requested to `run`

ensureCoordinatorReady Method

```
boolean ensureCoordinatorReady(final Timer timer)
```

```
ensureCoordinatorReady ...FIXME
```

Note

- `ensureCoordinatorReady` is used when:
- `AbstractCoordinator` is requested to `ensureActiveGroup` and `joinGroupIfNeeded`
 - `ConsumerCoordinator` is requested to `poll`, `fetchCommittedOffsets`, `close`, and `commitOffsetsSync`

Sending JoinGroupRequest to Group Coordinator (Kafka Broker) — `sendJoinGroupRequest` Method

```
RequestFuture<ByteBuffer> sendJoinGroupRequest()
```

`sendJoinGroupRequest` prints out the following INFO message to the logs:

```
(Re-)joining group
```

`sendJoinGroupRequest` creates a new `JoinGroupRequest.Builder` (for the `groupId`, the `sessionTimeoutMs`, the `member ID`, `consumer` protocol type, the `supportedProtocolMetadata`, and the `rebalanceTimeoutMs`).

`sendJoinGroupRequest` prints out the following DEBUG message to the logs:

```
Sending JoinGroup ([requestBuilder]) to coordinator [coordinator]
```

In the end, `sendJoinGroupRequest` requests the `ConsumerNetworkClient` to `send` the `JoinGroupRequest` to the `group coordinator` node and then creates a new `JoinGroupResponseHandler` to handle a response.

`sendJoinGroupRequest` returns immediately with a `RequestFuture.coordinatorNotAvailable()` when the `coordinator has not been discovered yet`.

Note

`sendJoinGroupRequest` is used exclusively when `AbstractCoordinator` is requested to `initiateJoinGroup`.

Sending SyncGroupRequest (to Kafka Broker) — `sendSyncGroupRequest` Internal Method

```
RequestFuture<ByteBuffer> sendSyncGroupRequest(  
    SyncGroupRequest.Builder requestBuilder)
```

```
sendSyncGroupRequest ...FIXME
```

Note

`sendSyncGroupRequest` is used when...FIXME

Sending FindCoordinatorRequest (to Kafka Broker) — `sendFindCoordinatorRequest` Internal Method

```
RequestFuture<Void> sendFindCoordinatorRequest(Node node)
```

`sendFindCoordinatorRequest` prints out the following DEBUG message to the logs:

```
Sending FindCoordinator request to broker [node]
```

`sendFindCoordinatorRequest` creates a new `FindCoordinatorRequest` with `GROUP` type and the `groupId`.

In the end, `sendFindCoordinatorRequest` requests the `ConsumerNetworkClient` to `send` the `FindCoordinatorRequest` (to the given Kafka node) and then creates a new `FindCoordinatorResponseHandler` to handle a response.

Note

`sendFindCoordinatorRequest` is used exclusively when `AbstractCoordinator` is requested to `lookupCoordinator`.

maybeLeaveGroup Method

```
void maybeLeaveGroup()
```

`maybeLeaveGroup` ...FIXME

Note

`maybeLeaveGroup` is used when...FIXME

Sending HeartbeatRequest to Group Coordinator (Kafka Broker)— `sendHeartbeatRequest` Method

```
RequestFuture<Void> sendHeartbeatRequest()
```

`sendHeartbeatRequest` prints out the following DEBUG message to the logs:

```
Sending Heartbeat request to coordinator [coordinator]
```

`sendHeartbeatRequest` creates a new `HeartbeatRequest.Builder` (for the `groupId`, `generation` and `member` IDs).

In the end, `sendHeartbeatRequest` requests the `ConsumerNetworkClient` to send the `HeartbeatRequest` to the `group coordinator` node and then creates a new `HeartbeatResponseHandler` to handle a response.

Note

`sendHeartbeatRequest` is used exclusively when `HeartbeatThread` is requested to run.

Performing Partition Assignment and Notifying Group Coordinator— `onJoinLeader` Internal Method

```
RequestFuture<ByteBuffer> onJoinLeader(JoinGroupResponse joinResponse)
```

`onJoinLeader` performs partition assignment with the leader ID, the group protocol, and the member metadata (all from the given `JoinGroupResponse`).

`onJoinLeader` creates a new `SyncGroupRequest.Builder` (with the `group`, `generation`, and `member` IDs as well as the group partition assignment).

`onJoinLeader` prints out the following DEBUG message to the logs:

```
Sending leader SyncGroup to coordinator [coordinator]: [requestBuilder]
```

In the end, `onJoinLeader` sends a `SyncGroupRequest` (to the group coordinator node).

Note

`onJoinLeader` is used exclusively when `JoinGroupResponseHandler` is requested to handle a successful response from the group coordinator (and the consumer is the leader of the consumer group).

`resetJoinGroupFuture` Internal Method

```
void resetJoinGroupFuture()
```

`resetJoinGroupFuture` simply resets the `joinFuture` internal registry (i.e. sets it to `null`).

Note

`resetJoinGroupFuture` is used exclusively when `AbstractCoordinator` is requested to `joinGroupIfNeeded`.

Clearing FindCoordinatorFuture

— `clearFindCoordinatorFuture` Internal Method

```
void clearFindCoordinatorFuture()
```

`clearFindCoordinatorFuture` simply resets the `findCoordinatorFuture` internal registry (to be `null`).

Note

`clearFindCoordinatorFuture` is used exclusively when `FindCoordinatorResponseHandler` is requested to `onSuccess` and `onFailure`.

rejoinNeededOrPending Internal Method

```
boolean rejoinNeededOrPending()
```

`rejoinNeededOrPending` is positive (`true`) when `rejoinNeeded` and `joinFuture` is initialized (i.e. not `null`).

Note

`rejoinNeededOrPending` is used when:

- `ConsumerCoordinator` is requested to `rejoinNeededOrPending` and `poll`
- `KafkaConsumer` is requested to `pollForFetches`
- `AbstractCoordinator` is requested to `joinGroupIfNeeded`

pollHeartbeat Method

```
void pollHeartbeat(long now)
```

`pollHeartbeat` ...FIXME

Note

`pollHeartbeat` is used exclusively when `consumerCoordinator` is requested to poll for Coordinator events.

Starting HeartbeatThread

— `startHeartbeatThreadIfNeeded` Internal Method

```
void startHeartbeatThreadIfNeeded()
```

`startHeartbeatThreadIfNeeded` ...FIXME

Note

`startHeartbeatThreadIfNeeded` is used exclusively when `AbstractCoordinator` is requested to [ensureActiveGroup](#).

needRejoin Method

```
boolean needRejoin()
```

`needRejoin` simply returns the [rejoinNeeded](#) flag.

Note

`needRejoin` is used when...FIXME

requestRejoin Method

```
void requestRejoin()
```

`requestRejoin` simply turns the [rejoinNeeded](#) flag on.

Note

`requestRejoin` is used when:

- `HeartbeatResponseHandler` is requested to [handle a response](#) that the consumer group is rebalancing
- `SyncGroupResponseHandler` is requested to [handle a response](#) with an error

resetGeneration Method

```
void resetGeneration()
```

`resetGeneration` simply resets the following internal registries:

- `generation` becomes `NO_GENERATION`
- `rejoinNeeded` is turned on
- `state` is `UNJOINED`

Note

- `resetGeneration` is used when:
- `AbstractCoordinator` is requested to `maybeLeaveGroup`
 - `HeartbeatResponseHandler`, `JoinGroupResponseHandler`, `SyncGroupResponseHandler`, and `OffsetCommitResponseHandler` are requested to handle a response with an error (`UNKNOWN_MEMBER_ID` , `ILLEGAL_GENERATION` , or `REBALANCE_IN_PROGRESS`)

coordinatorUnknown Method

```
boolean coordinatorUnknown()
```

`coordinatorUnknown` ...FIXME

Note

`coordinatorUnknown` is used when...FIXME

disableHeartbeatThread Internal Method

```
void disableHeartbeatThread()
```

`disableHeartbeatThread` simply requests the `HeartbeatThread` to `disable` (if available, i.e. not `null`).

Note

`disableHeartbeatThread` is used exclusively when `AbstractCoordinator` is requested to `initiateJoinGroup`.

checkAndGetCoordinator Method

```
Node checkAndGetCoordinator()
```

`checkAndGetCoordinator` ...FIXME

Note

`checkAndGetCoordinator` is used when...FIXME

HeartbeatThread

`HeartbeatThread` is a daemon Kafka thread of execution that `AbstractCoordinator` starts to send HeartbeatRequests to the group coordinator (aka *heartbeating*) regularly every `request.timeout.ms` (default: 30000 ms).

`HeartbeatThread` is created and immediately started exclusively for `AbstractCoordinator` when requested to start a `HeartbeatThread` (when `ConsumerCoordinator` is requested to poll for coordinator events after `KafkaConsumer` was requested to poll for records).

`HeartbeatThread` uses **kafka-coordinator-heartbeat-thread** prefix followed by the `group ID` as the thread name.

```
kafka-coordinator-heartbeat-thread | [groupId]
```

`HeartbeatThread` uses `closed` flag to indicate...FIXME

`HeartbeatThread` uses `enabled` flag to indicate whether to wait for `AbstractCoordinator` to notify it (*release*). The flag is off (`false`) initially and can be turned on (`true`) and off by `enable` or `disable`, respectively.

`HeartbeatThread` takes no arguments to be created.

Starting Thread — run Method

```
void run()
```

Note	<code>run</code> is part of the <code>java.lang.Runnable</code> to start itself as a separately-executing thread.
------	---

`run` prints out the following DEBUG message to the logs:

```
Heartbeat thread started
```

In an infinite `while` loop, `run` does the following (infinitely):

1. Checks the internal state to exit or wait:
 - i. Exits when `closed`
 - ii. Waits for `AbstractCoordinator` to wake it up when not `enabled`

2. Requests the [ConsumerNetworkClient](#) to `pollNoWakeup`
3. Sends a [HeartbeatRequest](#) to the group coordinator
4. (when the response eventually arrives) Requests the [Heartbeat](#) to record a success (`receiveHeartbeat`) or a failure (`failHeartbeat`)

In the end, when the infinite `while` loop is terminated for any reason, `run` prints out the following DEBUG message to the logs:

```
Heartbeat thread has closed
```

Warning	<code>run</code> does some other state control checks but they look uninteresting...to me...now.
---------	--

Disabling HeartbeatThread — `disable` Method

```
void disable()
```

`disable` prints out the following DEBUG message to the logs and turns the `enabled` flag off (`false`).

```
Disabling heartbeat thread
```

Note	<code>disable</code> is used when: <ul style="list-style-type: none"> • <code>AbstractCoordinator</code> is requested to disableHeartbeatThread • <code>HeartbeatThread</code> is requested to <code>run</code> (when in <code>UNJOINED</code> or <code>REBALANCING states</code>)
------	--

Enabling HeartbeatThread — `enable` Method

```
void enable()
```

`enable` prints out the following DEBUG message to the logs:

```
Enabling heartbeat thread
```

`enable` turns the `enabled` flag on (`true`) and requests the [Heartbeat](#) to `resetTimeouts`.

In the end, `enable` notifies (*wakes up*) the owning [AbstractCoordinator](#).

Note

`enable` is used exclusively when `AbstractCoordinator` is requested to [initiateJoinGroup](#).

GroupCoordinatorMetrics

`GroupCoordinatorMetrics` is a set of [metrics](#) to measure performance of [AbstractCoordinator](#).

Table 1. GroupCoordinatorMetrics's Sensors and Metrics

Sensor Name	Metric Name	Description
heartbeat-latency	heartbeat-rate	The number of heartbeats per second
heartbeat-latency	heartbeat-response-time-max	The max time taken to receive a response to a heartbeat request
heartbeat-latency	heartbeat-total	The total number of heartbeats
join-latency	join-rate	The number of group joins per second
join-latency	join-time-avg	The average time taken for a group rejoin
join-latency	join-time-max	The max time taken for a group rejoin
join-latency	join-total	The total number of group joins
	last-heartbeat-seconds-ago	The number of seconds since the last coordinator heartbeat was sent
sync-latency	sync-rate	The number of group syncs per second
sync-latency	sync-time-avg	The average time taken for a group sync
sync-latency	sync-time-max	The max time taken for a group sync
sync-latency	sync-total	The total number of group syncs

`GroupCoordinatorMetrics` is registered in **consumer-coordinator-metrics** group.

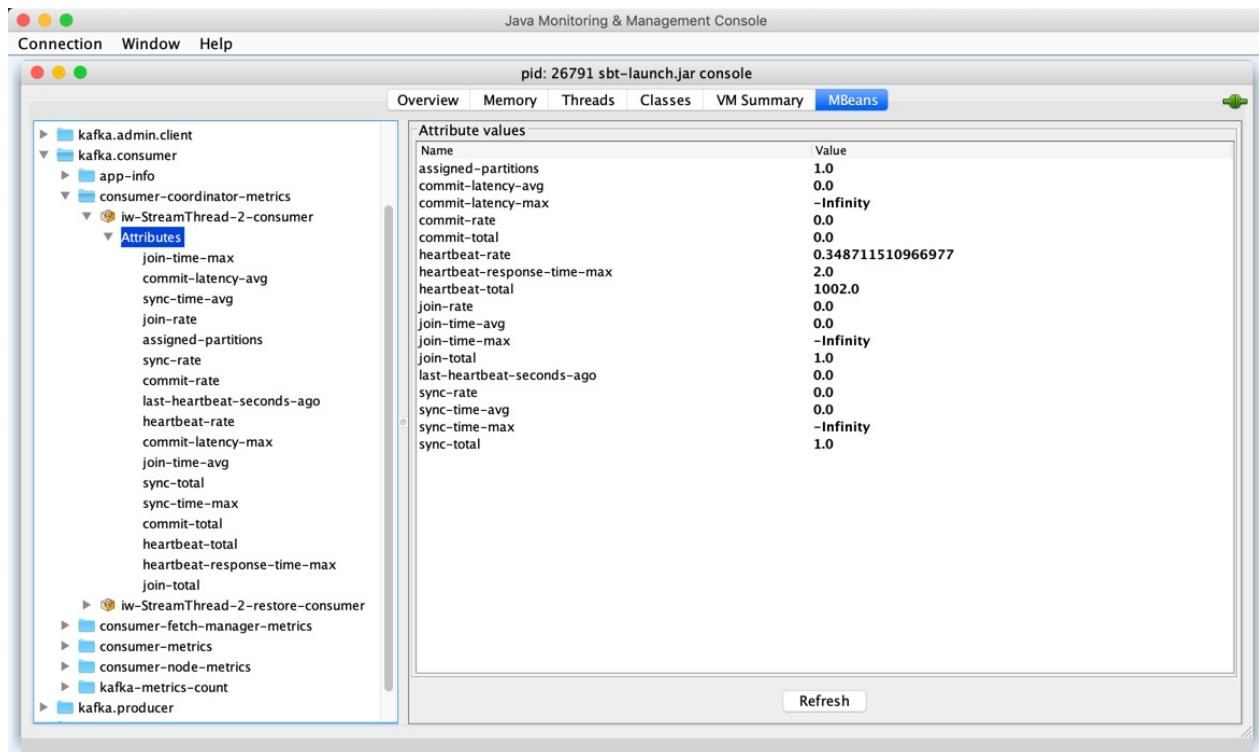


Figure 1. GroupCoordinatorMetrics in jconsole

`GroupCoordinatorMetrics` is created exclusively for `AbstractCoordinator`.

`GroupCoordinatorMetrics` takes the following to be created:

- Metrics
- Prefix of the metric group name (i.e. **consumer**)

`GroupCoordinatorMetrics` uses the `prefix` followed by `-coordinator-metrics` for the **metric group name** (e.g. `consumer-coordinator-metrics`).

```
[prefix]-coordinator-metrics
```

ConsumerNetworkClient

`ConsumerNetworkClient` is a high-level Kafka consumer that...FIXME

`ConsumerNetworkClient` is created for [KafkaConsumer](#) and [AdminClient](#).

Table 1. ConsumerNetworkClient's Internal Properties (e.g. Registries and Counters)

Name	Description
<code>pendingCompletion</code>	
<code>unsent</code>	<code>UnsentRequests</code> with pending requests per node that have not been sent yet (i.e. awaiting transmission).

Tip	<p>Enable <code>DEBUG</code> logging level for <code>org.apache.kafka.clients.consumer.internals.ConsumerNetworkClient</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.clients.consumer.internals.ConsumerNetworkClient=DEBUG</pre> <p>Refer to Logging.</p>
------------	--

checkDisconnects Internal Method

```
void checkDisconnects(long now)
```

`checkDisconnects` ...FIXME

Note	<p><code>checkDisconnects</code> is used exclusively when <code>ConsumerNetworkClient</code> is requested to <code>poll</code>.</p>
-------------	---

"Sending" Request to Broker— send Method

```
RequestFuture<ClientResponse> send(Node node, AbstractRequest.Builder<?> requestBuilder)
```

`send` creates a `RequestFutureCompletionHandler` and requests the [KafkaClient](#) for a new [ClientRequest](#) (with the `RequestFutureCompletionHandler` and expecting a response).

`send` records the new `ClientRequest` with the `node` in `unsent` internal registry.

`send` requests the `KafkaClient` to `wake up`.

Note	<code>send</code> is used...FIXME
------	-----------------------------------

wakeup Method

```
void wakeup()
```

`wakeup` prints out the following DEBUG message to the logs:

```
Received user wakeup
```

`wakeup` turns the internal `wakeup` flag on and requests `KafkaClient` to `wakeup`

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

Updating Cluster Metadata (and Possibly Blocking Until It Refreshes) — ensureFreshMetadata Method

```
void ensureFreshMetadata()
```

`ensureFreshMetadata` waits for metadata update when `Metadata` was requested for `update` or `time to the next update` is now.

Note	<code>ensureFreshMetadata</code> is used when <code>ConsumerCoordinator</code> does <code>onJoinComplete</code> , <code>performAssignment</code> and <code>poll</code> .
------	--

pendingRequestCount Method

```
int pendingRequestCount()
int pendingRequestCount(Node node)
```

`pendingRequestCount` ...FIXME

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

Least-Loaded Kafka Node — leastLoadedNode Method

```
Node leastLoadedNode()
```

leastLoadedNode ...FIXME

Note

- `leastLoadedNode` is used when:
- `AbstractCoordinator` is requested to `lookupCoordinator`
 - `Fetcher` is requested to `sendMetadataRequest`

poll Method

```
void poll(RequestFuture<?> future)
boolean poll(RequestFuture<?> future, Timer timer)
void poll(Timer timer)
void poll(Timer timer, PollCondition pollCondition)
void poll(Timer timer, PollCondition pollCondition, boolean disableWakeup)
```

poll ...FIXME

Note

`poll` is used when...FIXME

Waiting for Metadata Update — awaitMetadataUpdate Method

```
boolean awaitMetadataUpdate(long timeout)
```

awaitMetadataUpdate ...FIXME

Note

`awaitMetadataUpdate` is used when...FIXME

awaitPendingRequests Method

```
boolean awaitPendingRequests(Node node, Timer timer)
```

awaitPendingRequests ...FIXME

Note

`awaitPendingRequests` is used when...FIXME

pollNowakeup Method

```
void pollNowakeup()
```

pollNowakeup ...FIXME

Note

pollNowakeup is used when:

- KafkaConsumer is requested to [poll for records](#)
- AbstractCoordinator is requested to [maybeLeaveGroup](#)
- HeartbeatThread is requested to [run](#)
- ConsumerCoordinator is requested to [commitOffsetsAsync](#)

Creating ConsumerNetworkClient Instance

ConsumerNetworkClient takes the following when created:

- LogContext
- KafkaClient
- Metadata
- Time
- retryBackoffMs
- requestTimeoutMs

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

trySend Internal Method

```
long trySend(long now)
```

trySend ...FIXME

Note

trySend is used exclusively when ConsumerNetworkClient is requested to [poll](#).

Initiating Connector to Kafka Node — tryConnect Method

```
void tryConnect(Node node)
```

`tryConnect` simply requests the [KafkaClient](#) to [initiate a connection](#) to the given broker [Node](#).

Note

`tryConnect` is used exclusively when `FindCoordinatorResponseHandler` is requested to [onSuccess](#).

handlePendingDisconnects Internal Method

```
void handlePendingDisconnects()
```

`handlePendingDisconnects` ...FIXME

Note

`handlePendingDisconnects` is used exclusively when `consumerNetworkClient` is requested to [poll](#).

maybeTriggerWakeup Method

```
void maybeTriggerWakeup()
```

`maybeTriggerWakeup` ...FIXME

Note

`maybeTriggerWakeup` is used when...FIXME

ConsumerMetrics

ConsumerMetrics is...FIXME

Fetcher

Fetcher is created exclusively when KafkaConsumer is created.



Figure 1. Fetcher and KafkaConsumer

KafkaConsumer uses the following configuration properties to create a Fetcher:

- `fetch.min.bytes` for minimum number of bytes
- `fetch.max.bytes` for maximum number of bytes
- `fetch.max.wait.ms` for maximum wait time
- `max.partition.fetch.bytes` for fetch size
- `max.poll.records` for how many records to poll
- `check.crcs` for flag to check CRC or not
- `retry.backoff.ms` for retry backoff (in milliseconds)
- `request.timeout.ms` for request timeout (in milliseconds)

Table 1. Fetcher's Internal Properties (e.g. Registries and Counters) (in alphabetical order)

Name	Description
<code>client</code>	<code>ConsumerNetworkClient</code> that is given when Fetcher is created.

Creating Fetcher Instance

Fetcher takes the following when created:

- `ConsumerNetworkClient`
- Minimum number of bytes
- Maximum number of bytes
- Maximum wait time
- Fetch size
- How many records to poll

- Flag to check CRC or not
- [Deserializer](#) for keys
- [Deserializer](#) for values
- [Metadata](#)
- [SubscriptionState](#)
- [Metrics](#)
- [FetcherMetricsRegistry](#)
- [Time](#)
- Retry backoff (in milliseconds)
- Request timeout (in milliseconds)
- [IsolationLevel](#)

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

`Fetcher` registers itself with [SubscriptionState](#) as a listener to receive notifications about...
FIXME

sendFetches Method

Caution

FIXME

sendMetadataRequest Internal Method

```
RequestFuture<ClientResponse> sendMetadataRequest(MetadataRequest.Builder request)
```

Internally, `sendMetadataRequest` requests [ConsumerNetworkClient](#) for the [least loaded node](#).

With the node, `sendMetadataRequest` requests [ConsumerNetworkClient](#) to [send the request to the node](#).

When no node was found, `sendMetadataRequest` returns a [RequestFuture](#) with [NoAvailableBrokersException](#).

Note

`sendMetadataRequest` is used exclusively when `Fetcher` is requested for a [topic metadata](#).

beginningOffsets Method

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

retrieveOffsetsByTimes Method

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

Getting Topic Metadata — getTopicMetadata Method

```
Map<String, List<PartitionInfo>> getTopicMetadata(
    MetadataRequest.Builder request,
    long timeout)
```

Internally, `getTopicMetadata` sends the metadata request and requests `ConsumerNetworkClient` to poll until it finishes successfully or `timeout` expires.

After `poll` finishes, `getTopicMetadata` takes the cluster information from `MetadataResponse`.

When `MetadataResponse` is successful, `getTopicMetadata` takes topics (from `cluster`) and requests `cluster` for available partitions for every topic.

In the end, `getTopicMetadata` creates a collection of topic and partitions pairs.

Caution	FIXME Describe the failure path
---------	---------------------------------

Note	<code>getTopicMetadata</code> is used when: <ul style="list-style-type: none"> • <code>Fetcher</code> is requested to find metadata for all topics • <code>KafkaConsumer</code> is requested to find partitions for a topic
------	---

Finding Metadata for All Topics — getAllTopicMetadata Method

```
Map<String, List<PartitionInfo>> getAllTopicMetadata(long timeout)
```

`getAllTopicMetadata` gets topic metadata specifying no topics (which means all topics available).

Note

`getAllTopicMetadata` is used exclusively when `KafkaConsumer` requests metadata for all topics.

sendListOffsetRequest Internal Method

```
RequestFuture<ListOffsetResult> sendListOffsetRequest(  
    final Node node,  
    final Map<TopicPartition, ListOffsetRequest.PartitionData> timestampsToSearch,  
    boolean requireTimestamp)
```

sendListOffsetRequest ...FIXME

Note

sendListOffsetRequest is used when...FIXME

RequestFutureListener

RequestFutureListener is...FIXME

SubscriptionState

`SubscriptionState` allows for tracking the topics, partitions, and offsets assigned to a Kafka consumer.

`SubscriptionState` is in **partitionsAutoAssigned** if it is `AUTO_TOPICS` or `AUTO_PATTERN`.

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

Name	Description
<code>listener</code>	<p><code>ConsumerRebalanceListener</code> for notifications about partition assignment changes</p> <p>Used exclusively when <code>ConsumerCoordinator</code> is notified about:</p> <ul style="list-style-type: none"> • <code>onJoinComplete</code> (that triggers <code>ConsumerRebalanceListener.onPartitionsAssigned</code>) • <code>onJoinPrepare</code> (that triggers <code>ConsumerRebalanceListener.onPartitionsRevoked</code>)
<code>groupSubscription</code>	

subscribe Method

```
void subscribe(Pattern pattern, ConsumerRebalanceListener listener)
```

subscribe ...FIXME

Note	<code>subscribe</code> is used exclusively when <code>KafkaConsumer</code> is requested to subscribe to the topics that match a pattern with a <code>ConsumerRebalanceListener</code> .
------	---

Adding Topics to Group Subscription — groupSubscribe Method

```
void groupSubscribe(Collection<String> topics)
```

groupSubscribe ...FIXME

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

RequestFuture

RequestFuture is...FIXME

compose Method

```
RequestFuture<S> compose(final RequestFutureAdapter<T, S> adapter)
```

compose ...FIXME

Note	<p>compose is used when:</p> <ul style="list-style-type: none">• AbstractCoordinator is requested to send a JoinGroupRequest to the group coordinator, sendSyncGroupRequest, sendFindCoordinatorRequest, maybeLeaveGroup, and sendHeartbeatRequest• ConsumerCoordinator is requested to send an OffsetCommitRequest to the group coordinator (a Kafka broker) and sendOffsetFetchRequest• Fetcher is requested to sendListOffsetRequest
------	---

RequestFutureAdapter Contract

`RequestFutureAdapter` is the base of `RequestFuture` type converters that can do type conversion when a `RequestFuture` completes with a `success` or a `failure`.

Table 1. RequestFutureAdapter Contract

Method	Description
<code>onFailure</code>	<pre>void onFailure(RuntimeException e, RequestFuture<T> future)</pre> <p>Performs a type conversion when a <code>RequestFuture</code> completes with a failure</p> <p>Used when:</p> <ul style="list-style-type: none"> • <code>FindCoordinatorResponseHandler</code> is requested to <code>onFailure</code> • <code>RequestFuture</code> is requested to <code>compose</code>
<code>onSuccess</code>	<pre>void onSuccess(F value, RequestFuture<T> future)</pre> <p>Performs type conversion when a <code>RequestFuture</code> completes with a success</p> <p>Used exclusively when <code>RequestFuture</code> is requested to <code>compose</code></p>

Table 2. RequestFutureAdapters (Direct Implementations)

RequestFutureAdapter	Description
<code>CoordinatorResponseHandler</code>	
<code>FindCoordinatorResponseHandler</code>	

CoordinatorResponseHandler Contract

`CoordinatorResponseHandler` is the extension of the [RequestFutureAdapter contract](#) for type converters that handle ClientResponses.

Table 1. CoordinatorResponseHandler Contract

Method	Description
<code>handle</code>	<pre>void handle(R response, RequestFuture<T> future)</pre> <p>Handles a successful response from a group coordinator Used exclusively when <code>CoordinatorResponseHandler</code> is requested to onSuccess</p>

Table 2. CoordinatorResponseHandlers

CoordinatorResponseHandler	Description
HeartbeatResponseHandler	
JoinGroupResponseHandler	Handles JoinGroupResponses
LeaveGroupResponseHandler	
OffsetCommitResponseHandler	
OffsetFetchResponseHandler	
SyncGroupResponseHandler	

onFailure Method

```
void onFailure(RuntimeException e, RequestFuture<T> future)
```

Note	<code>onFailure</code> is part of the RequestFutureAdapter Contract to perform a type conversion when a <code>RequestFuture</code> completes with a failure.
------	--

`onFailure ...FIXME`

onSuccess Method

```
void onSuccess(ClientResponse clientResponse, RequestFuture<T> future)
```

Note	<p>onSuccess is part of the RequestFutureAdapter Contract to perform a type conversion when a RequestFuture completes with a success.</p>
------	---

onSuccess ...FIXME

FindCoordinatorResponseHandler

`FindCoordinatorResponseHandler` is a concrete [RequestFutureAdapter](#) of [ClientResponse](#).

`FindCoordinatorResponseHandler` is used exclusively when `AbstractCoordinator` is requested to [discover the coordinator for a consumer group](#) (and when `successful` sets the [group coordinator](#) as a Kafka node with the hostname and the port).

Performing Response Conversion on Success — `onSuccess` Method

```
void onSuccess(ClientResponse resp, RequestFuture<Void> future)
```

Note	<code>onSuccess</code> is part of the RequestFutureAdapter Contract to perform a type conversion when a <code>RequestFuture</code> completes with a success.
------	--

`onSuccess` prints out the following DEBUG message to the logs:

```
Received FindCoordinator response [resp]
```

`onSuccess` [clears the findCoordinatorFuture internal registry](#).

`onSuccess` takes the [FindCoordinatorResponse](#) (from the given `ClientResponse`) and branches off per `error`.

No Error

With no error, `onSuccess` [sets the group coordinator](#) (as a Kafka node with the hostname and the port).

`onSuccess` prints out the following INFO message to the logs:

```
Discovered group coordinator [coordinator]
```

`onSuccess` [requests the ConsumerNetworkClient to initiate a connection to the group coordinator](#).

In the end, `onSuccess` [requests the Heartbeat to resetSessionTimeout](#) and completes the given `RequestFuture<Void>`.

GROUP_AUTHORIZATION_FAILED Error

In case of `GROUP_AUTHORIZATION_FAILED` error, `onSuccess` raises a `GroupAuthorizationException` in the given `RequestFuture<Void>`.

Other Errors

All other errors lead `onSuccess` to print out the following DEBUG message to the logs and raise the error in the given `RequestFuture<Void>`.

```
Group coordinator lookup failed: [message]
```

Performing Response Conversion on Failure

— onFailure Method

```
void onFailure(RuntimeException e, RequestFuture<Void> future)
```

Note	<code>onFailure</code> is part of the RequestFutureAdapter Contract to perform a type conversion when a <code>RequestFuture</code> completes with a failure.
------	--

`onFailure ...FIXME`

HeartbeatResponseHandler

HeartbeatResponseHandler is...FIXME

JoinGroupResponseHandler

`JoinGroupResponseHandler` is a concrete [CoordinatorResponseHandler](#) to handle a successful [JoinGroupResponse](#) from a group coordinator (and convert it to a `ByteBuffer`).

Note

`JoinGroupResponseHandler` is a Java private class of [AbstractCoordinator](#) and so can only be used by [AbstractCoordinators](#).

`JoinGroupResponseHandler` is created exclusively when `AbstractCoordinator` is requested to send a [JoinGroupRequest](#) to the group coordinator (when requested to [initiateJoinGroup](#) when [joinGroupIfNeeded](#) which is when `ConsumerCoordinator` is requested to [poll for coordinator events](#)).

`JoinGroupResponseHandler` takes no arguments to be created.

Handling Successful Response From Group Coordinator — `handle` Method

```
void handle(
    JoinGroupResponse joinResponse,
    RequestFuture<ByteBuffer> future)
```

Note

`handle` is part of the [CoordinatorResponseHandler Contract](#) to handle a successful response from a group coordinator.

`handle` branches off per the error in the given [JoinGroupResponse](#).

No Errors

`handle` prints out the following DEBUG message to the logs:

```
Received successful JoinGroup response: [joinResponse]
```

`handle` requests the [GroupCoordinatorMetrics](#) for the [joinLatency](#) sensor to record the request latency.

When in `REBALANCING` state, `handle` creates a new [Generation](#) (with the generation ID, the member ID, and the group protocol from the given `JoinGroupResponse`).

In the end, `handle` checks if the given `JoinGroupResponse` is for the leader of the consumer group or a follower and performs partition assignment and notifies the group coordinator or `onJoinFollower`, respectively.

When not in `REBALANCING state`, `handle` requests the raises the given `RequestFuture<ByteBuffer>` to raise a `UnjoinedGroupException`.

Caution	There are other errors, but of little interest currently (aka <i>FIXME</i>).
---------	---

OffsetCommitResponseHandler

OffsetCommitResponseHandler is...FIXME

SyncGroupResponseHandler

SyncGroupResponseHandler is...FIXME

Broker Nodes — Kafka Servers

Note

A **Kafka server**, a **Kafka broker** and a **Kafka node** all refer to the same concept and are synonyms (see the scaladoc of [KafkaServer](#)).

A **Kafka broker** is modelled as [KafkaServer](#) that hosts [topics](#).

Note

Given topics are always partitioned across brokers in a cluster a single broker hosts topic partitions of one or more topics actually (even when a topic is only partitioned to just a single partition).

Quoting [Broker](#) article (from Wikipedia, the free encyclopedia):

A broker is an individual person who arranges transactions between a buyer and a seller for a commission when the deal is executed.

A broker's prime responsibility is to bring sellers and buyers together and thus a broker is the third-person facilitator between a buyer and a seller.

A Kafka broker receives messages from producers and stores them on disk keyed by unique **offset**.

A Kafka broker allows consumers to fetch messages by topic, partition and offset.

Kafka brokers can create a Kafka cluster by sharing information between each other directly or indirectly using Zookeeper.

A Kafka cluster has exactly one broker that acts as the [Controller](#).

You can [start a single Kafka broker](#) using `kafka-server-start.sh` script.

Starting Kafka Broker

Start Zookeeper.

```
./bin/zookeeper-server-start.sh config/zookeeper.properties
```

Only when Zookeeper is up and running you can start a Kafka server (that will connect to Zookeeper).

```
./bin/kafka-server-start.sh config/server.properties
```

Tip

Read [kafka-server-start.sh](#) script.

kafka-server-start.sh script

`kafka-server-start.sh` starts a [Kafka broker](#).

```
$ ./bin/kafka-server-start.sh  
USAGE: ./bin/kafka-server-start.sh [-daemon] server.properties [--override property=value]*
```

Note Before you run `kafka-server-start.sh` make sure that Zookeeper is up and running. Use `zookeeper-server-start` shell script.

`kafka-server-start.sh` uses `config/log4j.properties` for logging configuration that you can override using `KAFKA_LOG4J_OPTS` environment variable.

```
KAFKA_LOG4J_OPTS="-Dlog4j.configuration=file:config/log4j.properties"
```

`kafka-server-start.sh` accepts `KAFKA_HEAP_OPTS` and `EXTRA_ARGS` environment variables.

Command-line options:

1. `-name` — defaults to `kafkaServer` when in daemon mode.
2. `-loggc` — enabled when in daemon mode.
3. `-daemon` — enables daemon mode.
4. `--override property=value` — `value` that should override the value set for `property` in `server.properties` file.

```
$ ./bin/kafka-server-start.sh config/server.properties --override broker.id=100  
...  
INFO [KafkaServer id=100] started (kafka.server.KafkaServer)
```

Broker

`Broker` represents a Kafka broker that has an id, a host, a port, communication endpoints (and few other properties).

```
// List the brokers in a cluster
$ ./bin/zookeeper-shell.sh :2181 ls /brokers/ids
[0]

$ ./bin/zookeeper-shell.sh :2181 get /brokers/ids/0
Connecting to :2181
>{"listener_security_protocol_map":{"PLAINTEXT":"PLAINTEXT"},"endpoints":["PLAINTEXT://
192.168.0.187:9092"],"jmx_port":-1,"host":"192.168.0.187","timestamp":"1543830073084",
"port":9092,"version":4}
```

createBroker Method

```
createBroker(id: Int, brokerInfoString: String): Broker
```

`createBroker ...FIXME`

Note

`createBroker` is used when...FIXME

Topics

Topics are virtual groups of one or many [partitions](#) across [Kafka brokers](#) in a Kafka cluster.

A single Kafka broker stores messages in a partition in an ordered fashion, i.e. appends them one message after another and creates a log file.

[Producers](#) write messages to the tail of these logs that [consumers](#) read at their own pace.

Kafka scales topic consumption by distributing partitions among a [consumer group](#), which is a set of consumers sharing a common group identifier.

Note	Due to limitations in metric names, topics with a period (.) or underscore (_) could collide. To avoid issues it is best to use either, but not both.
------	---

Use [kafka-topics](#) shell script to manage topics.

Partitions

Partitions with messages — topics can be partitioned to improve read/write performance and resiliency. You can lay out a topic (as partitions) across a cluster of machines to allow data streams larger than the capability of a single machine. Partitions are log files on disk with sequential write only. Kafka guarantees message ordering in a partition.

The **log end offset** is the offset of the last message written to a log.

The **high watermark offset** is the offset of the last message that was successfully copied to all of the log's replicas.

Note	A consumer can only read up to the high watermark offset to prevent reading unreplicated messages.
------	--

Messages

Messages are the data that brokers store in the [partitions of a topic](#).

Messages are sequentially appended to the end of the partition log file and numbered by unique [offsets](#). They are persisted on disk (aka *disk-based persistence*) and replicated within the cluster to prevent data loss. It has an in-memory page cache to improve data reads. Messages are in partitions until deleted when [TTL](#) occurs or after [compaction](#).

Offsets

Offsets are message positions in a topic.

Kafka Clients

- Producers
- Consumers

Producers

Multiple concurrent **producers** that send (aka *push*) messages to topics which is appending the messages to the end of partitions. They can batch messages before they are sent over the wire to a topic. Producers support message compression. Producers can send messages in synchronous (with acknowledgement) or asynchronous mode.

```
import collection.JavaConversions._
import org.apache.kafka.common.serialization._
import org.apache.kafka.clients.producer.KafkaProducer
import org.apache.kafka.clients.producer.ProducerRecord

val cfg = Map(
  "bootstrap.servers" -> "localhost:9092",
  "key.serializer" -> classOf[IntegerSerializer],
  "value.serializer" -> classOf[StringSerializer])
val producer = new KafkaProducer[Int, String](cfg)
val msg = new ProducerRecord(topic = "my-topic", key = 1, value = "hello")

scala> val f = producer.send(msg)
f: java.util.concurrent.Future[org.apache.kafka.clients.producer.RecordMetadata] = org.apache.kafka.clients.producer.internals.FutureRecordMetadata@2e9e8fe

scala> f.get
res7: org.apache.kafka.clients.producer.RecordMetadata = my-topic-0@1

producer.close
```

Kafka Consumers

Multiple concurrent **consumers** read (aka *pull*) messages from topics however they want using [offsets](#). Unlike typical messaging systems, Kafka consumers pull messages from a topic using offsets.

A **Kafka consumer** consume records from a Kafka cluster.

Note	Kafka 0.9.0.0 was about introducing a brand new Consumer API aka New Consumer .
------	--

When a consumer is created, it requires [bootstrap.servers](#) which is the initial list of brokers to discover the full set of alive brokers in a cluster from.

A consumer has to subscribe to the topics it wants to read messages from called [topic subscription](#).

Caution	FIXME Building a custom consumption strategy
---------	--

Using Kafka Consumer API requires **kafka-clients** dependency in your project.

```
val kafka = "0.10.2.1"
libraryDependencies += "org.apache.kafka" % "kafka-clients" % kafka

// You should also define the logging binding for slf4j
// Kafka uses slf4j for logging
val logback = "1.2.3"
libraryDependencies += "ch.qos.logback" % "logback-core" % logback
libraryDependencies += "ch.qos.logback" % "logback-classic" % logback
```

Consumer Contract

```
public interface Consumer<K, V> extends Closeable {
    // FIXME more methods...
    ConsumerRecords<K, V> poll(long timeout)
}
```

Table 1. Consumer Contract

Method	Description
poll	Used to...

Topic Subscription

Topic Subscription is the process of announcing the topics a consumer wants to read messages from.

```
void subscribe(Collection<String> topics)
void subscribe(Collection<String> topics, ConsumerRebalanceListener callback)
void subscribe(Pattern pattern, ConsumerRebalanceListener callback)
```

Note

`subscribe` method is not incremental and you always must include the full list of topics that you want to consume from.

You can change the set of topics a consumer is subscribed to at any time and (given the note above) any topics previously subscribed to will be replaced by the new list after `subscribe`.

Automatic and Manual Partition Assignment

Caution

FIXME

Consumer Groups

A **consumer group** is a set of Kafka consumers that share a common link:a set of consumers sharing a common group identifier#group.id[group identifier].

Partitions in a [topic](#) are assigned to exactly one member in a consumer group.

Group Coordination Protocol

Caution

FIXME

- the new consumer uses a group coordination protocol built into Kafka
- For each group, one of the brokers is selected as the group coordinator. The coordinator is responsible for managing the state of the group. Its main job is to mediate partition assignment when new members arrive, old members depart, and when topic metadata changes. The act of reassigning partitions is known as rebalancing the group.
- When a group is first initialized, the consumers typically begin reading from either the earliest or latest offset in each partition. The messages in each partition log are then read sequentially. As the consumer makes progress, it commits the offsets of messages it has successfully processed.

- When a partition gets reassigned to another consumer in the group, the initial position is set to the last committed offset. If a consumer suddenly crashed, then the group member taking over the partition would begin consumption from the last committed offset (possibly reprocessing messages that the failed consumer would have processed already but not committed yet).

Further reading or watching

1. [Introducing the Kafka Consumer: Getting Started with the New Apache Kafka 0.9 Consumer Client](#)

Clusters

A Kafka **cluster** is the central data exchange backbone for an organization.

Metrics

Metrics is...FIXME

Sensor

Sensor is...FIXME

MetricsReporter

JmxReporter

`JmxReporter` is a metrics reporter that is always included in `metric.reporters` setting with `kafka.consumer` metrics prefix.

ProducerMetrics

ProducerMetrics is...FIXME

```
// Generate HTML with the metrics
$ ./bin/kafka-run-class.sh org.apache.kafka.clients.producer.internals.ProducerMetrics
> metrics.html
```

SenderMetrics

SenderMetrics is...FIXME

updateProduceRequestMetrics Method

```
void updateProduceRequestMetrics(Map<Integer, List<ProducerBatch>> batches)
```

updateProduceRequestMetrics ...FIXME

Note	updateProduceRequestMetrics is used exclusively when Sender is sendProducerData.
------	--

updateProduceRequestMetrics is used exclusively when Sender is sendProducerData.

Kafka Tools

ConsoleProducer

```
kafka.tools.ConsoleProducer
```

```
./bin/kafka-console-producer.sh --broker-list localhost:9092 --topic my-topic
```

ConsoleConsumer

```
kafka.tools.ConsoleConsumer
```

```
./bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic my-topic
```

kafka-configs.sh Shell Script

kafka-configs.sh is a shell script that...FIXME

```
./bin/kafka-configs.sh \
--alter \
--zookeeper localhost:2181 \
--entity-type topics \
--entity-name test \
--add-config retention.ms=5000
```

```
./bin/kafka-configs.sh \
--alter \
--bootstrap-server localhost:9092 \
--entity-type brokers \
--entity-name 0 \
--add-config advertised.listeners=PLAINTEXT://localhost:9092
```

kafka-topics.sh Shell Script

`kafka-topics.sh` is a shell script that...FIXME

Kafka Properties

Table 1. Properties

Name	Description
advertised.listeners	<p>Comma-separated list of URLs to publish to clients to use, if different than the listeners.</p> <p>Default: <code>null</code></p> <p>In IaaS environments (e.g. Docker, Kubernetes), <code>advertised.listeners</code> may need to be different from the routable meta-address. This is because the Kafka broker advertises an address that can be reached from both local and external hosts. If not set, listeners is used.</p> <p>Unlike listeners it is invalid to advertise the routable meta-address.</p> <ul style="list-style-type: none"> • Use KafkaConfig.AdvertisedListenersFactory to reference the property • Use KafkaConfig.advertisedListeners to get the current value <p>Dynamic reconfigurable configuration</p>
authorizer.class.name	
auto.commit.interval.ms	How often (in milliseconds) consumer offsets are auto-committed when enable.auto.commit is enabled.
auto.create.topics.enable	<p>Enables auto creation of a topic</p> <p>Default: <code>true</code></p> <ul style="list-style-type: none"> • Use KafkaConfig.AutoCreateTopicsEnableFactory to reference the property • Use KafkaConfig.autoCreateTopicsEnable to get the current value
auto.leader.rebalance.enable	<p>Enables auto leader balancing</p> <p>Default: <code>true</code></p> <p>A background thread checks and triggers leader rebalances as required.</p> <ul style="list-style-type: none"> • Use KafkaConfig.AutoLeaderRebalanceEnableFactory to reference the property

	<ul style="list-style-type: none"> • Use KafkaConfig.autoLeaderRebalance to access the current value
auto.offset.reset	<p>Reset policy — what to do when there is no Kafka or if the current offset does not exist on the server (e.g. because that data has been deleted).</p> <ul style="list-style-type: none"> • earliest — automatically reset the offset to the earliest offset • latest — automatically reset the offset to the latest offset • none — throw an exception to the consumer if no previous offset is found for the consumer • anything else: throw an exception to a consumer <p>Default: <code>latest</code></p>
background.threads	<p>The number of threads to use for various background processing tasks.</p> <p>Default: <code>10</code></p> <p>Must be at least <code>1</code></p> <p>Dynamic reconfigurable configuration</p>
bootstrap.servers	<p>A comma-separated list of host:port pairs representing the addresses of one or more brokers in a Kafka cluster. For example, <code>localhost:9092</code> or <code>localhost:9092,anotherhost:9092</code>.</p> <p>Default: (empty)</p>
broker.id	<p>The broker id of a Kafka broker for identification purposes. If unset, a unique broker id will be generated. This can cause conflicts between zookeeper generated broker id's and user configured broker id's, generated broker id's must be greater than <code>reserved.broker.max.id</code> + 1.</p> <p>Use KafkaConfig.brokerId to access the current broker id.</p>
broker.id.generation.enable	<p>Enables automatic broker id generation for a Kafka broker.</p> <p>Default: <code>true</code></p> <p>When enabled (<code>true</code>) the value configured for <code>reserved.broker.max.id</code> should be checked.</p> <ul style="list-style-type: none"> • Use KafkaConfig.BrokerIdGenerationEnabled to reference the property

	<ul style="list-style-type: none"> • Use KafkaConfig.brokerIdGenerationEpoch to access the current value 		
broker.rack			
check.crcs	<p>Automatically check the CRC32 of the reconstructed messages. This ensures no on-the-wire or on-disk corruption of messages occurred. This check adds some overhead to each message. It may be disabled in cases seeking extreme performance.</p> <p>Use <code>ConsumerConfig.CHECK_CRCS_CONFIG</code></p>		
client.id	<p>Default: (randomly-generated)</p>		
connection.failed.authentication.delay.ms	<p>Connection close delay on failed authentication. The time (in milliseconds) by which connection will be delayed on authentication failure. This must be less than connections.max.idle.ms to prevent connection timeout.</p> <p>Default: 100</p> <p>Has to be at least 0</p>		
connections.max.idle.ms	<p>Idle connections timeout: the server socket threads close the connections that idle more than this duration.</p> <p>Default: 10 * 60 * 1000L</p>		
default.replication.factor	<p>The default replication factor that is used for topics</p> <p>Default: 1</p> <p>Increase the default value to at least 2</p>		
delegation.token.master.key			
delete.topic.enable	<p>Enables topic deletion</p> <table border="1"> <tr> <td>Note</td><td>Deleting topic through the admin API will not effect with the property disabled.</td></tr> </table> <p>Default: true</p>	Note	Deleting topic through the admin API will not effect with the property disabled.
Note	Deleting topic through the admin API will not effect with the property disabled.		
	<p>When enabled (i.e. true) consumer offset will be committed automatically in the background (consumer auto commit) every auto.commit.interval.ms.</p> <p>Default: true</p>		

<code>enable.auto.commit</code>	<p>When disabled, offsets have to be committed (synchronously using KafkaConsumer.commit) or asynchronously KafkaConsumer.commitAsynchronous to restart restore the position of a consumer using KafkaConsumer.seek.</p> <p>Used when <code>KafkaConsumer</code> is created and controlled by ConsumerCoordinator.</p>
<code>fetch.max.bytes</code>	<p>The maximum amount of data the server should return in a fetch request. Records are fetched in batches by the consumer, and if the first record batch in the non-empty partition of the fetch is larger than this value, the record batch will still be returned to ensure the consumer can make progress. As such, this is an absolute maximum. The maximum record batch size accepted by the broker is defined via message.max.bytes (broker config) or max.message.bytes (topic config) so it's possible that the consumer performs multiple fetches.</p> <p>Use <code>ConsumerConfig.FETCH_MAX_BYTES_CONFIG</code></p>
<code>fetch.max.wait.ms</code>	<p>The maximum amount of time the server will wait for the client to answer the fetch request if there isn't sufficient data to immediately satisfy the requirement given to <code>fetch.min.bytes</code>.</p> <p>Use <code>ConsumerConfig.FETCH_MAX_WAIT_MS_CONFIG</code></p>
<code>fetch.min.bytes</code>	<p>The minimum amount of data the server should return in a fetch request. If insufficient data is available, the server will wait for that much data to accumulate before answering the request. The default setting of 1 means that fetch requests are answered as soon as a single byte of data is available or the fetch times out waiting for data to arrive. Setting this to a value greater than 1 will cause the server to wait for larger amounts of data to accumulate which can increase throughput a bit at the cost of some additional latency.</p> <p>Use <code>ConsumerConfig.FETCH_MIN_BYTES_CONFIG</code></p>
<code>group.id</code>	<p>The name of the consumer group the consumer belongs to.</p>
<code>heartbeat.interval.ms</code>	<p>The expected time between heartbeats to the coordinator when using Kafka's group management facilities.</p>
<code>host.name</code>	<p>The hostname a Kafka broker listens on.</p> <p>Default: (empty)</p>

<code>inter.broker.listener.name</code>	<p>Name of the listener that is used for inter-broker communication</p> <p>Default: <code>security.inter.broker.protocol</code></p> <p>It is not allowed to set <code>inter.broker.listener.name</code> and <code>security.inter.broker.protocol</code> properties at the same time.</p>
<code>inter.broker.protocol.version</code>	<p>Version of the inter-broker protocol</p> <p>Default: the latest <code>ApiVersion</code> (e.g. <code>2.1-IV</code>)</p> <p>Typically bumped up after all brokers were updated to a new version</p> <ul style="list-style-type: none"> • Use <code>KafkaConfig.InterBrokerProtocolV</code> to reference the property • Use <code>KafkaConfig.interBrokerProtocolV</code> to access the current value
<code>interceptor.classes</code>	<p>Comma-separated list of <code>ConsumerInterceptor</code> and <code>ProducerInterceptor</code> class names.</p> <p>Default: (empty)</p>
<code>key.deserializer</code>	How to deserialize message keys.
<code>leader.imbalance.check.interval.seconds</code>	<p>How often the active <code>KafkaController</code> schedules a <code>partition rebalance check</code> (aka <code>AutoLeaderImbalanceCheck</code>, <code>AutoPreferredReplicaLeaderElection</code> or <code>autoImbalanceCheck</code>).</p> <p>Default: <code>300</code></p> <ul style="list-style-type: none"> • Use <code>KafkaConfig.LeaderImbalanceCheckInterval</code> to reference the property • Use <code>KafkaConfig.leaderImbalanceCheckInterval</code> to access the current value
<code>listeners</code>	<p>Comma-separated list of URIs and listener names that the Kafka broker will listen on.</p> <p>Default: <code>PLAINTEXT://host.name:port</code></p> <p>Use <code>0.0.0.0</code> to bind to all the network interfaces of the machine or leave it empty to bind to the default port.</p> <ul style="list-style-type: none"> • Use <code>KafkaConfig.ListenersProp</code> to reference the property • Use <code>KafkaConfig.listeners</code> to access the current value

	Dynamic reconfigurable configuration		
listener.security.protocol.map			
log.cleaner.enable			
log.cleaner.threads	Dynamic reconfigurable configuration		
log.cleaner.dedupe.buffer.size	Dynamic reconfigurable configuration		
log.cleaner.io.buffer.load.factor	Dynamic reconfigurable configuration		
log.cleaner.io.buffer.size	Dynamic reconfigurable configuration		
log.cleaner.io.max.bytes.per.second	Dynamic reconfigurable configuration		
log.cleaner.backoff.ms	Dynamic reconfigurable configuration		
log.dir	The directory in which the log data is kept Default: /tmp/kafka-logs		
log.dirs	The directories in which the log data is kept Default: log.dir Use KafkaConfig.logDirs to access the current value.		
max.block.ms			
max.partition.fetch.bytes	The maximum amount of data per-partition return. Records are fetched in batches by the consumer. If the first record batch in the first non-empty partition fetch is larger than this limit, the batch will be split to ensure that the consumer can make progress. This is the maximum record batch size accepted by the broker defined via message.max.bytes (broker config) or max.message.bytes (topic config). Use ConsumerConfig.MAX_PARTITION_FETCH_BY		
	<table border="1"> <tr> <td>Note</td> <td>Use fetch.max.bytes for limiting the request size.</td> </tr> </table>	Note	Use fetch.max.bytes for limiting the request size.
Note	Use fetch.max.bytes for limiting the request size.		
	(KafkaConsumer) The maximum number of records returned from a Kafka Consumer when polling.		

	<p>The default setting (<code>-1</code>) sets no upper bound on the number of records, i.e. <code>Consumer.poll()</code> will return as soon as either any data is available or the poll timeout expires.</p> <p><code>max.poll.records</code> was added to Kafka in 0.11.0.1: KafkaConsumer Max Records.</p> <p>From kafka-clients mailing list:</p> <p><code>max.poll.records</code> only controls the number of records returned from poll, but does not control the fetching. The consumer will try to prefetch from all partitions it is assigned. It will then store those records and return them in batches of <code>max.poll.records</code> each (either all from the same topic partition if there are enough left to satisfy the number of records, or from multiple topic partitions if the data from the last fetch for one of the partitions does not cover the <code>max.poll.records</code> limit).</p> <p>Use <code>ConsumerConfig.MAX_POLL_RECORDS_CONFIG</code> to set this value.</p> <p>Internally, <code>max.poll.records</code> is used exclusively by the <code>KafkaConsumer</code> to create a <code>Fetcher</code> for each partition.</p>
<code>message.max.bytes</code>	Dynamic reconfigurable configuration
<code>metadata.max.age.ms</code>	
<code>metric.reporters</code>	<p>The list of fully-qualified classes names of the reporters.</p> <p>Default: JmxReporter</p>
<code>metrics.num.samples</code>	Number of samples to compute metrics.
<code>metrics.sample.window.ms</code>	Time window (in milliseconds) a metrics sample is computed over.
	<p>The minimum number of replicas in ISR that must commit a produce request with <code>required.acks = all</code> (the default)</p> <p>Default: <code>1</code></p> <p>When a Kafka producer sets <code>acks = all</code> (the default), this configuration specifies the minimum number of replicas that must acknowledge a write for the write to be considered successful.</p>

<code>min.insync.replicas</code>	<p>If this minimum cannot be met, then the producer will raise an exception (either <code>NotEnoughReplicas</code> or <code>NotEnoughReplicasAfterAppend</code>).</p> <p>When used together, <code>min.insync.replicas</code> allows you to enforce greater durability guarantees.</p> <p>A typical scenario would be to create a topic with a replication factor of 3, set <code>min.insync.replicas</code> to 2 and produce with acks of "all". This will ensure that if a majority of replicas receive a write, the producer raises an exception if a majority of replicas do not receive a write.</p>
<code>num.io.threads</code>	<p>The number of threads that KafkaServer uses for processing requests, which may include disk I/O.</p> <p>Default: 8</p> <p>Must be at least 1</p> <p>Dynamic reconfigurable configuration</p>
<code>max.connections.per.ip</code>	<p>The maximum number of connections allowed per ip address.</p> <p>Default: <code>Int.MaxValue</code></p> <p>Must be at least 0 (with 0 if there are overconnections, configured using <code>max.connections.per.ip.overrides</code> property)</p> <p>Dynamic reconfigurable configuration</p>
<code>max.connections.per.ip.overrides</code>	<p>A comma-separated list of per-ip or hostname overrides for the default maximum number of connections. For example: <code>hostName:100,127.0.0.1:200</code></p> <p>Default: (empty)</p> <p>Dynamic reconfigurable configuration</p>
<code>num.network.threads</code>	<p>The number of threads that SocketServer uses for number of processors per endpoint (for receiving from the network and sending responses to clients).</p> <p>Default: 3</p> <p>Must be at least 1</p> <p>Dynamic reconfigurable configuration</p>
<code>num.partitions</code>	<p>The number of log partitions for auto-create topics.</p> <p>Default: 1</p>

	<p>Increase the default value (1) since it is better to partition a topic that leads to a better data distribution and aids consumer parallelism.</p>
<code>num.recovery.threads.per.data.dir</code>	<p>The number of threads per log data directory for recovery at startup and flushing at shutdown.</p> <p>Default: 1</p> <p>Must be at least 1</p> <p>Dynamic reconfigurable configuration</p>
<code>num.replica.fetchers</code>	<p>The number of fetcher threads used to replicate messages from a source broker.</p> <p>Increasing this value can increase the degree of parallelism in the follower broker.</p> <p>Default: 1</p> <p>Dynamic reconfigurable configuration</p>
<code>port</code>	<p>The port a Kafka broker listens on.</p> <p>Default: 9092</p>
<code>principal.builder.class</code>	
<code>replica.fetch.backoff.ms</code>	<p>The amount of time to sleep when fetch pause occurs.</p> <p>Default: 1000</p> <p>Must be at least 0</p> <ul style="list-style-type: none"> • Use KafkaConfig.ReplicaFetchBackoff to reference the property • Use KafkaConfig.replicaFetchBackoffMs to get the current value
<code>queued.max.requests</code>	<p>The number of queued requests allowed before the network threads</p> <p>Default: 500</p> <p>Must be at least 1</p>
<code>rebalance.timeout.ms</code>	<p>The maximum allowed time for each worker group once a rebalance has begun.</p>
<code>receive.buffer.bytes</code>	<p>The hint about the size of the TCP network (SO_RCVBUF) to use (for a socket) when creating a new socket. If the value is -1, the OS default will be used.</p>

<code>replica.lag.time.max.ms</code>	
<code>replica.socket.timeout.ms</code>	
<code>request.timeout.ms</code>	<p>The configuration controls the maximum amount of time the client will wait for the response of a request. If no response is not received before the timeout, the client will resend the request if necessary or cancel the request if retries are exhausted.</p> <p>Use <code>ConsumerConfig.REQUEST_TIMEOUT_MS_CONFIG</code> to change the value.</p>
<code>reserved.broker.max.id</code>	<p>Maximum number that can be used for broker IDs. Must be at least <code>0</code>.</p> <p>Default: <code>1000</code></p> <ul style="list-style-type: none"> * Use <code>KafkaConfig.MaxReservedBrokerIdProperty</code> to change the property * Use <code>KafkaConfig.maxReservedBrokerId</code> to get the current value
<code>retry.backoff.ms</code>	<p>Time to wait before attempting to retry a failed request for a given topic partition. This avoids repeated requests in a tight loop under some failure scenarios.</p> <p>Use <code>ConsumerConfig.RETRY_BACKOFF_MS_CONFIG</code> to change the value.</p>
<code>sasl.mechanism.inter.broker.protocol</code>	
<code>sasl.jaas.config</code>	
<code>sasl.enabled.mechanisms</code>	
<code>sasl.kerberos.service.name</code>	
<code>sasl.kerberos.kinit.cmd</code>	
<code>sasl.kerberos.ticket.renew.window.factor</code>	
<code>sasl.kerberos.ticket.renew.jitter</code>	
<code>sasl.kerberos.min.time.before.relogin</code>	
<code>sasl.kerberos.principal.to.local.rules</code>	
<code>sasl.login.refresh.window.factor</code>	
<code>sasl.login.refresh.window.jitter</code>	

Properties

sasl.login.refresh.min.period.seconds	
sasl.login.refresh.buffer.seconds	
send.buffer.bytes	The hint about the size of the TCP network (SO_SNDBUF) to use (for a socket) when sending data. If the value is -1, the OS default will be used.
session.timeout.ms	The timeout used to detect worker failures. Default: 10000
socket.request.max.bytes	The maximum number of bytes in a socket Default: 100 * 1024 * 1024 Must be at least 1
ssl.protocol	
ssl.provider	
ssl.cipher.suites	
ssl.enabled.protocols	
ssl.keystore.type	
ssl.keystore.location	
ssl.keystore.password	
ssl.key.password	
ssl.truststore.type	
ssl.truststore.location	
ssl.truststore.password	
ssl.keymanager.algorithm	
ssl.trustmanager.algorithm	
ssl.endpoint.identification.algorithm	
ssl.secure.random.implementation	
ssl.client.auth	
	The maximum allowed timeout for transactions.

<code>transaction.max.timeout.ms</code>	If a client's requested transaction time exceeds this value, the broker will return an error in <code>InitProducerId</code> . This prevents a client from a too large timeout from stalling consumers reading from topics included in the transaction. Default: 15 minutes Must be at least 1 <ul style="list-style-type: none"> • Use KafkaConfig.transactionMaxTimeOut to access the current value
<code>unclean.leader.election.enable</code>	Controls whether to enable replicas not in the current leader's log to be elected as leader as the last resort, even though they may have older data. Default: 1
<code>value.deserializer</code>	How to deserialize message values
<code>zookeeper.connect</code>	Comma-separated list of Zookeeper hosts (IP:port pairs) that brokers register to, e.g. localhost:2181, 127.0.0.1:3000, 127.0.0.1:3001, 127.0.0.1:3002 Default: (empty) Zookeeper URIs can have an optional chroot path suffix at the end, e.g. 127.0.0.1:3000, 127.0.0.1:3001, 127.0.0.1:3002 If the optional chroot path suffix is used, all paths are relative to this path. It is recommended to include all the hosts in the ensemble (cluster) <ul style="list-style-type: none"> • Available as KafkaConfig.ZkConnectProps • Use KafkaConfig.zkConnect to access the current value
<code>zookeeper.connection.timeout.ms</code>	The max time that the client waits to establish a connection to zookeeper Default: zookeeper.session.timeout.ms <ul style="list-style-type: none"> • Available as KafkaConfig.ZkConnectionProps • Use KafkaConfig.zkConnectionTimeout to access the current value
	The maximum number of unacknowledged messages a client will send to Zookeeper before blocking. At least 1

<code>zookeeper.max.in.flight.requests</code>	<p>Default: 10</p> <ul style="list-style-type: none">• Available as <code>KafkaConfig.ZkMaxInFlight</code>• Use <code>KafkaConfig.zkMaxInFlightRequests</code> to get the current value
<code>zookeeper.session.timeout.ms</code>	<p>Zookeeper session timeout</p> <p>Default: 6000</p> <ul style="list-style-type: none">• Available as <code>KafkaConfig.ZkSessionTimeoutMs</code>• Use <code>KafkaConfig.zkSessionTimeoutMs</code> to get the current value
<code>zookeeper.set.acl</code>	<p>Enables secure ACLs</p> <p>Default: false</p> <ul style="list-style-type: none">• Available as <code>KafkaConfig.ZkEnableSecureAcls</code>• Use <code>KafkaConfig.zkEnableSecureAcls</code> to get the current value

bootstrap.servers Property

`bootstrap.servers` is a comma-separated list of host and port pairs that are the addresses of the Kafka brokers in a "bootstrap" [Kafka cluster](#) that a Kafka client connects to initially to bootstrap itself.

A host and port pair uses `:` as the separator.

```
localhost:9092  
localhost:9092,another.host:9092
```

`bootstrap.servers` provides the initial hosts that act as the starting point for a Kafka client to discover the full set of alive servers in the cluster.

Note

Since these servers are just used for the initial connection to discover the full cluster membership (which may change dynamically), this list does not have to contain the full set of servers (you may want more than one, though, in case a server is down).

Note

Clients (producers or consumers) make use of all servers irrespective of which servers are specified in `bootstrap.servers` for bootstrapping.

Tip

Use `org.apache.kafka.clients.CommonClientConfigs.BOOTSTRAP_SERVERS_CONFIG` public value to refer to the property.

client.id Property

An optional identifier of a [Kafka consumer](#) (in a consumer group) that is passed to a Kafka broker with every request.

The sole purpose of this is to be able to track the source of requests beyond just ip and port by allowing a logical application name to be included in Kafka logs and monitoring aggregates.

enable.auto.commit Property

```
enable.auto.commit ...FIXME
```

By default, as the consumer reads messages from Kafka, it will periodically commit its current offset (defined as the offset of the next message to be read) for the partitions it is reading from back to Kafka. Often you would like more control over exactly when offsets are committed. In this case you can set `enable.auto.commit` to `false` and call the `commit` method on the consumer.

group.id Property

`group.id` specifies the name of the consumer group a [Kafka consumer](#) belongs to.

When the Kafka consumer is constructed and `group.id` does not exist yet (i.e. there are no existing consumers that are part of the group), the consumer group will be created automatically.

Note

If all consumers in a group leave the group, the group is automatically destroyed.

retry.backoff.ms Property

`retry.backoff.ms` is the time to wait before attempting to retry a failed request to a given topic partition.

This avoids repeatedly sending requests in a tight loop under some failure scenarios.

Logging

Kafka Streams uses [Apache Log4j 2](#) for logging service.

The main logger is `org.apache.kafka.streams`. Use `log4j.properties` to set the logging levels.

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

`log4j.properties`

```
log4j.rootLogger=INFO
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.streams=DEBUG
log4j.additivity.org.apache.kafka.streams=false
```

Tip

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

Simple Logging Facade for Java (SLF4J)

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/logback-classic.jar!/org/slf4j/impl/Log4jLoggerFactory.class]  
SLF4J: Found binding in [jar:file:/Users/jacek/.ivy2/cache/ch.qos.logback/logback-core/1.2.3/logback-core.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?

Gradle Tips

Building Kafka Distribution

```
gradle -PscalaVersion=2.11.8 clean releaseTarGz install
```

It takes around 2 minutes (after all the dependencies were downloaded once).

After the command, you can unpack the release as follows:

```
tar -zxvf core/build/distributions/kafka_2.11-0.10.1.0-SNAPSHOT.tgz
```

Executing Single Test

```
gradle -PscalaVersion=2.11.8 :core:test --no-rebuild --tests "*PlaintextProducerSendTe  
st"
```

Zookeeper Tips

The zookeeper shell shipped with Kafka works with no support for command line history because jline jar is missing (see [KAFKA-2385](#)).

A solution is to use the official distribution of Apache Zookeeper (**3.4.10** as of this writing) from [Apache ZooKeeper Releases](#).

Once downloaded, use `./bin/zkCli.sh` to connect to Zookeeper that is used for Kafka.

```
$ ./bin/zkCli.sh -server localhost:2181
...
JLine support is enabled

[zk: localhost:2181(CONNECTED) 0] ls /
[cluster, controller_epoch, controller, brokers, zookeeper, admin, isr_change_notification, consumers, log_dir_event_notification, latest_producer_id_block, config]

[zk: localhost:2181(CONNECTED) 5] get /controller
{"version":1,"brokerid":200,"timestamp":"1506417983145"}
cZxid = 0x11b
ctime = Tue Sep 26 11:26:23 CEST 2017
mZxid = 0x11b
mtime = Tue Sep 26 11:26:23 CEST 2017
pZxid = 0x11b
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x15ebd811a0e0001
dataLength = 56
numChildren = 0
```

Kafka in Scala REPL for Interactive Exploration

Use the following `build.sbt` and execute `sbt` followed by `console` command (while inside the `sbt` shell).

Note

The reason for executing `console` command after `sbt` has started up is that command history did not work using the key-up and key-down keys. YMMV

build.sbt

```
name := "kafka-sandbox"

version := "1.0"

scalaVersion := "2.12.3"

//val kafkaVer = "0.11.0.1"
resolvers += Resolver.mavenLocal
val kafkaVer = "1.0.0-SNAPSHOT"
libraryDependencies += "org.apache.kafka" % "kafka-clients" % kafkaVer
libraryDependencies += "org.apache.kafka" %% "kafka" % kafkaVer

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

sbt with console command

```
→ kafka-sandbox sbt
[info] Loading settings from plugins.sbt ...
[info] Loading project definition from /Users/jacek/dev/sandbox/kafka-sandbox/project
[info] Loading settings from build.sbt ...
[info] Set current project to kafka-sandbox (in build file:/Users/jacek/dev/sandbox/ka
fka-sandbox/)
[info] sbt server started at 127.0.0.1:4408
sbt:kafka-sandbox> console
[info] Starting scala interpreter...
Welcome to Scala 2.12.3 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_144).
Type in expressions for evaluation. Or try :help.

scala> import kafka.utils._
import kafka.utils._

scala> ZkUtils.controllerZkData(1, System.currentTimeMillis())
res0: String = {"version":1,"brokerid":1,"timestamp":"1506162122097"}
```


Running Kafka Broker in Docker

A very basic installation of Apache Kafka is made up of the following components:

1. An instance of Apache Zookeeper
2. At least one Kafka broker

You can use [Docker Compose](#) to run such an installation where all the components are *dockerized* (i.e. run as Docker containers).

There are two projects with the Docker images for the components that seem to have been trusted the most:

1. [wurstmeister/kafka](#)
2. [spotify/kafka](#)

Note	ches/docker-kafka is another Docker image (that I have not tried myself yet).
------	---

`wurstmeister/kafka` gives separate images for Apache Zookeeper and Apache Kafka while `spotify/kafka` runs both Zookeeper and Kafka in the same container.

With the separate images for Apache Zookeeper and Apache Kafka in `wurstmeister/kafka` project and a `docker-compose.yml` configuration for Docker Compose that is a very good starting point as well as allows for further customizations.

Tip	Read the official tutorial on how to use <code>wurstmeister/kafka</code> project.
-----	---

Let's start a very basic one-broker Kafka cluster using Docker and `wurstmeister/kafka` project.

```
// Clone wurstmeister/kafka repo
$ git clone https://github.com/wurstmeister/kafka-docker

$ cd kafka-docker

// Edit `docker-compose.yml`
// 1. Change the docker host IP in `KAFKA_ADVERTISED_HOST_NAME`
// e.g. KAFKA_ADVERTISED_HOST_NAME: docker.for.mac.localhost on macOS
// See https://docs.docker.com/docker-for-mac/networking/
// 2. Expose port `9092` of `kafka` service to the host
// i.e. change it to `9092:9092`

// Start services from `docker-compose.yml`
$ docker-compose up

// Check the connection from the host to the single Kafka broker
$ nc -vz localhost 9092
found 0 associations
found 1 connections:
  1: flags=82<CONNECTED, PREFERRED>
    outif lo0
    src ::1 port 60643
    dst ::1 port 9092
    rank info not available
    TCP aux info available

Connection to localhost port 9092 [tcp/XmlIpcRegSvc] succeeded!
```

KafkaClient

`KafkaClient` is the abstraction of Kafka clients.

Note

`NetworkClient` is the one and only known implementation of the `KafkaClient Contract` in Apache Kafka.

Table 1. KafkaClient Contract

Method	Description
<code>authenticationException</code>	<pre>AuthenticationException authenticationException(Node node)</pre> <p>Used when...FIXME</p>
<code>close</code>	<pre>void close(String nodeId)</pre> <p>Closing the Kafka client (releasing resources)</p> <p>Used when:</p> <ul style="list-style-type: none"> • <code>RequestSendThread</code> is requested to <code>dowork</code> and <code>brokerReady</code> • <code>KafkaServer</code> is requested to <code>controlledShutdown</code> • <code>ReplicaFetcherBlockingSend</code> is requested to <code>send a client request and wait for a response</code>
<code>connectionDelay</code>	<pre>long connectionDelay(Node node, long now)</pre> <p>Used when...FIXME</p>
<code>connectionFailed</code>	<pre>boolean connectionFailed(Node node)</pre> <p>Used when...FIXME</p>
<code>disconnect</code>	<pre>void disconnect(String nodeId)</pre> <p>Used when:</p> <ul style="list-style-type: none"> • <code>AdminClientRunnable</code> is requested to <code>timeoutCallsInFlight</code> and <code>handleResponses</code> • <code>ConsumerNetworkClient</code> is requested to <code>handlePendingDisconnects</code>

	<pre><code>boolean hasInFlightRequests() boolean hasInFlightRequests(String nodeId);</code></pre> <p>Used when...FIXME</p>
<pre><code>hasReadyNodes</code></pre>	<pre><code>boolean hasReadyNodes(long now)</code></pre> <p>Used when...FIXME</p>
<pre><code>inFlightRequestCount</code></pre>	<pre><code>int inFlightRequestCount() int inFlightRequestCount(String nodeId)</code></pre> <p>Used when...FIXME</p>
<pre><code>isReady</code></pre>	<pre><code>boolean isReady(Node node, long now)</code></pre> <p>Used when:</p> <ul style="list-style-type: none"> • <code>NetworkClient</code> is requested to <code>ready</code> and <code>leastLoadedNode</code> • <code>NetworkClientUtils</code> is requested to <code>isReady</code> and <code>wait until a connection to a broker node is ready</code>
<pre><code>leastLoadedNode</code></pre>	<pre><code>Node leastLoadedNode(long now)</code></pre> <p>Used when:</p> <ul style="list-style-type: none"> • <code>DefaultMetadataUpdater</code> is requested to <code>maybeUpdate</code> • <code>ConsumerNetworkClient</code> is requested to <code>leastLoadedNode</code> • <code>Sender</code> is requested to <code>awaitLeastLoadedNodeReady</code> • <code>LeastLoadedNodeProvider</code> and <code>MetadataUpdateNodeIdProvider</code> are requested to provide the least-loaded node

	<pre><code>ClientRequest newClientRequest(String nodeId, AbstractRequest.Builder<?> requestBuilder, long createdTimeMs, boolean expectResponse) ClientRequest newClientRequest(String nodeId, AbstractRequest.Builder<?> requestBuilder, long createdTimeMs, boolean expectResponse, int requestTimeoutMs, RequestCompletionHandler callback)</code></pre>
	<p>Creates a new <code>ClientRequest</code> for a given <code>AbstractRequest.Builder</code> and <code>RequestCompletionHandler</code></p> <p>Used when...FIXME</p>
<p>poll</p>	<pre><code>List<ClientResponse> poll(long timeout, long now)</code></pre> <p>Used when:</p> <ul style="list-style-type: none"> • <code>InterBrokerSendThread</code> is requested to <code>doWork</code> • <code>NetworkClientUtils</code> is requested to <code>isReady</code>, <code>awaitReady</code> and <code>sendAndReceive</code> • <code>AdminClientRunnable</code> is requested to run • <code>ConsumerNetworkClient</code> is requested to <code>poll</code> • <code>Sender</code> is requested to <code>start</code> (as a thread of execution)
<p>pollDelayMs</p>	<pre><code>long pollDelayMs(Node node, long now)</code></pre> <p>Used when...FIXME</p>
<p>ready</p>	<pre><code>boolean ready(Node node, long now)</code></pre> <p>Initiates a connection to the given broker <code>Node</code> (i.e. a connection to a broker node is ready)</p> <p>Used when:</p> <ul style="list-style-type: none"> • <code>NetworkClientUtils</code> helper object is requested to <code>wait until a connection to a broker node is ready</code> • <code>AdminClientRunnable</code> is requested to <code>sendEligibleCalls</code> • <code>ConsumerNetworkClient</code> is requested to <code>trySend</code> and <code>tryConnect</code> • <code>Sender</code> is requested to <code>sendProducerData</code> (when requested to <code>run a single iteration of sending</code>)

	<ul style="list-style-type: none">• <code>InterBrokerSendThread</code> is requested to <code>sendRequests</code>
	<pre>void send(ClientRequest request, long now)</pre> <p>Queues up the <code>ClientRequest</code> for sending</p> <p>Used when:</p> <ul style="list-style-type: none">• <code>InterBrokerSendThread</code> is requested to <code>sendRequests</code>• <code>NetworkClientUtils</code> is requested to <code>sendAndReceive</code>• <code>AdminClientRunnable</code> is requested to <code>sendEligibleCalls</code>• <code>ConsumerNetworkClient</code> is requested to <code>trySend</code>• <code>Sender</code> is requested to <code>maybeSendTransactionalRequests</code> and <code>sendProduceRequest</code>
wakeup	<pre>void wakeup()</pre> <p>Used when...FIXME</p>

NetworkClient — Non-Blocking Network KafkaClient

`NetworkClient` is a non-blocking `KafkaClient` that uses a `Selectable` for network communication (i.e. sending and receiving messages).

Note

`Selector` is the one and only `Selectable` that uses Java's selectable channels for stream-oriented connecting sockets (i.e. Java's `java.nio.channels.SocketChannel`).

`NetworkClient` does the actual reads and writes (to sockets) every `poll`.

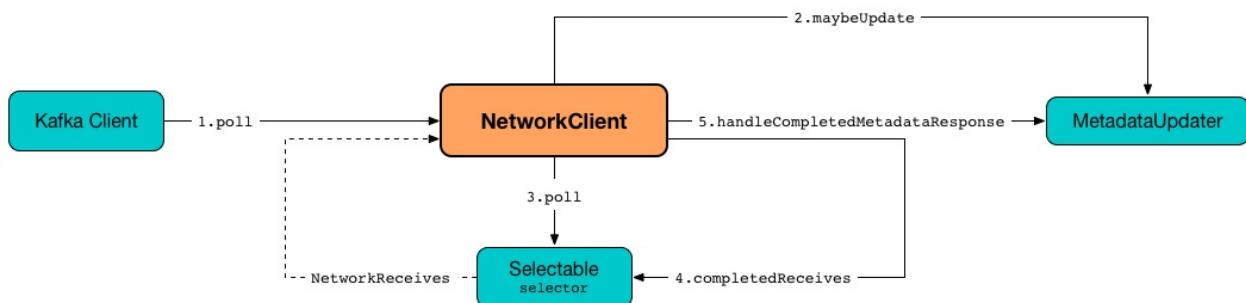


Figure 1. NetworkClient

`NetworkClient` uses the `MetadataUpdater` for the following:

- `disconnect`
- `close`
- `isReady`
- `poll`
- `close`
- `leastLoadedNode`
- `processDisconnection`
- `handleTimedOutRequests`
- `handleCompletedReceives`
- `handleDisconnections`
- `initiateConnect`

`NetworkClient` is created when:

- `KafkaConsumer` is created (with `ConsumerNetworkClient`)
- `KafkaProducer` is created (with `Sender`)
- `KafkaAdminClient` is created (using `createInternal`)
- `AdminClient` is created (with `ConsumerNetworkClient`)
- `ControllerChannelManager` is requested to `addNewBroker` (and creates a `RequestSendThread` daemon thread and a `ControllerBrokerStateInfo`)
- `TransactionMarkerChannelManager` is created
- `KafkaServer` does `doControlledShutdown`
- `ReplicaFetcherBlockingSend` is created

Table 1. NetworkClient's Internal Properties (e.g. Registries and Counters)

Name	Description
<code>connectionStates</code>	<code>ClusterConnectionStates</code> Used when...FIXME

Tip

Enable `DEBUG` logging level for `org.apache.kafka.clients.NetworkClient` logger to see what happens inside.

Add the following line to `config/tools-log4j.properties` (for Kafka tools):

```
log4j.logger.org.apache.kafka.clients.NetworkClient=DEBUG
```

Add the following line to `config/log4j.properties`:

```
log4j.logger.org.apache.kafka.clients.NetworkClient=DEBUG
```

Refer to [Logging](#).

Establishing Connection to Broker Node

— `initiateConnect` Internal Method

```
void initiateConnect(Node node, long now)
```

`initiateConnect` prints out the following DEBUG message to the logs:

```
Initiating connection to node [node]
```

`initiateConnect` requests the [ClusterConnectionStates](#) to enter the connecting state for the connection to the broker `node`.

`initiateConnect` requests the [Selectable](#) to connect to the broker `node` (at a given host and port).

Note	<code>initiateConnect</code> passes the sizes of send and receive buffers for the socket connection.
------	--

In case of an IO failure, `initiateConnect` requests the [ClusterConnectionStates](#) to enter the disconnected state for the connection to the broker `node`.

`initiateConnect` requests the [MetadataUpdater](#) for update.

You should see the following DEBUG message in the logs:

Error connecting to node [node]	
---------------------------------	--

Note	<code>initiateConnect</code> is used when: <ul style="list-style-type: none"><code>NetworkClient</code> is requested to attempt to connect to a broker node<code>DefaultMetadataUpdater</code> is requested to maybeUpdate
------	---

ready Method

boolean ready(Node node, long now)	
------------------------------------	--

Note	<code>ready</code> is a part of KafkaClient Contract .
------	--

`ready` ...FIXME

wakeup Method

void wakeup()	
---------------	--

Note	<code>wakeup</code> is a part of KafkaClient Contract .
------	---

`wakeup` simply requests the internal [Selectable](#) to `wakeup`

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

Reading and Writing to Socket— `poll` Method

```
List<ClientResponse> poll(long timeout, long now)
```

Note	<code>poll</code> is a part of KafkaClient Contract .
------	---

`poll` requests the [MetadataUpdater](#) for [cluster metadata update](#) (if needed and possible).

`poll` then requests [Selectable](#) to `poll`.

In the end, `poll` handles completed request sends, receives, disconnected connections, records any connections to new brokers, initiates API version requests, expire in-flight requests, and finally triggers their [RequestCompletionHandlers](#).

In case [abortedSends](#) is not empty, `poll` creates a collection of [ClientResponse](#) with [abortedSends](#), triggers their [RequestCompletionHandlers](#) and returns them.

handleCompletedReceives Internal Method

```
void handleCompletedReceives(List<ClientResponse> responses, long now)
```

`handleCompletedReceives` ...FIXME

Note	<code>handleCompletedReceives</code> is used exclusively when NetworkClient is requested to poll .
------	--

Creating NetworkClient Instance

`NetworkClient` takes the following when created:

- [MetadataUpdater](#) (i.e. [DefaultMetadataUpdater](#))
- [Metadata](#)
- [Selectable](#)
- Client ID that is used to identify the client in requests to a Kafka server (when `NetworkClient` is requested to [create a new ClientRequest](#))
- `maxInFlightRequestsPerConnection`
- `reconnectBackoffMs`
- `reconnectBackoffMax`

- Size of the TCP send buffer (SO_SNDBUF) for socket connection (in bytes)

Use `send.buffer.bytes` property to configure it.

Used when `NetworkClient` establishes connection to a broker node.

- Size of the TCP receive buffer (SO_RCVBUF) for socket connection (in bytes)

Use `receive.buffer.bytes` property to configure it.

Used when `NetworkClient` establishes connection to a broker node

- `defaultRequestTimeoutMs`
- `Time`
- `discoverBrokerVersions` flag
- `ApiVersions`
- `throttleTimeSensor` `Sensor`
- `LogContext`

`NetworkClient` initializes the internal registries and counters.

Informing ClientResponse about Response Being Completed — `completeResponses` Internal Method

```
void completeResponses(List<ClientResponse> responses)
```

`completeResponses` informs every `ClientResponse` (in the input `responses`) that a response has been completed.

In case of any exception, `completeResponses` prints out the following ERROR message to the logs:

```
Uncaught error in request completion: [exception]
```

Note	<code>completeResponses</code> is used when <code>NetworkClient</code> is requested to <code>poll</code> (for both <code>abortedSends</code> and completed actions).
------	--

Creating ClientRequest — `newClientRequest` Method

```
ClientRequest newClientRequest(  
    String nodeId,  
    AbstractRequest.Builder<?> requestBuilder,  
    long createdTimeMs,  
    boolean expectResponse,  
    int requestTimeoutMs,  
    RequestCompletionHandler callback)
```

Note `newClientRequest` is part of the [KafkaClient Contract](#) to...FIXME.

`newClientRequest` simply creates a new `ClientRequest` (with the input parameters and the `correlation` incremented, the `clientId` and the `defaultRequestTimeoutMs`).

sendInternalMetadataRequest Internal Method

```
void sendInternalMetadataRequest(  
    MetadataRequest.Builder builder,  
    String nodeConnectionId,  
    long now)
```

`sendInternalMetadataRequest` ...FIXME

Note `sendInternalMetadataRequest` is used exclusively when `DefaultMetadataUpdater` is requested to [maybeUpdate](#).

doSend Internal Method

```
void doSend(  
    ClientRequest clientRequest,  
    boolean isInternalRequest,  
    long now)  
void doSend(  
    ClientRequest clientRequest,  
    boolean isInternalRequest,  
    long now,  
    AbstractRequest request)
```

`doSend` ...FIXME

Note `doSend` is used when `NetworkClient` is requested to [send](#), [sendInternalMetadataRequest](#) and [handleInitiateApiVersionRequests](#).

send Method

```
void send(ClientRequest request, long now)
```

Note	send is part of the KafkaClient Contract to...FIXME.
------	--

send ...FIXME

handleDisconnections Internal Method

```
void handleDisconnections(List<ClientResponse> responses, long now)
```

handleDisconnections ...FIXME

Note	handleDisconnections is used exclusively when NetworkClient is requested to poll.
------	---

handleTimedOutRequests Internal Method

```
void handleTimedOutRequests(List<ClientResponse> responses, long now)
```

handleTimedOutRequests ...FIXME

Note	handleTimedOutRequests is used exclusively when NetworkClient is requested to poll.
------	---

processDisconnection Internal Method

```
void processDisconnection(
    List<ClientResponse> responses,
    String nodeId,
    long now,
    ChannelState disconnectState)
```

processDisconnection ...FIXME

Note	processDisconnection is used when NetworkClient is requested to handleTimedOutRequests, handleApiVersionsResponse, and handleDisconnections.
------	--

handleApiVersionsResponse Internal Method

```
void handleApiVersionsResponse(  
    List<ClientResponse> responses,  
    InFlightRequest req,  
    long now,  
    ApiVersionsResponse apiVersionsResponse)
```

handleApiVersionsResponse ...FIXME

Note

`handleApiVersionsResponse` is used exclusively when `NetworkClient` is requested to `handleCompletedReceives` (when requested to `poll`).

leastLoadedNode Method

```
Node leastLoadedNode(long now)
```

Note

`leastLoadedNode` is part of the [KafkaClient Contract](#) to...FIXME.

leastLoadedNode ...FIXME

close Method

```
void close()
```

Note

`close` is part of Java's [java.io.Closeable](#) to close this stream and releases any system resources associated with it.

close ...FIXME

close Method

```
void close(String nodeId)
```

Note

`close` is part of the [KafkaClient Contract](#) to...FIXME.

close ...FIXME

isReady Method

```
boolean isReady(Node node, long now)
```

Note	<code>isReady</code> is part of the KafkaClient Contract to...FIXME.
------	--

`isReady` ...FIXME

disconnect Method

```
void disconnect(String nodeId)
```

Note	<code>disconnect</code> is part of the KafkaClient Contract to...FIXME.
------	---

`disconnect` ...FIXME

RequestCompletionHandler Contract

`RequestCompletionHandler` is the [contract](#) of request completion handlers that are [executed when a request is complete](#), i.e. the corresponding response has been received or there was a disconnection while handling the request.

Table 1. RequestCompletionHandler Contract

Method	Description
<code>onComplete</code>	<pre>void onComplete(ClientResponse response)</pre> <p>Action to be executed when a request is complete</p> <p>Used when:</p> <ul style="list-style-type: none"> <code>ClientResponse</code> is requested to onComplete (when <code>NetworkClient</code> is requested to poll) <code>ConsumerNetworkClient</code> is requested to checkDisconnects (when <code>ConsumerNetworkClient</code> is requested to poll) <code>InterBrokerSendThread</code> is requested to completeWithDisconnect (when <code>InterBrokerSendThread</code> is requested to doWork)

Table 2. RequestCompletionHandlers (Direct Implementations)

RequestCompletionHandler	Description
<code>RequestFutureCompletionHandler</code>	
<code>TransactionMarkerRequestCompletionHandler</code>	
<code>TxnRequestHandler</code>	

MetadataUpdater Contract

`MetadataUpdater` is the abstraction of [FIXME](#) that [FIXME](#).

Table 1. MetadataUpdater Contract

Method	Description
<code>close</code>	<pre>void close()</pre> <p>Used when...FIXME</p>
<code>fetchNodes</code>	<pre>List<Node> fetchNodes()</pre> <p>Used when...FIXME</p>
<code>handleAuthenticationFailure</code>	<pre>void handleAuthenticationFailure(AuthenticationException e)</pre> <p>Used when...FIXME</p>
<code>handleCompletedMetadataResponse</code>	<pre>void handleCompletedMetadataResponse(RequestHeader requestHeader, long now, MetadataResponse metadataResponse)</pre> <p>Used when...FIXME</p>
<code>handleDisconnection</code>	<pre>void handleDisconnection(String destination)</pre> <p>Used when...FIXME</p>
<code>isUpdateDue</code>	<pre>boolean isUpdateDue(long now)</pre> <p>Used when...FIXME</p>
<code>maybeUpdate</code>	<pre>long maybeUpdate(long now)</pre> <p>Starts a cluster metadata update if needed and possible. Used exclusively when <code>NetworkClient</code> is requested to use <code>Sockets</code>.</p>

	<pre>void requestUpdate()</pre>
requestUpdate	<p>Schedules an update of the current cluster metadata</p> <p>Used when <code>NetworkClient</code> is requested to handleTim handleDisconnections and establish a connection to a</p>

DefaultMetadataUpdater

`DefaultMetadataUpdater` is a [MetadataUpdater](#) that `NetworkClient` uses to...FIXME

`DefaultMetadataUpdater` takes a single [Metadata](#) when created.

`DefaultMetadataUpdater` is [created](#) exclusively for a [NetworkClient](#).

Table 1. DefaultMetadataUpdater's Internal Properties (e.g. Registries and Counters)

Name	Description
<code>metadataFetchInProgress</code>	<p>Flag to control whether a cluster metadata update is in progress, i.e. FIXME</p> <ul style="list-style-type: none"> • Disabled when <code>DefaultMetadataUpdater</code> is created • Turned on exclusively when <code>DefaultMetadataUpdater</code> does maybeUpdate (with a timestamp and a broker node) • Turned off when <code>DefaultMetadataUpdater</code> handles completed metadata response, disconnection or authentication failure
Tip	<p>Enable <code>WARN</code>, <code>DEBUG</code> or <code>TRACE</code> logging levels for <code>org.apache.kafka.clients.NetworkClient</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.clients.NetworkClient=DEBUG</pre> <p>Refer to Logging.</p>

isUpdateDue Method

Caution

FIXME

maybeUpdate Internal Method (with timestamp only)

```
maybeUpdate(long now)
```

Note

`maybeUpdate` is a part of [MetadataUpdater Contract](#).

`maybeUpdate` requests [Metadata](#) for `timeToNextUpdate` (with the input `now`).

`maybeUpdate` takes `requestTimeoutMs` for the time to wait till metadata fetch in progress finishes if `metadataFetchInProgress` flag is turned on or `0` otherwise.

`maybeUpdate` takes the maximum of the two values above to check if the current cluster metadata has expired.

If not, `maybeUpdate` gives the maximum value (that says how long to wait till the current cluster metadata expires).

Otherwise, `maybeUpdate` selects the node to request a cluster metadata from and `maybeUpdate` (with the input `now` timestamp and the node).

If no node was found, `maybeUpdate` prints out the following DEBUG message to the logs and gives `reconnectBackoffMs`.

```
Give up sending metadata request since no node is available
```

maybeUpdate Internal Method (with timestamp and node)

```
long maybeUpdate(long now, Node node)
```

```
maybeUpdate ...FIXME
```

Note	<code>maybeUpdate</code> is used exclusively when <code>DefaultMetadataUpdater</code> is requested to <code>maybeUpdate</code> (with the timestamp only).
------	---

handleAuthenticationFailure Callback Method

```
void handleAuthenticationFailure(AuthenticationException exception)
```

Note	<code>handleAuthenticationFailure</code> is a part of MetadataUpdater Contract .
------	--

`handleCompletedMetadataResponse` turns `metadataFetchInProgress` flag off.

`handleCompletedMetadataResponse` asks `Metadata` whether metadata update was requested and if so requests it to record a failure (passing on the `exception`).

handleCompletedMetadataResponse Callback Method

```
void handleCompletedMetadataResponse(RequestHeader requestHeader, long now, MetadataResponse response)
```

Note	<code>handleCompletedMetadataResponse</code> is a part of MetadataUpdater Contract .
------	--

`handleCompletedMetadataResponse` turns [metadataFetchInProgress](#) flag off.

`handleCompletedMetadataResponse` takes the [cluster](#) from the `response`.

`handleCompletedMetadataResponse` requests [Metadata](#) to [update](#) (with the cluster and unavailable topics) when there is at least one node in the cluster.

When there are no nodes in the cluster, `handleCompletedMetadataResponse` prints out the following TRACE message to the logs and requests [Metadata](#) to [record a failure](#) (with no exception).

Ignoring empty metadata response with correlation id [correlationId].

In case `response` has errors, `handleCompletedMetadataResponse` prints out the following WARN message to the logs:

Error while fetching metadata with correlation id [correlationId] : [errors]"

maybeUpdate Method

<code>long maybeUpdate(long now)</code>

Note	<code>maybeUpdate</code> is part of the MetadataUpdater Contract to...FIXME.
------	--

`maybeUpdate` ...FIXME

maybeUpdate Internal Method

<code>long maybeUpdate(long now, Node node)</code>
--

`maybeUpdate` ...FIXME

Note	<code>maybeUpdate</code> is used exclusively when <code>DefaultMetadataUpdater</code> is requested to maybeUpdate .
------	---

Scheduling Update Of Cluster Metadata — requestUpdate Method

```
void requestUpdate()
```

Note

`requestUpdate` is part of the [MetadataUpdater Contract](#) to schedule an update of the current cluster metadata.

`requestUpdate` simply requests the [Metadata](#) for a [metadata update](#).

Metadata

`Metadata` describes a Kafka cluster (and is [created](#)) for [KafkaConsumer](#) and [KafkaProducer](#).

Table 1. Metadata's Properties When Created by Clients

Client	<code>refreshBackoffMs</code>	<code>metadataExpireMs</code>	<code>allowAutoTopicCreate</code>
<code>KafkaConsumer</code>	<code>retry.backoff.ms</code>	<code>metadata.max.age.ms</code>	on
<code>KafkaProducer</code>	<code>retry.backoff.ms</code>	<code>metadata.max.age.ms</code>	on

A (*seemingly*) common usage pattern is as follows:

1. Request `Metadata` for a [update](#) (that simply turns the `needUpdate` flag on)
2. Request (indirectly) `KafkaClient` to [wake up](#) if blocked on I/O
3. Request `Metadata` to [wait for metadata change](#) (i.e. until the `metadata version` has changed)

`Metadata` manages [metadata update listeners](#) that want to be [notified](#) about [metadata updates](#). Listeners can be [registered](#) and [deregistered](#).

Note	<code>ConsumerCoordinator</code> is the only entity to register a metadata update listener.
------	---

Table 2. Metadata's Internal Properties (e.g. Registries and Counters)

Name	Description
<code>cluster</code>	<p><code>Cluster</code> with a subset of the nodes and topic partitions in a Kafka cluster.</p> <ul style="list-style-type: none"> • Empty (with no nodes and no topic partitions) when <code>Metadata</code> is created • Updated when: <ul style="list-style-type: none"> ◦ <code>DefaultMetadataUpdater</code> handles <code>MetadataResponse</code> ◦ <code>KafkaConsumer</code>, <code>KafkaProducer</code>, <code>KafkaAdminClient</code> and <code>AdminClient</code> are created and update the cluster with a "bootstrap" cluster with bootstrap brokers • Can be accessed using <code>fetch</code>
	The time (in millis) of the last successful update (and failed update)

<code>lastRefreshMs</code>	<ul style="list-style-type: none"> Used in <code>timeToNextUpdate</code> Starts <code>0</code> when <code>Metadata</code> is <code>created</code> Reset (to <code>0</code>) in <code>requestUpdateForNewTopics</code>
<code>lastSuccessfulRefreshMs</code>	
<code>needMetadataForAllTopics</code>	<p>Flag...FIXME</p> <ul style="list-style-type: none"> Disabled (i.e. <code>false</code>) when <code>Metadata</code> is <code>created</code> Updated when <code>Metadata</code> is requested to set state to indicate that metadata for all topics in Kafka cluster is required
<code>needUpdate</code>	<p>Flag that controls whether a metadata update has been requested (enabled) or not (disabled).</p> <ul style="list-style-type: none"> Starts turned off when <code>Metadata</code> is <code>created</code> Turned on exclusively when <code>Metadata</code> is requested for an <code>update</code> Turned off when <code>Metadata</code> is <code>updated</code> <p>Use <code>updateRequested</code> to know the current value.</p>
<code>version</code>	<p>Metadata version</p> <ul style="list-style-type: none"> <code>0</code> when <code>Metadata</code> is <code>created</code> Incremented every <code>update</code>
Tip	<p>Enable <code>INFO</code>, <code>DEBUG</code> or <code>TRACE</code> logging levels for <code>org.apache.kafka.clients.Metadata</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.clients.Metadata=TRACE</pre> <p>Refer to Logging.</p>

Checking if Metadata Update was Requested

— `updateRequested` Method

```
synchronized boolean updateRequested()
```

`updateRequested` ...FIXME

	<p><code>updateRequested</code> is used when:</p> <ul style="list-style-type: none"> • <code>DefaultMetadataUpdater</code> handles an authentication failure • <code>ConsumerCoordinator</code> polls for coordinator events • <code>ConsumerNetworkClient</code> makes sure that the metadata is fresh
--	--

Recording Update Request Failure — `failedUpdate` Method

```
synchronized void failedUpdate(long now, AuthenticationException authenticationException)
```

`failedUpdate` ...FIXME

	<p><code>failedUpdate</code> is used when...FIXME</p>
--	---

`getClusterForCurrentTopics` Internal Method

```
Cluster getClusterForCurrentTopics(Cluster cluster)
```

`getClusterForCurrentTopics` ...FIXME

	<p><code>getClusterForCurrentTopics</code> is used exclusively when <code>Metadata</code> is requested to update.</p>
--	---

`timeToNextUpdate` Method

```
synchronized long timeToNextUpdate(long nowMs)
```

`timeToNextUpdate` ...FIXME

	<p><code>timeToNextUpdate</code> is used when:</p> <ul style="list-style-type: none"> • <code>ConsumerNetworkClient</code> <code>ensureFreshMetadata</code> • <code>DefaultMetadataUpdater</code> (of <code>NetworkClient</code>) <code>isUpdateDue</code> and <code>maybeUpdate</code>
--	---

`add` Method

```
synchronized void add(String topic)
```

add ...FIXME

Note

add is used when...FIXME

Requesting Metadata Update — `requestUpdate` Method

```
int requestUpdate()
```

`requestUpdate` simply turns the `needUpdate` flag on and returns the `current version`.

Note

`requestUpdate` is used when:

- `Metadata` is requested to `requestUpdateForNewTopics`
- `DefaultMetadataUpdater` is requested to `requestUpdate`
- `KafkaConsumer` is requested to `subscribe`
- `ConsumerCoordinator` is `created`, `addMetadataListener`, `poll`
- `ConsumerNetworkClient` is requested to `awaitMetadataUpdate`
- `Fetcher` is requested to `resetOffsetsAsync`, `groupListOffsetRequests`, `prepareFetchRequests`, `parseCompletedFetch`
- `KafkaProducer` is requested to `wait` for cluster metadata
- `Sender` is requested to `sendProducerData`, `maybeSendTransactionalRequest`, `maybeWaitForProducerId`, and `completeBatch`

Waiting for Metadata Update (i.e. Metadata Version Change) — `awaitUpdate` Method

```
synchronized void awaitUpdate(
    final int lastVersion,
    final long maxWaitMs) throws InterruptedException
```

`awaitUpdate` ...FIXME

Note

`awaitUpdate` is used when...FIXME

Getting Current Cluster Metadata — `fetch` Method

```
synchronized Cluster fetch()
```

`fetch` returns current [cluster](#) information.

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

Setting Topics to Maintain — `setTopics` Method

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

Updating Cluster Metadata — `update` Method

```
void update(
    Cluster newCluster,
    Set<String> unavailableTopics,
    long now)
```

`update` turns the [needUpdate](#) flag off and increments [version](#).

`update` sets [lastRefreshMs](#) and [lastSuccessfulRefreshMs](#) internal registries to the given `now`.

(only when [topicExpiryEnabled](#) is enabled, e.g. [KafkaProducer](#)) `update` ...FIXME

`update` notifies the [Metadata.Listeners](#) that the [metadata has been updated](#).

`update` does [getClusterForCurrentTopics](#) for the `cluster` when [needMetadataForAllTopics](#) flag is on and turns [needUpdate](#) flag off (that may have been turned on...FIXME).

`update` sets the `cluster` to the given `cluster` when the [needMetadataForAllTopics](#) flag is off.

`update` prints out the following INFO message to the logs with the cluster ID and notifies [clusterResourceListeners](#) that [cluster has changed](#) (only for a non-bootstrap cluster).

```
Cluster ID: [clusterId]
```

In the end, `update` prints out the following DEBUG message to the logs:

```
Updated cluster metadata version [version] to [cluster]
```

	<p><code>update</code> is used when:</p> <ul style="list-style-type: none"> • <code>DefaultMetadataUpdater</code> is requested to handle a MetadataResponse • KafkaConsumer and KafkaProducer are created (and request updating the cluster metadata with a "bootstrap" cluster, i.e. with the bootstrap servers only)
--	--

Creating Metadata Instance

`Metadata` takes the following when created:

- `refreshBackoffMs`
- `metadataExpireMs`
- `allowAutoTopicCreation` flag
- `topicExpiryEnabled` flag
- [ClusterResourceListeners](#)

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

Conditionally Requesting Update For New Topics (for KafkaConsumer) — `needMetadataForAllTopics` Method

```
synchronized void needMetadataForAllTopics(boolean needMetadataForAllTopics)
```

`needMetadataForAllTopics` [requestUpdateForNewTopics](#) when the input `needMetadataForAllTopics` flag is enabled (i.e. `true`) and the current `needMetadataForAllTopics` is disabled (i.e. `false`).

`needMetadataForAllTopics` sets `needMetadataForAllTopics` to be the input `needMetadataForAllTopics`.

	<p><code>needMetadataForAllTopics</code> is used when <code>KafkaConsumer</code>:</p> <ul style="list-style-type: none"> • Subscribes to topics matching specified pattern (and <code>needMetadataForAllTopics</code> flag is then enabled) • Unsubscribes from topics (and <code>needMetadataForAllTopics</code> flag is then disabled)
--	--

requestUpdateForNewTopics Internal Method

```
synchronized void requestUpdateForNewTopics()
```

requestUpdateForNewTopics sets lastRefreshMs to 0 and requests update.

Note

requestUpdateForNewTopics is used when Metadata :

- add
- needMetadataForAllTopics
- setTopics

Registering Metadata.Listener— addListener Method

```
void addListener(Listener listener)
```

addListener simply adds the given Listener to the listeners internal registry.

Note

addListener is used exclusively when ConsumerCoordinator is requested to addMetadataListener (when created).

Listener Contract — Intercepting Metadata Updates

Listener is the [contract](#) of...FIXME

```
package org.apache.kafka.clients;

public final class Metadata {
    public interface Listener {
        void onMetadataUpdate(Cluster cluster, Set<String> unavailableTopics);
    }
}
```

ClientRequest

`ClientRequest` is...FIXME

`ClientRequest` is [created](#) exclusively when `NetworkClient` is requested to [create one](#).

`ClientRequest` uses the [client ID](#) when requested to [makeHeader](#).

Creating ClientRequest Instance

`ClientRequest` takes the following when created:

- Destination
- [AbstractRequest.Builder](#)
- Correlation ID
- Client ID
- `createdTimeMs`
- `expectResponse` flag
- `requestTimeoutMs`
- [RequestCompletionHandler](#)

`ClientRequest` initializes the internal registries and counters.

Creating RequestHeader — `makeHeader` Method

[RequestHeader](#) `makeHeader(short version)`

`makeHeader` creates a `RequestHeader` (with the [client](#) and the [correlation IDs](#))

Note	<p><code>makeHeader</code> is used when:</p> <ul style="list-style-type: none"> • <code>InterBrokerSendThread</code> is requested to completeWithDisconnect • <code>NetworkClient</code> is requested to doSend • <code>ConsumerNetworkClient</code> is requested to checkDisconnects
-------------	--

ClientResponse

`ClientResponse` is...FIXME

onComplete Method

```
void onComplete()
```

`onComplete` triggers [RequestCompletionHandler](#) if defined.

Note	<code>onComplete</code> is used exclusively when <code>NetworkClient</code> is requested to completeResponses (when <code>NetworkClient</code> is requested to poll).
------	--

`onComplete` is used exclusively when `NetworkClient` is requested to [completeResponses](#) (when `NetworkClient` is requested to [poll](#)).

StaleMetadataException

```
StaleMetadataException is...FIXME
```

NetworkClientUtils

NetworkClientUtils is...FIXME

sendAndReceive Static Method

```
ClientResponse sendAndReceive(  
    KafkaClient client  
    ClientRequest request  
    Time time) throws IOException
```

sendAndReceive ...FIXME

Note	sendAndReceive is used when...FIXME
------	-------------------------------------

Waiting Until Connection to Broker Node is Ready — awaitReady Static Method

```
boolean awaitReady(  
    KafkaClient client  
    Node node  
    Time time  
    long timeoutMs) throws IOException
```

awaitReady ...FIXME

Note	awaitReady is used when...FIXME
------	---------------------------------

isReady Static Method

```
boolean isReady(  
    KafkaClient client,  
    Node node,  
    long currentTime)
```

isReady ...FIXME

Note	isReady is used when...FIXME
------	------------------------------

Cluster

`Cluster` represents a subset of the nodes and topic partitions in a Kafka cluster.

Note	<code>org.apache.kafka.common.Cluster</code> is a <code>public final class</code> .
------	---

A special variant of a cluster is **bootstrap cluster** that is made up of the [bootstrap brokers](#) that are mandatory (and specified explicitly) when Kafka clients are created, i.e. [KafkaAdminClient](#), [AdminClient](#), [KafkaConsumer](#) and [KafkaProducer](#).

Note	A bootstrap cluster does not hold all information about the cluster.
------	--

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

- `AdminMetadataManager` is requested to [clearController](#)
- `Cluster` is requested to create an [empty Cluster metadata](#) and [Cluster metadata with given partitions](#), and [bootstrap](#)
- `MetadataCache` is requested to [getClusterMetadata](#)
- `Metadata` is requested to [getClusterForCurrentTopics](#)
- `MetadataResponse` is requested to `cluster`

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

Name	Description
<code>isBootstrapConfigured</code>	Flag...FIXME Used when...FIXME
<code>partitionsByTopic</code>	

bootstrap Factory Method

```
static Cluster bootstrap(List<InetSocketAddress> addresses)
```

`bootstrap` ...FIXME

Note	<code>bootstrap</code> is used when KafkaAdminClient , KafkaConsumer and KafkaProducer are created (and update their internal metadata).
------	--

isBootstrapConfigured Method

```
boolean isBootstrapConfigured()
```

`isBootstrapConfigured` gives `isBootstrapConfigured` internal flag.

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

Getting Partitions for Topic — partitionsForTopic Method

```
List<PartitionInfo> partitionsForTopic(String topic)
```

`partitionsForTopic` returns a collection of zero or more partition of the input `topic` from `partitionsByTopic` internal lookup table.

Note	<p><code>partitionsForTopic</code> is used when:</p> <ul style="list-style-type: none"> • <code>Metadata getClusterForCurrentTopics</code> • <code>KafkaAdminClient describeTopics</code> • <code>KafkaConsumer partitionsFor</code> • <code>KafkaProducer requests partitions for a topic</code> • <code>DefaultPartitioner assigns the partition for a record</code>
------	---

availablePartitionsForTopic Method

```
List<PartitionInfo> availablePartitionsForTopic(String topic)
```

`availablePartitionsForTopic` ...FIXME

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

Creating Cluster Instance

`cluster` takes the following when created:

- **Cluster ID**
- `isBootstrapConfigured` flag

- Kafka [Nodes](#) (`Collection<Node>`)
- Kafka [PartitionInfos](#) (`Collection<PartitionInfo>`)
- Names of the **unauthorized topics**
- Names of the **invalid topics**
- Names of the **internal topics**
- **Controller** (as a Kafka [Node](#))

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

Creating Empty Cluster Metadata — `empty` Factory Method

```
static Cluster empty()
```

`empty` ...FIXME

	<code>empty</code> is used when:
Note	<ul style="list-style-type: none"> • <code>Metadata</code> is created • <code>AdminMetadataManager</code> is created

Creating Cluster Metadata With Given Partitions — `withPartitions` Method

```
Cluster withPartitions(Map<TopicPartition, PartitionInfo> partitions)
```

`withPartitions` ...FIXME

`withPartitions` is used exclusively when Kafka Streams' `streamsPartitionAssignor` is requested to `assign` and `onAssignment`.

Cluster (deprecated)

Important	<p>It seems that <code>cluster</code> class is created using <code>ZkUtils.getCluster</code> that is used exclusively when <code>ZKRebalancerListener</code> does <code>syncedRebalance</code> (that in turn happens for the currently-deprecated <code>zookeeperConsumerConnector</code>).</p> <p>In other words, <code>cluster</code> class and the page are soon to be removed.</p>
-----------	---

`kafka.cluster.Cluster` private class represents a set of active brokers in a Kafka cluster.

Note	There is also <code>org.apache.kafka.common.Cluster</code> .
------	--

topics Method

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

availablePartitionsForTopic Method

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

ClusterConnectionStates

ClusterConnectionStates is...FIXME

connecting Method

```
void connecting(String id, long now)
```

connecting ...FIXME

Note	connecting is used when...FIXME
------	---------------------------------

disconnected Method

```
void disconnected(String id, long now)
```

disconnected ...FIXME

Note	disconnected is used when...FIXME
------	-----------------------------------

ClusterResourceListener (and ClusterResourceListeners Collection)

`ClusterResourceListener` is the [contract](#) for objects that want to be notified about changes in the cluster metadata.

```
package org.apache.kafka.common;

public interface ClusterResourceListener {
    void onUpdate(ClusterResource clusterResource);
}
```

You can register a `clusterResourceListener` for the following Kafka services:

- `KafkaServer` and get notified when the server [starts up](#)
- `KafkaProducer` and get notified when it is [created](#) and...FIXME
- `KafkaConsumer` and get notified when it is [created](#) and...FIXME

ClusterResourceListeners Collection

`ClusterResourceListeners` collection holds zero or more `ClusterResourceListener` objects and uses them as if there were one.

`ClusterResourceListeners` is used when:

- `Metadata` notifies [ClusterResourceListeners](#) about every cluster metadata change
- `KafkaServer` [starts up](#)

NotificationHandler Contract

`NotificationHandler` is the [contract](#) of [change notification handlers](#) that can [process change notifications](#).

```
package kafka.common

trait NotificationHandler {
  def processNotification(notificationMessage: Array[Byte])
}
```

Table 1. NotificationHandler Contract

Property	Description
<code>processNotification</code>	Handles a change notification Used exclusively when <code>ZkNodeChangeNotificationListener</code> is requested to processNotification

Table 2. NotificationHandlers

NotificationHandler	Description
rawHandler	
ConfigChangedNotificationHandler	
TokenChangedNotificationHandler	

ZkNodeChangeNotificationListener

ZkNodeChangeNotificationListener is...FIXME

processNotification Internal Method

```
processNotification(notification: String): Unit
```

processNotification ...FIXME

Note	processNotification is used when...FIXME
------	--

init Method

```
init(): Unit
```

init ...FIXME

Note	init is used when...FIXME
------	---------------------------

close Method

```
close(): Unit
```

close ...FIXME

Note	close is used when...FIXME
------	----------------------------

Configurable Contract

Configurable is...FIXME

Reconfigurable Contract

`Reconfigurable` is the abstraction of reconfigurables that support dynamic reconfiguration.

Table 1. Reconfigurable Contract

Method	Description
<code>reconfigurableConfigs</code>	<pre>Set<String> reconfigurableConfigs()</pre> <p>Used when...FIXME</p>
<code>reconfigure</code>	<pre>void reconfigure(Map<String, ?> configs)</pre> <p>Reconfigures the <code>Reconfigurable</code> Used when <code>DynamicBrokerConfig</code> is requested to <code>maybeReco</code> <code>processReconfigurable</code>.</p>
<code>validateReconfiguration</code>	<pre>void validateReconfiguration(Map<String, ?> configs) throws</pre> <p>Used when...FIXME</p>

Table 2. Reconfigurables (Direct Implementations)

Reconfigurable	Description
<code>DynamicClientQuotaCallback</code>	
<code>DynamicLogConfig</code>	
<code>DynamicMetricsReporters</code>	
<code>ListenerReconfigurable</code>	
<code>SslFactory</code>	

MemoryRecordsBuilder

MemoryRecordsBuilder is...FIXME

AbstractRequestResponse Contract

`AbstractRequestResponse` is the [abstraction](#) of [messages](#) exchanged between Kafka clients and brokers.

Note

`AbstractRequestResponse` is a Java abstract class and cannot be created directly. It is created indirectly for the [concrete AbstractRequestResponses](#).

`AbstractRequestResponse` defines the single `serialize` static method.

```
ByteBuffer serialize(Struct headerStruct, Struct bodyStruct)
```

`serialize` simply allocates a new `java.nio.ByteBuffer` for the given header and body structs followed by writing them out to the buffer (and rewinding the buffer).

`serialize` is used when `AbstractRequest` and `AbstractResponse` are requested to serialize.

Table 1. AbstractRequestResponses (Direct Extensions and Implementations)

AbstractRequestResponse	Description
<code>AbstractRequest</code>	
<code>AbstractResponse</code>	
<code>RequestHeader</code>	
<code>ResponseHeader</code>	

AbstractRequest Contract

`AbstractRequest` is the [extension](#) of the [AbstractRequestResponse contract](#) for [message requests](#).

`AbstractRequest` takes the following to be created:

- `ApiKeys`
- `Version`

Note

`AbstractRequest` is a Java abstract class and cannot be [created](#) directly. It is created indirectly for the [concrete AbstractRequests](#).

Table 1. AbstractRequest Contract

Method	Description
<code>getErrorResponse</code>	<pre>AbstractResponse getErrorResponse(int throttleTimeMs, Throwable e)</pre> <p>Used when...FIXME</p>
<code>toStruct</code>	<pre>Struct toStruct()</pre> <p>Used when...FIXME</p>

Table 2. AbstractRequests

AbstractRequest	Description
<code>AddOffsetsToTxnRequest</code>	
<code>AddPartitionsToTxnRequest</code>	
<code>AlterConfigsRequest</code>	
<code>AlterReplicaLogDirsRequest</code>	
<code>ApiVersionsRequest</code>	
<code>ControlledShutdownRequest</code>	
<code>CreateAclsRequest</code>	

CreateDelegationTokenRequest	
CreatePartitionsRequest	
CreateTopicsRequest	
DeleteAclsRequest	
DeleteGroupsRequest	
DeleteRecordsRequest	
DeleteTopicsRequest	
DescribeAclsRequest	
DescribeConfigsRequest	
DescribeDelegationTokenRequest	
DescribeGroupsRequest	
DescribeLogDirsRequest	
EndTxnRequest	
ExpireDelegationTokenRequest	
FetchRequest	
FindCoordinatorRequest	
HeartbeatRequest	
InitProducerIdRequest	
JoinGroupRequest	
LeaderAndIsrRequest	
LeaveGroupRequest	
ListGroupsRequest	
ListOffsetRequest	

MetadataRequest	
OffsetCommitRequest	
OffsetFetchRequest	
OffsetsForLeaderEpochRequest	
ProduceRequest	
RenewDelegationTokenRequest	
SaslAuthenticateRequest	
SaslHandshakeRequest	
StopReplicaRequest	
SyncGroupRequest	
TxnOffsetCommitRequest	
UpdateMetadataRequest	
WriteTxnMarkersRequest	

Creating AbstractRequest For API Key — `parseRequest` Factory Method

```
AbstractRequest parseRequest(
    ApiKeys apiKey,
    short apiVersion,
    Struct struct)
```

`parseRequest` simply creates a [concrete AbstractRequest](#) for the given `ApiKey`.

Note	<code>parseRequest</code> is used exclusively when <code>RequestContext</code> is requested to parse a request .
------	--

AbstractRequest.Builder Contract

`AbstractRequest.Builder` is the [abstraction](#) of request builders that can [build a request](#).

Table 1. AbstractRequest.Builder Contract

Method	Description
<code>build</code>	<p><code>T build(short version)</code></p> <p>Builds a concrete AbstractRequest (of type <code>T</code>)</p> <p>Used when:</p> <ul style="list-style-type: none">• <code>NetworkClient</code> is requested to doSend• <code>ReplicaAlterLogDirsThread</code> is requested to fetchFromLeader• <code>OffsetFetchRequest</code> is requested to forAllPartitions• <code>SaslClientAuthenticator</code> is requested to createSaslHandshakeRequest and sendSaslClientToken

AbstractResponse

AbstractResponse is...FIXME

DescribeLogDirsRequest

`DescribeLogDirsRequest` is...FIXME

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

Name	Description
<code>topicPartitions</code>	<code>TopicPartitions</code> that...FIXME Used when...FIXME

isAllTopicPartitions Method

```
boolean isAllTopicPartitions()
```

`isAllTopicPartitions` is `true` only if the `topicPartitions` is not specified.

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

FindCoordinatorRequest

`FindCoordinatorRequest` is a concrete `AbstractRequest` with **FindCoordinator** API key and the following:

- Coordinator type (i.e. `GROUP` or `TRANSACTION`)
- Coordinator key
- Version

`FindCoordinatorRequest` is **created** when:

- `AbstractRequest` is requested to **parse a request** (with the **FindCoordinator** API key)
- `FindCoordinatorRequest` is requested to **parse a byte buffer**
- `FindCoordinatorRequest.Builder` is requested to **build a FindCoordinatorRequest**

FindCoordinatorRequest.Builder Factory Object

`FindCoordinatorRequest.Builder` is a concrete `AbstractRequest.Builder` factory object that can **build** a `FindCoordinatorRequest`.

```
FindCoordinatorRequest build(short version)
```

Creating FindCoordinatorRequest from Byte Buffer — parse Factory Method

```
FindCoordinatorRequest parse(ByteBuffer buffer, short version)
```

`parse` ...FIXME

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

FindCoordinatorResponse

FindCoordinatorResponse is...FIXME

HeartbeatRequest

`HeartbeatRequest` is a concrete `AbstractRequest` with `Heartbeat` API key and the following:

- Group ID
- Group Generation ID
- Member ID

`HeartbeatRequest` is created when:

- `AbstractRequest` is requested to parse a request (with the `Heartbeat` API key)
- `HeartbeatRequest` is requested to parse a byte buffer
- `HeartbeatRequest.Builder` is requested to build a `HeartbeatRequest`

HeartbeatRequest.Builder Factory Object

`HeartbeatRequest.Builder` is a concrete `AbstractRequest.Builder` factory object that can build a `HeartbeatRequest`.

```
HeartbeatRequest build(short version)
```

Creating HeartbeatRequest from Byte Buffer — parse Factory Method

```
HeartbeatRequest parse(ByteBuffer buffer, short version)
```

parse ...FIXME

Note

parse is used when...FIXME

JoinGroupRequest

`JoinGroupRequest` is a concrete `AbstractRequest` with `JoinGroup` API key and the following:

- Version
- Group ID
- Session timeout
- Rebalance timeout
- Member ID
- Protocol type (e.g. `consumer`)
- Supported `ProtocolMetadata`

`JoinGroupRequest` is created when:

- `AbstractRequest` is requested to parse a request (with the `JoinGroup` API key)
- `JoinGroupRequest` is requested to parse a byte buffer
- `JoinGroupRequest.Builder` is requested to build a `JoinGroupRequest`

JoinGroupRequest.Builder Factory Object

`JoinGroupRequest.Builder` is a concrete `AbstractRequest.Builder` factory object that can build a `JoinGroupRequest`.

```
JoinGroupRequest build(short version)
```

Creating JoinGroupRequest from Byte Buffer — parse Factory Method

```
JoinGroupRequest parse(ByteBuffer buffer, short version)
```

parse ...FIXME

Note	parse is used when...FIXME
------	----------------------------

getErrorResponse Method

```
AbstractResponse getErrorResponse(int throttleTimeMs, Throwable e)
```

Note

`getErrorResponse` is part of the [AbstractRequest Contract](#) to create a [AbstractResponse](#) for a failure.

`getErrorResponse ...FIXME`

JoinGroupResponse

`JoinGroupResponse` is a concrete [AbstractResponse](#) that...FIXME

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

- `JoinGroupRequest` is requested to [getErrorResponse](#)
- `KafkaApis` is requested to [handle a JoinGroupRequest](#)
- `AbstractResponse` is requested to [parse a JoinGroup response](#)
- `JoinGroupResponse` is requested to [parse a ByteBuffer](#)

`JoinGroupResponse` takes the following to be created:

- Throttle time (default: 0 ms)
- Errors
- Generation ID
- Group protocol
- Member ID
- Leader ID
- Group members (`Map<String, ByteBuffer>`)

Creating `JoinGroupResponse` from `ByteBuffer` — [parse](#) Factory Method

```
JoinGroupResponse parse(ByteBuffer buffer, short version)
```

`parse` ...FIXME

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

LeaderAndIsrRequest

`LeaderAndIsrRequest` is a concrete `AbstractRequest` with `LeaderAndIsr` API key and the following properties:

- Controller ID
- Controller Epoch
- `PartitionStates` by `TopicPartition` (`Map<TopicPartition, PartitionState>`)
- Live broker nodes
- Version

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

- `AbstractRequest` is requested to [parse a request](#) (with the `LeaderAndIsr` API key)
- `LeaderAndIsrRequest` is requested to [parse a byte buffer](#)
- `LeaderAndIsrRequest.Builder` is requested to [build a LeaderAndIsrRequest](#)

LeaderAndIsrRequest . Builder Factory Object

`LeaderAndIsrRequest` comes with a concrete `AbstractRequest.Builder` factory object that can [build a LeaderAndIsrRequest](#).

```
LeaderAndIsrRequest build(short version)
```

`LeaderAndIsrRequest.Builder` is used exclusively when `ControllerBrokerRequestBatch` is requested to [sendRequestsToBrokers](#).

Creating LeaderAndIsrRequest from Byte Buffer — parse Factory Method

```
LeaderAndIsrRequest parse(ByteBuffer buffer, short version)
```

`parse` ...FIXME

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

MetadataRequest

`MetadataRequest` is an `AbstractRequest` with `METADATA` API key that is used (and [created](#)) when:

- `AdminClientRunnable` is requested to `makeMetadataCall`
- `DefaultMetadataUpdater` is requested to `maybeUpdate`
- `KafkaAdminClient` is requested to `describeCluster`, `describeTopics`, `deleteRecords`, `listConsumerGroups`, `listTopics`
- `Fetcher` is requested to `getAllTopicMetadata`
- `KafkaConsumer` is requested to `partitionsFor`

`MetadataRequest` takes the following when created:

- List of topics to request metadata for (can be empty)
- `allowAutoTopicCreation` flag (that controls whether to auto-create a topic if unavailable at the time of a request or not)

`MetadataRequest` can be [created](#) using the following factory methods:

- `MetadataRequest.Builder.allTopics`
- `MetadataRequest.Builder`

```
import org.apache.kafka.common.requests.MetadataRequest

val r = MetadataRequest.Builder.allTopics
scala> println(r)
(type=MetadataRequest, topics=<ALL>

import collection.JavaConverters._
val topics = Seq("t1", "t2").asJava
val allowAutoTopicCreation = false
val r = new MetadataRequest.Builder(topics, allowAutoTopicCreation)
scala> println(r)
(type=MetadataRequest, topics=t1,t2)
```

allTopics Factory Method

```
// MetadataRequest.Builder.allTopics
Builder allTopics()
```

`allTopics` creates a new `MetadataRequest` for all `topics` and `allowAutoTopicCreation` flag on.

MetadataResponse

MetadataResponse holds information about a Kafka cluster, i.e. the broker nodes, the controller and the topics.

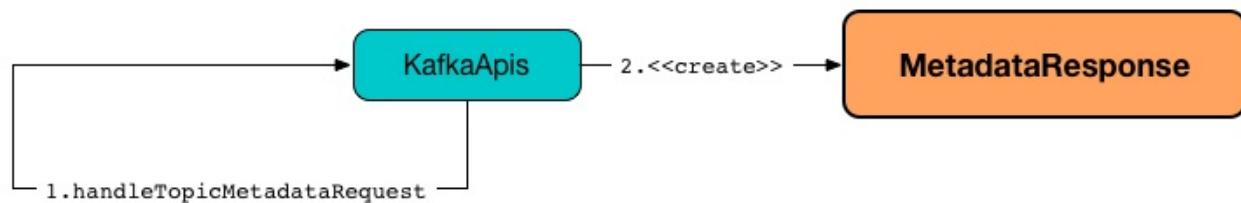


Figure 1. MetadataResponse

MetadataResponse is created mainly when KafkaApis handles a Metadata request.

cluster Method

```
Cluster cluster()
```

```
cluster ...FIXME
```

Note	cluster is used when...FIXME
------	------------------------------

Creating MetadataResponse Instance

MetadataResponse takes the following when created:

- throttleTimeMs
- Broker nodes
- cluster ID
- controller ID
- Collection of TopicMetadata

MetadataResponse initializes the internal registries and counters.

OffsetCommitRequest

`OffsetCommitRequest` is a concrete `AbstractRequest` with **OffsetCommit** API key and the following:

- Group ID
- Generation ID
- Member ID
- Retention time
- Offset data (`Map<TopicPartition, PartitionData>`)
- Version

`OffsetCommitRequest` is used exclusively when `ConsumerCoordinator` is requested to send an `OffsetCommitRequest` to the group coordinator (a Kafka broker).

`OffsetCommitRequest` is created when:

- `AbstractRequest` is requested to parse a request (with the `OffsetCommit` API key)
- `OffsetCommitRequest` is requested to parse a byte buffer
- `OffsetCommitRequest.Builder` is requested to build an `OffsetCommitRequest`

OffsetCommitRequest.Builder Factory Object

`OffsetCommitRequest.Builder` is a concrete `AbstractRequest.Builder` factory object that can build an `OffsetCommitRequest`.

```
OffsetCommitRequest build(short version)
```

Creating OffsetCommitRequest from Byte Buffer — parse Factory Method

```
OffsetCommitRequest parse(ByteBuffer buffer, short version)
```

`parse` ...FIXME

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

getErrorResponse Method

```
AbstractResponse getErrorResponse(int throttleTimeMs, Throwable e)
```

Note	getErrorResponse is part of the AbstractRequest Contract to create a AbstractResponse for a failure.
------	--

getErrorResponse ...FIXME

ProduceRequest

`ProduceRequest` is a concrete `AbstractRequest` with **Produce** API key and the following properties:

- Version
- acks
- timeout
- Partition records (`Map<TopicPartition, MemoryRecords>`)
- Transactional ID

`ProduceRequest` is used by a Kafka producer ([Kafka Producer I/O Thread](#) actually) to send [partition records](#) (with a given `acks`).

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

- `ProduceRequest.Builder` is requested to [build](#) a `ProduceRequest`
- `ProduceRequest` is requested to [parse](#) a byte buffer
- `AbstractRequest` is requested to [parse](#) a request (with the `Produce` API key)

ProduceRequest.Builder Factory Object

`ProduceRequest` comes with a concrete `AbstractRequest.Builder` factory object that can [build](#) a [ProduceRequest](#).

```
ProduceRequest build(short version, boolean validate)
```

`ProduceRequest.Builder` is used when...FIXME

Creating ProduceRequest.Builder Instance (For Given Magic Number) — `forMagic` Factory Method

```
Builder forMagic(
    byte magic,
    short acks,
    int timeout,
    Map<TopicPartition, MemoryRecords> partitionRecords,
    String transactionalId)
```

forMagic ...FIXME

Note

forMagic is used exclusively when Sender is requested to send a ProduceRequest.

Creating ProduceRequest from Byte Buffer — parse Factory Method

```
ProduceRequest parse(ByteBuffer buffer, short version)
```

parse ...FIXME

Note

parse is used when...FIXME

SyncGroupRequest

`SyncGroupRequest` is a concrete `AbstractRequest` with **SyncGroup** API key and the following:

- Group ID
- Generation ID
- Member ID
- Group partition assignment (as `Map<String, ByteBuffer>`)
- Version

`SyncGroupRequest` is created when:

- `AbstractRequest` is requested to `parse` a request (with the **SyncGroup** API key)
- `SyncGroupRequest` is requested to `parse` a byte buffer
- `SyncGroupRequest.Builder` is requested to `build` a `SyncGroupRequest`

SyncGroupRequest.Builder Factory Object

`SyncGroupRequest.Builder` is a concrete `AbstractRequest.Builder` factory object that can build a `SyncGroupRequest`.

```
SyncGroupRequest build(short version)
```

Creating SyncGroupRequest from Byte Buffer — parse Factory Method

```
SyncGroupRequest parse(ByteBuffer buffer, short version)
```

`parse` ...FIXME

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

UpdateMetadataRequest

`UpdateMetadataRequest` is an `AbstractRequest` with `updateMetadata` API key that is used (and [created](#)) when:

- `AdminClientRunnable` is requested to `makeMetadataCall`

RequestContext

RequestContext is...FIXME

RequestContext is created when:

- Processor is requested to processCompletedReceives
- SaslServerAuthenticator is requested to handleSaslToken and handleKafkaRequest

parseRequest Method

```
RequestAndSize parseRequest(ByteBuffer buffer)
```

parseRequest ...FIXME

Note

parseRequest is used when:

- RequestChannel.Request is created
- SaslServerAuthenticator is requested to handleSaslToken and handleKafkaRequest

Creating RequestContext Instance

RequestContext takes the following to be created:

- RequestHeader
- Connection ID
- Client address (as a java.net.InetAddress)
- KafkaPrincipal
- Listener name
- SecurityProtocol

Serializer Contract

```
Serializer is...FIXME
```

Deserializer Contract

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

Serde Contract

Serde is...FIXME

SerdeS Factory Object

SerdeS is...FIXME

SimpleAclAuthorizer

`SimpleAclAuthorizer` is...FIXME

startZkChangeListeners Method

`startZkChangeListeners(): Unit`

`startZkChangeListeners` ...FIXME

Note	<code>startZkChangeListeners</code> is used exclusively when <code>SimpleAclAuthorizer</code> is requested to configure .
------	---

configure Method

`configure(javaConfigs: util.Map[String, _]): Unit`

Note	<code>configure</code> is part of the Configurable Contract to...FIXME.
------	---

`configure` ...FIXME

KafkaScheduler

`KafkaScheduler` is a concrete [task scheduler](#) that allows for [scheduling tasks](#) using Java's [ScheduledThreadPoolExecutor](#).

`KafkaScheduler` uses the requested [number of threads](#) that is usually `1` except for `KafkaServer` that uses [background.threads](#) configuration property (default: `10`).

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

- `KafkaController` is [created](#) (and initializes `kafkaScheduler` and `tokenCleanScheduler`)
- `GroupMetadataManager` is [created](#)
- `KafkaServer` is requested to [start up](#)
- `ZooKeeperClient` is [created](#)

`KafkaScheduler` is used to create a [LogManager](#) and a [TransactionCoordinator](#).

Table 1. KafkaScheduler's Internal Properties (e.g. Registries and Counters)

Name	Description
<code>executor</code>	Java's ScheduledThreadPoolExecutor that schedules tasks . Initialized when <code>KafkaScheduler</code> starts up and shut down when <code>KafkaScheduler</code> shuts down.
<code>schedulerThreadId</code>	Java's AtomicInteger with initial value 0.

Tip	Enable <code>INFO</code> , <code>DEBUG</code> or <code>TRACE</code> logging level for <code>kafka.utils.KafkaScheduler</code> logger to see what happens in <code>KafkaScheduler</code> . Add the following line to <code>config/log4j.properties</code> : <pre>log4j.logger.kafka.utils.KafkaScheduler=TRACE</pre> Refer to Logging .

Starting Up — `startup` Method

```
startup(): Unit
```

Note

`startup` is part of the [Scheduler Contract](#) to initialize the scheduler so it is ready to schedule tasks.

When executed, `startup` prints out the following DEBUG message to the logs:

```
Initializing task scheduler.
```

`startup` creates a [ScheduledThreadPoolExecutor](#) with the [requested number of threads](#).

`startup` requests the [ScheduledThreadPoolExecutor](#) to disable executing existing [periodic](#) and [delayed](#) tasks after the executor has been shutdown.

`startup` requests the [ScheduledThreadPoolExecutor](#) to [use a custom thread factory](#) that creates a new `KafkaThread` with the `threadNamePrefix` followed by the `schedulerThreadId` whenever requested for a new thread (e.g. `kafka-scheduler-0`).

`startup` throws an `IllegalStateException` when the `KafkaScheduler` has already been [started](#):

```
This scheduler has already been started!
```

shutdown Method

```
shutdown(): Unit
```

Note

`shutdown` is part of the [Scheduler Contract](#) to...FIXME.

`shutdown` ...FIXME

ensureRunning Internal Method

```
ensureRunning(): Unit
```

`ensureRunning` ...FIXME

Note

`ensureRunning` is used when...FIXME

Scheduling Tasks — schedule Method

```
def schedule(
    name: String,
    fun: () => Unit,
    delay: Long = 0,
    period: Long = -1,
    unit: TimeUnit = TimeUnit.MILLISECONDS)
```

Note `schedule` is part of the [Scheduler Contract](#) to schedule a task.

When `schedule` is executed, you should see the following DEBUG message in the logs:

```
DEBUG Scheduling task [name] with initial delay [delay] ms and period [period] ms. (kafka.utils.KafkaScheduler)
```

Note `schedule` uses Java's [java.util.concurrent.TimeUnit](#) to convert `delay` and `period` to milliseconds.

`schedule` first [makes sure that](#) `KafkaScheduler` is running (which simply means that the internal `executor` has been initialized).

`schedule` creates an execution thread for the input `fun`.

For positive `period`, `schedule` schedules the thread every `period` after the initial `delay`. Otherwise, `schedule` schedules the thread once.

Note `schedule` uses the internal `executor` to schedule `fun` using [ScheduledThreadPoolExecutor.scheduleAtFixedRate](#) and [ScheduledThreadPoolExecutor.schedule](#) for periodic and one-off executions, respectively.

Whenever the thread is executed, and before `fun` gets triggered, you should see the following TRACE message in the logs:

```
Beginning execution of scheduled task '[name]'.
```

After the execution thread is finished, you should see the following TRACE message in the logs:

```
Completed execution of scheduled task '[name]'.
```

In case of any exceptions, the execution thread catches them and you should see the following ERROR message in the logs:

```
Uncaught exception in scheduled task '[name]'
```

Creating KafkaScheduler Instance

`KafkaScheduler` takes the following when created:

- Number of threads
- Thread name prefix (default: `kafka-scheduler-`)
- `daemon` flag (default: `true`)

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

scheduleOnce Method

```
scheduleOnce(name: String, fun: () => Unit): Unit
```

`scheduleOnce` ...FIXME

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

resizeThreadPool Method

```
resizeThreadPool(newSize: Int): Unit
```

`resizeThreadPool` ...FIXME

Note	<code>resizeThreadPool</code> is used exclusively when <code>DynamicThreadPool</code> is requested to reconfigure.
------	--

Scheduler Contract

`Scheduler` is the [contract](#) of task schedulers that allow for [scheduling tasks](#).

Note

[Kafka Server and Periodic Tasks](#) describes the scheduled tasks.

```
package kafka.utils

trait Scheduler {
  def isStarted: Boolean
  def shutdown()
  def schedule(
    name: String,
    fun: () => Unit,
    delay: Long = 0,
    period: Long = -1,
    unit: TimeUnit = TimeUnit.MILLISECONDS)
  def startup()
}
```

Table 1. Scheduler Contract

Property	Description
<code>isStarted</code>	<p>Flag that indicates whether the scheduler has been started or not</p> <p>Used when:</p> <ul style="list-style-type: none"> • <code>KafkaController</code> is requested to onControllerResignation • <code>GroupMetadataManager</code> is requested to shutdown • <code>LogManager</code> is requested to deleteLogs • <code>KafkaScheduler</code> is requested to startup and ensureRunning
<code>shutdown</code>	<p>Shuts down the scheduler. When this method is complete no more executions of background tasks will occur.</p> <p>Used when:</p> <ul style="list-style-type: none"> • <code>KafkaController</code> is requested to onControllerResignation • <code>GroupMetadataManager</code> is requested to shutdown • <code>TransactionCoordinator</code> is requested to shutdown • <code>KafkaServer</code> is requested to shutdown

	<ul style="list-style-type: none"> <code>ZooKeeperClient</code> is requested to close
schedule	<p>Schedules a task</p> <p>Used when:</p> <ul style="list-style-type: none"> <code>KafkaController</code> is requested to onControllerFailover and scheduleAutoLeaderRebalanceTask <code>GroupMetadataManager</code> is requested to start up, scheduleLoadGroupAndOffsets, removeGroupsForPartition and scheduleHandleTxnCompletion <code>TransactionCoordinator</code> is requested to start up <code>TransactionStateManager</code> is requested to enableTransactionalIdExpiration, loadTransactionsForTxnTopicPartition and removeTransactionsForTxnTopicPartition <code>Log</code> is created and requested to roll and asyncDeleteSegment <code>LogManager</code> is requested to start up and deleteLogs <code>ReplicaManager</code> is requested to startHighWaterMarksCheckPointThread and startup <code>KafkaScheduler</code> is requested to scheduleOnce
startup	<p>Initializes the scheduler so it is ready to schedule tasks</p> <p>Used when:</p> <ul style="list-style-type: none"> <code>KafkaController</code> is requested to onControllerFailover <code>GroupMetadataManager</code> is requested to start up <code>TransactionCoordinator</code> is requested to start up <code>KafkaServer</code> is requested to [kafka-server-KafkaServer#startup] <code>ZooKeeperClient</code> is created

Note

[KafkaScheduler](#) is the one and only known implementation of the [Scheduler Contract](#) in Apache Kafka.

ZooKeeperClient

`ZooKeeperClient` is...FIXME

`ZooKeeperClient` is a [KafkaMetricsGroup](#).

`ZooKeeperClient` is [created](#) when...FIXME

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

Name	Description
<code>expiryScheduler</code>	<code>KafkaScheduler</code> with 1 daemon thread with zk-session-expiry-handler prefix Used when...FIXME

Creating ZooKeeperClient Instance

`ZooKeeperClient` takes the following when created:

- Comma-separated host:port pairs of Zookeeper servers
- `sessionTimeoutMs`
- `connectionTimeoutMs`
- `maxInFlightRequests`
- `Time`
- Metric group
- Metric type

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

While being created, `ZooKeeperClient` ...FIXME

close Method

`close(): Unit`

`close` ...FIXME

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

KafkaZkClient — Higher-Level Kafka-Specific ZooKeeper Client

`KafkaZkClient` is a higher-level Kafka-specific ZooKeeper client.

`KafkaZkClient` is created (using the `apply` factory method) when:

- `KafkaServer` is requested to `initZkClient` (right after `kafka.Kafka` command-line application is executed)
- `AclCommand`, `ConfigCommand`, `PreferredReplicaLeaderElectionCommand`, `ReassignPartitionsCommand`, `TopicCommand` commands are executed

`KafkaZkClient` is a `KafkaMetricsGroup` and registers performance metrics.

Table 1. KafkaZkClient's Performance Metrics

Name	Description
<code>ZooKeeperRequestLatencyMs</code>	Histogram that is updated when <code>KafkaZkClient</code> is requested to <code>retryRequestsUntilConnected</code>

The performance metrics are registered in `kafka.server:type=ZooKeeperClientMetrics` group.

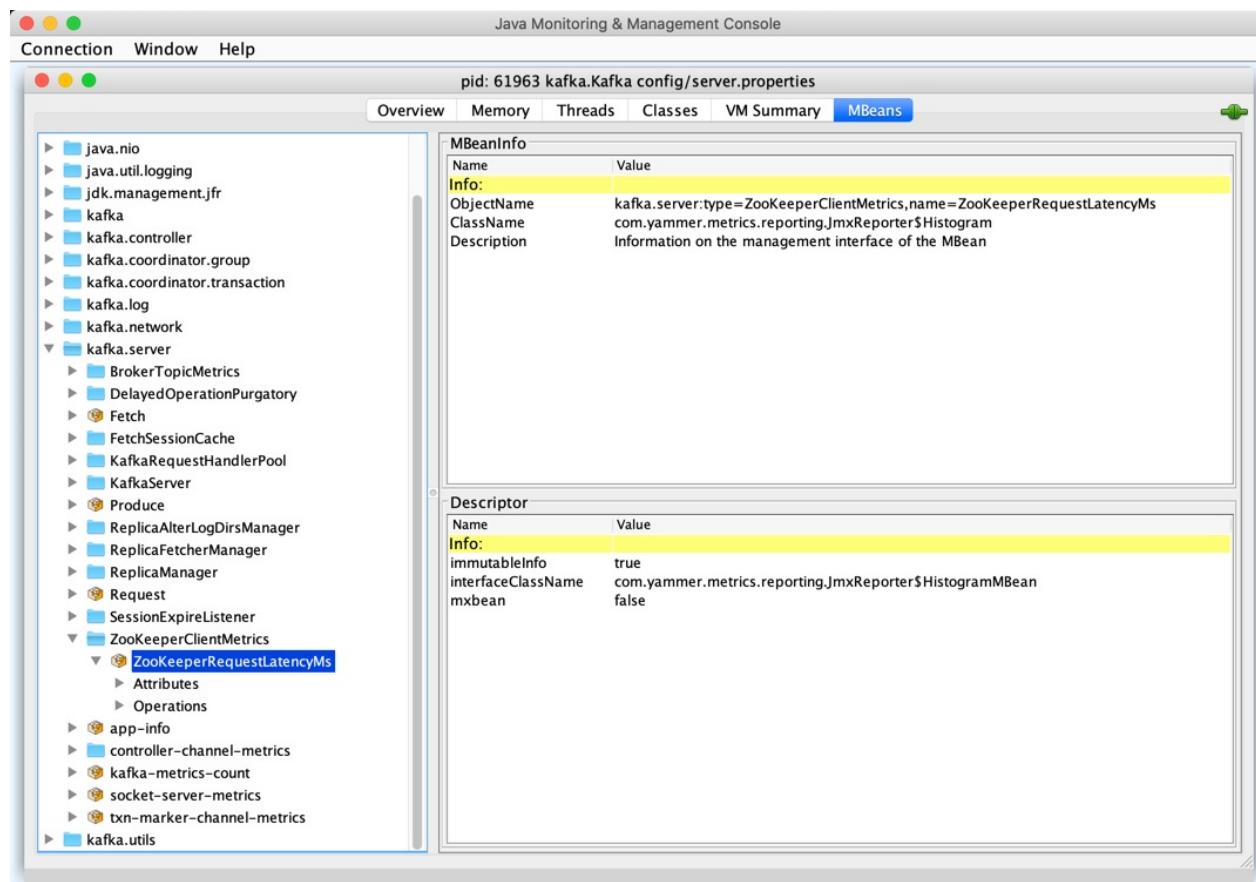


Figure 1. KafkaZkClient Metrics in JConsole JMX Tool

Tip Enable `DEBUG` logging level for `kafka.zk.KafkaZkClient` logger to see what happens inside.

Add the following line to `config/log4j.properties` :

```
log4j.logger.kafka.zk.KafkaZkClient=DEBUG
```

Refer to [Logging](#).

getTopicPartitionCount Method

```
getTopicPartitionCount(topic: String): Option[Int]
```

`getTopicPartitionCount ...FIXME`

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

Registering StateChangeHandler — registerStateChangeHandler Method

```
registerStateChangeHandler(stateChangeHandler: StateChangeHandler): Unit
```

```
registerStateChangeHandler ...FIXME
```

Note

`registerStateChangeHandler` is used when...FIXME

Creating KafkaZkClient Instance

`KafkaZkClient` takes the following when created:

- `ZooKeeperClient`
- `isSecure` flag
- `Time`

`KafkaZkClient` initializes the internal registries and counters.

Creating KafkaZkClient Instance — `apply` Factory Method

```
apply(  
    connectString: String,  
    isSecure: Boolean,  
    sessionTimeoutMs: Int,  
    connectionTimeoutMs: Int,  
    maxInFlightRequests: Int,  
    time: Time,  
    metricGroup: String = "kafka.server",  
    metricType: String = "SessionExpireListener"): KafkaZkClient
```

`apply` creates a `ZooKeeperClient` that is then used to create a `KafkaZkClient`.

Note

`apply` is used when:

- `KafkaServer` is requested to `initZkClient` (right after `kafka.Kafka` command-line application is `executed`)
- The following commands are executed (`main`):
 - `AclCommand` (when `SimpleAclAuthorizer` is used and requested to `configure`)
 - `ConfigCommand`
 - `PreferredReplicaLeaderElectionCommand`
 - `ReassignPartitionsCommand`
 - `TopicCommand`

createTopLevelPaths Method

```
createTopLevelPaths(): Unit
```

`createTopLevelPaths` ...FIXME

Note

`createTopLevelPaths` is used when...FIXME

updateBrokerInfoInZk Method

```
updateBrokerInfoInZk(brokerInfo: BrokerInfo): Unit
```

`updateBrokerInfoInZk` ...FIXME

Note

`updateBrokerInfoInZk` is used when...FIXME

registerZNodeChildChangeHandler Method

```
registerZNodeChildChangeHandler(zNodeChildChangeHandler: ZNodeChildChangeHandler): Unit
```

`registerZNodeChildChangeHandler` ...FIXME

Note

`registerZNodeChildChangeHandler` is used when...FIXME

registerZNodeChangeHandlerAndCheckExistence Method

```
registerZNodeChangeHandlerAndCheckExistence(zNodeChangeHandler: ZNodeChangeHandler): Boolean
```

registerZNodeChangeHandlerAndCheckExistence ...FIXME

Note	registerZNodeChangeHandlerAndCheckExistence is used when...FIXME
------	--

deleteLogDirEventNotifications Method

```
deleteLogDirEventNotifications(): Unit
```

deleteLogDirEventNotifications ...FIXME

Note	deleteLogDirEventNotifications is used when...FIXME
------	---

deleteIsrChangeNotifications Method

```
deleteIsrChangeNotifications(): Unit
```

deleteIsrChangeNotifications ...FIXME

Note	deleteIsrChangeNotifications is used when...FIXME
------	---

unregisterZNodeChildChangeHandler Method

```
unregisterZNodeChildChangeHandler(path: String): Unit
```

unregisterZNodeChildChangeHandler ...FIXME

Note	unregisterZNodeChildChangeHandler is used when...FIXME
------	--

unregisterZNodeChangeHandler Method

```
unregisterZNodeChangeHandler(path: String): Unit
```

```
unregisterZNodeChangeHandler ...FIXME
```

Note	unregisterZNodeChangeHandler is used when...FIXME
------	---

setControllerEpochRaw Method

```
setControllerEpochRaw(epoch: Int, epochZkVersion: Int): SetDataResponse
```

```
setControllerEpochRaw ...FIXME
```

Note	setControllerEpochRaw is used when...FIXME
------	--

createControllerEpochRaw Method

```
createControllerEpochRaw(epoch: Int): CreateResponse
```

```
createControllerEpochRaw ...FIXME
```

Note	createControllerEpochRaw is used when...FIXME
------	---

Fetching Metadata of Brokers in Cluster — getAllBrokersInCluster Method

```
getAllBrokersInCluster: Seq[Broker]
```

getAllBrokersInCluster [fetches broker IDs](#) followed by fetching the metadata of every broker (which is the data associated with a `/brokers/ids/[brokerId]` znode).

Note	getAllBrokersInCluster is used when:
------	--------------------------------------

- `ConfigCommand` is requested to [alterConfig](#)
- `ReassignPartitionsCommand` is requested to [removeThrottle](#)
- `TopicCommand` is requested to [describeTopic](#)
- `KafkaController` is requested to [initializeControllerContext](#) and at [BrokerChange](#) controller event
- `KafkaServer` is requested to [createBrokerInfo](#)
- `AdminZkClient` is requested to [getBrokerMetadatas](#)

getAllTopicsInCluster Method

```
getAllTopicsInCluster: Seq[String]
```

```
getAllTopicsInCluster ...FIXME
```

Note	getAllTopicsInCluster is used when...FIXME
------	--

getReplicaAssignmentForTopics Method

```
getReplicaAssignmentForTopics(topics: Set[String]): Map[TopicPartition, Seq[Int]]
```

```
getReplicaAssignmentForTopics ...FIXME
```

Note	getReplicaAssignmentForTopics is used when...FIXME
------	--

getPartitionReassignment Method

```
getPartitionReassignment: collection.Map[TopicPartition, Seq[Int]]
```

```
getPartitionReassignment ...FIXME
```

Note	getPartitionReassignment is used when...FIXME
------	---

getTopicDeletions Method

```
getTopicDeletions: Seq[String]
```

```
getTopicDeletions ...FIXME
```

Note	getTopicDeletions is used when...FIXME
------	--

Retrieving Partition State — getTopicPartitionStates Method

```
getTopicPartitionStates(partitions: Seq[TopicPartition]): Map[TopicPartition, LeaderIs  
rAndControllerEpoch]
```

`getTopicPartitionStates` [getTopicPartitionStatesRaw](#) for the given `TopicPartitions`.

For every response, `getTopicPartitionStates` decodes the JSON-encoded partition state data (for the partitions that were found in ZooKeeper).

Note

`getTopicPartitionStates` is used when `KafkaController` is requested to [updateLeaderAndIsrCache](#), [areReplicasInIsr](#), [updateLeaderEpoch](#) and process a [PartitionReassignmentIsrChange](#) controller event.

registerZNodeChangeHandler Method

`registerZNodeChangeHandler(zNodeChangeHandler: ZNodeChangeHandler): Unit`

`registerZNodeChangeHandler ...FIXME`

Note

`registerZNodeChangeHandler` is used when...FIXME

getControllerEpoch Method

`getControllerEpoch: Option[(Int, Stat)]`

`getControllerEpoch ...FIXME`

Note

`getControllerEpoch` is used when...FIXME

deletePartitionReassignment Method

`deletePartitionReassignment(): Unit`

`deletePartitionReassignment ...FIXME`

Note

`deletePartitionReassignment` is used when...FIXME

setOrCreatePartitionReassignment Method

`setOrCreatePartitionReassignment(reassignment: collection.Map[TopicPartition, Seq[Int]]): Unit`

`setOrCreatePartitionReassignment ...FIXME`

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

Getting Active Controller ID — `getControllerId` Method

```
getControllerId: Option[Int]
```

`getControllerId` sends a request to Zookeeper for the data of the `/controller` znode and returns the following:

- The `brokerid` field of the JSON data when the response is `ok`
- `None` for a `NONODE` response
- Throws a `KeeperException` with the response code and the `/controller` path

Note	<code>getControllerId</code> is used when:
------	--

- `KafkaController` is requested to `elect`
- `ControllerEventThread` is requested to `process controller events` (and processes `ControllerChange` and `Reelect` events)
- `KafkaServer` is requested to `perform a controlled shutdown`

Creating Ephemeral Znode (And Throwing Exception When Unsuccessful)-- `checkedEphemeralCreate` Method

```
checkedEphemeralCreate(path: String, data: Array[Byte]): Unit
```

`checkedEphemeralCreate` ...FIXME

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

`registerControllerAndIncrementControllerEpoch` Method

```
registerControllerAndIncrementControllerEpoch(controllerId: Int): (Int, Int)
```

`registerControllerAndIncrementControllerEpoch` ...FIXME

Note

`registerControllerAndIncrementControllerEpoch` is used exclusively when `KafkaController` is requested to `elect`.

retryRequestsUntilConnected Internal Method

```
retryRequestsUntilConnected[Req <: AsyncRequest](requests: Seq[Req]): Seq[Req#Response]
```

`retryRequestsUntilConnected` ...FIXME

createSequentialPersistentPath Method

```
createSequentialPersistentPath(path: String, data: Array[Byte]): String
```

`createSequentialPersistentPath` ...FIXME

Note

`createSequentialPersistentPath` is used when `KafkaZkClient` is requested to `propagateLogDirEvent` and `propagateIsrChanges`.

propagateLogDirEvent Method

```
propagateLogDirEvent(brokerId: Int): Unit
```

`propagateLogDirEvent` ...FIXME

Note

`propagateLogDirEvent` is used exclusively when `ReplicaManager` is requested to `handleLogDirFailure`.

propagateIsrChanges Method

```
propagateIsrChanges(isrChangeSet: collection.Set[TopicPartition]): Unit
```

`propagateIsrChanges` ...FIXME

Note

`propagateIsrChanges` is used exclusively when `ReplicaManager` is requested to `maybePropagateIsrChanges`.

getTopicPartitionStatesRaw Method

```
getTopicPartitionStatesRaw(partitions: Seq[TopicPartition]): Seq[GetDataResponse]
```

`getTopicPartitionStatesRaw` creates a ZooKeeper `GetDataRequest` for the path `/brokers/topics/[topic]/partitions/[partition]/state` for every partition in the given partitions.

In the end, `getTopicPartitionStatesRaw` [retryRequestsUntilConnected](#) the `GetDataRequests`.

Note

`getTopicPartitionStatesRaw` is used when:

- `PartitionStateMachine` is requested to [doElectLeaderForPartitions](#)
- `ReplicaStateMachine` is requested to [getTopicPartitionStatesFromZk](#)
- `KafkaZkClient` is requested to [getTopicPartitionStates](#) and [getTopicPartitionState](#)

getTopicPartitionState Method

```
getTopicPartitionState(partition: TopicPartition): Option[LeaderIsrAndControllerEpoch]
```

`getTopicPartitionState` ...FIXME

Note

`getTopicPartitionState` is used when...FIXME

Fetching Broker IDs — getSortedBrokerList Method

```
getSortedBrokerList(): Seq[Int]
```

`getSortedBrokerList` gets the child znodes at `/brokers/ids` path and sorts it by broker ID (according to the natural ordering).

Note

`getSortedBrokerList` is used when:

- `ReassignPartitionsCommand` is requested to [parseAndValidate](#)
- `KafkaZkClient` is requested to [getAllBrokersInCluster](#)

Fetching Child ZNodes — getChildren Method

```
getChildren(path : String): Seq[String]
```

getChildren ...FIXME

Note

getChildren is used when...FIXME

getClusterId Method

getClusterId: Option[String]

getClusterId ...FIXME

Note

getClusterId is used when...FIXME

createOrGetClusterId Method

createOrGetClusterId(proposedClusterId: String): String

createOrGetClusterId ...FIXME

Note

createOrGetClusterId is used when...FIXME

AdminZkClient

`AdminZkClient` is created when:

- `ConfigCommand` is requested to `processCommandWithZk`
- `ReassignPartitionsCommand` is requested to `verifyAssignment`, `generateAssignment`, `executeAssignment`
- `TopicCommand` is requested to `createTopic`, `alterTopic`, `describeTopic`
- `Partition` is requested to `getOrCreateReplica`
- `DynamicBrokerConfig` is requested to `initialize`
- `AdminManager`, `DynamicConfigManager` and `KafkaApis` are created

`AdminZkClient` takes a single `KafkaZkClient` when created.

fetchEntityConfig Method

```
fetchEntityConfig(rootEntityType: String, sanitizedEntityName: String): Properties
```

`fetchEntityConfig` ...FIXME

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

writeTopicPartitionAssignment Internal Method

```
writeTopicPartitionAssignment(
    topic: String,
    replicaAssignment: Map[Int, Seq[Int]],
    update: Boolean): Unit
```

`writeTopicPartitionAssignment` ...FIXME

Note	<code>writeTopicPartitionAssignment</code> is used exclusively when <code>AdminZkClient</code> is requested to <code>createOrUpdateTopicPartitionAssignmentPathInZK</code> .
------	--

createOrUpdateTopicPartitionAssignmentPathInZK Internal Method

```
createOrUpdateTopicPartitionAssignmentPathInZK(
    topic: String,
    partitionReplicaAssignment: Map[Int, Seq[Int]],
    config: Properties = new Properties,
    update: Boolean = false): Unit
```

`createOrUpdateTopicPartitionAssignmentPathInZK` ...FIXME

Note

`createOrUpdateTopicPartitionAssignmentPathInZK` is used when:

- `TopicCommand` is requested to [create a topic](#)
- `AdminManager` is requested to [create topics](#)
- `AdminZkClient` is requested to [create a topic](#) and [addPartitions](#)

Creating Topic— `createTopic` Method

```
createTopic(
    topic: String,
    partitions: Int,
    replicationFactor: Int,
    topicConfig: Properties = new Properties,
    rackAwareMode: RackAwareMode = RackAwareMode.Enforced): Unit
```

`createTopic` [fetches the metadata of the brokers in the cluster](#) (given the `RackAwareMode`).

`createTopic` [requests Adminutils helper object to assignReplicasToBrokers](#) (given the broker metadata, partitions and replicationFactor).

In the end, `createTopic` [createOrUpdateTopicPartitionAssignmentPathInZK](#).

Note

`createTopic` is used when:

- `TopicCommand` is requested to [create a topic](#)
- `KafkaApis` is requested to [create a topic](#)

addPartitions Method

```
addPartitions(
    topic: String,
    existingAssignment: Map[Int, Seq[Int]],
    allBrokers: Seq[BrokerMetadata],
    numPartitions: Int = 1,
    replicaAssignment: Option[Map[Int, Seq[Int]]] = None,
    validateOnly: Boolean = false): Map[Int, Seq[Int]]
```

`addPartitions` ...FIXME

Note

`addPartitions` is used when...FIXME

Fetching Metadata of Brokers in Cluster (Broker ID and Rack Information) — `getBrokerMetadatas` Method

```
getBrokerMetadatas(
    rackAwareMode: RackAwareMode = RackAwareMode.Enforced,
    brokerList: Option[Seq[Int]] = None): Seq[BrokerMetadata]
```

`getBrokerMetadatas` requests the [KafkaZkClient](#) for the metadata of the brokers in the cluster.

Note

The broker metadata includes a broker ID, endpoints, and an optional rack information.

`getBrokerMetadatas` takes the input broker IDs (`brokerList`) if defined and leaves only those that are available (i.e. among the brokers in the cluster) or falls back to all brokers in the cluster.

`getBrokerMetadatas` branches off per `RackAwareMode` as follows:

- For `RackAwareMode.Disabled` or `RackAwareMode.Safe`, `getBrokerMetadatas` returns `BrokerMetadata` with broker IDs and no rack information
- For `RackAwareMode.Enforced`, `getBrokerMetadatas` returns `BrokerMetadata` with broker IDs and rack information

In the end, `getBrokerMetadatas` sorts the brokers by broker ID.

`getBrokerMetadatas` throws an `AdminOperationException` for `RackAwareMode.Enforced` mode with some brokers without rack information:

Not all brokers have rack information. Add `--disable-rack-aware` in command line to make replica assignment without rack information.

Note

`getBrokerMetadatas` is used when:

- `AdminManager` is requested to [createPartitions](#)
- `AdminZkClient` is requested to [create a topic](#)
- `ReassignPartitionsCommand` is requested to [generateAssignment](#)
- `TopicCommand` is requested to [alterTopic](#)

ZkAclChangeStore Contract

`ZkAclChangeStore` is the [contract](#) of [ZkAclChangeStores](#) that [FIXME](#).

```
package kafka.zk

sealed trait ZkAclChangeStore {
    // only required properties (vals and methods) that have no implementation
    // the others follow
    val aclChangePath: String
    def decode(bytes: Array[Byte]): Resource
    protected def encode(resource: Resource): Array[Byte]
}
```

Note	<code>ZkAclChangeStore</code> is a Scala sealed trait and so all the available implementations are in a single compilation unit (i.e. the file with the Scala source code).
------	---

Table 1. (Subset of) ZkAclChangeStore Contract

Property	Description
<code>aclChangePath</code>	Used when...FIXME
<code>decode</code>	Used when...FIXME
<code>encode</code>	Used when...FIXME

Table 2. ZkAclChangeStores

ZkAclChangeStore	Description
<code>ExtendedAclChangeStore</code>	
<code>LiteralAclChangeStore</code>	

createListener Method

```
createListener(
    handler: AclChangeNotificationHandler,
    zkClient: KafkaZkClient): AclChangeSubscription
```

`createListener` creates a [NotificationHandler](#) that simply requests the input `AclChangeNotificationHandler` to `processNotification` after [decoding](#) the input notification message.

`createListener` creates a [ZkNodeChangeNotificationListener](#) (with the input [KafkaZkClient](#), the [aclChangePath](#), [acl_changes_](#) node prefix and the `NotificationHandler`).

`createListener` requests the [ZkNodeChangeNotificationListener](#) to `init`.

In the end, `createListener` returns a new `AclChangeSubscription` that requests the [ZkNodeChangeNotificationListener](#) to `close` when requested to close.

Note	<code>createListener</code> is used exclusively when <code>SimpleAclAuthorizer</code> is requested to startZkChangeListeners .
------	--

WorkerGroupMember

Caution	FIXME WorkerCoordinator? DistributedHerder?
---------	---

ConnectDistributed

`ConnectDistributed` is a command-line utility that runs [Kafka Connect](#) in distributed mode.

Caution	FIXME Doh, I'd rather not enter Kafka Connect yet. Not interested in it yet.
---------	--

Demo: Kafka Controller Election

Use the following setup with one Zookeeper server and two Kafka brokers to observe the Kafka controller election.

Start Zookeeper server.

```
$ ./bin/zookeeper-server-start.sh config/zookeeper.properties
...
INFO binding to port 0.0.0.0/0.0.0.0:2181 (org.apache.zookeeper.server.NIOServerCnxnFactory)
```

Add the following line to `config/log4j.properties` to enable `DEBUG` logging level for `kafka.controller.KafkaController` logger.

```
log4j.logger.kafka.controller.KafkaController=DEBUG
```

Start a Kafka broker.

```
$ ./bin/kafka-server-start.sh config/server.properties \
  --override broker.id=100 \
  --override log.dirs=/tmp/kafka-logs-100 \
  --override port=9192
...
INFO Registered broker 100 at path /brokers/ids/100 with addresses: EndPoint(192.168.1
.4,9192,ListenerName(PLAINTEXT),PLAINTEXT) (kafka.utils.ZkUtils)
INFO Kafka version : 1.0.0-SNAPSHOT (org.apache.kafka.common.utils.AppInfoParser)
INFO Kafka commitId : 852297efd99af04d (org.apache.kafka.common.utils.AppInfoParser)
INFO [KafkaServer id=100] started (kafka.server.KafkaServer)
```

Start another Kafka broker.

```
# Note the different properties
$ ./bin/kafka-server-start.sh config/server.properties \
  --override broker.id=200 \
  --override log.dirs=/tmp/kafka-logs-200 \
  --override port=9292
...
INFO Registered broker 200 at path /brokers/ids/200 with addresses: EndPoint(192.168.1
.4,9292,ListenerName(PLAINTEXT),PLAINTEXT) (kafka.utils.ZkUtils)
INFO Kafka version : 1.0.0-SNAPSHOT (org.apache.kafka.common.utils.AppInfoParser)
INFO Kafka commitId : 852297efd99af04d (org.apache.kafka.common.utils.AppInfoParser)
INFO [KafkaServer id=200] started (kafka.server.KafkaServer)
```

Connect to Zookeeper using Zookeeper CLI (command-line interface). Use the official distribution of Apache Zookeeper as described in [Zookeeper Tips](#).

```
$ ./bin/zkCli.sh -server localhost:2181
```

Once connected, execute `get /controller` to get the data associated with `/controller` znode where the active Kafka controller stores the controller ID.

```
[zk: localhost:2181(CONNECTED) 0] get /controller
>{"version":1,"brokerid":100,"timestamp":"1506423376977"}
cZxid = 0x191
ctime = Tue Sep 26 12:56:16 CEST 2017
mZxid = 0x191
mtime = Tue Sep 26 12:56:16 CEST 2017
pZxid = 0x191
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x15ebdd241840002
dataLength = 56
numChildren = 0
```

(optional) Clear the consoles of the two Kafka brokers so you have the election logs only.

Delete `/controller` znode and observe the controller election.

```
[zk: localhost:2181(CONNECTED) 2] delete /controller
```

You should see the following in the logs in the consoles of the two Kafka brokers.

```
DEBUG [Controller id=100] Resigning (kafka.controller.KafkaController)
DEBUG [Controller id=100] De-registering IsrChangeNotificationListener (kafka.controller.KafkaController)
DEBUG [Controller id=100] De-registering logDirEventNotificationListener (kafka.controller.KafkaController)
INFO [Controller id=100] Resigned (kafka.controller.KafkaController)
DEBUG [Controller id=100] Broker 200 has been elected as the controller, so stopping the election process. (kafka.controller.KafkaController)
```

and

```
INFO Creating /controller (is it secure? false) (kafka.utils.ZKCheckedEphemeral)
INFO Result of znode creation is: OK (kafka.utils.ZKCheckedEphemeral)
INFO [Controller id=200] 200 successfully elected as the controller (kafka.controller.KafkaController)
INFO [Controller id=200] Starting become controller state transition (kafka.controller.KafkaController)
INFO [Controller id=200] Initialized controller epoch to 39 and zk version 38 (kafka.controller.KafkaController)
INFO [Controller id=200] Incremented epoch to 40 (kafka.controller.KafkaController)
DEBUG [Controller id=200] Registering IsrChangeNotificationListener (kafka.controller.KafkaController)
DEBUG [Controller id=200] Registering logDirEventNotificationListener (kafka.controller.KafkaController)
INFO [Controller id=200] Partitions being reassigned: Map() (kafka.controller.KafkaController)
INFO [Controller id=200] Partitions already reassigned: Set() (kafka.controller.KafkaController)
INFO [Controller id=200] Resuming reassignment of partitions: Map() (kafka.controller.KafkaController)
INFO [Controller id=200] Currently active brokers in the cluster: Set(100, 200) (kafka.controller.KafkaController)
INFO [Controller id=200] Currently shutting brokers in the cluster: Set() (kafka.controller.KafkaController)
INFO [Controller id=200] Current list of topics in the cluster: Set(my-topic2, NEW, my-topic, my-topic1) (kafka.controller.KafkaController)
INFO [Controller id=200] List of topics to be deleted: (kafka.controller.KafkaController)
INFO [Controller id=200] List of topics ineligible for deletion: (kafka.controller.KafkaController)
INFO [Controller id=200] Ready to serve as the new controller with epoch 40 (kafka.controller.KafkaController)
INFO [Controller id=200] Partitions undergoing preferred replica election: (kafka.controller.KafkaController)
INFO [Controller id=200] Partitions that completed preferred replica election: (kafka.controller.KafkaController)
INFO [Controller id=200] Skipping preferred replica election for partitions due to topic deletion: (kafka.controller.KafkaController)
INFO [Controller id=200] Resuming preferred replica election for partitions: (kafka.controller.KafkaController)
INFO [Controller id=200] Starting preferred replica leader election for partitions (kafka.controller.KafkaController)
INFO [Controller id=200] Starting the controller scheduler (kafka.controller.KafkaController)
```

Further Reading or Watching

Videos

1. [Apache Kafka Core Internals: A Deep Dive](#) by Jun Rao, Confluent at Kafka Summit 2017 in New York City

Articles

1. [Apache Kafka for Beginners](#) - an excellent article that you should start your Kafka journey with.
2. [Introduction to Apache Kafka™ for Python Programmers](#) - using Python as the programming language, but offers enough exposure to the topics of the Kafka architecture.