

---

# Table of Contents

Introduction	1.1
Overview of Kafka	1.2
Kafka Server and Periodic Tasks	1.3

---

## Kafka Broker Services

AdminManager	2.1
Authorizer	2.2
Cluster	2.3
Cluster (deprecated)	2.3.1
ClusterConnectionStates	2.4
ClusterResourceListener (and ClusterResourceListeners Collection)	2.5
DynamicBrokerConfig	2.6
DynamicLogConfig	2.7
DynamicMetricsReporters	2.8
DynamicClientQuotaCallback	2.9
ConfigHandler Contract	2.10
BrokerConfigHandler	2.10.1
ClientIdConfigHandler	2.10.2
TopicConfigHandler	2.10.3
UserConfigHandler	2.10.4
DelegationTokenManager	2.11
DynamicConfigManager	2.12
BrokerReconfigurable Contract — Reconfigurable Broker Services	2.13
DynamicThreadPool	2.13.1
DynamicListenerConfig	2.13.2
LogCleaner	2.13.3
Fetcher	2.14
GroupCoordinator	2.15
GroupMetadataManager	2.16

---

---

InterBrokerSendThread	2.17
Kafka	2.18
KafkaApis — API Request Handler	2.19
KafkaHealthcheck	2.20
KafkaServerStartable — Thin Management Layer over KafkaServer	2.21
KafkaServer	2.22
KafkaConfig	2.23
KafkaController	2.24
ControllerEventManager	2.24.1
ControllerEventThread	2.24.2
ControllerEvent	2.24.3
TopicDeletion Controller Event	2.24.3.1
ControllerBrokerRequestBatch	2.24.4
KafkaMetricsReporter	2.25
KafkaRequestHandler	2.26
KafkaRequestHandlerPool — Pool of Daemon KafkaRequestHandler Threads	2.27
KafkaScheduler	2.28
Log	2.29
LogCleanerManager	2.30
LogDirFailureHandler	2.31
LogManager	2.32
ProducerStateManager	2.33
Metadata	2.34
Metadata Update Listener	2.34.1
MetadataCache	2.35
MetadataResponse	2.36
MetadataUpdater	2.37
DefaultMetadataUpdater	2.37.1
NetworkClient — Non-Blocking KafkaClient	2.38
KafkaClient	2.38.1
NetworkClientUtils	2.38.2
OffsetConfig	2.39
ReplicaManager	2.40
ReplicaFetcherManager	2.40.1

---

---

AbstractFetcherManager	2.40.1.1
ReplicaFetcherThread	2.40.2
AbstractFetcherThread	2.40.2.1
ReplicaFetcherBlockingSend	2.40.2.2
ReplicationQuotaManager	2.40.3
ReplicationUtils	2.40.4
ReplicaStateMachine	2.40.5
Selector — Selectable on Socket Channels (from Java's New IO API)	2.41
Selectable	2.41.1
ShutdownableThread	2.42
SocketServer	2.43
TopicDeletionManager	2.44
TransactionCoordinator	2.45
TransactionStateManager	2.46
ZkUtils	2.47
ZKRebalancerListener	2.48

## Kafka Controller

ControllerBrokerRequestBatch	3.1
------------------------------	-----

## Kafka Cluster

Partition	4.1
Replica	4.2
PartitionStateMachine	4.3

## Kafka Metrics

BrokerTopicStats	5.1
BrokerTopicMetrics	5.2
KafkaMetricsGroup	5.3

---

# Kafka Features

Topic Replication	6.1
Topic Deletion	6.2
Kafka Controller Election	6.3

## Kafka Producer API

KafkaProducer — Main Class For Kafka Producers	7.1
Producer	7.1.1
DefaultPartitioner	7.2
Partitioner	7.2.1
ProducerInterceptor	7.3
Sender	7.4
Serializer	7.5

## Kafka Consumer API

KafkaConsumer — Main Class For Kafka Consumers	8.1
Consumer	8.1.1
Deserializer	8.2
ConsumerConfig	8.3
ConsumerCoordinator	8.4
AbstractCoordinator	8.4.1
ConsumerInterceptor	8.5
ConsumerNetworkClient	8.6
ConsumerRebalanceListener	8.7
SubscriptionState	8.8

## Kafka Architecture

Broker Nodes — Kafka Servers	9.1
Broker	9.1.1
Topics	9.2

---

Messages	9.3
Kafka Clients	9.4
Producers	9.4.1
Consumers	9.4.2
RequestCompletionHandler	9.5
ClientResponse	9.6
Clusters	9.7

## Kafka Operations and Administration

kafka-consumer-groups.sh	10.1
ConsumerGroupCommand	10.1.1
KafkaConsumerGroupService	10.1.2
ConsumerGroupService	10.1.2.1
KafkaAdminClient	10.2
AdminClient	10.2.1
ReassignPartitionsCommand	10.3
TopicCommand	10.4

## Kafka Monitoring (Metrics)

Metrics	11.1
Sensor	11.2
MetricsReporter	11.3
ProducerMetrics	11.4
SenderMetrics	11.5

## Kafka Tools

Kafka Tools	12.1
kafka-configs.sh	12.1.1
kafka-topics.sh	12.1.2

---

# Kafka Configuration

Properties	13.1
<a href="#">bootstrap.servers</a>	13.1.1
<a href="#">client.id</a>	13.1.2
<a href="#">enable.auto.commit</a>	13.1.3
<a href="#">group.id</a>	13.1.4
<a href="#">retry.backoff.ms</a>	13.1.5
Logging	13.2

## Tips and Tricks

<a href="#">Gradle Tips</a>	14.1
<a href="#">Zookeeper Tips</a>	14.2
<a href="#">Kafka in Scala REPL for Interactive Exploration</a>	14.3
<a href="#">Running Kafka Broker in Docker</a>	14.4

## Kafka Clients

<a href="#">Configurable Contract</a>	15.1
<a href="#">Reconfigurable</a>	15.2
<a href="#">DescribeLogDirsRequest</a>	15.3

## Kafka Commons

<a href="#">NotificationHandler Contract</a>	16.1
<a href="#">ZkNodeChangeNotificationListener</a>	16.2

## Kafka Security

<a href="#">SimpleAclAuthorizer</a>	17.1
-------------------------------------	------

## Varia / Misc

---

<a href="#">Scheduler</a>	18.1
<a href="#">ZooKeeperClient</a>	18.2
<a href="#">KafkaZkClient</a>	18.3
<a href="#">AdminZkClient</a>	18.4
<a href="#">ZkAclChangeStore</a>	18.5

## Kafka Connect

<a href="#">WorkerGroupMember</a>	19.1
<a href="#">ConnectDistributed</a>	19.2

## Appendix

<a href="#">Further reading or watching</a>	20.1
---	------

# Mastering Apache Kafka (2.0.0)

Welcome to **Mastering Apache Kafka** gitbook! I'm very excited to have you here and hope you will enjoy exploring the internals of Spark SQL as much as I have.

I write to discover what I know.

— Flannery O'Connor

I'm [Jacek Laskowski](#), an independent consultant, software developer and technical instructor specializing in **Apache Spark**, Apache Kafka and Kafka Streams (with Scala, sbt, Kubernetes, DC/OS, Apache Mesos, and Hadoop YARN).

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

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

Tip	I'm also writing <a href="#">Mastering Apache Spark 2</a> , <a href="#">Mastering Spark SQL</a> , <a href="#">Mastering Kafka Streams</a> and <a href="#">Spark Structured Streaming Notebook</a> gitbooks.
-----	---

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](#).

# 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	Description
KafkaController	delete-expired-tokens	Executes <a href="#">expireTokens</a>
	auto-leader-rebalance-task	Sends <a href="#">AutoPreferredReplicaLeaderChange</a> controller event to <a href="#">ControllerEventManager</a>
GroupMetadataManager	delete-expired-group-metadata	Executes <a href="#">cleanupGroupMetadata</a>
	__consumer_offsets-[offsetsPartition]	Executes <a href="#">loadGroupsAndOffsets</a>
	__consumer_offsets-[offsetsPartition]	Executes <a href="#">removeGroupMetadata</a>
	handleTxnCompletion-[producerId]	Executes <a href="#">handleTxnCompletion</a>
TransactionCoordinator	transaction-abort	Executes <a href="#">abortTimedOutTransactions</a>
TransactionStateManager	transactionalId-expiration	Executes <a href="#">enableTransactionalIdExpiration</a>
	load-txns-for-partition-[topicPartition]	Executes <a href="#">loadTransactions</a>
	remove-txns-for-partition-[topicPartition]	Executes <a href="#">removeTransactions</a>
Log	PeriodicProducerExpirationCheck	Executes <a href="#">removeExpiredProducers</a>
	flush-log	Executes <a href="#">flush</a>
	delete-file	Executes <a href="#">deleteSegment</a>
	kafka-log-retention	Executes <a href="#">cleanupLogs</a>

LogManager	kafka-log-flusher	Executes <code>flushDirtyLog</code>
	kafka-recovery-point-checkpoint	Executes <code>checkpointLogRecovery</code>
	kafka-log-start-offset-checkpoint	Executes <code>checkpointLogStartOffset</code>
	kafka-delete-logs	Executes <code>deleteLogs</code>
ReplicaManager	highwatermark-checkpoint	Executes <code>checkpointHighWatermark</code>
	isr-expiration	Executes <code>maybeShrinkIsr</code>
	isr-change-propagation	Executes <code>maybePropagateIsrChange</code>
	shutdown-idle-replica-alter-log-dirs-thread	Executes <code>shutdownIdleReplicaAlterLogDirsThread</code>

# AdminManager

AdminManager is...FIXME

AdminManager is created exclusively when KafkaServer is started.

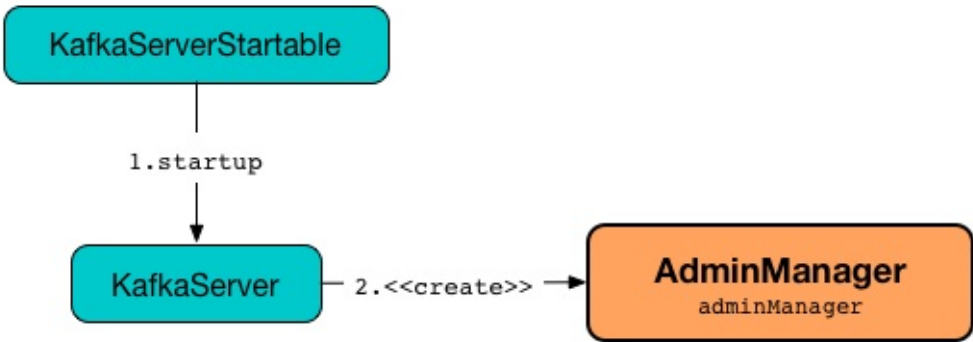


Figure 1. AdminManager

logIdent is [Admin Manager on Broker [brokerId]].

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

Name	Description
alterConfigPolicy	
createTopicPolicy	
topicPurgatory	

Tip

Enable INFO , DEBUG , TRACE logging levels for kafka.server.AdminManager logger to see what happens inside.

Add the following line to config/log4j.properties :

log4j.logger.kafka.server.AdminManager=TRACE

Refer to [Logging](#).

## Creating Topics — createTopics Method

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

createTopics ...FIXME

**Note**

`createTopics` is used exclusively when `KafkaApis` handles a [CREATE\\_TOPICS request](#).

## Creating AdminManager Instance

`AdminManager` takes the following when created:

- [KafkaConfig](#)
- `Metrics`
- [MetadataCache](#)
- [ZkUtils](#)

`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 when...FIXME

## configSynonyms Internal Method

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

configSynonyms ...FIXME

Note	configSynonyms is used when...FIXME
------	-------------------------------------

# Authorizer

Authorizer is...FIXME

configure

Method

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

# Cluster

`Cluster` represents a subset of the nodes and topic partitions in a Kafka cluster.

## Note

`org.apache.kafka.common.Cluster` is a `public final class`.

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.

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 Method

```
static Cluster bootstrap(List<InetSocketAddress> addresses)
```

`bootstrap` ...FIXME

## Note

`bootstrap` is used when [KafkaAdminClient](#), [AdminClient](#), [KafkaConsumer](#) and [KafkaProducer](#) are created.

## isBootstrapConfigured Method

```
boolean isBootstrapConfigured()
```

`isBootstrapConfigured` gives `isBootstrapConfigured` internal flag.

## Note

`isBootstrapConfigured` 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"><li>• <code>Metadata</code> <code>getClusterForCurrentTopics</code></li><li>• <code>KafkaAdminClient</code> <code>describeTopics</code></li><li>• <code>KafkaConsumer</code> <code>partitionsFor</code></li><li>• <code>KafkaProducer</code> <code>requests partitions for a topic</code></li><li>• <code>DefaultPartitioner</code> <code>assigns the partition for a record</code></li></ul>
------	--

# Cluster (deprecated)

Important	<p>It <i>seems</i> that <code>cluster</code> class is created using <a href="#">ZkUtils.getCluster</a> that is used exclusively when <code>ZKRebalancerListener</code> does <a href="#">syncedReaderRebalance</a> (that in turn happens for the currently-deprecated <code>ZookeeperConsumerConnector</code> ).</p> <p>In other words, <code>cluster</code> class and the page are <i>soon</i> 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](#)

# DynamicBrokerConfig

DynamicBrokerConfig is...FIXME

DynamicBrokerConfig is created exclusively when KafkaConfig is created.

DynamicBrokerConfig` is used to create a KafkaConfig.

Note	DynamicBrokerConfig is created when KafkaConfig is, but the reverse is also true (!) Isn't it a dependency cycle? Not really since KafkaConfig will create a DynamicBrokerConfig 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
currentConfig	Current KafkaConfig Initialized with the input KafkaConfig. Used when...FIXME

Tip	Enable INFO , DEBUG or TRACE logging levels for kafka.server.DynamicBrokerConfig logger to see what happens inside. Add the following line to config/log4j.properties : <div>log4j.logger.kafka.server.DynamicBrokerConfig=TRACE</div> 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...FIXME

## updateCurrentConfig Internal Method

```
updateCurrentConfig(): Unit
```

updateCurrentConfig ...FIXME

Note

updateCurrentConfig is used when...FIXME

## currentDynamicDefaultConfigs Method

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

currentDynamicDefaultConfigs ...FIXME

Note

currentDynamicDefaultConfigs is used when...FIXME

## 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 <code>KafkaServer</code> is requested to <a href="#">start up</a> .
------	--

## updateDefaultConfig Method

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

`updateDefaultConfig` ...FIXME

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

## updateBrokerConfig Method

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

`updateBrokerConfig` ...FIXME

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

## maybeReEncodePasswords Internal Method

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

`maybeReEncodePasswords` ...FIXME

Note	<code>maybeReEncodePasswords</code> 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`.

<b>Note</b>	<code>addReconfigurables</code> is used when <code>KafkaServer</code> is requested to <a href="#">start up</a> .
-------------	--



# DynamicLogConfig

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

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

## Creating DynamicLogConfig Instance

`DynamicLogConfig` takes the following when created:

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

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

# DynamicMetricsReporters

DynamicMetricsReporters is...FIXME

Table 1. DynamicMetricsReporters’s Internal Properties (e.g. Registries, Counters and Flags)

Name	Description
dynamicConfig	<a href="#">DynamicBrokerConfig</a> Used when...FIXME

# DynamicClientQuotaCallback

DynamicClientQuotaCallback is...FIXME

reconfigure

Method

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

Note	reconfigure is part of the <a href="#">Reconfigurable Contract</a> to...FIXME.
------	--

reconfigure ...FIXME

# ConfigHandler Contract

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

```
package kafka.server

trait ConfigHandler {
  def processConfigChanges(entityName: String, value: Properties)
}
```

Table 1. ConfigHandler Contract

Property	Description
<code>processConfigChanges</code>	Used when <code>DynamicConfigManager</code> is requested to <a href="#">start up</a> , <a href="#">processEntityConfigChangeVersion1</a> , <a href="#">processEntityConfigChangeVersion2</a>

Table 2. ConfigHandlers

ConfigHandler	Description
<a href="#">BrokerConfigHandler</a>	
<a href="#">ClientIdConfigHandler</a>	
<a href="#">TopicConfigHandler</a>	
<a href="#">UserConfigHandler</a>	

# BrokerConfigHandler

BrokerConfigHandler is...FIXME

## processConfigChanges Method

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

Note	processConfigChanges is part of the <a href="#">ConfigHandler Contract</a> to...FIXME.
------	--

processConfigChanges ...FIXME

# ClientIdConfigHandler

ClientIdConfigHandler is...FIXME

processConfigChanges

Method

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

Note	processConfigChanges is part of the <a href="#">ConfigHandler Contract</a> 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 <a href="#">ConfigHandler Contract</a> to...FIXME.
------	--

processConfigChanges ...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 <a href="#">NotificationHandler Contract</a> to...FIXME.
------	---

processNotification ...FIXME

# DynamicConfigManager

DynamicConfigManager is...FIXME

DynamicConfigManager uses the [ConfigChangedNotificationHandler](#) to process config change notifications.

DynamicConfigManager is created exclusively when `KafkaServer` is requested to start up.

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

Name	Description
<code>adminZkClient</code>	<a href="#">AdminZkClient</a> (for the <a href="#">KafkaZkClient</a> )  Used when...FIXME

## startup Method

startup

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`

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 <a href="#">NotificationHandler Contract</a> 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	<code>processEntityConfigChangeVersion2</code> is used when...FIXME
------	---

# BrokerReconfigurable Contract — Reconfigurable Broker Services

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

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

```
package kafka.server

trait BrokerReconfigurable {
  def reconfigurableConfigs: Set[String]
  def reconfigure(oldConfig: KafkaConfig, newConfig: KafkaConfig): Unit
  def validateReconfiguration(newConfig: KafkaConfig): Unit
}
```

Table 1. BrokerReconfigurable Contract

Property	Description
<code>reconfigurableConfigs</code>	Used when <code>DynamicBrokerConfig</code> is requested to <a href="#">addBrokerReconfigurable</a> and <a href="#">processReconfiguration</a>
<code>reconfigure</code>	Used exclusively when <code>DynamicBrokerConfig</code> is requested to <a href="#">updateCurrentConfig</a>
<code>validateReconfiguration</code>	Used exclusively when <code>DynamicBrokerConfig</code> is requested to <a href="#">processReconfiguration</a>

Table 2. BrokerReconfigurables (Direct Implementations)

BrokerReconfigurable	Description
<a href="#">DynamicListenerConfig</a>	
<a href="#">DynamicThreadPool</a>	
<a href="#">LogCleaner</a>	

# DynamicThreadPool

DynamicThreadPool is...FIXME

reconfigure

Method

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

Note	reconfigure is part of the <a href="#">BrokerReconfigurable Contract</a> to...FIXME.
------	--

reconfigure ...FIXME

# DynamicListenerConfig

DynamicListenerConfig is...FIXME

reconfigure

Method

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

Note	reconfigure is part of the <a href="#">BrokerReconfigurable Contract</a> to...FIXME.
------	--

reconfigure ...FIXME

# LogCleaner

LogCleaner is...FIXME

## 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

cleanSegments is used when...FIXME

## buildOffsetMap Internal Method

```
buildOffsetMap(  
  log: Log,  
  start: Long,  
  end: Long,  
  map: OffsetMap,  
  stats: CleanerStats): Unit
```

buildOffsetMap ...FIXME

Note

buildOffsetMap is used when...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>	<a href="#">ConsumerNetworkClient</a> that is given when <code>Fetcher</code> is <a href="#">created</a> .

## 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 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	<p><code>getTopicMetadata</code> is used when:</p> <ul style="list-style-type: none"> <li><code>Fetcher</code> is requested to <a href="#">find metadata for all topics</a></li> <li><code>KafkaConsumer</code> is requested to <a href="#">find partitions for a topic</a>.</li> </ul>
------	---

## 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.

# GroupCoordinator

GroupCoordinator is...FIXME

GroupCoordinator is created when...FIXME

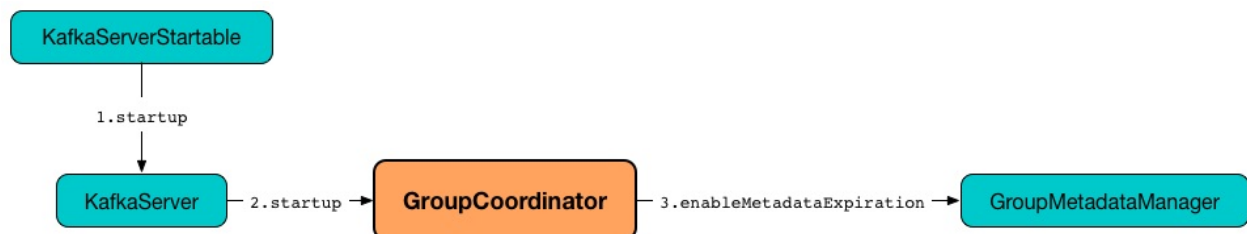


Figure 1. GroupCoordinator's Startup

logIdent is [GroupCoordinator [brokerId]].

GroupCoordinator uses isActive flag to...FIXME

## Creating GroupCoordinator Instance — apply Factory Method

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

## Creating GroupCoordinator Instance

GroupCoordinator takes the following when created:

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

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

## Starting Up (and GroupMetadataManager) — startup Method

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

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

```
INFO [GroupCoordinator [brokerId]]: Starting up. (kafka.coordinator.group.GroupCoordinator)
```

With `enableMetadataExpiration` input flag enabled, `startup` requests [GroupMetadataManager](#) to [enableMetadataExpiration](#).

`startup` turns [isActive](#) flag on.

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

```
INFO [GroupCoordinator [brokerId]]: Startup complete. (kafka.coordinator.group.GroupCoordinator)
```

Note	<code>startup</code> is used exclusively when <code>KafkaServer</code> <a href="#">starts up</a> .
------	--

# GroupMetadataManager

GroupMetadataManager is...FIXME

GroupMetadataManager is created exclusively when GroupCoordinator is created.

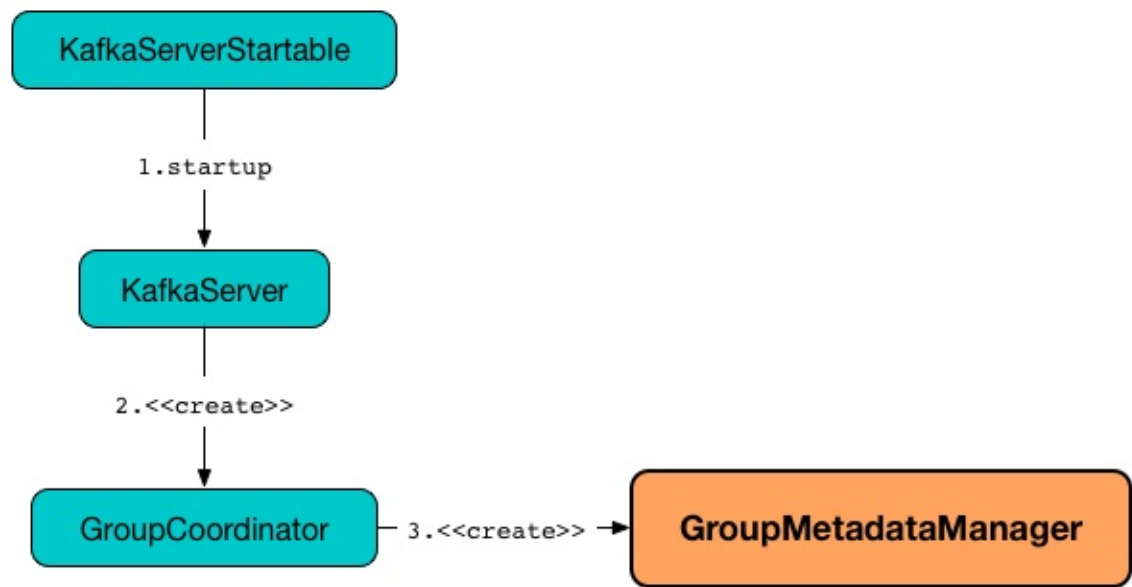


Figure 1. Creating GroupMetadataManager

Table 1. GroupMetadataManager’s Internal Properties (e.g. Registries and Counters)

Name	Description
groupMetadataCache	Pool of GroupMetadata by group ID
groupMetadataTopicPartitionCount	
scheduler	KafkaScheduler with 1 daemon thread with group-metadata-manager- prefix

## 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
- ZkUtils
- 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 [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 Topic — getGroupMetadataTopicPartitionCount Internal Method

```
getGroupMetadataTopicPartitionCount: Int
```



`getGroupMetadataTopicPartitionCount` requests `ZkUtils` for `getTopicPartitionCount` of `__consumer_offsets` topic.

If not available, `getGroupMetadataTopicPartitionCount` requests `OffsetConfig` for `offsetsTopicNumPartitions`.

**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 when...FIXME

## 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 when...FIXME
------	---

# InterBrokerSendThread

InterBrokerSendThread ...FIXME

doWork

Method

```
def doWork(): Unit
```

Note	doWork is a part of <a href="#">ShutdownableThread Contract</a> .
------	---

doWork ...FIXME

Note	doWork is used when...FIXME
------	-----------------------------

# 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 <a href="#">Runtime.addShutdownHook</a> 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 <a href="#">System.exit</a> 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` handles API requests (by means of `handlers`).

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

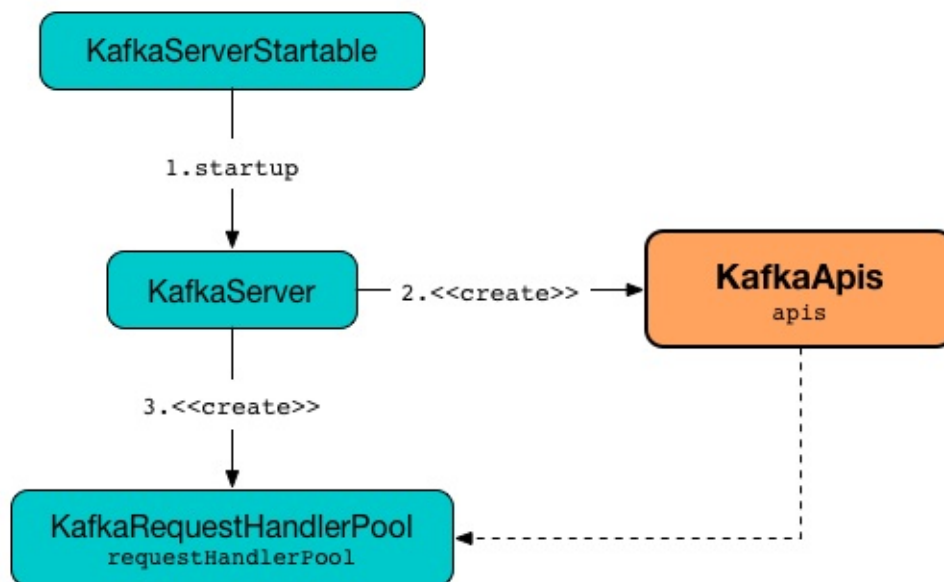


Figure 1. `KafkaApis` is Created for `KafkaRequestHandlerPool` when `KafkaServer` Starts Up

Table 1. KafkaApis's API Keys and Handlers (in alphabetical order)

Key	Handler
AlterReplicaLogDirs	<a href="#">handleLeaderAndIsrRequest</a>
ControlledShutdown	<a href="#">handleControlledShutdownRequest</a>
CreatePartitions	<a href="#">handleCreatePartitionsRequest</a>
CreateTopics	<a href="#">handleCreateTopicsRequest</a>
DeleteTopics	<a href="#">handleDeleteTopicsRequest</a>
DescribeLogDirs	<a href="#">handleDescribeLogDirsRequest</a>
Fetch	<a href="#">handleFetchRequest</a>
LeaderAndIsr	<a href="#">handleAlterReplicaLogDirsRequest</a>
Metadata	<a href="#">handleTopicMetadataRequest</a>
OffsetFetch	<a href="#">handleOffsetFetchRequest</a>

KafkaApis uses the [ReplicaManager](#) in the following handlers:

- [handleAlterReplicaLogDirsRequest](#)
- [handleDeleteRecordsRequest](#)
- [handleDescribeLogDirsRequest](#)
- [handleFetchRequest](#)
- [handleLeaderAndIsrRequest](#)
- [handleListOffsetRequest](#)
- [handleOffsetForLeaderEpochRequest](#)
- [handleProduceRequest](#)
- [handleStopReplicaRequest](#)
- [handleUpdateMetadataRequest](#)
- [handleWriteTxnMarkersRequest](#)



## Tip

Enable `INFO`, `DEBUG` or `TRACE` 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=TRACE
```

Refer to [Logging](#).

## Routing API Requests — `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 `request` to the corresponding [handler](#) per the [apiKey](#) (from the header of the input `request` ).

## Note

`handle` is used exclusively when `KafkaRequestHandler` thread is [started](#).

## Handling `LeaderAndIsr` Request — `handleLeaderAndIsrRequest` Handler

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

`handleLeaderAndIsrRequest` ...FIXME

## Note

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

## Handling `AlterReplicaLogDirs` Request — `handleAlterReplicaLogDirsRequest` Handler

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

`handleAlterReplicaLogDirsRequest` ...FIXME

**Note**

`handleAlterReplicaLogDirsRequest` is used exclusively to handle [AlterReplicaLogDirs](#) requests.

## 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 to handle [CreateTopics](#) requests.

## Handling OffsetFetch Request — `handleOffsetFetchRequest` Handler

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

`handleOffsetFetchRequest` ...FIXME

**Note**

`handleOffsetFetchRequest` is used exclusively to handle [OffsetFetch](#) requests.

## Handling Fetch Request — `handleFetchRequest` Handler

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

`handleFetchRequest` ...FIXME

**Note**

`handleFetchRequest` is used exclusively to handle [Fetch](#) requests.

## Handling Metadata Request — `handleTopicMetadataRequest` Method

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

`handleTopicMetadataRequest` takes the body (from the input `request` ) as `MetadataRequest` .

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

Note	<code>handleTopicMetadataRequest</code> is used exclusively to handle <a href="#">Metadata</a> requests.
------	--

## authorize Internal Method

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

`authorize` ...FIXME

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**

```
handleControlledShutdownRequest
```

 is used when...FIXME

## Creating KafkaApis Instance

`KafkaApis` takes the following when created:

- `RequestChannel`
- [ReplicaManager](#)
- [AdminManager](#)
- [GroupCoordinator](#)
- `TransactionCoordinator`
- [KafkaController](#)
- [ZkUtils](#)
- Broker ID
- [KafkaConfig](#)
- [MetadataCache](#)
- `Metrics`
- Optional [Authorizer](#)
- `QuotaManagers`
- `BrokerTopicStats`
- Cluster ID
- `Time`

## fetchOffsetForTimestamp Internal Method

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

fetchOffsetForTimestamp ...FIXME

Note

`fetchOffsetForTimestamp` is used exclusively when `KafkaApis` is requested to [handleListOffsetRequestV1AndAbove](#).

## handleListOffsetRequestV0 Internal Method

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

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
```

`handleDescribeLogDirsRequest` takes the `DescribeLogDirsRequest` (from the body of the input `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 to 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

<b>Note</b>	<code>fetchOffsets</code> is used exclusively when <code>KafkaApis</code> is requested to <a href="#">handleListOffsetRequestV0</a> .
-------------	---

# 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](#).

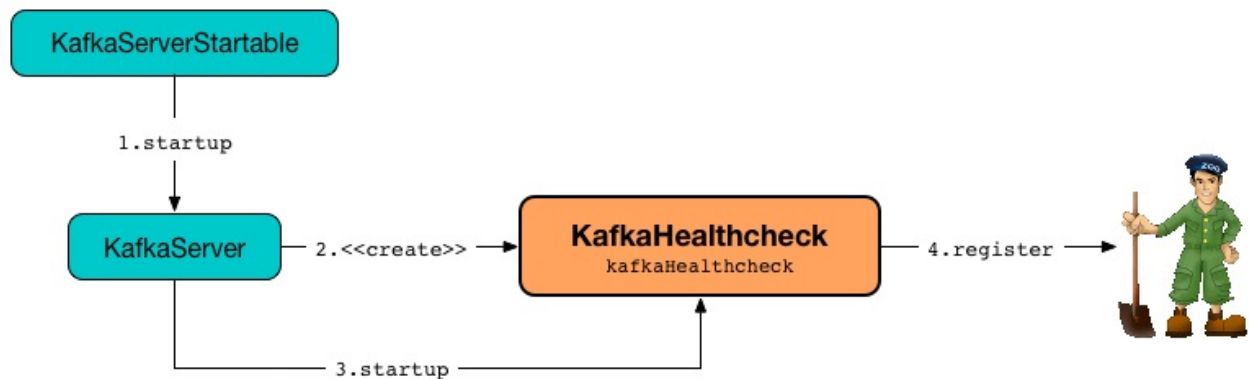


Figure 1. KafkaHealthcheck

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

Name	Description
<code>sessionExpireListener</code>	<code>SessionExpireListener</code>

## 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

```
register(): Unit
```

`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.

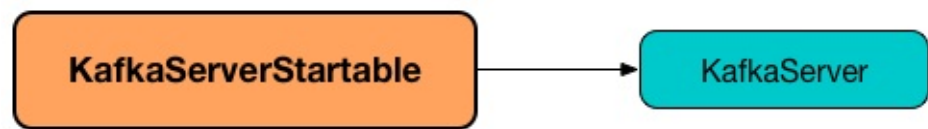


Figure 1. `KafkaServerStartable` manages `KafkaServer`

`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

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

## 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	fromProps is used when kafka.Kafka 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	startup uses Java’s <a href="#">System.exit</a> to terminate a JVM.
------	---

Note	startup is used when a <a href="#">Kafka Broker starts (on command line)</a> .
------	--

# KafkaServer

`KafkaServer` is a Kafka broker that wires (creates and starts) Kafka services together.

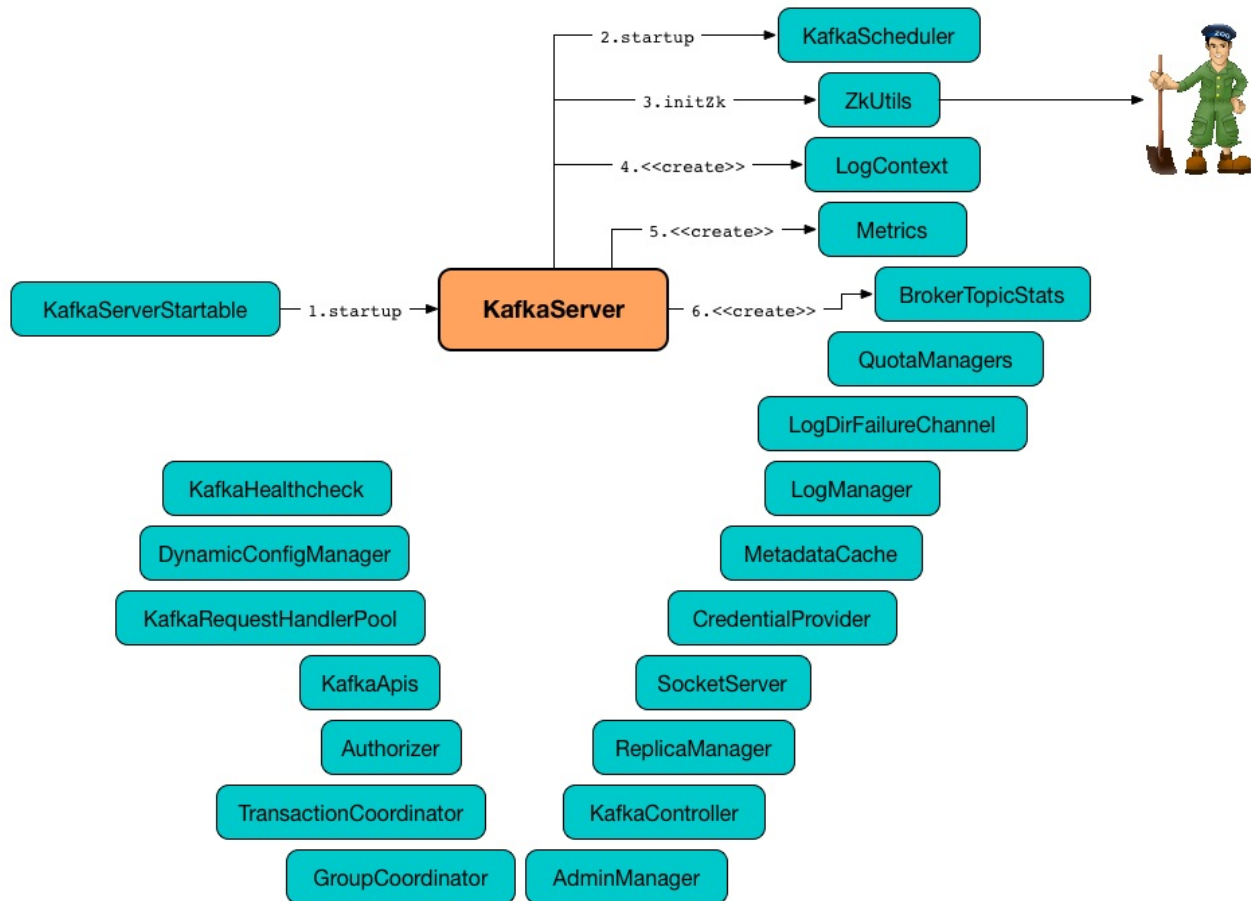


Figure 1. KafkaServer's Startup and Auxiliary Services

`KafkaServer` registers itself in the JMX system under **kafka.server**.

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

Name	Description
<code>adminManager</code>	<code>AdminManager</code>
<code>apis</code>	<code>KafkaApis</code>
<code>authorizer</code>	Optional <code>Authorizer</code>
<code>brokerMetadataCheckpoints</code>	
<code>brokerState</code>	<code>BrokerState</code>
<code>_brokerTopicStats</code>	<code>BrokerTopicStats</code>
<code>_clusterId</code>	Cluster ID

credentialProvider	CredentialProvider
dynamicConfigHandlers	
dynamicConfigManager	<a href="#">DynamicConfigManager</a>
groupCoordinator	<a href="#">GroupCoordinator</a>
isStartingUp	Flag for...FIXME
kafkaController	<a href="#">KafkaController</a>
kafkaHealthcheck	<a href="#">KafkaHealthcheck</a>
kafkaScheduler	<a href="#">KafkaScheduler</a> with the number of daemon threads as configured using <a href="#">background.threads</a> configuration property (default: <code>10</code> )
logContext	LogContext
logDirFailureChannel	<code>LogDirFailureChannel</code>
logManager	<p><a href="#">LogManager</a></p> <p>Used when:</p> <ul style="list-style-type: none"> <li>• <a href="#">checkpointBrokerId</a> (when <code>KafkaServer</code> is requested to <a href="#">start up</a>)</li> <li>• <code>DynamicBrokerConfig</code> is requested to <a href="#">addReconfigurables</a></li> <li>• <code>KafkaServer</code> is requested to <a href="#">start up</a> (and creates a <code>TopicConfigHandler</code> for topics for <a href="#">dynamicConfigHandlers</a> and a <a href="#">ReplicaManager</a>)</li> </ul> <hr/> <p><a href="#">Created</a> and immediately <a href="#">started</a> when <code>KafkaServer</code> is requested to <a href="#">start up</a>.</p> <p><a href="#">Shut down</a> when <code>KafkaServer</code> is requested to <a href="#">shut down</a>.</p>
metadataCache	<a href="#">MetadataCache</a>
	<p><a href="#">ReplicaManager</a> used to create:</p> <ul style="list-style-type: none"> <li>• <a href="#">KafkaApis</a></li> <li>• <a href="#">GroupCoordinator</a></li> </ul>

replicaManager	<ul style="list-style-type: none"> <li>• <a href="#">TransactionCoordinator</a></li> </ul> <hr/> <ul style="list-style-type: none"> <li>• <a href="#">Created</a> (and <a href="#">started</a> immediately) when <code>KafkaServer</code> is requested to <a href="#">start up</a></li> <li>• <a href="#">Shut down</a> when <code>KafkaServer</code> <a href="#">shuts down</a></li> </ul>
reporters	Collection of <a href="#">MetricsReporter</a> Used when...FIXME
requestHandlerPool	<a href="#">KafkaRequestHandlerPool</a>
socketServer	<a href="#">SocketServer</a>
transactionCoordinator	<a href="#">TransactionCoordinator</a>
quotaManagers	<code>QuotaManagers</code>
shutdownLatch	Java's <a href="#">java.util.concurrent.CountDownLatch</a>
startupComplete	Flag for...FIXME
zkUtils	<a href="#">ZkUtils</a>

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

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

`getBrokerIdAndOfflineDirs` ...FIXME

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

## `getOrCreateClusterId` Internal Method

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

## Connecting to Zookeeper — `initZk` Internal Method

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

## checkpointBrokerId Internal Method

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

checkpointBrokerId ...FIXME

### Note

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

## 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

KafkaServer is created when [KafkaServerStartable](#) is created.

## Starting Up — startup Method

```
startup(): Unit
```

startup starts a single Kafka server.

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

```
INFO starting (kafka.server.KafkaServer)
```

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:

```
INFO Cluster ID = [clusterId] (kafka.server.KafkaServer)
```

startup gets broker id and initial offline directories.

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

startup creates and configures metrics.

1. Requests `KafkaConfig` for configured instances of metric reporters
2. Adds a `JmxReporter` (with **kafka.server** prefix)
3. Creates the `MetricConfig`
4. Initializes `Metrics` internal registry

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 <b>log ident</b> with the value of <code>broker.id</code> property and is always in the format <code>[Kafka Server [brokerId]]</code> , after a Kafka server has fully started.
------	---

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

## 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 exclusively when `KafkaServer` starts up (with `clusterListeners` as `kafkaMetricsReporters` and the `MetricsReporter` reporters from `metric.reporters` Kafka property).

## 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.

## shutdown Method

```
shutdown(): Unit
```

`shutdown` ...FIXME

**Note**

`shutdown` is used when...FIXME

# KafkaConfig

`KafkaConfig` is the configuration of a Kafka server 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
<code>backgroundThreads</code>	<a href="#">background.threads</a>
<code>brokerId</code>	<a href="#">broker.id</a>
<code>deleteTopicEnable</code>	<a href="#">delete.topic.enable</a>
<code>hostName</code>	<a href="#">host.name</a>
<code>listeners</code>	<a href="#">listeners</a> (see <a href="#">getListeners</a> )
<code>logDirs</code>	<p><a href="#">log.dirs</a> or <a href="#">log.dir</a></p> <p>At least one log directory must be defined via <a href="#">log.dirs</a> or <a href="#">log.dir</a> properties.</p> <p>Used when:</p> <ul style="list-style-type: none"> <li>• <code>LogManager</code> is <a href="#">created</a></li> <li>• <code>LogDirFailureChannel</code> is created (when <code>KafkaServer</code> is requested to <a href="#">start up</a>)</li> <li>• <code>KafkaConfig</code> is requested to <a href="#">getNumReplicaAlterLogDirsThreads</a></li> <li>• <code>KafkaServer</code> is <a href="#">created</a> and is requested to <a href="#">getBrokerIdAndOfflineDirs</a> and <a href="#">checkpointBrokerId</a></li> <li>• <code>ReplicaManager</code> is requested to <a href="#">describeLogDirs</a></li> </ul>
<code>numNetworkThreads</code>	<a href="#">num.network.threads</a>
<code>port</code>	<a href="#">port</a>
<code>replicaLagTimeMaxMs</code>	

Table 2. KafkaConfig's Internal Properties (e.g. Registries, Counters and Flags)

Name	Description
<code>dynamicConfig</code>	<p><code>DynamicBrokerConfig</code> (that could be <a href="#">provided</a> or is going to be <a href="#">created from scratch</a>)</p> <p>Used when:</p> <ul style="list-style-type: none"> <li><code>AdminManager</code> is requested to <a href="#">describeConfigs</a> (for <code>BROKER</code> resources), <a href="#">alterConfigs</a> and <a href="#">configSynonyms</a></li> <li><code>BrokerConfigHandler</code> is requested to <a href="#">processConfigChanges</a></li> <li><code>DynamicMetricsReporters</code> is <a href="#">created</a></li> <li><code>DynamicClientQuotaCallback</code> is <a href="#">reconfigure</a></li> <li><code>KafkaConfig</code> is requested to <a href="#">addReconfigurable</a></li> <li><code>KafkaServer</code> is requested to <a href="#">start up</a></li> </ul>

## 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

`fromProps` is used when:

- `KafkaServerStartable` is requested to [create a KafkaServerStartable from Java Properties](#)
- `SimpleAclAuthorizer` is requested to `configure` 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

`apply` seems to be used in tests only.

## `addReconfigurable` Method

```
addReconfigurable(reconfigurable: Reconfigurable): Unit
```

`addReconfigurable` ...FIXME

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

# KafkaController

`KafkaController` is a Kafka service responsible for:

- [topic deletion](#)
- ...FIXME

`KafkaController` uses [listeners](#) as a notification system to monitor znodes in Zookeeper and react accordingly.

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` [starts up](#).

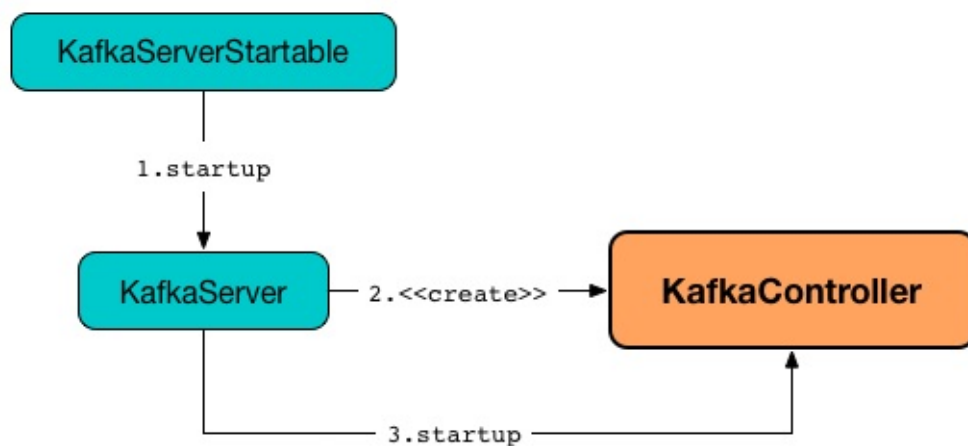


Figure 1. KafkaController

`KafkaController` is part of every Kafka broker, but only one `KafkaController` is [active](#) at all times.

`KafkaController` emulates a state machine using [controller events](#).

Table 1. KafkaController's Controller Events

Event	ControllerState	process Handler
BrokerChange	BrokerChange	
ControllerChange <ul style="list-style-type: none"><li>newControllerId : Int</li></ul>	ControllerChange	<div><div>1. Assigns the <code>current controller</code> as the input <code>newControllerId</code></div><div>2. (only when the broker is no longer an <code>active controller</code>) <code>Resigns as the active controller</code></div></div> <div><div>Note</div><div>Similar to <code>Reelect</code> with the only difference that it does <b>not</b> trigger an <code>election</code></div></div>
ControlledShutdown <ul style="list-style-type: none"><li>ID</li><li>controlledShutdownCallback : Try[Set[TopicAndPartition]] ⇒ Unit</li></ul>	ControlledShutdown	
Reelect	ControllerChange	<div><div>1. Assigns the <code>current controller</code> as <code>activeControllerId</code></div><div>2. (only when the broker is no longer an <code>active controller</code>) <code>Resigns as the active controller</code></div><div>3. <code>elect</code></div></div>
Startup	ControllerChange	<div><div>1. <code>registerSessionExpiration</code></div><div>2. <code>registerControllerChange</code></div><div>3. <code>elect</code></div></div>

`logIdent` is **[Controller id=[brokerId]]**.



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

Name	Description
<code>activeControllerId</code>	The ID of the active <code>KafkaController</code> <ul style="list-style-type: none"> <li>Initialized to <code>-1</code></li> </ul>
<code>controllerContext</code>	
<code>eventManager</code>	<code>ControllerEventManager</code> for <code>controllerContext.stats.rateAndTimeMetrics</code> and <code>updateMetrics</code> listener
<code>kafkaScheduler</code>	<code>KafkaScheduler</code> with 1 daemon thread with <b>kafka-scheduler</b> 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 <b>delegation-token-cleaner</b> prefix
<code>topicDeletionManager</code>	<code>TopicDeletionManager</code>

Table 3. KafkaController's Listeners

Listener	Description
<code>brokerChangeListener</code>	<code>BrokerChangeListener</code> for this <code>KafkaController</code> and <code>EventManager</code>
<code>isrChangeNotificationListener</code>	<p><code>IsrChangeNotificationListener</code> for this <code>KafkaController</code> and <code>EventManager</code></p> <p>Registered in <code>registerIsrChangeNotificationListener</code> when <code>KafkaController</code> does <code>onControllerFailover</code>.</p> <p>De-registered in <code>deregisterIsrChangeNotificationListener</code> when <code>KafkaController</code> resigns as the active controller.</p>
<code>logDirEventNotificationListener</code>	<code>LogDirEventNotificationListener</code>
<code>partitionModificationsListeners</code>	<code>PartitionModificationsListener</code> by name
<code>partitionReassignmentListener</code>	<code>PartitionReassignmentListener</code> for this <code>KafkaController</code> and <code>ControllerEventManager</code>
<code>preferredReplicaElectionListener</code>	<code>PreferredReplicaElectionListener</code> for this <code>KafkaController</code> and <code>ControllerEventManager</code>
<code>topicDeletionListener</code>	<p><code>TopicDeletionListener</code> (for this <code>KafkaController</code> and <code>ControllerEventManager</code>)</p> <p>Registered in <code>registerTopicDeletionListener</code> when <code>KafkaController</code> does <code>onControllerFailover</code>.</p> <p>De-registered in <code>deregisterTopicDeletionListener</code> when <code>KafkaController</code> resigns as the active controller.</p>

Tip	<p>Enable <code>WARN</code>, <code>INFO</code> or <code>DEBUG</code> logging levels for <code>kafka.controller.KafkaController</code> logger to see what happens inside.</p> <p>Add the following line to <code>config/log4j.properties</code> :</p> <pre>log4j.logger.kafka.controller.KafkaController=DEBUG</pre> <p>Refer to <a href="#">Logging</a>.</p>
-----	--

## initiateReassignReplicasForTopicPartition Method

```
initiateReassignReplicasForTopicPartition
```

```
initiateReassignReplicasForTopicPartition ...FIXME
```

**Note**

`initiateReassignReplicasForTopicPartition` is used when...FIXME

## deregisterPartitionReassignmentIsrChangeListener Method

```
deregisterPartitionReassignmentIsrChangeListeners
```

```
deregisterPartitionReassignmentIsrChangeListeners ...FIXME
```

**Note**

`deregisterPartitionReassignmentIsrChangeListeners` is used when...FIXME

## resetControllerContext Method

```
resetControllerContext
```

```
resetControllerContext ...FIXME
```

**Note**

`resetControllerContext` is used when...FIXME

## deregisterBrokerChangeListener Method

```
deregisterBrokerChangeListener
```

```
deregisterBrokerChangeListener ...FIXME
```

**Note**

`deregisterBrokerChangeListener` is used when...FIXME

## deregisterTopicChangeListener Method

```
deregisterTopicChangeListener
```

```
deregisterTopicChangeListener ...FIXME
```

**Note**

`deregisterTopicChangeListener` 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](#)
- [preferredReplicaImbalanceCount](#)
- [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

`onControllerResignation` is used when:

1. `ControllerEventThread` [processes ControllerChange](#) and [Reelect](#) controller events
2. [triggerControllerMove](#)
3. `KafkaController` [shuts 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 a [IllegalStateException](#) in [updateLeaderEpochAndSendRequest](#), [sendUpdateMetadataRequest](#) and [ControlledShutdown](#) event.

## sendUpdateMetadataRequest Method

```
sendUpdateMetadataRequest(): Unit
```

sendUpdateMetadataRequest ...FIXME

### Note

sendUpdateMetadataRequest is used when:

- KafkaController is requested to [onControllerFailover](#), [onBrokerStartup](#), [onBrokerUpdate](#), [onReplicasBecomeOffline](#), [onPartitionReassignment](#), [processUpdateNotifications](#)
- TopicDeletionManager is requested to [onTopicDeletion](#)

## updateLeaderEpochAndSendRequest Internal Method

```
updateLeaderEpochAndSendRequest(): Unit
```

updateLeaderEpochAndSendRequest ...FIXME

Note

updateLeaderEpochAndSendRequest is used when...FIXME

## shutdown Method

```
shutdown(): Unit
```

shutdown ...FIXME

Note

shutdown is used when...FIXME

## updateMetrics Internal Method

Caution

FIXME

## onBrokerStartup Method

```
onBrokerStartup(newBrokers: Seq[Int]): Unit
```

onBrokerStartup ...FIXME

Note

onBrokerStartup is used exclusively when KafkaController processes BrokerChange controller event.

## elect Method

```
elect(): Unit
```

elect ...FIXME

Note

elect is used when KafkaController enters Startup and Reelect states.

## onControllerFailover Method

Caution

FIXME

Note

onControllerFailover is used exclusively when KafkaController is requested to [elect](#).



## isActive Method

```
isActive: Boolean
```

`isActive` says whether the `activeControllerId` equals the broker ID (from `KafkaConfig`).

Caution	FIXME When could they be different?
---------	-------------------------------------

## 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`
- `ZkUtils`
- `Time`
- `Metrics`
- Optional thread name prefix

`KafkaController` initializes the `internal registries and counters`.

## Starting ControllerEventManager (and Putting Startup Event in Event Queue) — startup Method

```
startup(): Unit
```

startup puts startup event at the end of the event queue of [ControllerEventManager](#) and requests it to [start](#).

Note	startup is used exclusively when <a href="#">KafkaServer</a> is <a href="#">started up</a> .
------	--

## 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	<a href="#">SessionExpirationListener</a> puts <a href="#">Reelect</a> event on the <a href="#">event queue</a> of <a href="#">ControllerEventManager</a> every time the Zookeeper session has expired and a new session has been created.
------	--

Note	registerSessionExpirationListener is used exclusively when <a href="#">Startup</a> event is processed (after <a href="#">ControllerEventThread</a> is <a href="#">started</a> ).
------	--

## 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	<p><code>ControllerChangeListener</code> emits:</p> <ol style="list-style-type: none"> <li>1. <code>ControllerChange</code> event with the current controller ID (on the <code>event queue</code> of <code>ControllerEventManager</code> ) every time the data of a znode changes</li> <li>2. <code>Reelect</code> event when the data associated with a znode has been deleted</li> </ol>
Note	<p><code>registerControllerChangeListener</code> is used exclusively when <code>Startup</code> event is processed (after <code>ControllerEventThread</code> is <code>started</code>).</p>

## registerBrokerChangeListener Internal Method

```
registerBrokerChangeListener(): Option[Seq[String]]
```

`registerBrokerChangeListener` requests `ZkUtils` to `subscribeChildChanges` for `/brokers/ids` path with `BrokerChangeListener`.

Note	<p><code>registerBrokerChangeListener</code> is used exclusively when <code>KafkaController</code> does <code>onControllerFailover</code>.</p>
------	--

## 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 0.0.0.0:2181
Connecting to 0.0.0.0:2181
Welcome to ZooKeeper!
...
get /controller
{"version":1,"brokerid":100,"timestamp":"1506197069724"}
cZxid = 0xf9
ctime = Sat Sep 23 22:04:29 CEST 2017
mZxid = 0xf9
mtime = Sat Sep 23 22:04:29 CEST 2017
pZxid = 0xf9
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x15eaa3a4fdd000d
dataLength = 56
numChildren = 0
```

Otherwise, when no `/controller` znode is available, `getControllerID` returns `-1`.

Note	<p><code>getControllerID</code> is used when:</p> <ol style="list-style-type: none"> <li>1. Processing <code>Reelect</code> controller event</li> <li>2. <code>elect</code></li> </ol>
------	--

## 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	<p><code>registerTopicDeletionListener</code> is used exclusively when <code>KafkaController</code> does <code>onControllerFailover</code>.</p>
------	---

## 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.

## processUpdateNotifications Internal Method

```
processUpdateNotifications(partitions: Seq[TopicPartition]): Unit
```

`processUpdateNotifications` ...FIXME

**Note**

`processUpdateNotifications` is used when...FIXME

## 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...FIXME

## onBrokerUpdate Internal Method

```
onBrokerUpdate(updatedBrokerId: Int): Unit
```

`onBrokerUpdate` ...FIXME

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

**scheduleAutoLeaderRebalanceTask**

**Internal Method**

```
scheduleAutoLeaderRebalanceTask(delay: Long, unit: TimeUnit): Unit
```

`scheduleAutoLeaderRebalanceTask` ...FIXME

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

# ControllerEventManager

`ControllerEventManager` is...FIXME

`ControllerEventManager` is **created** when `KafkaController` is **created**.

`ControllerEventManager` is **started** when `KafkaController` is **started up**.

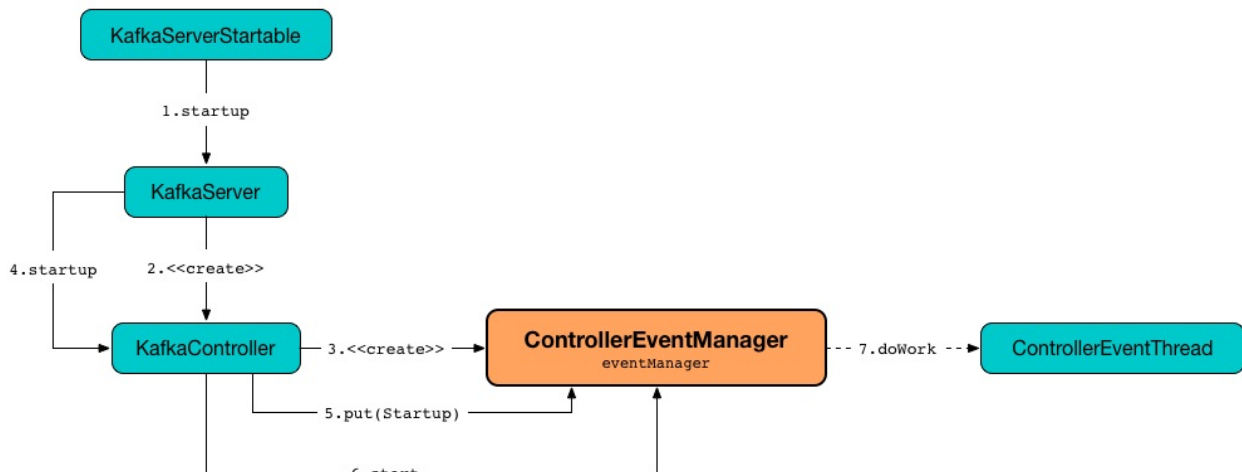


Figure 1. `ControllerEventManager` is Created and Started Alongside `KafkaController`  
Table 1. `ControllerEventManager`'s Internal Properties (e.g. Registries and Counters)

Name	Description
<code>queue</code>	Java's <a href="#">java.util.concurrent.LinkedBlockingQueue</a> (i.e. an optionally-bounded blocking queue based on linked nodes that orders elements in first-in-first-out fashion) of <a href="#">ControllerEvents</a> .
<code>_state</code>	<code>ControllerState</code> with <code>Idle</code> being the initial state
<code>thread</code>	<a href="#">ControllerEventThread</a> with <b>controller-event-thread</b> thread name

## Creating ControllerEventManager Instance

`ControllerEventManager` takes the following when created:

- `rateAndTimeMetrics` collection of `ControllerState` and `KafkaTimer`
- `eventProcessedListener` Procedure of `ControllerEvent`

`ControllerEventManager` initializes the **internal registries and counters**.

## Inserting Controller Event to Event Queue — `put` Method

```
put(event: ControllerEvent): Unit
```

`put` inserts `event` at the tail of [event queue](#).

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

## Starting ControllerEventManager (and ControllerEventThread) — `start` Method

```
start(): Unit
```

`start` requests [ControllerEventThread](#) to [do the work](#).

Note	<code>ControllerEventThread</code> is a <code>ShutdownableThread</code> that, once started, triggers <code>doWork()</code> method.
------	--

Note	<code>start</code> is used exclusively when <code>KafkaController</code> is <a href="#">started up</a> .
------	--



# ControllerEventThread

`ControllerEventThread` is a `ShutdownableThread` that is started when `ControllerEventManager` is started

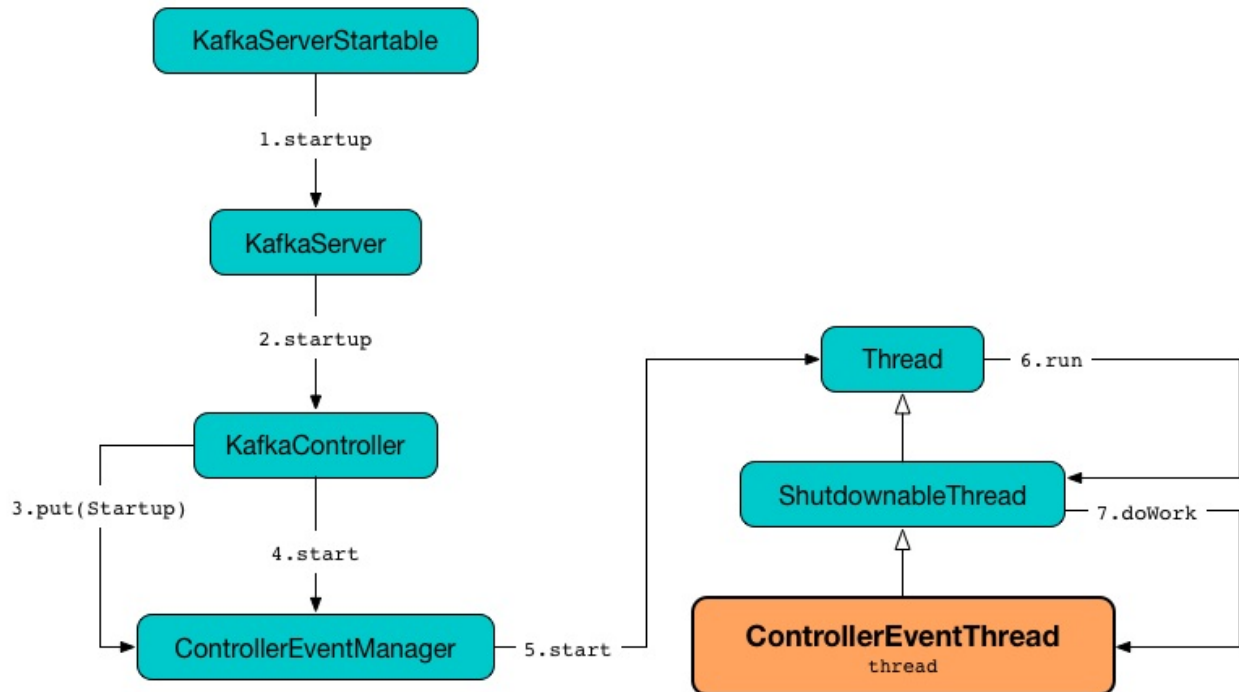


Figure 1. `ControllerEventThread` is Started Alongside `ControllerEventManager`  
When created, `ControllerEventThread` takes the name of the thread (which `ControllerEventManager` sets as **controller-event-thread**).

```

"controller-event-thread" #44 prio=5 os_prio=31 tid=0x00007fac45730800 nid=0xad03 wait
ing on condition [0x00000000178b30000]
  java.lang.Thread.State: WAITING (parking)
    at sun.misc.Unsafe.park(Native Method)
      - parking to wait for <0x000000007bcb03938> (a java.util.concurrent.locks.Abst
ractQueuedSynchronizer$ConditionObject)
    at java.util.concurrent.locks.LockSupport.park(LockSupport.java:175)
    at java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.await
(AbstractQueuedSynchronizer.java:2039)
    at java.util.concurrent.LinkedBlockingQueue.take(LinkedBlockingQueue.java:442)
    at kafka.controller.ControllerEventManager$ControllerEventThread.doWork(Contro
llerEventManager.scala:48)
    at kafka.utils.ShutdownableThread.run(ShutdownableThread.scala:64)
  
```

## Processing ControllerEvents — `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 events in the lifecycle of `KafkaController` state machine that, once [emitted](#), triggers [state](#) change and the corresponding [process](#) action.

```
package kafka.controller

sealed trait ControllerEvent {
  def state: ControllerState
  def process(): Unit
}
```

Note	<code>ControllerEvent</code> is a Scala sealed trait and so all the available events are in a single compilation unit (i.e. a file).
------	--

Table 1. ControllerEvent Contract

Method	Description
<code>state</code>	<code>ControllerState</code> of the <a href="#">ControllerEventManager</a> Used when <code>ControllerEventThread</code> <a href="#">does the work</a>
<code>process</code>	Used when <code>ControllerEventThread</code> <a href="#">does the work</a> to trigger an action associated with <a href="#">state</a> change.

Table 2. Known ControllerEvents

ControllerEvent	ControllerState
<a href="#">TopicDeletion</a>	<code>TopicDeletion</code>

# TopicDeletion Controller Event

`TopicDeletion` is a [ControllerEvent](#) that is [executed](#) on the active `KafkaController` (and does nothing otherwise).

Note	<code>TopicDeletion</code> uses <a href="#">delete.topic.enable</a> Kafka property.
------	---

Note	Topics to be deleted are created in <code>/admin/delete_topics</code> path in Zookeeper.
------	--

`state` is `TopicDeletion` .

## process Method

```
process(): Unit
```

Note	<code>process</code> is a part of <a href="#">ControllerEvent Contract</a> .
------	--

Note	<code>process</code> is executed on the <a href="#">active</a> controller only (and does nothing otherwise).
------	--

`process` prints out the following DEBUG message to the logs:

```
Delete topics listener fired for topics [topicsToBeDeleted] 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 [topicsToBeDeleted]
```

`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).

# ControllerBrokerRequestBatch

ControllerBrokerRequestBatch is...FIXME

## sendRequestsToBrokers Method

```
sendRequestsToBrokers(controllerEpoch: Int): Unit
```

sendRequestsToBrokers ...FIXME

### Note

sendRequestsToBrokers is used when:

- `KafkaController` is requested to [updateLeaderEpochAndSendRequest](#), [sendUpdateMetadataRequest](#) and `ControlledShutdown` is processed.
- `PartitionStateMachine` is requested to [handleStateChanges](#) and [triggerOnlinePartitionStateChange](#)
- `ReplicaStateMachine` is requested to [handleStateChanges](#)

# KafkaMetricsReporter

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

# KafkaRequestHandler

`KafkaRequestHandler` is a thread of execution (i.e. Java’s `Runnable`) that is responsible for relaying client requests (from `RequestChannel`) to `KafkaApis` (except `ShutdownRequest` requests that are handled directly).

`KafkaRequestHandler` is created exclusively when `KafkaRequestHandlerPool` is created (and starts the internal `runnables` threads).

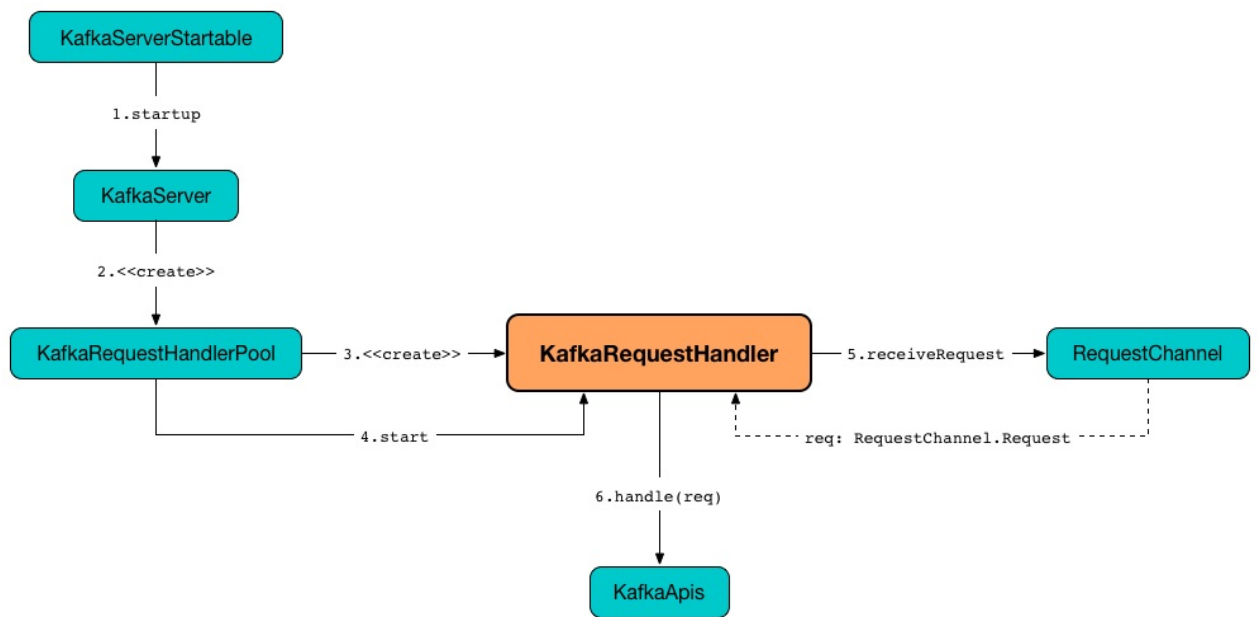


Figure 1. KafkaRequestHandler’s Startup and Request Relay

`logIdent` is **[Kafka Request Handler [id] on Broker [brokerId]]**.

Tip

Enable `DEBUG` or `TRACE` logging levels for `kafka.server.KafkaRequestHandler` logger to see what happens inside.

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

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

Refer to [Logging](#).

## Starting Thread — `run` Method

```
run(): Unit
```

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



**Note**

`run` is used when `KafkaRequestHandlerPool` is [created](#).

## Creating KafkaRequestHandler Instance

`KafkaRequestHandler` takes the following when created:

- ID
- Broker ID
- Aggregate Idle `Meter`
- Total number of handler threads
- `RequestChannel`
- [KafkaApis](#)
- `Time`

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

# KafkaRequestHandlerPool — Pool of Daemon KafkaRequestHandler Threads

KafkaRequestHandlerPool is a pool of daemon **kafka-request-handler** threads that are started immediately when KafkaRequestHandlerPool is created.

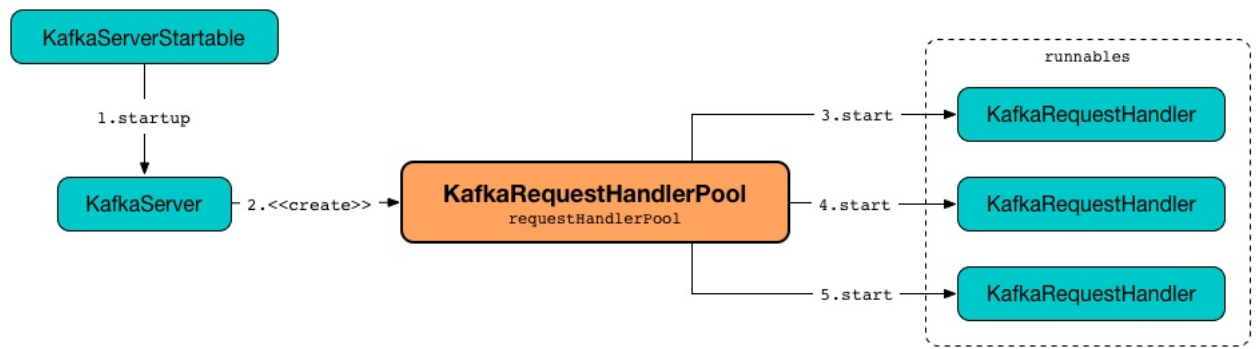


Figure 1. KafkaRequestHandlerPool and KafkaRequestHandler Threads

Note	The number of <b>kafka-request-handler</b> threads is controlled by <code>num.network.threads</code> Kafka property and defaults to <code>3</code> .
------	--

KafkaRequestHandlerPool is created exclusively when KafkaServer is started.

logIdent is **[Kafka Request Handler on Broker [brokerId]]**.

Table 1. KafkaRequestHandlerPool’s Internal Properties (e.g. Registries and Counters)

Name	Description
aggregateIdleMeter	
runnables	Collection of <code>KafkaRequestHandler</code>

shutdown

Method

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

## Creating KafkaRequestHandlerPool Instance

KafkaRequestHandlerPool takes the following when created:

- Broker ID
- `RequestChannel`
- `KafkaApis`

- `Time`
- Number of threads (i.e. instances of `KafkaRequestHandlers` as defined by `numNetworkThreads` property)

`KafkaRequestHandlerPool` initializes the `internal registries and counters` and starts `numThreads` daemon **kafka-request-handler** threads (as registered in `runnables`).

# 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 <a href="#">ScheduledThreadPoolExecutor</a> that <a href="#">schedules tasks</a> .  Initialized when <code>KafkaScheduler</code> <a href="#">starts up</a> and shut down when <code>KafkaScheduler</code> <a href="#">shuts down</a> .
<code>schedulerThreadId</code>	Java's <a href="#">AtomicInteger</a> with initial value 0.

Tip

Enable `INFO` , `DEBUG` or `TRACE` logging level for `kafka.utils.KafkaScheduler` logger to see what happens in `KafkaScheduler` .

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

`log4j.logger.kafka.utils.KafkaScheduler=TRACE`

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 <a href="#">reconfigure</a> .
------	---

# Partition Log

`Log` is a **partition log** that...FIXME

`Log` is **created** when...FIXME

## Creating Log Instance

`Log` takes the following when created:

- `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	apply is used when LogManager is requested to <a href="#">loadLog</a> and <a href="#">getOrCreateLog</a> .
------	--

## roll Method

```
roll(expectedNextOffset: Long = 0): LogSegment
```

roll ...FIXME

Note	roll is used when...FIXME
------	---------------------------

## asyncDeleteSegment Method

```
asyncDeleteSegment(segment: LogSegment): Unit
```

asyncDeleteSegment ...FIXME

Note	asyncDeleteSegment is used when...FIXME
------	---

## 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
------	--------------------------------

# LogCleanerManager

LogCleanerManager is...FIXME

## isCompactAndDelete Predicate

```
isCompactAndDelete(log: Log): Boolean
```

isCompactAndDelete ...FIXME

Note	isCompactAndDelete 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
------	---------------------------------------

# LogDirFailureHandler

LogDirFailureHandler is...FIXME

start

Method

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

# LogManager

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

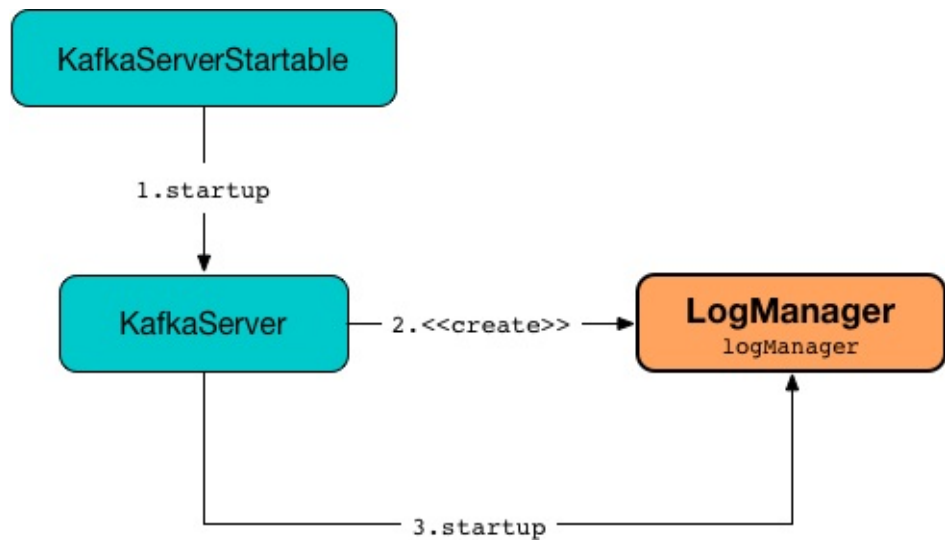


Figure 1. LogManager and KafkaServer

LogManager manages the log directories that are configured using log.dirs or log.dir configuration properties (default: /tmp/kafka-logs).

LogManager is used to create a ReplicaManager, a DynamicLogConfig, a TopicConfigHandler.

LogManager defaults to 30000ms for the initial task delay.

Table 1. LogManager’s Internal Properties (e.g. Registries, Counters and Flags)

Name	Description
_liveLogDirs	Java’s ConcurrentLinkedQueue of log directories Used when...FIXME

Tip

Enable INFO or DEBUG 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=DEBUG

Refer to Logging.

## 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, 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...FIXME

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

## cleanupLogs Method

```
cleanupLogs(): Unit
```

`cleanupLogs` ...FIXME

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

## Getting All Partition Logs — allLogs Method

```
allLogs: Iterable[Log]
```

`allLogs` ...FIXME

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

## addLogToBeDeleted 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...FIXME

## `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](#).

## `loadLog` Internal Method

```
loadLog(  
  logDir: File,  
  recoveryPoints: Map[TopicPartition, Long],  
  logStartOffsets: Map[TopicPartition, Long]): Unit
```

`loadLog` ...FIXME

Note

`loadLog` is used exclusively when `LogManager` is requested to [loadLogs](#).

## Loading Logs — `loadLogs` Internal Method

```
loadLogs(): Unit
```

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

Loading logs.

For every [live log directory](#), `loadLogs` ...FIXME

**Note**

`loadLogs` is used exclusively when `LogManager` is [created](#).

## Creating LogManager Instance

`LogManager` takes the following when created:

- Log directories
- Initial offline directories
- Topic configurations ( `Map[String, LogConfig]` )
- Initial default configuration ( `LogConfig` )
- `CleanerConfig`
- `recoveryThreadsPerDataDir`
- `flushCheckMs`
- `flushRecoveryOffsetCheckpointMs`
- `flushStartOffsetCheckpointMs`
- `retentionCheckMs`
- `maxPidExpirationMs`
- [Scheduler](#)
- `BrokerState`
- [BrokerTopicStats](#)
- `LogDirFailureChannel`
- `Time`

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

While being created, `LogManager` [load logs](#).

## 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 <a href="#">start up</a> .
------	---

## 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
------	--

# ProducerStateManager

ProducerStateManager is...FIXME

removeExpiredProducers

Method

```
removeExpiredProducers(currentTimeMs: Long): Unit
```

removeExpiredProducers ...FIXME

Note	removeExpiredProducers is used when...FIXME
------	---

# Metadata

Metadata describes a Kafka cluster...FIXME

Metadata is created right when `KafkaConsumer`, `KafkaProducer`, `KafkaAdminClient` and `AdminClient` are created.

Table 1. Metadata's Properties When Created by Clients

Client	refreshBackoffMs	metadataExpireMs	allowAutoTopicCr
KafkaConsumer	<code>retry.backoff.ms</code>	<code>metadata.max.age.ms</code>	enabled
KafkaProducer	<code>retry.backoff.ms</code>	<code>metadata.max.age.ms</code>	enabled
KafkaAdminClient	<code>retry.backoff.ms</code>	<code>metadata.max.age.ms</code>	
AdminClient			

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)

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

Name	Description
cluster	<p>Cluster with a subset of the nodes and topic partitions in a Kafka cluster.</p> <ul style="list-style-type: none"> <li>• Empty (with no nodes and no topic partitions) when Metadata is created</li> <li>• Updated when: <ul style="list-style-type: none"> <li>◦ <code>DefaultMetadataUpdater</code> handles <code>MetadataResponse</code></li> <li>◦ <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</li> </ul> </li> <li>• Can be accessed using <code>fetch</code></li> </ul>
listeners	

<code>lastRefreshMs</code>	<p>The time (in millis) of the last successful <a href="#">update</a> (and <a href="#">failed update</a>)</p> <ul style="list-style-type: none"> <li>Used in <a href="#">timeToNextUpdate</a></li> <li>Starts <code>0</code> when <code>Metadata</code> is <a href="#">created</a></li> <li>Reset (to <code>0</code>) in <a href="#">requestUpdateForNewTopics</a></li> </ul>
<code>lastSuccessfulRefreshMs</code>	
<code>needMetadataForAllTopics</code>	<p>Flag...FIXME</p> <ul style="list-style-type: none"> <li>Disabled (i.e. <code>false</code>) when <code>Metadata</code> is <a href="#">created</a></li> <li>Updated when <code>Metadata</code> is requested to <a href="#">set state to indicate that metadata for all topics in Kafka cluster is required</a></li> </ul>
<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"> <li>Starts turned off when <code>Metadata</code> is <a href="#">created</a></li> <li>Turned on exclusively when <code>Metadata</code> is requested for an <a href="#">update</a></li> <li>Turned off when <code>Metadata</code> is <a href="#">updated</a></li> </ul> <p>Use <a href="#">updateRequested</a> to know the current value.</p>
<code>version</code>	<p>Metadata version</p> <ul style="list-style-type: none"> <li><code>0</code> when <code>Metadata</code> is <a href="#">created</a></li> <li>Incremented every <a href="#">update</a></li> </ul>

Tip	<p>Enable <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>config/tools-log4j.properties</code> :</p> <pre>log4j.logger.org.apache.kafka.clients.Metadata=TRACE</pre> <p>Refer to <a href="#">Logging</a>.</p>
-----	--

## Checking if Metadata Update was Requested

### — `updateRequested` Method

```
synchronized boolean updateRequested()
```

updateRequested ...FIXME

#### Note

updateRequested is used when:

- DefaultMetadataUpdater handles an authentication failure
- ConsumerCoordinator polls for coordinator events
- ConsumerNetworkClient makes sure that the metadata is fresh

## Recording Update Request Failure — failedUpdate Method

```
synchronized void failedUpdate(long now, AuthenticationException authenticationException)
```

failedUpdate ...FIXME

#### Note

failedUpdate is used when...FIXME

## getClusterForCurrentTopics Internal Method

```
Cluster getClusterForCurrentTopics(Cluster cluster)
```

getClusterForCurrentTopics ...FIXME

#### Note

getClusterForCurrentTopics is used when...FIXME

## timeToNextUpdate Method

```
synchronized long timeToNextUpdate(long nowMs)
```

timeToNextUpdate ...FIXME

#### Note

timeToNextUpdate is used when:

- ConsumerNetworkClient ensureFreshMetadata
- DefaultMetadataUpdater (of NetworkClient ) isUpdateDue and maybeUpdate

## add Method

```
synchronized void add(String topic)
```

add ...FIXME

Note	add is used when...FIXME
------	--------------------------

## requestUpdate Method

```
synchronized int requestUpdate()
```

requestUpdate ...FIXME

Note	requestUpdate is used when...FIXME
------	------------------------------------

## 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 Information — fetch Method

```
synchronized Cluster fetch()
```

fetch returns current `cluster` information.

Note	fetch is used when...FIXME
------	----------------------------

## Setting Topics to Maintain — setTopics Method

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

## Updating Cluster Metadata — `update` Method

```
synchronized void update(Cluster cluster, Set<String> unavailableTopics, long now)
```

`update` turns `needUpdate` flag off and increments `version`.

`update` sets `lastRefreshMs` and `lastSuccessfulRefreshMs` internal registries to the input `now`.

(only when `topicExpiryEnabled` is enabled, e.g. `KafkaProducer`) `update` ...FIXME

`update` notifies `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 input `cluster`.

`update` prints out the cluster ID and notifies `clusterResourceListeners` that `cluster` has `changed` (only for a `non-bootstrap cluster`).

```
Cluster ID: [clusterId]
```

### Note

`update` is used when:

- `DefaultMetadataUpdater` `handles MetadataResponse`
- `KafkaConsumer` is `created` (and updates the cluster with a "bootstrap" cluster with bootstrap servers)
- `KafkaProducer` is `created` (and updates the cluster with a "bootstrap" cluster with bootstrap servers)
- `KafkaAdminClient` is `created` (and updates the cluster with a "bootstrap" cluster with bootstrap brokers)
- `AdminClient` is `created` (and updates the cluster with a "bootstrap" cluster with bootstrap brokers)

## 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`.

### Note

`needMetadataForAllTopics` is used when `KafkaConsumer` :

- [Subscribes to topics matching specified pattern](#) (and `needMetadataForAllTopics` flag is then enabled)
- [Unsubscribes from topics](#) (and `needMetadataForAllTopics` 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](#)

# Metadata Update Listener

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);
    }
}
```

# MetadataCache

MetadataCache is...FIXME

# MetadataResponse

`MetadataResponse` holds information about a Kafka cluster, i.e. the broker nodes, the controller and the topics.

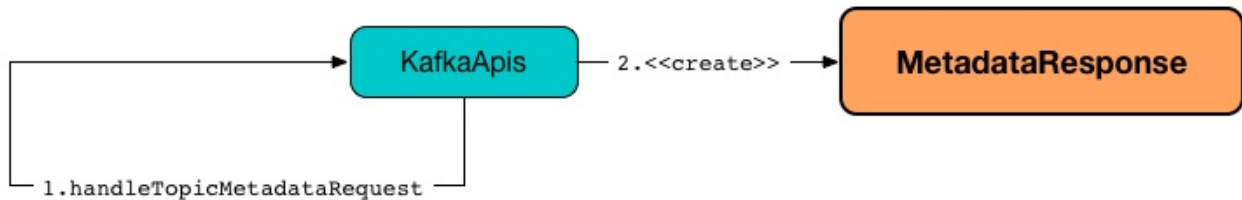


Figure 1. MetadataResponse

`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.

# MetadataUpdater

## MetadataUpdater Contract

```
package org.apache.kafka.clients;

interface MetadataUpdater {
    List<Node> fetchNodes();
    void handleDisconnection(String destination);
    void handleAuthenticationFailure(AuthenticationException exception);
    void handleCompletedMetadataResponse(RequestHeader requestHeader, long now, MetadataResponse metadataResponse);
    boolean isUpdateDue(long now);
    long maybeUpdate(long now);
    void requestUpdate();
}
```

Table 1. MetadataUpdater Contract (in alphabetical order)

Method	Description
maybeUpdate	Starts a cluster metadata update if needed and possible.  Used exclusively when <code>NetworkClient</code> is requested to <a href="#">read and write to sockets</a> .
requestUpdate	
handleAuthenticationFailure	
handleCompletedMetadataResponse	Used exclusively when <code>NetworkClient</code> <a href="#">handles completed receives</a>

# DefaultMetadataUpdater

DefaultMetadataUpdater is a MetadataUpdater that NetworkClient uses to...FIXME

DefaultMetadataUpdater is created when...FIXME

Table 1. DefaultMetadataUpdater’s Internal Properties (e.g. Registries and Counters)

Name	Description
metadata	Metadata
metadataFetchInProgress	<p>Flag to control whether a cluster metadata update is in progress, i.e. FIXME</p> <ul style="list-style-type: none"><li>Disabled when DefaultMetadataUpdater is created</li><li>Turned on exclusively when DefaultMetadataUpdater does maybeUpdate (with a timestamp and a broker node)</li><li>Turned off when DefaultMetadataUpdater handles completed metadata response, disconnection or authentication failure</li></ul>

Tip

Enable WARN , DEBUG or TRACE logging levels 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](#).

## Creating DefaultMetadataUpdater Instance

DefaultMetadataUpdater takes the following when created:

- FIXME

DefaultMetadataUpdater initializes the internal registries and counters.

## 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

`maybeUpdate` is used exclusively when `DefaultMetadataUpdater` is requested to [maybeUpdate](#) (with the timestamp only).

## handleAuthenticationFailure Callback Method

```
void handleAuthenticationFailure(AuthenticationException exception)
```

**Note**

`handleAuthenticationFailure` 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**

`handleCompletedMetadataResponse` 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]"
```



# NetworkClient — Non-Blocking KafkaClient

`NetworkClient` is a non-blocking `KafkaClient` that uses `Selectable` for network communication (i.e. sending and receiving messages).

Note	<code>Selector</code> is the one and only <code>Selectable</code> that uses Java's selectable channels for stream-oriented connecting sockets (i.e. Java's <code>java.nio.channels.SocketChannel</code> ).
------	--

`NetworkClient` does the actual reads and writes (to sockets) every `poll`.

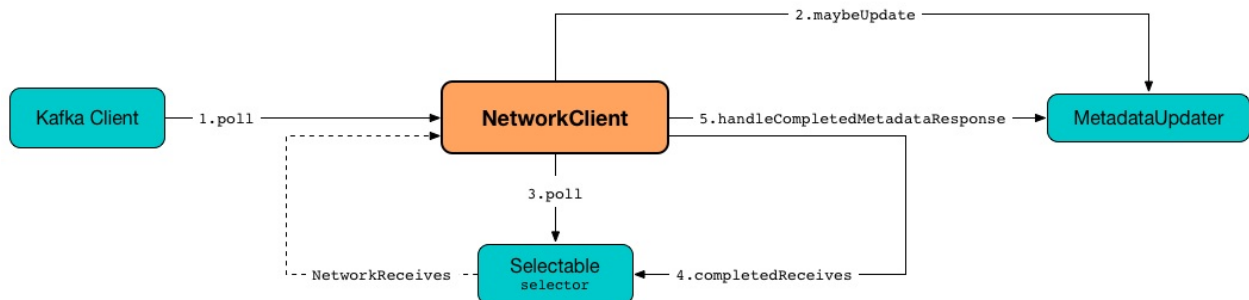


Figure 1. NetworkClient

`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` does `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](#).

## Creating ClientRequest — `newClientRequest` Method

```
ClientRequest newClientRequest(
    String nodeId
    AbstractRequest.Builder<?> requestBuilder
    long createdTimeMs,
    boolean expectResponse)
```

`newClientRequest` ...FIXME

## Note

`newClientRequest` is used when...FIXME

## 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 `ClusterConnectionStates` to enter the connecting state for the connection to the broker `node` .

`initiateConnect` requests `Selectable` to connect to the broker `node` (at a given host and port).

## Note

`initiateConnect` passes the sizes of `send` and `receive` buffers for the socket connection.

In case of an IO failure, `initiateConnect` requests `ClusterConnectionStates` to enter the `disconnected state for the connection` to the broker `node` .

`initiateConnect` requests `MetadataUpdater` for `update`.

You should see the following DEBUG message in the logs:

```
Error connecting to node [node]
```

#### Note

`initiateConnect` is used when:

- `NetworkClient` attempts to connect to a broker node
- `DefaultMetadataUpdater` maybeUpdate

## ready Method

```
boolean ready(Node node, long now)
```

#### Note

`ready` is a part of `KafkaClient Contract`.

`ready` ...FIXME

#### Note

`ready` is used when...FIXME

## wakeup Method

```
void wakeup()
```

#### Note

`wakeup` is a part of `KafkaClient Contract`.

`wakeup` simply requests the internal `Selectable` to `wakeup`

#### Note

`wakeup` is used when...FIXME

## Reading and Writing to Sockets — poll Method

```
List<ClientResponse> poll(long timeout, long now)
```

#### Note

`poll` is a part of `KafkaClient Contract`.

`poll` requests `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 Method

```
void handleCompletedReceives(List<ClientResponse> responses, long now)
```

`handleCompletedReceives` ...FIXME

Note
<code>handleCompletedReceives</code> is used exclusively when <code>NetworkClient</code> <code>polls</code> .

## Creating NetworkClient Instance

`NetworkClient` takes the following when created:

Arguments	Description
<a href="#">MetadataUpdater</a>	
<a href="#">Metadata</a>	
<a href="#">Selectable</a>	
Client ID	
<code>maxInFlightRequestsPerConnection</code>	
<code>reconnectBackoffMs</code>	
<code>reconnectBackoffMax</code>	
<code>socketSendBuffer</code>	<p>Size of the TCP send buffer (SO_SNDBUF) for socket connection (in bytes)</p> <div> <div>Note</div> <div>Use <a href="#">send.buffer.bytes</a> property to configure it.</div> </div> <p>Used when <code>NetworkClient</code> <a href="#">establishes connection to a broker node</a>.</p>
<code>socketReceiveBuffer</code>	<p>Size of the TCP receive buffer (SO_RCVBUF) for socket connection (in bytes)</p> <div> <div>Note</div> <div>Use <a href="#">receive.buffer.bytes</a> property to configure it.</div> </div> <p>Used when <code>NetworkClient</code> <a href="#">establishes connection to a broker node</a></p>
<code>requestTimeoutMs</code>	
<code>Time</code>	
<code>discoverBrokerVersions</code>	Flag...
<code>ApiVersions</code>	
<a href="#">Sensor</a>	
<code>LogContext</code>	

`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> <a href="#">poll</a> (for both <a href="#">abortedSends</a> and completed actions).
------	--

# KafkaClient

KafkaClient is the [contract](#) for...FIXME

Note

[NetworkClient](#) is the one and only implementation.

## KafkaClient Contract

```
package org.apache.kafka.clients;

public interface KafkaClient extends Closeable {
    void close(String nodeId);
    long connectionDelay(Node node, long now);
    boolean connectionFailed(Node node);
    void disconnect(String nodeId);
    boolean hasInFlightRequests();
    boolean hasInFlightRequests(String nodeId);
    boolean hasReadyNodes();
    int inFlightRequestCount();
    int inFlightRequestCount(String nodeId);
    boolean isReady(Node node, long now);
    Node leastLoadedNode(long now);
    ClientRequest newClientRequest(String nodeId, AbstractRequest.Builder<?> requestBuilder,
                                   long createdTimeMs, boolean expectResponse);
    ClientRequest newClientRequest(String nodeId, AbstractRequest.Builder<?> requestBuilder, long createdTimeMs,
                                   boolean expectResponse, RequestCompletionHandler callback);
    boolean ready(Node node, long now);
    List<ClientResponse> poll(long timeout, long now);
    void send(ClientRequest request, long now);
    void wakeup();
}
```

Table 1. KafkaClient Contract

Method	Description
<code>newClientRequest</code>	Used when: <ul style="list-style-type: none"><li>• ...FIXME</li></ul>
<code>wakeup</code>	Used when: <ul style="list-style-type: none"><li>• ...FIXME</li></ul>
<code>poll</code>	Used when: <ul style="list-style-type: none"><li>• <code>ConsumerNetworkClient</code> <a href="#">polls</a></li><li>• (Blocking) <code>NetworkClientUtils</code> does <code>awaitReady</code> , <code>isReady</code> OR <code>sendAndReceive</code></li><li>• <code>AdminClientRunnable</code> is started (and <code>run</code> is executed)</li><li>• <code>Sender</code> is requested to <a href="#">run once</a></li><li>• <code>InterBrokerSendThread</code> <a href="#">does its work</a></li></ul>



# NetworkClientUtils

NetworkClientUtils is...FIXME

## sendAndReceive Method

```
static 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 Method

```
static boolean awaitReady(  
    KafkaClient client  
    Node node  
    Time time  
    long timeoutMs) throws IOException
```

awaitReady ...FIXME

Note	awaitReady is used when...FIXME
------	---------------------------------

# OffsetConfig

OffsetConfig is...FIXME

Table 1. OffsetConfig's Properties (in alphabetical order)

Property	Default Value
offsetsTopicNumPartitions	50

# ReplicaManager

`ReplicaManager` is [created](#) and immediately [started](#) when `KafkaServer` is requested to [start up](#).

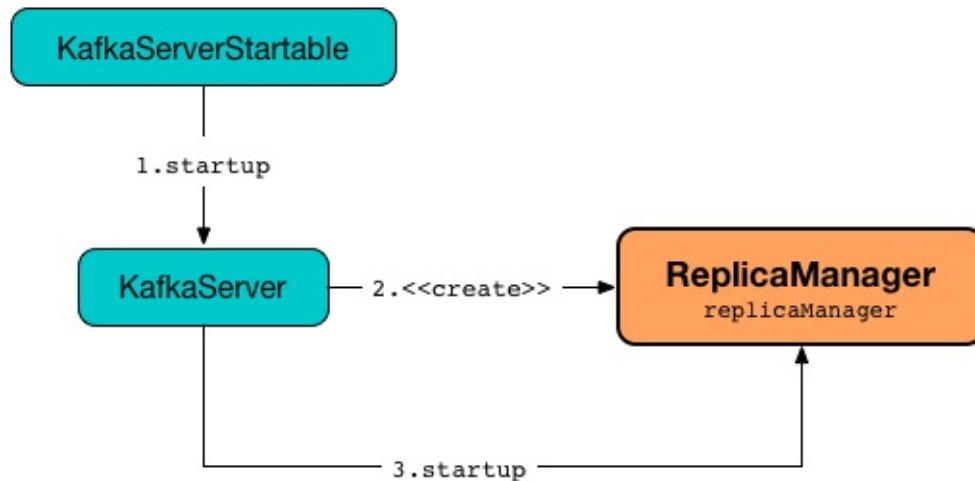


Figure 1. ReplicaManager and KafkaServer

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` is a [KafkaMetricsGroup](#).

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

Name	Description
<code>replicaFetcherManager</code>	<a href="#">ReplicaFetcherManager</a>
<code>allPartitions</code>	Pool of <code>TopicPartition</code> and <a href="#">Partitions</a>
<code>isrChangeSet</code>	Collection of <code>TopicPartition</code> that...FIXME
<code>lastIsrChangeMs</code>	Time when <a href="#">isrChangeSet</a> has a new <code>TopicPartition</code> <a href="#">added</a> .
<code>logDirFailureHandler</code>	<a href="#">LogDirFailureHandler</a>
<code>OfflinePartition</code>	

## `createReplicaFetcherManager` Internal Method

```
createReplicaFetcherManager(
  metrics: Metrics
  time: Time
  threadNamePrefix: Option[String]
  quotaManager: ReplicationQuotaManager): ReplicaFetcherManager
```

createReplicaFetcherManager ...FIXME

Note

createReplicaFetcherManager is used when...FIXME

## shutdown Method

```
shutdown(checkpointHW: Boolean = true): Unit
```

shutdown ...FIXME

Note

shutdown is used when...FIXME

## alterReplicaLogDirs Method

```
alterReplicaLogDirs(partitionDirs: Map[TopicPartition, String]): Map[TopicPartition, E
rrors]
```

alterReplicaLogDirs ...FIXME

Note

alterReplicaLogDirs is used exclusively when KafkaApis [handles AlterReplicaLogDirs request](#).

## becomeLeaderOrFollower Method

```
becomeLeaderOrFollower(
  correlationId: Int,
  leaderAndIsrRequest: LeaderAndIsrRequest,
  onLeadershipChange: (Iterable[Partition], Iterable[Partition]) => Unit): LeaderAndIs
rResponse
```

becomeLeaderOrFollower ...FIXME

Note

becomeLeaderOrFollower is used exclusively when KafkaApis 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

makeFollowers is used exclusively when ReplicaManager 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

recordIsrChange is used exclusively when Partition does [updateIsr](#)

## updateFollowerLogReadResults Internal Method

```
updateFollowerLogReadResults(
  replicaId: Int,
  readResults: Seq[(TopicPartition, LogReadResult)]: Seq[(TopicPartition, LogReadResult)])
```

updateFollowerLogReadResults ...FIXME

### Note

updateFollowerLogReadResults is used exclusively when ReplicaManager [fetches 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> handles a Fetch request.
------	---

## getLeaderPartitions Internal Method

```
getLeaderPartitions: List[Partition]
```

`getLeaderPartitions` gives the partitions from [allPartitions](#) that are not [offline](#) and their `leaderReplicaIfLocal` property is defined.

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

## isr-expiration Task

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

## isr-change-propagation Task

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

## maybePropagateIsrChanges Method

```
maybePropagateIsrChanges(): Unit
```

`maybePropagateIsrChanges` ...FIXME

Note	<code>maybePropagateIsrChanges</code> is used exclusively when <a href="#">isr-change-propagation</a> 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

`startup` uses `Scheduler` that was specified when `ReplicaManager` was created.

## Note

`startup` is used exclusively when `KafkaServer` 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](#).

## `getLog` Method

```
getLog(topicPartition: TopicPartition): Option[Log]
```

`getLog` ...FIXME

## Note

`getLog` is used when...FIXME

## `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

# ReplicaFetcherManager

`ReplicaFetcherManager` is a [AbstractFetcherManager](#) 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): AbstractFetcherThread
```

`createFetcherThread` ...FIXME

Note
<code>createFetcherThread</code> is used exclusively when <code>AbstractFetcherManager</code> <a href="#">addFetcherForPartitions</a>

## 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](#).

# AbstractFetcherManager

AbstractFetcherManager is...FIXME

## addFetcherForPartitions Method

```
addFetcherForPartitions(partitionAndOffsets: Map[TopicPartition, BrokerAndInitialOffset
]): Unit
```

addFetcherForPartitions ...FIXME

Note	<p>addFetcherForPartitions is used when:</p> <ul style="list-style-type: none"><li>• ReplicaManager alterReplicaLogDirs, becomeLeaderOrFollower, makeFollowers</li><li>• LeaderFinderThread (of the currently-deprecated ConsumerFetcherManager ) does doWork</li></ul>
------	---

# ReplicaFetcherThread

`ReplicaFetcherThread` is a [AbstractFetcherThread](#)...FIXME

`ReplicaFetcherThread` is [created](#) exclusively when `ReplicaFetcherManager` is requested to [create one](#) (when...FIXME).

## Creating ReplicaFetcherThread Instance

`ReplicaFetcherThread` takes the following when created:

- Name
- Fetcher ID
- Source `BrokerEndPoint`
- [KafkaConfig](#)
- [ReplicaManager](#)
- `Metrics`
- `Time`
- [ReplicationQuotaManager](#)
- Optional `BlockingSend` (undefined by default)

`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	<code>fetchEpochsFromLeader</code> is a part of <a href="#">AbstractFetcherThread Contract</a> .
------	--

`fetchEpochsFromLeader` ...FIXME

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

# AbstractFetcherThread

AbstractFetcherThread is...FIXME

# ReplicaFetcherBlockingSend

ReplicaFetcherBlockingSend is...FIXME

ReplicaFetcherBlockingSend is created exclusively when ReplicaFetcherThread is created.

ReplicaFetcherBlockingSend uses NetworkClient to send requests for...FIXME

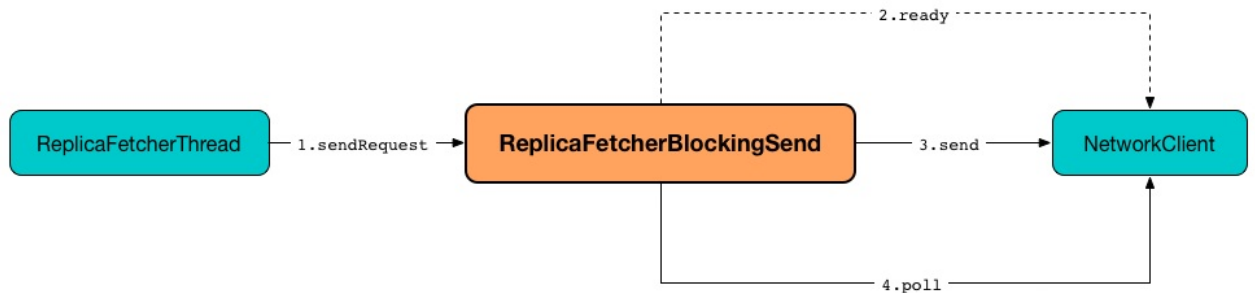


Figure 1. ReplicaFetcherBlockingSend’s Sending Client Request and Waiting for Response

ReplicaFetcherBlockingSend uses replica.socket.timeout.ms Kafka property for...FIXME

Table 1. ReplicaFetcherBlockingSend’s Internal Properties (e.g. Registries, Counters and Flags)

Name	Description
networkClient	NetworkClient Used when...FIXME
sourceNode	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

`sendRequest` 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

`sendRequest` is used when `ReplicaFetcherThread` [earliestOrLatestOffset](#) and [fetchEpochsFromLeader](#).

## `close` Method

```
close(): Unit
```

`close` ...FIXME

### Note

`close` is used when...FIXME



# ReplicationQuotaManager

ReplicationQuotaManager is...FIXME

# ReplicationUtils

ReplicationUtils ...FIXME

propagateIsrChanges

Method

```
propagateIsrChanges(zkUtils: ZkUtils, isrChangeSet: Set[TopicPartition]): Unit
```

propagateIsrChanges ...FIXME

Note	propagateIsrChanges is used exclusively when <a href="#">maybePropagateIsrChanges</a> .
------	---

# ReplicaStateMachine

ReplicaStateMachine is...FIXME

## handleStateChanges Method

```
handleStateChanges(  
  replicas: Set[PartitionAndReplica],  
  targetState: ReplicaState,  
  callbacks: Callbacks): Unit
```

handleStateChanges ...FIXME

Note	handleStateChanges is used when...FIXME
------	---

## shutdown Method

```
FIXME
```

shutdown ...FIXME

Note	shutdown is used when...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` .

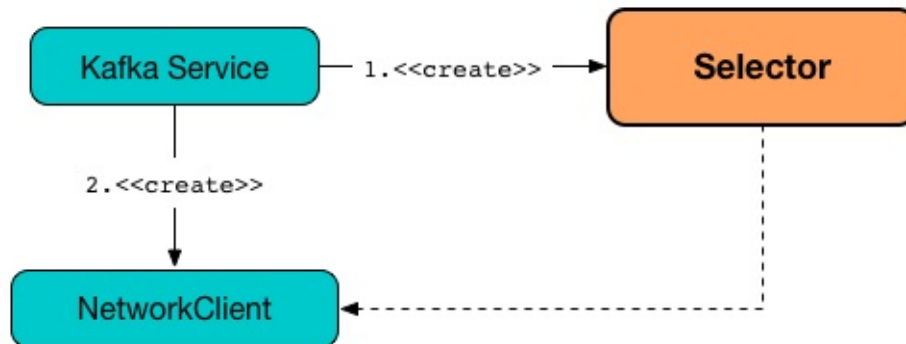


Figure 1. Selector is Created for Kafka Services For NetworkClient

`Selector` is created when:

- `KafkaAdminClient` is created (using `createInternal` )
- `KafkaConsumer` is created
- `KafkaProducer` is created
- `AdminClient` is created
- `ControllerChannelManager` does `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

# Selectable

Selectable is the [contract](#) for asynchronous, multi-channel network I/O.

Note	<a href="#">Selector</a> is the one and only <code>Selectable</code> .
------	--

```
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 <a href="#">establish a connection to a broker</a>
<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 <a href="#">run</a> method.
<code>shutdownLatch</code>	Java's <a href="#">java.util.concurrent.CountDownLatch</a> with the number of passes being 1

## run Method

```
run(): Unit
```

**Note** `run` 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.

SocketServer is created exclusively when KafkaServer is started.

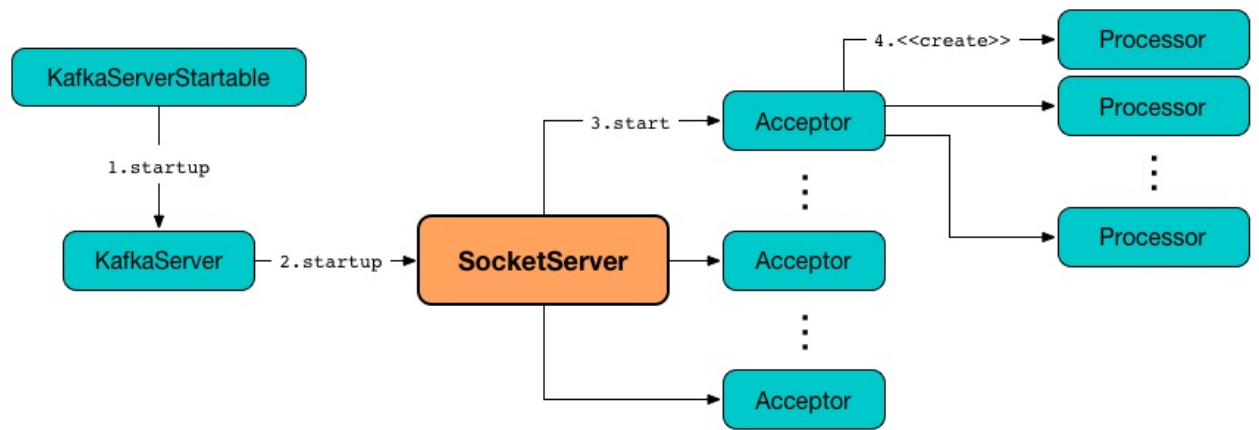


Figure 1. SocketServer’s Startup

Table 1. SocketServer’s Metrics (in kafka.network group)

Name	Description
NetworkProcessorAvgIdlePercent	
MemoryPoolAvailable	
MemoryPoolUsed	



Table 2. SocketServer's Internal Properties (e.g. Registries and Counters) (in alphabetical order)

Name	Description
<code>acceptors</code>	Acceptor threads per EndPoint
<code>connectionQuotas</code>	ConnectionQuotas
<code>endpoints</code>	EndPoints (aka <i>listeners</i> ) per name (as configured using <code>listeners</code> Kafka property)
<code>maxQueuedRequests</code>	
<code>maxConnectionsPerIp</code>	
<code>maxConnectionsPerIpOverrides</code>	
<code>memoryPool</code>	
<code>numProcessorThreads</code>	The number of processors per endpoint (as configured using <code>num.network.threads</code> Kafka property)
<code>processors</code>	Processor threads (initially <code>totalProcessorThreads</code> )
<code>requestChannel</code>	
<code>totalProcessorThreads</code>	Total number of <code>processors</code> , i.e. <code>numProcessorThreads</code> for every endpoint

## Creating Processor — `newProcessor` Internal Method

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

## Starting Up (and Auxiliary Services) — `startup` Method

```
startup(): 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> <a href="#">starts up</a> .
------	--

## Creating SocketServer Instance

`SocketServer` takes the following when created:

- `KafkaConfig`
- `Metrics`
- `Time`
- `CredentialProvider`

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

# TopicDeletionManager

TopicDeletionManager is...FIXME

TopicDeletionManager is **created** exclusively when KafkaController is **created**.

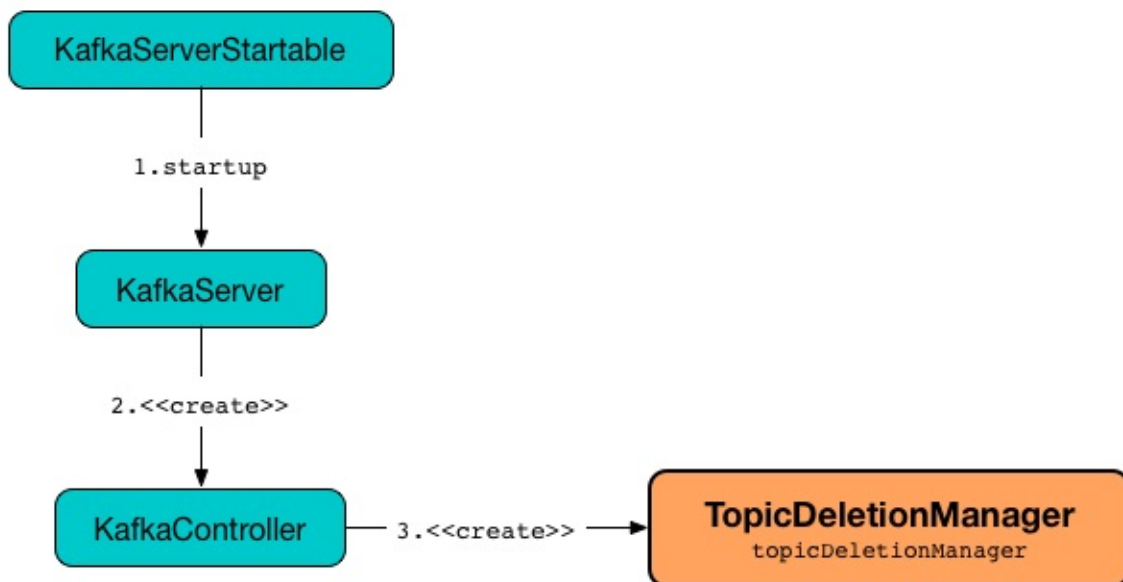


Figure 1. TopicDeletionManager is Created Alongside KafkaController

TopicDeletionManager is controlled by **delete.topic.enable** Kafka property and does nothing when it is turned off (i.e. `false` ).

logIdent is **[Topic Deletion Manager [brokerId]]**.

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

Name	Description
partitionsToBeDeleted	TopicAndPartitions to be deleted
topicsToBeDeleted	The names of the topics to be deleted
topicsIneligibleForDeletion	The names of the topics that must not be deleted (i.e. are ineligible for deletion)

## Tip

Enable `INFO` logging level for `kafka.controller.TopicDeletionManager` logger to see what happens inside.

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

```
log4j.logger.kafka.controller.TopicDeletionManager=INFO
```

Refer to [Logging](#).

## startReplicaDeletion Internal Method

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

## enqueueTopicsForDeletion Method

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

## failReplicaDeletion Method

Caution	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	<p><code>markTopicIneligibleForDeletion</code> is used when:</p> <ul style="list-style-type: none"> <li>• <code>KafkaController</code> <a href="#">initiateReassignReplicasForTopicPartition</a></li> <li>• <code>TopicDeletion</code> controller event is <a href="#">processed</a> (for topics to be deleted with partitions in <code>controllerContext.partitionsBeingReassigned</code> list)</li> <li>• <code>TopicDeletionManager</code> does <a href="#">startReplicaDeletion</a> and <a href="#">failReplicaDeletion</a></li> </ul>
------	--

## 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	<code>reset</code> does nothing when <a href="#">delete.topic.enable</a> Kafka property is <code>false</code> .
Note	<code>reset</code> is used exclusively when <code>KafkaController</code> <a href="#">resigns as the active controller</a> .

## `onTopicDeletion` Internal Method

```
onTopicDeletion(topics: Set[String]): Unit
```

`onTopicDeletion` ...FIXME

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

# TransactionCoordinator

TransactionCoordinator is...FIXME

TransactionCoordinator is [created](#) when...FIXME

## startup Method

```
startup
```

startup ...FIXME

Note	startup is used exclusively when KafkaServer is requested to <a href="#">start up</a> .
------	---

## 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 <a href="#">start up</a> .
------	---

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
------	--

# 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
------	---

# ZkUtils

`zkUtils` is...FIXME

`zkUtils` is `created` when...FIXME

Note

Quoting [Nodes and ephemeral nodes](#) from the Zookeeper documentation:

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.

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.

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.

Table 1. ZkUtils's ZNodes in Zookeeper

ZNode	Description
<code>ControllerPath</code>	<code>/controller</code>

Table 2. ZkUtils's Internal Properties (e.g. Registries and Counters) (in alphabetical order)

Name	Description
<code>persistentZkPaths</code>	
<code>zkPath</code>	

## `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	<code>getCluster</code> is used exclusively when <code>ZKRebalancerListener</code> does <a href="#">syncedRebalance</a> (that happens for the currently-deprecated <code>ZookeeperConsumerConnector</code> ).
------	---

## `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	<code>apply</code> is used when: <ol style="list-style-type: none"><li><code>KafkaServer</code> <a href="#">connects to Zookeeper</a></li><li>FIXME</li></ol>
------	--

## Registering Listener for State Changes — `subscribeStateChanges` Method

```
subscribeStateChanges(listener: IZkStateListener): Unit
```

`subscribeStateChanges` requests [ZkClient](#) to `subscribeStateChanges` with the `listener` .

## Note

`subscribeStateChanges` is used when:

1. `KafkaController` is requested to [register a SessionExpirationListener](#)
2. FIXME

## Registering Listener for Child Changes — `subscribeChildChanges` Method

```
subscribeChildChanges(path: String, listener: IZkChildListener): Option[Seq[String]]
```

`subscribeChildChanges` ...FIXME

## Note

`subscribeChildChanges` is used...FIXME

## 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

`unsubscribeChildChanges` is used when...FIXME

## 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

`unsubscribeDataChanges` is used when...FIXME

## `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

getTopicPartitionCount is used when:

1. GroupMetadataManager is requested for [getGroupMetadataTopicPartitionCount](#) of `__consumer_offsets` topic
2. 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	<code>controllerZkData</code> is used exclusively when <code>kafkaController</code> is requested for <a href="#">elect</a> .
------	--

## 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	<p>syncedRebalance is used when:</p> <ul style="list-style-type: none"><li>• ZKSessionExpireListener does handleNewSession</li><li>• That happens exclusively in (deprecated) ZookeeperConsumerConnector</li><li>• ZKRebalancerListener is created (and creates watcherExecutorThread )</li><li>• That happens exclusively in (deprecated) ZookeeperConsumerConnector</li><li>• (deprecated) ZookeeperConsumerConnector does reinitializeConsumer</li></ul>
------	---

# ControllerBrokerRequestBatch

ControllerBrokerRequestBatch is...FIXME

## sendRequestsToBrokers Method

```
sendRequestsToBrokers(controllerEpoch: Int): Unit
```

sendRequestsToBrokers ...FIXME

Note	<div><p>sendRequestsToBrokers is used when:</p><ul style="list-style-type: none"><li>• kafkaController is requested to <a href="#">updateLeaderEpochAndSendRequest</a>, <a href="#">sendUpdateMetadataRequest</a> and in <code>ControlledShutdown</code></li><li>• PartitionStateMachine is requested to <a href="#">handleStateChanges</a></li><li>• ReplicaStateMachine is requested to <a href="#">handleStateChanges</a></li></ul></div>
------	--



# 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...FIXME

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

Name	Description
<code>inSyncReplicas</code>	In-sync <a href="#">Replicas</a>
<code>leaderReplicaIdOpt</code>	Optional leader replica ID
<code>localBrokerId</code>	One of the following: <ul style="list-style-type: none"> <li><code>broker.id</code> configuration property (from the <a href="#">ReplicaManager</a>) if not <a href="#">offline</a></li> <li><code>-1</code> if <a href="#">offline</a></li> </ul>
<code>logManager</code>	<a href="#">LogManager</a> (of the <a href="#">ReplicaManager</a> ) if not <a href="#">offline</a> or <code>null</code>
<code>zkClient</code>	<a href="#">KafkaZkClient</a> (of the <a href="#">ReplicaManager</a> ) if not <a href="#">offline</a> or <code>null</code>
<code>topicPartition</code>	<code>TopicPartition</code> (with the <a href="#">topic</a> and <a href="#">partitionId</a> )

## maybeExpandIsr Method

```
maybeExpandIsr(replicaId: Int, logReadResult: LogReadResult): Boolean
```

`maybeExpandIsr` ...FIXME

Note	<code>maybeExpandIsr</code> is used exclusively when <code>Partition</code> is requested to <a href="#">updateReplicaLogReadResult</a> .
------	--

## maybeShrinkIsr Method

```
maybeShrinkIsr(replicaMaxLagTimeMs: Long): Unit
```

`maybeShrinkIsr` ...FIXME

Note	<code>maybeShrinkIsr</code> is used exclusively when <code>ReplicaManager</code> <a href="#">maybeShrinkIsr</a> .
------	---

## updateReplicaLogReadResult Method

```
updateReplicaLogReadResult(replica: Replica, logReadResult: LogReadResult): Boolean
```

`updateReplicaLogReadResult` ...FIXME

Note	<code>updateReplicaLogReadResult</code> is used exclusively when <code>ReplicaManager</code> <a href="#">updateFollowerLogReadResults</a> .
------	---

## updateIsr Internal Method

```
updateIsr(newIsr: Set[Replica]): Unit
```

`updateIsr` ...FIXME

Note	<code>updateIsr</code> is used when <code>Partition</code> is requested to <a href="#">expand</a> or <a href="#">shrink</a> the ISR.
------	--

## makeFollower Method

```
makeFollower(
  controllerId: Int,
  partitionStateInfo: LeaderAndIsrRequest.PartitionState,
  correlationId: Int): Boolean
```

`makeFollower` ...FIXME

Note	<code>makeFollower</code> is used exclusively when <code>ReplicaManager</code> is requested to <a href="#">makeFollowers</a> .
------	--

## leaderReplicaIfLocal Method

```
leaderReplicaIfLocal: Option[Replica]
```

leaderReplicaIfLocal gives...FIXME

Note

leaderReplicaIfLocal is used...FIXME

## maybeShrinkIsr Method

Caution

FIXME

## Creating Partition Instance

Partition takes the following when created:

- Topic name
- Partition ID
- Time
- [ReplicaManager](#)
- isOffline flag (default: false )

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

isUnderReplicated 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

`isUnderMinIsr` 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
<code>maybeCreateFutureReplica</code> is used exclusively when <code>Partition</code> is requested to <a href="#">alterReplicaLogDirs</a> .

## alterReplicaLogDirs Method

```
alterReplicaLogDirs(partitionDirs: Map[TopicPartition, String]): Map[TopicPartition, Errors]
```

alterReplicaLogDirs ...FIXME

Note
<code>alterReplicaLogDirs</code> is used exclusively when <code>KafkaApis</code> is requested to <a href="#">handle an AlterReplicaLogDirs request</a> .

# Replica

Replica is...FIXME

# PartitionStateMachine

PartitionStateMachine is...FIXME

## triggerOnlinePartitionStateChange Method

```
triggerOnlinePartitionStateChange(): Unit
```

triggerOnlinePartitionStateChange ...FIXME

Note	triggerOnlinePartitionStateChange is used when...FIXME
------	--

## handleStateChanges Method

```
handleStateChanges(  
  partitions: Set[TopicAndPartition],  
  targetState: PartitionState,  
  leaderSelector: PartitionLeaderSelector = noOpPartitionLeaders...,  
  callbacks: Callbacks): Unit
```

handleStateChanges ...FIXME

Note	handleStateChanges is used when...FIXME
------	---

## shutdown Method

```
FIXME
```

shutdown ...FIXME

Note	shutdown is used when...FIXME
------	-------------------------------

# BrokerTopicStats

BrokerTopicStats is...FIXME



# BrokerTopicMetrics

BrokerTopicMetrics is...FIXME

# KafkaMetricsGroup

`KafkaMetricsGroup` is the [contract](#) of [metrics groups](#) that can register new **performance metrics**, e.g. [gauge](#), [histogram](#), [meter](#) and [timer](#).

```
package kafka.metrics

trait KafkaMetricsGroup {
  // No required properties (vals and methods) that have no implementation
}
```

Table 1. KafkaMetricsGroups (Direct Implementations)

KafkaMetricsGroup	Description
<a href="#">AbstractFetcherManager</a>	
Acceptor	
<a href="#">AdminManager</a>	
AppInfo	
<a href="#">BrokerTopicMetrics</a>	
ControllerChannelManager	
<a href="#">ControllerEventManager</a>	
ControllerStats	
DelayedDeleteRecordsMetrics	
DelayedFetchMetrics	
DelayedOperationPurgatory	
DelayedProduceMetrics	
DelegationTokenManager	
FetcherLagMetrics	
FetcherStats	

FetchSessionCache	
GroupMetadataManager	
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
------	--

# 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 replica 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
[]
```

# 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)
```

# KafkaProducer — Main Class For Kafka Producers

`KafkaProducer` is the main public class that you and other Kafka developers use to write [Kafka producers](#) that publish records to a Kafka cluster.

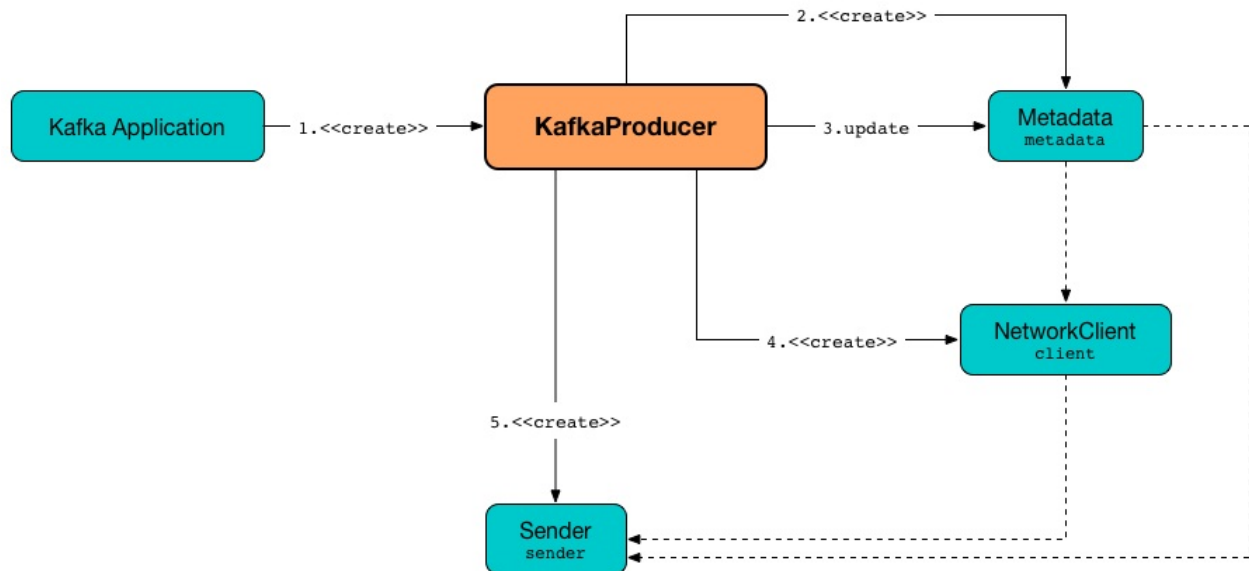


Figure 1. KafkaProducer

`KafkaProducer` is a part of the public API and is [created](#) with properties and (key and value) serializers as configuration.

`KafkaProducer` follows the [Producer contract](#).

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

Name	Description		
<code>maxBlockTimeMs</code>	<p>Time <code>KafkaProducer</code> uses to block when requesting <a href="#">partitions for a topic</a>.</p> <table border="1"> <tr> <td>Note</td><td>Use <code>max.block.ms</code> Kafka property to set the value.</td></tr> </table>	Note	Use <code>max.block.ms</code> Kafka property to set the value.
Note	Use <code>max.block.ms</code> Kafka property to set the value.		
<code>metadata</code>	<p><b>Metadata</b></p> <ul style="list-style-type: none"> <li>Created when <code>KafkaProducer</code> is <a href="#">created</a> with the following properties: <ul style="list-style-type: none"> <li><code>retry.backoff.ms</code> for <code>refreshBackoffMs</code></li> <li><code>metadata.max.age.ms</code> for <code>metadataExpireMs</code></li> <li><code>allowAutoTopicCreation</code> flag enabled</li> <li><code>topicExpiryEnabled</code> flag enabled</li> </ul> </li> <li><a href="#">Updated</a> with a bootstrap cluster when <code>KafkaProducer</code> is <a href="#">created</a></li> <li>Used in <a href="#">waitOnMetadata</a></li> </ul>		
<code>sender</code>	<p><a href="#">Sender</a> with the following:</p> <ul style="list-style-type: none"> <li>...FIXME</li> </ul>		
<code>ioThread</code>	Thread of execution...FIXME		

Tip	<p>Enable <code>DEBUG</code> or <code>TRACE</code> logging levels for <code>org.apache.kafka.clients.producer.KafkaProducer</code> logger to see what happens inside.</p> <p>Add the following line to <code>config/tools-log4j.properties</code> :</p> <pre>log4j.logger.org.apache.kafka.clients.producer.KafkaProducer=TRACE</pre> <p>Refer to <a href="#">Logging</a>.</p>
-----	--

## Asynchronously Sending Record to Topic and Notifying Producer Interceptors — `send` Method

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

## Note

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

`send ...FIXME`

## Asynchronously Sending Record to Topic — `doSend` Internal Method

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

`doSend ...FIXME`

## Note

`doSend` is used exclusively when `KafkaProducer` [asynchronously sends a record to a topic and notifies producer interceptors](#).

## Creating KafkaProducer Instance

`KafkaProducer` takes the following when created:

- `FIXME`

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

## 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

`configureClusterResourceListeners` is used exclusively when `KafkaProducer` is [created](#) (to create the [Metadata](#)) with the following input arguments:

- [key](#) and [value](#) serializers (defined when `KafkaProducer` is created)
- [ProducerInterceptors](#) from `interceptor.classes` Kafka property
- [MetricsReporters](#) from `metric.reporters` Kafka property

## Requesting Partitions for Topic — `partitionsFor` Method

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

### Note

`partitionsFor` 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	<code>waitOnMetadata</code> is used when <code>KafkaProducer</code> requests partitions for a topic and asynchronously sends a record to a topic.
------	---

# Producer

Producer is...FIXME

# DefaultPartitioner

DefaultPartitioner is...FIXME

# Partitioner

Partitioner is...FIXME

# ProducerInterceptor

ProducerInterceptor is...FIXME

# Sender

`Sender` is [thread of execution](#) that handles the sending of produce requests to a Kafka cluster.

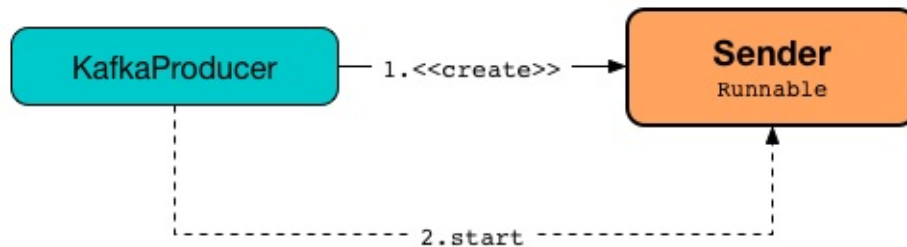


Figure 1. Sender and KafkaProducer

`Sender` is [created](#) exclusively when `KafkaProducer` is [created](#).

`Sender` is started under the name **kafka-producer-network-thread | [clientId]** when `KafkaProducer` is [created](#).

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

Name	Description
<code>running</code>	Flag that controls whether <a href="#">run</a> should finish or not

## sendProduceRequests Internal Method

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

`sendProduceRequests` ...FIXME

**Note** `sendProduceRequests` is used exclusively when `Sender` [sendProducerData](#).

## sendProduceRequest Internal Method

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

`sendProduceRequest` ...FIXME

Note	<code>sendProduceRequest</code> is used exclusively when <code>Sender</code> <a href="#">sendProduceRequests</a> .
------	--

## sendProducerData Internal Method

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

`sendProducerData` ...FIXME

Note	<code>sendProducerData</code> is used exclusively when <code>Sender</code> <a href="#">runs a single iteration of sending</a> .
------	---

## Running Single Iteration of Sending — run Method

```
void run(long now)
```

`run` ...FIXME

Note	<code>run</code> is used exclusively when <code>Sender</code> is <a href="#">started</a> (as a thread of execution).
------	--

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

```
void run()
```

Note	<code>run</code> is a part of <a href="#">java.lang.Runnable</a> 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](#) (with the current time in milliseconds) until [running](#) flag is turned off.

`run` ...FIXME

## Creating Sender Instance

`Sender` takes the following when created:

- `LogContext`

- [KafkaClient](#)
- [Metadata](#)
- `RecordAccumulator`
- `guaranteeMessageOrder` flag
- `maxRequestSize`
- `acks`
- The number of retries
- `SenderMetricsRegistry`
- `Time`
- `requestTimeout`
- `retryBackoffMs`
- `TransactionManager`
- `ApiVersions`

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



# Serializer

Serializer is...FIXME

# KafkaConsumer — Main Class For Kafka Consumers

`KafkaConsumer` is the main public class that you and other Kafka developers use to write [Kafka consumers](#) that consume records from a Kafka cluster.

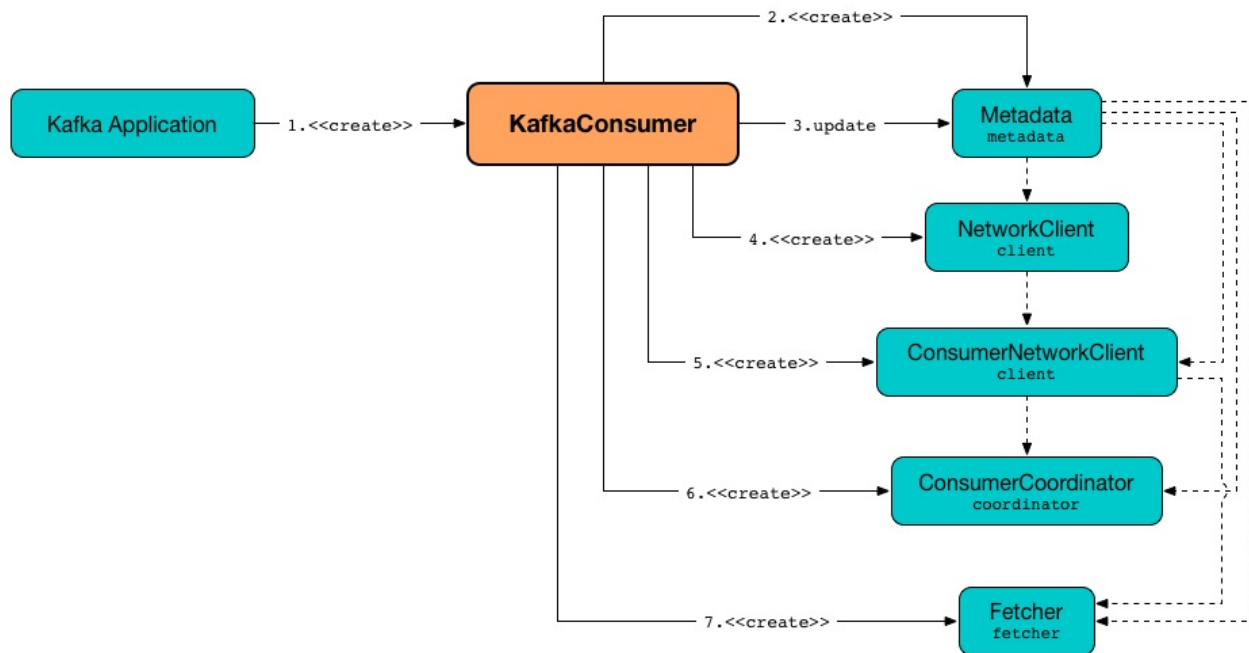


Figure 1. KafkaConsumer

`KafkaConsumer` is a part of the public API and is [created](#) with properties and (key and value) deserializers as configuration.

Note	<a href="#">bootstrap.servers</a> and <a href="#">group.id</a> properties are mandatory. They usually appear in the code as <code>ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG</code> and <code>ConsumerConfig.GROUP_ID_CONFIG</code> values, respectively.
------	--

`KafkaConsumer` follows the [Consumer contract](#).

```
// sandbox/kafka-sandbox
val bootstrapServers = "localhost:9092"
val groupId = "kafka-sandbox"
import org.apache.kafka.clients.consumer.ConsumerConfig
val configs: Map[String, Object] = Map(
  // required properties
  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](
  configs.asJava,
  keyDeserializer,
  valueDeserializer)
```

`KafkaConsumer` registers itself in JMX with **kafka.consumer** prefix.

Caution	FIXME How does the JMX registration happen?
Important	<p><code>KafkaConsumer</code> does not support multi-threaded access. You should use only one thread per <code>KafkaConsumer</code> instance.</p> <p><code>KafkaConsumer</code> uses light locks protecting itself from multi-threaded access and reports <code>ConcurrentModificationException</code> when it happens.</p> <p><code>KafkaConsumer</code> is not safe for multi-threaded access</p>

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

Name	Description
<code>client</code>	<p><a href="#">ConsumerNetworkClient</a></p> <p>Used mainly (?) to create the <a href="#">Fetcher</a> and <a href="#">ConsumerCoordinator</a></p> <p>Used also in <a href="#">poll</a>, <a href="#">pollOnce</a> and <a href="#">wakeup</a> (but I think the usage should be limited to create <a href="#">Fetcher</a> and <a href="#">ConsumerCoordinator</a>)</p>
<code>clientId</code>	
<code>coordinator</code>	<p><a href="#">ConsumerCoordinator</a></p> <ul style="list-style-type: none"> <li>• Initialized when <code>KafkaConsumer</code> is <a href="#">created</a></li> <li>• Used for...FIXME</li> </ul>
<code>fetcher</code>	<p><a href="#">Fetcher</a></p> <p>Created right when <code>KafkaConsumer</code> is <a href="#">created</a>.</p> <p>Used when...FIXME</p>
<code>interceptors</code>	<p><code>ConsumerInterceptors</code> that holds <a href="#">ConsumerInterceptor</a> instances (defined using <a href="#">interceptor.classes</a> setting).</p> <p>Used when...FIXME</p>
<code>metadata</code>	<p><a href="#">Metadata</a></p> <p>Created right when <code>KafkaConsumer</code> is <a href="#">created</a>.</p> <p>Used when...FIXME</p>
<code>metrics</code>	<code>Metrics</code>
<code>retryBackoffMs</code>	<a href="#">retry.backoff.ms</a> property or a user-defined value
<code>requestTimeoutMs</code>	<p>Corresponds to <a href="#">request.timeout.ms</a> property</p> <p><code>KafkaConsumer</code> reports <code>ConfigException</code> when smaller or equal than <a href="#">session.timeout.ms</a> and <a href="#">fetch.max.wait.ms</a> properties.</p>
<code>subscriptions</code>	<p><code>SubscriptionState</code> for <a href="#">auto.offset.reset</a> setting.</p> <p>Created when <code>KafkaConsumer</code> is <a href="#">created</a>.</p>

## Tip

Enable `DEBUG` or `TRACE` logging levels for `org.apache.kafka.clients.consumer.KafkaConsumer` logger to see what happens inside.

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

```
log4j.logger.org.apache.kafka.clients.consumer.KafkaConsumer=TRACE
```

Refer to [Logging](#).

## (Manually) Assigning Partitions — `assign` Method

```
void assign(Collection<TopicPartition> partitions)
```

## Note

`assign` is part of [Consumer Contract](#) to...FIXME.

`assign` ...FIXME

## `unsubscribe` Method

```
void unsubscribe()
```

## Note

`unsubscribe` 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`.

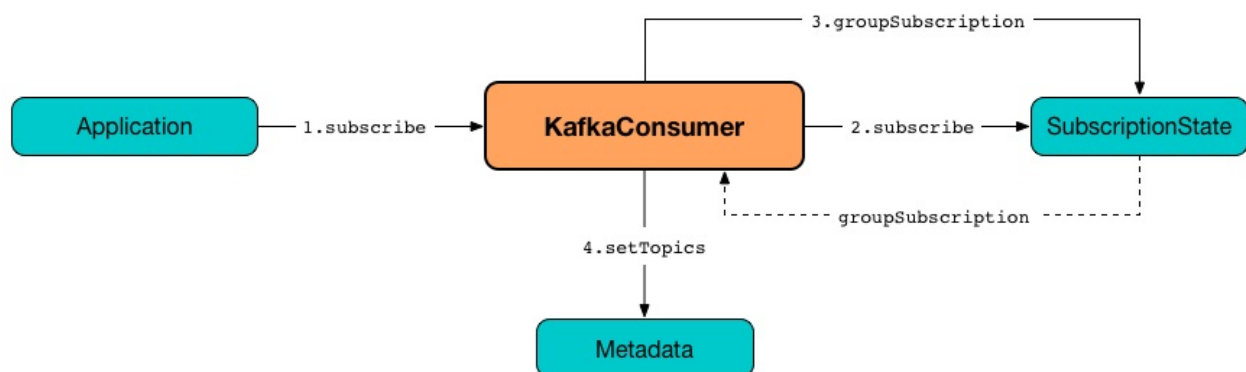


Figure 2. KafkaConsumer subscribes to topics

## Poll Specified Milliseconds For ConsumerRecords per TopicPartitions — `poll` Method

```
ConsumerRecords<K, V> poll(long timeout)
```

`poll` polls for new records until `timeout` expires.

Note	<code>KafkaConsumer</code> has to be subscribed to some topics or assigned partitions before calling <code>poll</code> .
------	--

Note	The input <code>timeout</code> should be <code>0</code> or greater and represents the milliseconds to poll for records.
------	---

```

val seconds = 10
while (true) {
    println(s"Polling for records for $seconds secs")
    val records = consumer.poll(seconds * 1000)
    // do something with the records here
}

```

Internally, `poll` starts by [polling once](#) (for `timeout` milliseconds).

If there are records available, `poll` checks [Fetcher](#) for [sendFetches](#) and [ConsumerNetworkClient](#) for [pendingRequestCount](#) flag. If either is positive, `poll` requests [ConsumerNetworkClient](#) to [pollNoWakeup](#).

Caution	FIXME Make the above more user-friendly
---------	---

`poll` returns the available `ConsumerRecords` directly when no [ConsumerInterceptors](#) are defined or passes them through [ConsumerInterceptors](#) using [onConsume](#).

Caution	FIXME Make the above more user-friendly, e.g. when could <code>interceptors</code> be empty?
---------	--

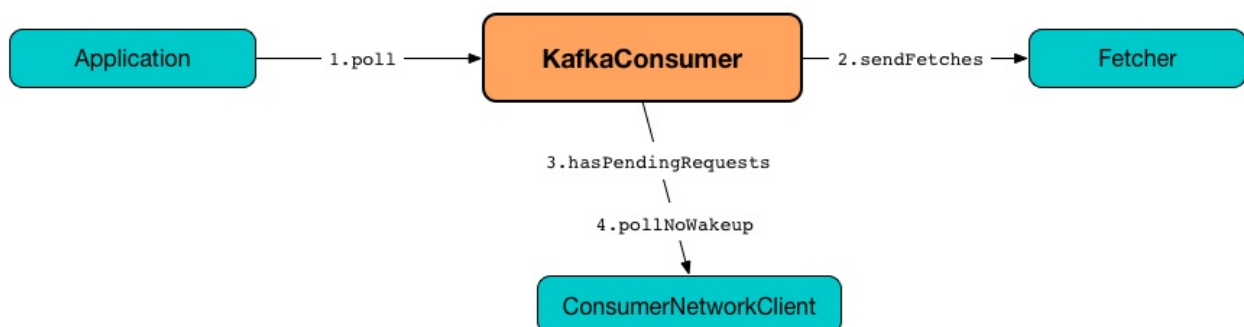


Figure 3. KafkaConsumer polls topics

Note	<code>poll</code> is a part of <a href="#">Consumer contract</a> to...FIXME
------	---

## commitSync Method

```
void commitSync()
```

Note	<code>commitSync</code> is part of <a href="#">Consumer Contract</a> to...FIXME.
------	--

`commitSync` ...FIXME

## seek Method

```
void seek(TopicPartition partition, long offset)
```

Note	<code>seek</code> is part of <a href="#">Consumer Contract</a> to...FIXME.
------	--

`seek` ...FIXME

## Getting Partitions For Topic — `partitionsFor` Method

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

## `endOffsets` Method

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

## `offsetsForTimes` Method

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

## `updateFetchPositions` Method

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

## Polling One-Off for ConsumerRecords per TopicPartition — `pollOnce` Internal Method

```
Map<TopicPartition, List<ConsumerRecord<K, V>>> pollOnce(long timeout)
```

`pollOnce` ...FIXME

Note	<code>pollOnce</code> is used exclusively when <code>KafkaConsumer</code> is requested to <a href="#">poll</a>
------	--

## 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`.

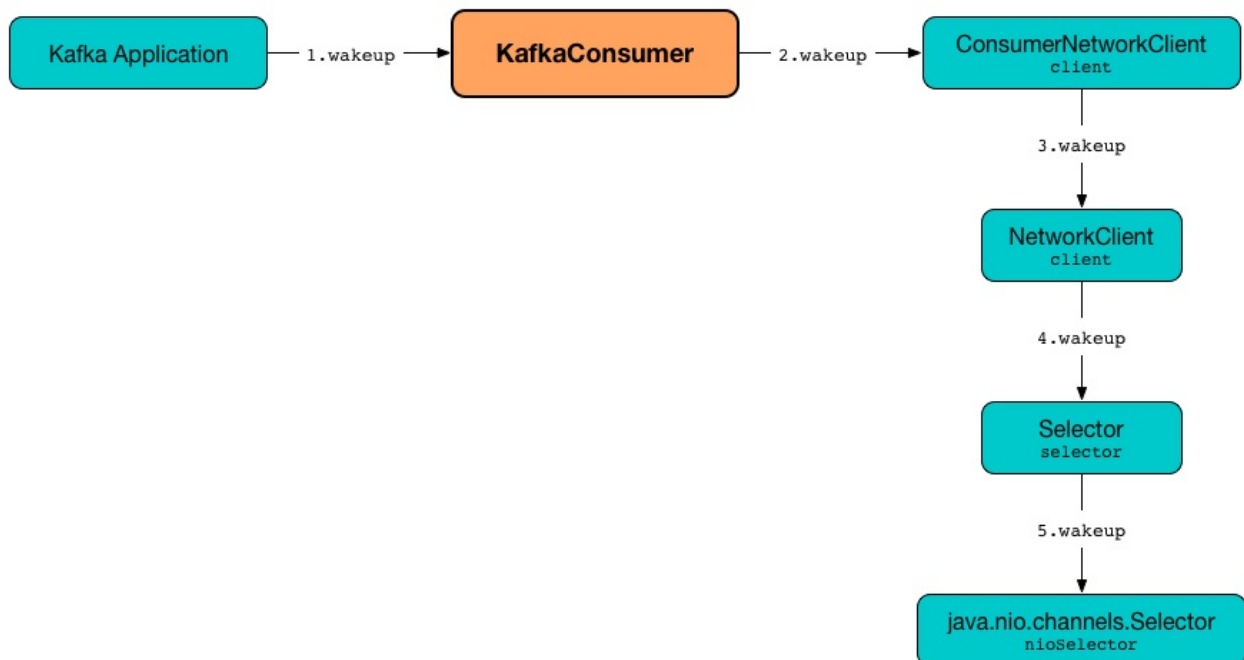


Figure 4. KafkaConsumer's wakeup Method

Note	<p>Quoting <code>wakeup</code> of Java's <a href="#">java.nio.channels.Selector</a> given <code>wakeup</code> simply passes through the intermediaries and in the end triggers it.</p> <p>Causes the first selection operation that has not yet returned to return immediately.</p> <p>Read about Selection in <a href="#">java.nio.channels.Selector</a>'s javadoc.</p>
------	--

Note	<code>wakeup</code> 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	<p><code>configureClusterResourceListeners</code> is used exclusively when <code>KafkaConsumer</code> is <a href="#">created</a> (to create the <a href="#">Metadata</a>) with the following input arguments:</p> <ul style="list-style-type: none"> <li>• <a href="#">key</a> and <a href="#">value</a> deserializers (defined when <code>KafkaConsumer</code> is created)</li> <li>• <a href="#">ConsumerInterceptors</a> from <a href="#">interceptor.classes</a> Kafka property</li> <li>• <a href="#">MetricsReporters</a> from <a href="#">metric.reporters</a> Kafka property</li> </ul>
------	---

# Consumer

Consumer is the [contract](#) for [Kafka consumers](#).

Note

[KafkaConsumer](#) is the main public class that Kafka developers use to write Kafka consumers.

## Consumer Contract

```
package org.apache.kafka.clients.consumer;

public interface Consumer<K, V> extends Closeable {
    void assign(Collection<TopicPartition> partitions);
    Set<TopicPartition> assignment();

    void subscribe(Collection<String> topics);
    void subscribe(Collection<String> topics, ConsumerRebalanceListener callback);
    void subscribe(Pattern pattern);
    void subscribe(Pattern pattern, ConsumerRebalanceListener callback);
    Set<String> subscription();

    void unsubscribe();

    ConsumerRecords<K, V> poll(long timeout);

    void commitSync();
    void commitSync(Map<TopicPartition, OffsetAndMetadata> offsets);

    void commitAsync();
    void commitAsync(OffsetCommitCallback callback);
    void commitAsync(Map<TopicPartition, OffsetAndMetadata> offsets, OffsetCommitCallback callback);

    void seek(TopicPartition partition, long offset);
    void seekToBeginning(Collection<TopicPartition> partitions);
    void seekToEnd(Collection<TopicPartition> partitions);

    long position(TopicPartition partition);
    OffsetAndMetadata committed(TopicPartition partition);
    Map<MetricName, ? extends Metric> metrics();
    List<PartitionInfo> partitionsFor(String topic);
    Map<String, List<PartitionInfo>> listTopics();
    Set<TopicPartition> paused();
    void pause(Collection<TopicPartition> partitions);
    void resume(Collection<TopicPartition> partitions);
    Map<TopicPartition, OffsetAndTimestamp> offsetsForTimes(Map<TopicPartition, Long> timestampsToSearch);
    Map<TopicPartition, Long> beginningOffsets(Collection<TopicPartition> partitions);
    Map<TopicPartition, Long> endOffsets(Collection<TopicPartition> partitions);

    void close();
    void close(long timeout, TimeUnit unit);

    void wakeup();
}
```

Table 1. Consumer Contract

Method	Description
assign	
commitSync	
poll	Polling topics for data
seek	
subscribe	
unsubscribe	
wakeup	

# Deserializer

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



# ConsumerConfig

Caution	FIXME
Caution	FIXME Describe <code>static</code> block when <code>ConsumerConfig</code> is created.

# ConsumerCoordinator

`ConsumerCoordinator` is a concrete [AbstractCoordinator](#) that...FIXME

`ConsumerCoordinator` is [created](#) exclusively when `KafkaConsumer` is [created](#).

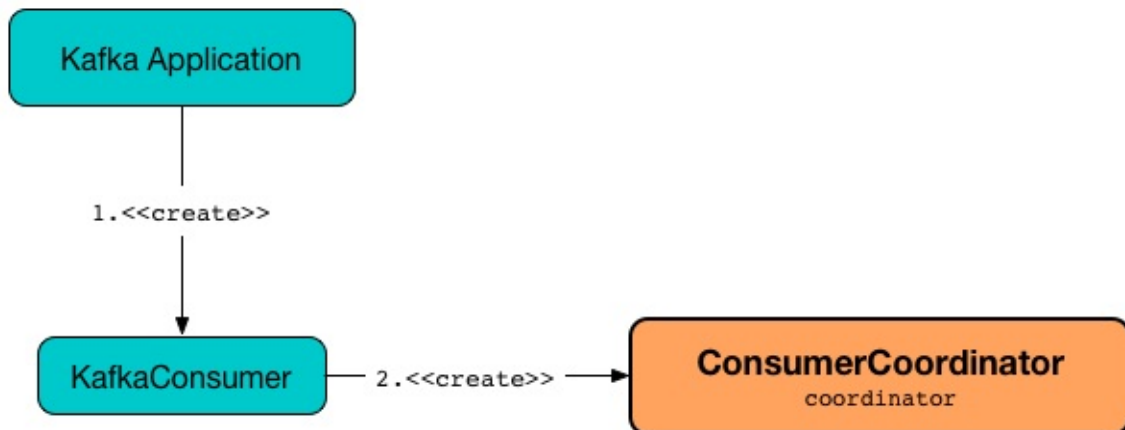


Figure 1. ConsumerCoordinator and KafkaConsumer

`KafkaConsumer` uses the [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>	<a href="#">rebalanceTimeoutMs</a>
<code>session.timeout.ms</code>	<a href="#">sessionTimeoutMs</a>
<code>heartbeat.interval.ms</code>	<a href="#">heartbeatIntervalMs</a>
<code>retry.backoff.ms</code>	<a href="#">retryBackoffMs</a>
<a href="#">enable.auto.commit</a>	<a href="#">autoCommitEnabled</a>
<code>auto.commit.interval.ms</code>	<a href="#">autoCommitIntervalMs</a>
<code>exclude.internal.topics</code>	<a href="#">excludeInternalTopics</a>
<code>internal.leave.group.on.close</code>	<a href="#">leaveGroupOnClose</a>

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

Table 2. ConsumerCoordinator's Internal Properties (e.g. Registries, Counters and Flags)

Name	Description
------	-------------

## Sending Asynchronous Auto-Commit of Offsets — maybeAutoCommitOffsetsAsync Method

```
void maybeAutoCommitOffsetsAsync(long now)
```

maybeAutoCommitOffsetsAsync ...FIXME

Note	<p>maybeAutoCommitOffsetsAsync is used when:</p> <ul style="list-style-type: none"> <li>• KafkaConsumer is requested to <a href="#">manually assign partitions</a></li> <li>• ConsumerCoordinator is requested to <a href="#">poll</a></li> </ul>
------	---

## Sending Synchronous Auto-Commit of Offsets — maybeAutoCommitOffsetsSync Internal Method

```
void maybeAutoCommitOffsetsSync(long timeoutMs)
```

maybeAutoCommitOffsetsSync ...FIXME

Note	<p>maybeAutoCommitOffsetsSync is used when ConsumerCoordinator is requested to <a href="#">close</a> or <a href="#">onJoinPrepare</a></p>
------	---

## doAutoCommitOffsetsAsync Internal Method

```
void doAutoCommitOffsetsAsync()
```

doAutoCommitOffsetsAsync ...FIXME

Note	<p>doAutoCommitOffsetsAsync is used when...FIXME</p>
------	--

## 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...FIXME

## `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` Method

```
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

## performAssignment Method

```
Map<String, ByteBuffer> performAssignment(  
    String leaderId,  
    String assignmentStrategy,  
    Map<String, ByteBuffer> allSubscriptions)
```

## Note

`performAssignment` is part of [AbstractCoordinator Contract](#) to...FIXME.

`performAssignment` ...FIXME

## maybeLeaveGroup Method

```
synchronized 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	<code>needRejoin</code> is part of <a href="#">AbstractCoordinator Contract</a> to...FIXME.
------	---

`needRejoin` ...FIXME

## timeToNextPoll Method

```
long timeToNextPoll(long now)
```

`timeToNextPoll` ...FIXME

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

## Polling for Coordinator Events — poll Method

```
void poll(long now, long remainingMs)
```

`poll` ...FIXME

Note	<code>poll</code> is used exclusively when <code>KafkaConsumer</code> is requested to <a href="#">pollOnce</a> .
------	--

## addMetadataListener Internal Method

```
void addMetadataListener()
```

`addMetadataListener` ...FIXME

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

## fetchCommittedOffsets Method

```
Map<TopicPartition, OffsetAndMetadata> fetchCommittedOffsets(Set<TopicPartition> partitions)
```

`fetchCommittedOffsets` ...FIXME

## Note

`fetchCommittedOffsets` is used when...FIXME

## Creating ConsumerCoordinator Instance

`ConsumerCoordinator` takes the following when created:

- `LogContext`
- [ConsumerNetworkClient](#)
- Group ID
- `rebalanceTimeoutMs`
- `sessionTimeoutMs`
- `heartbeatIntervalMs`
- Collection of `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 [Metadata](#) to [update](#) and [addMetadataListener](#).

# AbstractCoordinator

AbstractCoordinator is a [contract](#) for...FIXME

Note	<a href="#">ConsumerCoordinator</a> is the default AbstractCoordinator in Apache Kafka's Consumer API.
------	--

```
package org.apache.kafka.clients.consumer.internals;

abstract class AbstractCoordinator {
    void onJoinComplete(
        int generation,
        String memberId,
        String protocol,
        ByteBuffer memberAssignment);
    void onJoinPrepare(int generation, String memberId);
    Map<String, ByteBuffer> performAssignment(
        String leaderId,
        String protocol,
        Map<String, ByteBuffer> allMemberMetadata);
    ...FIXME
}
```

Table 1. AbstractCoordinator Contract

Method	Description
onJoinComplete	Used exclusively when AbstractCoordinator is requested to <a href="#">joinGroupIfNeeded</a> (and the request to <a href="#">initiateJoinGroup</a> succeeded)
onJoinPrepare	Used exclusively when AbstractCoordinator is requested to <a href="#">joinGroupIfNeeded</a> (and <a href="#">needsJoinPrepare</a> flag is on)
performAssignment	

Table 2. AbstractCoordinator's Internal Properties (e.g. Registries, Counters and Flags)

Name	Description
joinFuture	<code>RequestFuture&lt;ByteBuffer&gt;</code> for...FIXME
needsJoinPrepare	Controls when to execute <a href="#">onJoinPrepare</a> while performing <a href="#">joinGroupIfNeeded</a>
rejoinNeeded	Controls when to...FIXME



## Creating AbstractCoordinator Instance

`AbstractCoordinator` takes the following when created:

- `LogContext`
- [ConsumerNetworkClient](#)
- Group ID
- `rebalanceTimeoutMs`
- `sessionTimeoutMs`
- `heartbeatIntervalMs`
- `Metrics`
- Metric group prefix
- `Time`
- `retryBackoffMs`
- `leaveGroupOnClose` flag

### `joinGroupIfNeeded` Method

```
void joinGroupIfNeeded()
```

`joinGroupIfNeeded` ...FIXME

Note
<code>joinGroupIfNeeded</code> is used exclusively when <code>AbstractCoordinator</code> is requested to <a href="#">ensureActiveGroup</a> .

### `initiateJoinGroup` Internal Method

```
synchronized RequestFuture<ByteBuffer> initiateJoinGroup()
```

`initiateJoinGroup` ...FIXME

Note
<code>initiateJoinGroup</code> is used exclusively when <code>AbstractCoordinator</code> is requested to <a href="#">joinGroupIfNeeded</a> (when a rejoin is <a href="#">needed</a> or <a href="#">still in progress and incomplete</a> ).

## ensureActiveGroup Method

```
void ensureActiveGroup()
```

ensureActiveGroup ...FIXME

Note	ensureActiveGroup is used when ConsumerCoordinator is requested to poll for coordinator events
------	--

# 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] = {
      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
---------	-------

# 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

Enable `DEBUG` logging level for `org.apache.kafka.clients.consumer.internals.ConsumerNetworkClient` logger to see w happens inside.

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

```
log4j.logger.org.apache.kafka.clients.consumer.internals.ConsumerNetworkClient=DEBUG
```

Refer to [Logging](#).

## checkDisconnects Internal Method

```
void checkDisconnects(long now)
```

`checkDisconnects` ...FIXME

Note	<code>checkDisconnects</code> is used exclusively when <code>ConsumerNetworkClient</code> is requested to <a href="#">poll</a> .
------	--

## "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
------	--

## Ensuring Fresh 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

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

## `leastLoadedNode` Method

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

## `poll` Method

```
void poll(long timeout)
void poll(long timeout, long now, PollCondition pollCondition)
void poll(long timeout, long now, PollCondition pollCondition, boolean disableWakeup)
void poll(RequestFuture<?> future)
boolean poll(RequestFuture<?> future, long timeout)
```

poll ...FIXME

Note	<div><div>poll is used when:</div><ul style="list-style-type: none"><li>• KafkaConsumer polls once</li><li>• ConsumerNetworkClient does pollNoWakeup, awaitMetadataUpdate, awaitPendingRequests</li><li>• AbstractCoordinator does ensureCoordinatorReady and joinGroupIfNeeded</li><li>• ConsumerCoordinator commitOffsetsSync and fetchCommittedOffsets</li><li>• Fetcher is requested for topic metadata or retrieveOffsetsByTimes</li></ul></div>
------	---

Waiting for Metadata Update —

awaitMetadataUpdate Method

```
boolean awaitMetadataUpdate(long timeout)
```

awaitMetadataUpdate ...FIXME

Note	awaitMetadataUpdate is used when...FIXME
------	--

awaitPendingRequests

Method

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

pollNoWakeup

Method

```
void pollNoWakeup()
```

pollNoWakeup ...FIXME

Note	<p><code>pollNowWakeup</code> is used when:</p> <ul style="list-style-type: none"><li>• <code>KafkaConsumer</code> <a href="#">polls</a></li><li>• <code>AbstractCoordinator</code> <code>does</code> <code>maybeLeaveGroup</code></li><li>• <code>HeartbeatThread</code> <code>runs</code></li><li>• <code>ConsumerCoordinator</code> <code>does</code> <code>commitOffsetsAsync</code></li></ul>
------	--

## Creating ConsumerNetworkClient Instance

`ConsumerNetworkClient` takes the following when created:

- `LogContext`
- [KafkaClient](#)
- [Metadata](#)
- `Time`
- `retryBackoffMs`
- `requestTimeoutMs`

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

# 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 <a href="#">onJoinComplete</a>
<code>onPartitionsRevoked</code>	Used exclusively when <code>ConsumerCoordinator</code> is requested to <a href="#">onJoinPrepare</a>



# SubscriptionState

`SubscriptionState` allows for tracking the topics, partitions, and offsets assigned to a Kafka consumer.

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"><li>• <code>onJoinComplete</code> (that triggers <code>ConsumerRebalanceListener.onPartitionsAssigned</code>)</li><li>• <code>onJoinPrepare</code> (that triggers <code>ConsumerRebalanceListener.onPartitionsRevoked</code>)</li></ul>

## `subscribe` Method

```
void subscribe(Pattern pattern, ConsumerRebalanceListener listener)
```

`subscribe` ...FIXME

Note	<code>subscribe</code> is used exclusively when <code>KafkaConsumer</code> is requested to <code>subscribe to the topics that match a pattern with a ConsumerRebalanceListener</code> .
------	---

# Broker Nodes — Kafka Servers

Note	A <b>Kafka server</b> , a <b>Kafka broker</b> and a <b>Kafka node</b> all refer to the same concept and are synonyms (see the scaladoc of <a href="#">KafkaServer</a> ).
------	--

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 <a href="#">kafka-server-start.sh script</a> .
-----	---

## 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).

`createBroker`

Method

```
createBroker(id: Int, brokerInfoString: String): Broker
```

`createBroker` ...FIXME

Note	<code>createBroker</code> 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.

## 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 <b>New Consumer</b> .
------	--

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
<code>poll</code>	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	<code>subscribe</code> 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 subscrib 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](#)

# RequestCompletionHandler

`RequestCompletionHandler` is the [contract](#) to attach an action that is executed when a request is complete, i.e. the corresponding response has been received or there was a disconnection while handling the request.

```
package org.apache.kafka.clients;

public interface RequestCompletionHandler {
    public void onComplete(ClientResponse response);
}
```

Table 1. RequestCompletionHandler’s Known Implementations

Name	Description
<code>RequestFutureCompletionHandler</code>	
<code>TxnRequestHandler</code>	
<code>TransactionMarkerRequestCompletionHandler</code>	

# ClientResponse

ClientResponse is...FIXME

## onComplete Method

```
void onComplete()
```

onComplete triggers [RequestCompletionHandler](#) if defined.

Note
onComplete is used exclusively when <code>NetworkClient</code> <a href="#">completeResponses</a>

# Clusters

A Kafka **cluster** is the central data exchange backbone for an organization.

## 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.

Note

Quoting [Kafka 0.9.0.0](#):  
  
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 <a href="#">ConsumerGroupService Contract</a> .
------	--

`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 <a href="#">all consumer groups</a> 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, DescribeTopicOptions 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](#).

# AdminClient

AdminClient ...FIXME

AdminClient uses the `admin-client-network-thread` to poll continuously (using `ConsumerNetworkClient`).

Note	AdminClient is deprecated and is going to be replaced by KafkaAdminClient (after the public API becomes stable).
------	--

Table 1. ConsumerNetworkClient’s Internal Properties (e.g. Registries and Counters)

Name	Description
networkThread	admin-client-network-thread

Tip	<p>Enable <code>DEBUG</code> logging level for <code>kafka.admin.AdminClient</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.admin.AdminClient=DEBUG</pre> <p>Refer to <a href="#">Logging</a>.</p>
-----	---

## Creating AdminClient — create Method

```
create(config: AdminConfig): AdminClient
```

create ...FIXME

Note	create 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 <code>pendingFutures</code> ?
---------	--

When the future result has completed, `send` takes the response body for a successful result or reports a `RuntimeException` .

Note	<code>send</code> is used when <code>AdminClient</code> does <code>sendAnyNode</code> , <code>listGroups</code> , <code>getApiVersions</code> , <code>listGroupOffsets</code> and <code>describeConsumerGroupHandler</code> .
------	---

## 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	<code>findAllBrokers</code> is used when <code>AdminClient</code> does <code>awaitBrokers</code> , <a href="#">lists all groups per broker</a> and <a href="#">listAllBrokerVersionInfo</a> .
------	---

## 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	<code>sendAnyNode</code> is used when <code>AdminClient</code> is requested to <a href="#">find the coordinator</a> , <a href="#">finds all brokers</a> and <a href="#">deleteRecordsBefore</a> .
------	---

## findCoordinator Method

```
FIXME
```

```
findCoordinator ...FIXME
```

Note	<code>findCoordinator</code> is used when <code>KafkaConsumerGroupService</code> ...FIXME
------	---

## deleteRecordsBefore Method

```
FIXME
```

```
deleteRecordsBefore ...FIXME
```

Note	<code>deleteRecordsBefore</code> is used when <code>KafkaConsumerGroupService</code> ...FIXME
------	---

## listGroups Method

```
FIXME
```

```
listGroups ...FIXME
```

Note	<code>listGroups</code> is used when <code>KafkaConsumerGroupService</code> ...FIXME
------	--

## listAllBrokerVersionInfo Method

```
FIXME
```

`listAllBrokerVersionInfo` ...FIXME

Note	<code>listAllBrokerVersionInfo</code> is used when <code>KafkaConsumerGroupService</code> ...FIXME
------	--

## `awaitBrokers` Method

FIXME

`awaitBrokers` ...FIXME

Note	<code>awaitBrokers</code> is used when <code>KafkaConsumerGroupService</code> ...FIXME
------	--

## `listAllConsumerGroups` Method

FIXME

`listAllConsumerGroups` ...FIXME

Note	<code>listAllConsumerGroups</code> is used when <code>KafkaConsumerGroupService</code> ...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	<code>listAllGroups</code> is used when <code>AdminClient</code> does <a href="#">listAllConsumerGroups</a> and <a href="#">listAllGroupsFlattened</a> .
------	--

## `listAllGroupsFlattened` Method

```
listAllGroupsFlattened(): List[GroupOverview]
```

`listAllGroupsFlattened` simply [takes all groups](#) (across all brokers in a cluster).

Note	<code>listAllGroupsFlattened</code> is used exclusively when <code>AdminClient</code> <a href="#">lists all consumer groups</a> .
------	---

# 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
------	--

# ReassignPartitionsCommand

ReassignPartitionsCommand is...FIXME

## Starting ReassignPartitionsCommand on Command Line — main Entry Method

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

main ...FIXME

## executeAssignment Method

```
executeAssignment(
  zkUtils: ZkUtils,
  adminClientOpt: Option[AdminClient],
  opts: ReassignPartitionsCommand.ReassignPartitionsCommandOptions): Unit
```

```
executeAssignment(
  zkUtils: ZkUtils,
  adminClientOpt: Option[AdminClient],
  reassignmentJsonString: String,
  throttle: ReassignPartitionsCommand.Throttle,
  timeoutMs: Long = 10000L): Unit
```

executeAssignment ...FIXME

### Note

`executeAssignment` is used exclusively when `ReassignPartitionsCommand` is started on command line with `execute` option.

## 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 <a href="#">reassignPartitions</a>
------	--

# TopicCommand

kafka.admin.TopicCommand

```
./bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic my-topic
```

```
./bin/kafka-topics.sh --zookeeper localhost:2181 --describe --topic my-topic
```

# 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

# 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 \  
  --zookeeper localhost:2181 \  
  --alter \  
  --entity-type topics \  
  --entity-name test \  
  --add-config retention.ms=5000
```

## kafka-topics.sh Shell Script

`kafka-topics.sh` is a shell script that...FIXME

# Kafka Properties

Table 1. Properties

Name	Default Value	
<code>auto.commit.interval.ms</code>		How often (in milliseconds) consumer o enabled
<code>auto.offset.reset</code>	latest	<p>Reset policy — what to do when there is any more on the server (e.g. because the</p> <ul style="list-style-type: none"> <li>• <b>earliest</b> — automatically reset the c</li> <li>• <b>latest</b> — automatically reset the off</li> <li>• <b>none</b> — throw an exception to the c group</li> <li>• anything else: throw an exception t</li> </ul>
<code>authorizer.class.name</code>		
<code>broker.id</code>	-1	<p>The broker id for this server.</p> <p>If unset, a unique broker id will be gene id's and user configured broker id's, ger</p> <p>Use <a href="#">KafkaConfig.brokerId</a> to access the</p>
<code>background.threads</code>	10	<p>The number of threads to use for variou</p> <p>As least 1 thread is required</p>
<code>bootstrap.servers</code>	(empty)	A comma-separated list of <code>host:port</code> p <a href="#">cluster</a> , e.g. <code>localhost:9092</code> or <code>localho</code>
<code>broker.rack</code>		
<code>check.crcs</code>		<p>Automatically check the CRC32 of the r corruption to the messages occurred. T seeking extreme performance.</p> <p>Use <code>ConsumerConfig.CHECK_CRCS_CONFIG</code></p>
<code>client.id</code>	(random-generated)	
		Enables topic deletion

		Note	Deleting topic through the a
<code>enable.auto.commit</code>	<code>true</code>	<p>When enabled (i.e. <code>true</code>) consumer of <i>consumer auto commit</i>) every <a href="#">auto.com</a></p> <p>When disabled, offsets have to be comr <a href="#">KafkaConsumer.commitSync</a> or asynch the position of a consumer using <a href="#">KafkaC</a></p> <p>Used when <code>KafkaConsumer</code> is <a href="#">created</a> a</p>	
<code>fetch.min.bytes</code>		<p>The minimum amount of data the server available the request will wait for that m default setting of 1 byte means that fetc available or the fetch request times out than 1 will cause the server to wait for la throughput a bit at the cost of some add</p> <p>Use <code>ConsumerConfig.FETCH_MIN_BYTES_CO</code></p>	
<code>fetch.max.bytes</code>		<p>The maximum amount of data the server batches by the consumer, and if the first larger than this value, the record batch progress. As such, this is not a absolute broker is defined via <a href="#">message.max.byte</a> that the consumer performs multiple fet</p> <p>Use <code>ConsumerConfig.FETCH_MAX_BYTES_CO</code></p>	
<code>fetch.max.wait.ms</code>		<p>The maximum amount of time the server sufficient data to immediately satisfy the</p> <p>Use <code>ConsumerConfig.FETCH_MAX_WAIT_MS_C</code></p>	
<code>group.id</code>		The name of the consumer group the co	
<code>heartbeat.interval.ms</code>		The expected time between heartbeats management facilities.	
<code>host.name</code>	(empty)	The hostname the <a href="#">default endpoint</a> in s	
<code>inter.broker.protocol.version</code>			
<code>interceptor.classes</code>	(empty)	<p>Comma-separated list of <a href="#">ConsumerInte</a></p> <pre>props.put(ConsumerConfig.INTERCEPTOR_</pre>	
<code>listeners</code>	(empty)	The addresses the socket server listens	



<code>log.dir</code>	<code>/tmp/kafka-logs</code>	The directory in which the log data is kept.
<code>log.dirs</code>	(empty)	The directories in which the log data is kept. Use <code>KafkaConfig.logDirs</code> to access the configuration.
<code>key.deserializer</code>		How to deserialize message keys.
<code>max.block.ms</code>		
<code>max.partition.fetch.bytes</code>		<p>The maximum amount of data per-partition that the consumer can fetch. If the first record batch in the batch is over the limit, the batch will still be returned to the consumer. The default batch size accepted by the broker is <code>max.message.bytes</code> (topic config).</p> <p>Use <code>ConsumerConfig.MAX_PARTITION_FETCH_BYTES</code>.</p> <div> <div>Note</div> <div>Use <code>fetch.max.bytes</code> for limit.</div> </div>
<code>max.poll.records</code>	500	<p>(KafkaConsumer) The maximum number of records to return from a single poll. The default setting ( -1 ) sets no upper limit. The consumer will return as soon as either any data is available or the number of records returned equals <code>max.poll.records</code>.</p> <p><code>max.poll.records</code> was added to Kafka in version 0.10.0.</p> <p>From <a href="#">kafka-clients</a> mailing list:</p> <div> <pre><code>max.poll.records</code> only controls the number of records returned from a single poll. The consumer will try to prefetch those records and return them as soon as they are available. It will not return the same topic partition if there are enough records from other topic partitions if the data from the last poll is not yet returned (<code>max.poll.records</code>).</pre> </div> <p>Use <code>ConsumerConfig.MAX_POLL_RECORDS_CONFIG</code>.</p> <hr/> <p>Internally, <code>max.poll.records</code> is used exclusively by the consumer.</p>
<code>metadata.max.age.ms</code>		
<code>metric.reporters</code>	<code>JmxReporter</code>	The list of fully-qualified classes names of the reporters to use.
<code>metrics.num.samples</code>		Number of samples to compute metrics.
<code>metrics.sample.window.ms</code>		Time window (in milliseconds) a metrics sample covers.

<code>min.insync.replicas</code>	1	<p>When a producer sets acks to "all" (or "all" and "wait_for_replicas" that must acknowledge a write).</p> <p>If this minimum cannot be met, then the producer will throw an exception (<code>NotEnoughReplicasAfterAppend</code>).</p> <p>When used together, <code>min.insync.replicas</code> and <code>wait_for_replicas</code> guarantees.</p> <p>A typical scenario would be to create a topic with 3 replicas, 1 leader, 2 followers, and produce with acks of "all". This way all replicas do not receive a write.</p>
<code>num.io.threads</code>	8	The number of threads that <code>KafkaServer</code> uses for I/O.
<code>num.network.threads</code>	3	The number of threads that <code>SocketServer</code> uses for network I/O.
<code>port</code>	(empty)	The port the <code>default endpoint</code> in <code>SocketServer</code> listens on.
<code>rebalance.timeout.ms</code>		The maximum allowed time for each worker to complete a rebalance.
<code>receive.buffer.bytes</code>		The hint about the size of the TCP network socket for reading data. If the value is -1, the OS chooses the value.
<code>replica.lag.time.max.ms</code>		The maximum allowed time for a replica to be out of sync with the leader.
<code>replica.socket.timeout.ms</code>		The timeout for the replica socket.
<code>reserved.broker.max.id</code>	-1	The maximum ID of a reserved broker.
<code>retry.backoff.ms</code>		<p>Time to wait before attempting to retry a request after a failed attempt.</p> <p>Use <code>ConsumerConfig.RETRY_BACKOFF_MS_CONFIG</code>.</p>
<code>request.timeout.ms</code>		<p>The configuration controls the maximum time to wait for a response to a request. If the response is not received within the timeout, the request is considered failed and the consumer may retry the request if necessary or fail the request if retries are exhausted.</p> <p>Use <code>ConsumerConfig.REQUEST_TIMEOUT_MS_CONFIG</code>.</p>
<code>sasl.enabled.mechanisms</code>		
<code>send.buffer.bytes</code>		The hint about the size of the TCP network socket for sending data. If the value is -1, the OS chooses the value.
<code>session.timeout.ms</code>	10000	The timeout used to detect worker failure.
<code>value.deserializer</code>		How to deserialize message values.

```
// requires org.apache.kafka:connect-runtime:0.10.0.1 dependency

import org.apache.kafka.connect.runtime.distributed.DistributedConfig
DistributedConfig.SESSION_TIMEOUT_MS_CONFIG
```

Caution	FIXME How to know the current value of a setting on a producer's and a consumer's side?
---------	---

## 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 <code>bootstrap.servers</code> for bootstrapping.
Tip	Use <code>org.apache.kafka.clients.CommonClientConfigs.BOOTSTRAP_SERVERS_CONFIG</code> 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.

Tip

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"
```

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
```

	<p>With logback's configuration (as described in the <a href="#">above tip</a>) you may see the follow</p> <pre>SLF4J: Class path contains multiple SLF4J bindings. SLF4J: Found binding in [jar:file:/Users/jacek/.ivy2/cache/org.slf4j/slf4j-log4j-1.7.12.jar:/Users/jacek/.ivy2/cache/org.slf4j/slf4j-log4j-1.7.12.jar] SLF4J: Found binding in [jar:file:/Users/jacek/.ivy2/cache/ch.qos.logback/logback-classic-1.2.3.jar:/Users/jacek/.ivy2/cache/ch.qos.logback/logback-classic-1.2.3.jar] SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]</pre>
Note	<p>Commenting out <code>logback-classic</code> in <code>build.sbt</code> resolves it.</p> <p><code>build.sbt</code></p> <pre>val logback = "1.2.3" libraryDependencies += "ch.qos.logback" % "logback-core" % logback //libraryDependencies += "ch.qos.logback" % "logback-classic" % logback</pre> <p>FIXME: Explain why the commenting out is required?</p>

# 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 "*PlaintextProducerSendTest"
```

## 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_notifica
tion, 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/kafka-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	<a href="#">ches/docker-kafka</a> 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 <a href="#">official tutorial</a> 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!
```



# Configurable Contract

Configurable is...FIXME

# Reconfigurable Contract

Reconfigurable is...FIXME

# DescribeLogDirsRequest

DescribeLogDirsRequest is...FIXME

Table 1. DescribeLogDirsRequest’s Internal Properties (e.g. Registries, Counters and Flags)

Name	Description
topicPartitions	TopicPartitions that...FIXME Used when...FIXME

## isAllTopicPartitions Method

```
boolean isAllTopicPartitions()
```

isAllTopicPartitions is true only if the [topicPartitions](#) is not specified.

Note	isAllTopicPartitions is used when...FIXME
------	---

# 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 <a href="#">processNotification</a>

Table 2. NotificationHandlers

NotificationHandler	Description
<a href="#">rawHandler</a>	
<a href="#">ConfigChangedNotificationHandler</a>	
<a href="#">TokenChangedNotificationHandler</a>	

# ZkNodeChangeListener

ZkNodeChangeListener 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
------	----------------------------

# SimpleAclAuthorizer

SimpleAclAuthorizer is...FIXME

## startZkChangeListenerListeners Method

```
startZkChangeListenerListeners(): Unit
```

startZkChangeListenerListeners ...FIXME

Note	startZkChangeListenerListeners is used exclusively when SimpleAclAuthorizer is requested to <a href="#">configure</a> .
------	---

## configure Method

```
configure(javaConfigs: util.Map[String, _]): Unit
```

Note	configure is part of the <a href="#">Configurable Contract</a> to...FIXME.
------	--

configure ...FIXME

# Scheduler Contract

`Scheduler` is the [contract](#) of [task schedulers](#) that allow for [scheduling tasks](#).

Note	<a href="#">Kafka Server and Periodic Tasks</a> 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"><li><code>KafkaController</code> is requested to <a href="#">onControllerResignation</a></li><li><code>GroupMetadataManager</code> is requested to <a href="#">shutdown</a></li><li><code>LogManager</code> is requested to <a href="#">deleteLogs</a></li><li><code>KafkaScheduler</code> is requested to <a href="#">startup</a> and <a href="#">ensureRunning</a></li></ul>
<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"><li><code>KafkaController</code> is requested to <a href="#">onControllerResignation</a></li><li><code>GroupMetadataManager</code> is requested to <a href="#">shutdown</a></li><li><code>TransactionCoordinator</code> is requested to <a href="#">shutdown</a></li><li><code>KafkaServer</code> is requested to <a href="#">shutdown</a></li></ul>

	<ul style="list-style-type: none"> <li><code>ZooKeeperClient</code> is requested to <a href="#">close</a></li> </ul>
<code>schedule</code>	<p>Schedules a task</p> <p>Used when:</p> <ul style="list-style-type: none"> <li><code>KafkaController</code> is requested to <a href="#">onControllerFailover</a> and <a href="#">scheduleAutoLeaderRebalanceTask</a></li> <li><code>GroupMetadataManager</code> is requested to <a href="#">start up</a>, <a href="#">scheduleLoadGroupAndOffsets</a>, <a href="#">removeGroupsForPartition</a> and <a href="#">scheduleHandleTxnCompletion</a></li> <li><code>TransactionCoordinator</code> is requested to <a href="#">start up</a></li> <li><code>TransactionStateManager</code> is requested to <a href="#">enableTransactionalIdExpiration</a>, <a href="#">loadTransactionsForTxnTopicPartition</a> and <a href="#">removeTransactionsForTxnTopicPartition</a></li> <li><code>Log</code> is <a href="#">created</a> and requested to <a href="#">roll</a> and <a href="#">asyncDeleteSegment</a></li> <li><code>LogManager</code> is requested to <a href="#">start up</a> and <a href="#">deleteLogs</a></li> <li><code>ReplicaManager</code> is requested to <a href="#">startHighWaterMarksCheckPointThread</a> and <a href="#">startup</a></li> <li><code>KafkaScheduler</code> is requested to <a href="#">scheduleOnce</a></li> </ul>
<code>startup</code>	<p>Initializes the scheduler so it is ready to schedule tasks</p> <p>Used when:</p> <ul style="list-style-type: none"> <li><code>KafkaController</code> is requested to <a href="#">onControllerFailover</a></li> <li><code>GroupMetadataManager</code> is requested to <a href="#">start up</a></li> <li><code>TransactionCoordinator</code> is requested to <a href="#">start up</a></li> <li><code>KafkaServer</code> is requested to <a href="#">[kafka-KafkaServer#startup]</a></li> <li><code>ZooKeeperClient</code> is <a href="#">created</a></li> </ul>
Note	<p><a href="#">KafkaScheduler</a> is the one and only known implementation of the <a href="#">Scheduler Contract</a> in Apache Kafka.</p>



# 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>	<a href="#">KafkaScheduler</a> with 1 daemon thread with <b>zk-session-expiry-handler</b> 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

KafkaZkClient is...FIXME

# AdminZkClient

AdminZkClient is...FIXME

fetchEntityConfig

Method

```
fetchEntityConfig(rootEntityType: String, sanitizedEntityName: String): Properties
```

fetchEntityConfig ...FIXME

Note	fetchEntityConfig is used when...FIXME
------	--

# 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	ZkAclChangeStore is a Scala sealed trait and so all the <a href="#">available implementations</a> are in a single compilation unit (i.e. the file with the Scala source code).
------	--

Table 1. (Subset of) ZkAclChangeStore Contract

Property	Description
aclChangePath	Used when...FIXME
decode	Used when...FIXME
encode	Used when...FIXME

Table 2. ZkAclChangeStores

ZkAclChangeStore	Description
ExtendedAclChangeStore	
LiteralAclChangeStore	

## 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 <a href="#">startZkChangeListeners</a> .
------	--

# 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.
---------	--



## 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.