Table of Contents

Introduction 1.1

Overview of Kafka 1.2

Kafka Features

Partition Leader Election 2.1

- Preferred Replica Election 2.1.1
- Demo: Using kafka-leader-election.sh 2.1.2

Log Cleanup Policies 2.2

Kafka Controller Election 2.3

Topic Replication 2.4

Topic Deletion 2.5

Transactional Producer 2.6

Idempotent Producer 2.7

Log Management System 2.8

Kafka Cluster

Broker 3.1

EndPoint 3.2

Partition 3.3

Replica 3.4

PartitionStateStore 3.5

- ZkPartitionStateStore 3.5.1

ReplicationUtils 3.6

CoreUtils 3.7

Kafka Security / Transport Layer Security (TLS) and Secure Sockets Layer (SSL)

Kafka Security / Communications Security 4.1
Demo: Securing Communication Between Clients and Brokers Using SSL 4.1.1
SslFactory 4.2
SslEngineBuilder 4.3
SslPrincipalMapper 4.4
SslConfigs 4.5
BrokerSecurityConfigs 4.6

Kafka Security / SSL Authentication and Authorization

SSL Authentication and Authorization 5.1
Demo: Secure Inter-Broker Communication 5.1.1
Demo: SSL Authentication 5.1.2
Demo: ACL Authorization 5.1.3
Authorizer 5.2
AclAuthorizer 5.2.1
AuthorizerWrapper 5.2.2
KafkaPrincipal 5.3
KafkaPrincipalBuilder 5.4
DefaultKafkaPrincipalBuilder 5.5

Kafka Networking Layer

ChannelBuilder 6.1
SaslChannelBuilder 6.1.1
SslChannelBuilder 6.1.2
SslAuthenticator 6.1.2.1
ChannelBuilders 6.2
Authenticator 6.3
KafkaChannel 6.4
Selectable 6.5
Selector 6.5.1

Kafka Tools (Utilities)

Kafka Tools 7.1
kafka-configs.sh 7.2
kafka-topics.sh 7.3
kafka-leader-election.sh 7.4
LeaderElectionCommand 7.4.1
kafka-acls.sh 7.5
AclCommand 7.5.1
JAuthorizerService 7.5.1.1
AclCommandService 7.5.2
AdminClientService 7.5.3
kafka-consumer-groups.sh 7.6
ConsumerGroupCommand 7.6.1
ConsumerGroupService 7.6.2
BrokerApiVersionsCommand 7.7
JmxTool 7.8
GetOffsetShell 7.9
ReplicaVerificationTool 7.10
kafka-dump-log.sh 7.11
DumpLogSegments 7.11.1
OffsetsMessageParser 7.11.2
TransactionLogMessageParser 7.11.3
ConfigCommand 7.12
ReassignPartitionsCommand — Partition Reassignment on Command Line 7.13
TopicCommand — Topic Management on Command Line 7.14
KafkaAdminClient 7.15
AdminClient 7.15.1
AdminMetadataManager 7.15.2
AdminClientRunnable 7.15.3
Admin 7.15.4
AdminUtils 7.16

Controller Broker

Controller Broker 8.1
KafkaController 8.2
StateChangeLogger 8.3
ControllerContext 8.4
ControllerState 8.5
PartitionStateMachine 8.6
ZkPartitionStateMachine 8.6.1
Election 8.6.1.1
PartitionLeaderElectionAlgorithms 8.6.1.2
ReplicaStateMachine 8.7
ZkReplicaStateMachine 8.7.1
ControllerEventManager 8.8
ControllerEventThread 8.8.1
QueuedEvent 8.8.2
ControllerEventProcessor 8.9
ControllerEvent 8.10
AutoPreferredReplicaLeaderElection 8.10.1
BrokerChange 8.10.2
LeaderAndIsrResponseReceived 8.10.3
PreferredReplicaLeaderElection 8.10.4
Reelect 8.10.5
Startup 8.10.6
TopicDeletion 8.10.7
TopicUncleanLeaderElectionEnable 8.10.8
UncleanLeaderElectionEnable 8.10.9
ControllerChannelManager 8.11
RequestSendThread 8.11.1
ControllerBrokerRequestBatch 8.12
AbstractControllerBrokerRequestBatch 8.12.1
TopicDeletionManager 8.13
AbstractControlRequest 8.14
LeaderAndIsrRequest 8.14.1

StopReplicaRequest 8.14.2
UpdateMetadataRequest 8.14.3

Kafka Clients / Producer API

KafkaProducer 9.1
Producer 9.1.1
ProducerConfig 9.2
ProducerRecord 9.3
Callback 9.4
Partitioner 9.5
ProducerInterceptor 9.6

Internals of Kafka Producer

Sender 10.1
RecordAccumulator 10.2
ProducerBatch 10.2.1
ProducerInterceptors 10.3
DefaultPartitioner 10.4
TransactionManager 10.5

Kafka Broker Services

KafkaServer — Kafka Broker 11.1
Kafka Server and Periodic Tasks 11.2
AdminManager 11.3
DelegationTokenManager 11.4
DynamicConfigManager 11.5
ConfigHandler 11.5.1
BrokerConfigHandler 11.5.2
ClientIdConfigHandler 11.5.3
TopicConfigHandler 11.5.4
UserConfigHandler 11.5.5

DynamicBrokerConfig 11.6
BrokerReconfigurable 11.6.1
DynamicConnectionQuota 11.6.1.1
DynamicListenerConfig 11.6.1.2
DynamicThreadPool 11.6.1.3
DynamicClientQuotaCallback 11.6.2
DynamicLogConfig 11.6.3
DynamicMetricsReporters 11.6.4
FetchManager 11.7
GroupCoordinator 11.8
GroupMetadataManager 11.8.1
GroupMetadata 11.8.2
Kafka 11.9
KafkaApis 11.10
KafkaHealthcheck 11.11
KafkaServerStartable 11.12
KafkaConfig 11.13
KafkaMetricsReporter 11.14
KafkaRequestHandlerPool 11.15
KafkaRequestHandler 11.15.1
LogManager 11.16
Log 11.16.1
LogSegment 11.16.2
LogCleanerManager 11.16.3
LogCleaner 11.16.4
CleanerConfig 11.16.4.1
CleanerThread 11.16.4.2
Cleaner 11.16.4.3
CleanedTransactionMetadata 11.16.4.4
ProducerStateManager 11.16.5
LogConfig 11.16.6
RollParams 11.16.7
AbstractIndex 11.16.8
OffsetIndex 11.16.8.1

TimeIndex 11.16.8.2
TransactionIndex 11.16.9
LazyIndex 11.16.10
MetadataCache 11.17
OffsetConfig 11.18
ReplicaManager 11.19
ReplicaFetcherManager 11.19.1
ReplicaFetcherThread 11.19.1.1
ReplicaAlterLogDirsManager 11.19.2
ReplicaAlterLogDirsThread 11.19.2.1
AbstractFetcherManager 11.19.3
AbstractFetcherThread 11.19.3.1
ReplicaFetcherBlockingSend 11.19.4
ReplicationQuotaManager 11.19.5
LogDirFailureHandler 11.19.6
ShutdownableThread 11.20
SocketServer 11.21
Network Processor Thread 11.21.1
RequestChannel 11.22
RequestChannel.Request 11.22.1
TransactionCoordinator 11.23
TransactionMarkerChannelManager 11.23.1
InterBrokerSendThread 11.23.2
TransactionStateManager 11.24
QuotaManagers 11.25
ZkUtils 11.26
ZKRebalancerListener 11.27

Kafka Performance Metrics

KafkaMetricsGroup 12.1
BrokerTopicStats 12.2
BrokerTopicMetrics 12.3

Kafka Clients / Consumer API

Consumer Contract — Kafka Clients for Consuming Records 13.1

- KafkaConsumer 13.1.1
- MockConsumer 13.1.2

ConsumerRecord 13.2

OffsetAndMetadata 13.3

OffsetAndTimestamp 13.4

OffsetCommitCallback 13.5

ConsumerRebalanceListener 13.6

ConsumerConfig — Configuration Properties for KafkaConsumer 13.7

CommitFailedException 13.8

InvalidOffsetException 13.9

NoOffsetForPartitionException 13.10

OffsetOutOfRangeException 13.11

RetriableCommitFailedException 13.12

ConsumerInterceptor 13.13

PartitionAssignor Contract 13.14

- RangeAssignor 13.14.1
- RoundRobinAssignor 13.14.2
- StickyAssignor 13.14.3
- AbstractPartitionAssignor 13.14.4

ConsumerCoordinator 13.15

- AbstractCoordinator Contract 13.15.1
- HeartbeatThread 13.15.2
- GroupCoordinatorMetrics 13.15.3

ConsumerNetworkClient 13.16

ConsumerMetrics 13.17

Fetcher 13.18

RequestFutureListener 13.19

SubscriptionState 13.20

RequestFuture 13.21

RequestFutureAdapter Contract 13.22

- CoordinatorResponseHandler Contract 13.22.1

FindCoordinatorResponseHandler 13.22.2
HeartbeatResponseHandler 13.22.3
JoinGroupResponseHandler 13.22.4
OffsetCommitResponseHandler 13.22.5
SyncGroupResponseHandler 13.22.6

Kafka Architecture

Broker Nodes — Kafka Servers 14.1
Topics 14.2
Messages 14.3
Kafka Clients 14.4
Producers 14.4.1
Consumers 14.4.2
Clusters 14.5

Kafka Monitoring (Metrics)

Metrics 15.1
Sensor 15.2
MetricsReporter 15.3
ProducerMetrics 15.4
SenderMetrics 15.5

Kafka Configuration

Properties 16.1
bootstrap.servers 16.1.1
client.id 16.1.2
enable.auto.commit 16.1.3
group.id 16.1.4
retry.backoff.ms 16.1.5
Logging 16.2

Tips and Tricks

Gradle Tips 17.1
Zookeeper Tips 17.2
Kafka in Scala REPL for Interactive Exploration 17.3
Running Kafka Broker in Docker 17.4

Kafka Clients

KafkaClient 18.1
NetworkClient — Non-Blocking Network KafkaClient 18.1.1
RequestCompletionHandler Contract 18.2
MetadataUpdater 18.3
DefaultMetadataUpdater 18.3.1
Metadata 18.4
Listener Contract — Intercepting Metadata Updates 18.4.1
ClientRequest 18.5
ClientResponse 18.6
StaleMetadataException 18.7
NetworkClientUtils 18.8
CommonClientConfigs 18.9
ClientUtils 18.10

Kafka Common

Cluster 19.1
Cluster (deprecated) 19.1.1
ClusterConnectionStates 19.2
ClusterResourceListener (and ClusterResourceListeners Collection) 19.3
TopicConfig 19.4
NotificationHandler Contract 19.5
ZkNodeChangeNotificationListener 19.6
Configurable Contract 19.7
Reconfigurable 19.8

MemoryRecordsBuilder 19.9
Records 19.10
FileRecords 19.10.1
MemoryRecords 19.10.2
AbstractRecords 19.10.3
BaseRecords 19.10.4

Kafka Common / Requests

AbstractRequestResponse Contract 20.1
AbstractRequest Contract 20.1.1
AbstractRequest.Builder Contract 20.1.2
AbstractResponse 20.1.3
DescribeLogDirsRequest 20.2
DescribeConfigsRequest 20.3
ElectPreferredLeadersRequest 20.4
FetchRequest 20.5
FindCoordinatorRequest 20.6
FindCoordinatorResponse 20.7
HeartbeatRequest 20.8
JoinGroupRequest 20.9
JoinGroupResponse 20.10
MetadataRequest 20.11
MetadataResponse 20.12
OffsetCommitRequest 20.13
ProduceRequest 20.14
SyncGroupRequest 20.15
RequestContext 20.16

Kafka Common / Serialization

Serializer Contract 21.1
Deserializer Contract 21.2
Serde Contract 21.3

Serdes Factory Object 21.4

Varia / Misc

KafkaScheduler 22.1
Scheduler 22.2

Zookeeper

ZooKeeperClient 23.1
KafkaZkClient 23.2
AdminZkClient 23.3
ZkAclChangeStore 23.4
BrokerIdZNode 23.5

Kafka Connect

WorkerGroupMember 24.1
ConnectDistributed 24.2

Kafka Demos

Demo: Kafka Controller Election 25.1

Appendix

Further reading or watching 26.1

The Internals of Apache Kafka 2.4.0

Welcome to **The Internals of Apache Kafka** online book!

I'm Jacek Laskowski, a freelance IT consultant specializing in Apache Spark, Apache Kafka, Delta Lake and Kafka Streams.

I'm very excited to have you here and hope you will enjoy exploring the internals of Apache Kafka as much as I have.

> I write to discover what I know.

— Flannery O'Connor

Tip	I'm also writing other books in the "The Internals Of" series about Apache Spark, Spark SQL, Spark Structured Streaming, Delta Lake, and Kafka Streams.

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

Now, let me introduce you to Apache Kafka.

Overview of Kafka

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

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

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

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

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

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

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

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

Partition Leader Election

Partition Leader Election is a process of electing a broker to be the leader of a partition.

Use kafka-leader-election.sh utility for preferred or unclean leader election.

Tip	Consult Demo: Using kafka-leader-election.sh.

Note	`kafka-preferred-replica-election.sh` tool has been deprecated since Kafka 2.4.0 (cf. KIP-460: Admin Leader Election RPC).

Observe `state.change.logger` (default: `state-change.log`) to trace the process in the logs.

Internally, Kafka controller uses Election utility (and PartitionLeaderElectionAlgorithms) for the algorithms for partition leader election.

Preferred Partition Leader Election

Preferred Partition Leader Election is…FIXME

Unclean Partition Leader Election

Unclean Partition Leader Election allows a non-ISR replica to be elected as a partition leader (as the last resort, even though doing so may result in data loss).

unclean.leader.election.enable configuration property is used to enable it cluster-wide (for any topic) or per topic.

Enable INFO logging level for kafka.controller.KafkaController logger to observe the process in the logs.

Preferred Replica Leader Election (Preferred Replica Election)

Preferred Replica Leader Election (*Preferred Replica Election*) is a process…FIXME

Tip	Read up on Preferred Replica Leader Election in the official documentation.

Preferred Replica Election is triggered using kafka-leader-election.sh shell script for partitions as defined in a JSON file.

Preferred Replica Election can also be triggered using AdminClient that sends out the request to the controller broker (that simply requests the `ReplicaManager` to trigger the election)

Tip	Use KafkaController logging to monitor the process.

```
INFO [Controller id=1001] Starting preferred replica leader election for partitions  (
kafka.controller.KafkaController)
DEBUG [Controller id=1001] PreferredReplicaLeaderElection waiting: Set(), results: Map
(t1-0 -> ApiError(error=NONE, message=null), t1-1 -> ApiError(error=NONE, message=null
)) (kafka.controller.KafkaController)
```

In the very end, it is one of the many responsibilities of the controller broker to actually perform Preferred Replica Leader Election.

ACL

Either tool requires a principal to be authorized to perform `Alter` operation on `cluster` resource.

```
$ kafka-acls.sh \
    --authorizer-properties zookeeper.connect=zookeeper:2181 \
    --list

// Disable Alter on cluster resource
// The principal type is always User for the simple authorizer
$ kafka-acls.sh \
    --authorizer-properties zookeeper.connect=zookeeper:2181 \
    --add \
    --cluster \
    --operation Alter \
    --deny-principal User:ANONYMOUS
Adding ACLs for resource `Cluster:LITERAL:kafka-cluster`:
```

```
        User:ANONYMOUS has Deny permission for operations: Alter from hosts: *

Current ACLs for resource `Cluster:LITERAL:kafka-cluster`:
        User:ANONYMOUS has Deny permission for operations: Alter from hosts: *

$ kafka-acls.sh \
    --authorizer-properties zookeeper.connect=zookeeper:2181 \
    --list
Current ACLs for resource `Cluster:LITERAL:kafka-cluster`:
        User:ANONYMOUS has Deny permission for operations: Alter from hosts: *

$ kafka-preferred-replica-election.sh \
    --bootstrap-server :9092 \
    --path-to-json-file parts.json
Error completing preferred replica election for partition t1-1: org.apache.kafka.commo
n.errors.ClusterAuthorizationException: Cluster authorization failed.
Error completing preferred replica election for partition t1-0: org.apache.kafka.commo
n.errors.ClusterAuthorizationException: Cluster authorization failed.
Exception in thread "main" kafka.common.AdminCommandFailedException: 2 preferred repli
ca(s) could not be elected
        at kafka.admin.PreferredReplicaLeaderElectionCommand$AdminClientCommand.electP
referredLeaders(PreferredReplicaLeaderElectionCommand.scala:263)
        at kafka.admin.PreferredReplicaLeaderElectionCommand$.run(PreferredReplicaLead
erElectionCommand.scala:78)
        at kafka.admin.PreferredReplicaLeaderElectionCommand$.main(PreferredReplicaLea
derElectionCommand.scala:42)
        at kafka.admin.PreferredReplicaLeaderElectionCommand.main(PreferredReplicaLead
erElectionCommand.scala)
        Suppressed: org.apache.kafka.common.errors.ClusterAuthorizationException: Clus
ter authorization failed.
        [CIRCULAR REFERENCE:org.apache.kafka.common.errors.ClusterAuthorizationExcepti
on: Cluster authorization failed.]

$ kafka-acls.sh \
    --authorizer-properties zookeeper.connect=zookeeper:2181 \
    --remove \
    --cluster \
    --operation Alter \
    --deny-principal User:ANONYMOUS \
    --force

$ kafka-acls.sh \
    --authorizer-properties zookeeper.connect=zookeeper:2181 \
    --list

$ kafka-preferred-replica-election.sh \
    --bootstrap-server :9092 \
    --path-to-json-file parts.json
Successfully completed preferred replica election for partitions t1-1, t1-0
```

Demo: Using kafka-leader-election.sh

The demo shows how to use kafka-leader-election.sh command-line utility for Partition Leader Election.

The demo is made up of the following steps:

1. Start Kafka Cluster
2. Create Topic
3. Shutting Down Leader Broker
4. Trigger Preferred Leader Election

Start Kafka Cluster

You'll be using kafka-docker project to run Apache Kafka in Docker (using Docker Compose).

```
docker-compose up -d --scale=kafka=3
```

The goal is to run 3 brokers as easily as possible (*let me know if you know a simpler way. Thanks!*).

Create Topic

In this step you'll create a topic with 3 partitions (one for every broker in the cluster) and 2 replicas.

Log in to `kafka-docker_kafka_1` broker first.

```
docker exec -it kafka-docker_kafka_1 /bin/bash
```

```
kafka-topics.sh \
    --bootstrap-server :9092 \
    --create \
    --topic demo-kafka-leader-election \
    --partitions 3 \
    --replication-factor 2
```

Review the leader and ISR brokers. Yours may be different.

```
$ kafka-topics.sh \
    --bootstrap-server :9092 \
    --describe \
    --topic demo-kafka-leader-election
Topic: demo-kafka-leader-election       PartitionCount: 3       ReplicationFactor: 2
        Configs: segment.bytes=1073741824
        Topic: demo-kafka-leader-election       Partition: 0    Leader: 1002    Replic
as: 1002,1001   Isr: 1002,1001
        Topic: demo-kafka-leader-election       Partition: 1    Leader: 1001    Replic
as: 1001,1003   Isr: 1001,1003
        Topic: demo-kafka-leader-election       Partition: 2    Leader: 1003    Replic
as: 1003,1002   Isr: 1003,1002
```

Shutting Down Leader Broker

In this step, you’re going to shut down a leader broker to simulate a broker failure. That triggers a leader election and promote the other broker (in the ISR list) to become the leader.

```
docker stop kafka-docker_kafka_2
```

You should have 2 brokers up and running.

```
$ docker ps
CONTAINER ID        IMAGE                     COMMAND                  CREATED
    STATUS              PORTS                                                   NAMES
f7224b2bdf90        kafka-docker_kafka        "start-kafka.sh"         2 minutes ago
    Up 2 minutes        0.0.0.0:32785->9092/tcp                                 kafka-doc
ker_kafka_3
52ac4b64648e        wurstmeister/zookeeper    "/bin/sh -c '/usr/sb…"   2 minutes ago
    Up 2 minutes        22/tcp, 2888/tcp, 3888/tcp, 0.0.0.0:2181->2181/tcp   kafka-doc
ker_zookeeper_1
ad05db816dc1        kafka-docker_kafka        "start-kafka.sh"         2 minutes ago
    Up 2 minutes        0.0.0.0:32784->9092/tcp                                 kafka-doc
ker_kafka_1
```

Review the leaders.

```
$ kafka-topics.sh \
    --bootstrap-server :9092 \
    --describe \
    --topic demo-kafka-leader-election
Topic: demo-kafka-leader-election       PartitionCount: 3       ReplicationFactor: 2
        Configs: segment.bytes=1073741824
        Topic: demo-kafka-leader-election       Partition: 0    Leader: 1001    Replic
as: 1002,1001   Isr: 1001
        Topic: demo-kafka-leader-election       Partition: 1    Leader: 1001    Replic
as: 1001,1003   Isr: 1001,1003
        Topic: demo-kafka-leader-election       Partition: 2    Leader: 1003    Replic
as: 1003,1002   Isr: 1003
```

Trigger Preferred Leader Election

In this step, you will trigger preferred leader election for the partitions with the other leader elected (*non-preferred leader*).

Let's bring the stopped broker up.

```
$ docker start kafka-docker_kafka_2
kafka-docker_kafka_2
```

Review the leaders.

```
$ kafka-topics.sh \
    --bootstrap-server :9092 \
    --describe \
    --topic demo-kafka-leader-election
Topic: demo-kafka-leader-election       PartitionCount: 3       ReplicationFactor: 2
        Configs: segment.bytes=1073741824
        Topic: demo-kafka-leader-election       Partition: 0    Leader: 1001    Replic
as: 1002,1001   Isr: 1001,1002
        Topic: demo-kafka-leader-election       Partition: 1    Leader: 1001    Replic
as: 1001,1003   Isr: 1001,1003
        Topic: demo-kafka-leader-election       Partition: 2    Leader: 1003    Replic
as: 1003,1002   Isr: 1003,1002
```

Time to fix the partition `0` so the broker `1002` is the leader again.

```
kafka-leader-election.sh \
  --bootstrap-server :9092 \
  --topic demo-kafka-leader-election \
  --partition 0 \
  --election-type preferred
```

You should see the following message for a successful election.

```
Successfully completed leader election (PREFERRED) for partitions demo-kafka-leader-el
ection-0
```

Review the leaders.

```
$ kafka-topics.sh \
    --bootstrap-server :9092 \
    --describe \
    --topic demo-kafka-leader-election
Topic: demo-kafka-leader-election       PartitionCount: 3       ReplicationFactor: 2
        Configs: segment.bytes=1073741824
        Topic: demo-kafka-leader-election       Partition: 0    Leader: 1002    Replic
as: 1002,1001   Isr: 1001,1002
        Topic: demo-kafka-leader-election       Partition: 1    Leader: 1001    Replic
as: 1001,1003   Isr: 1001,1003
        Topic: demo-kafka-leader-election       Partition: 2    Leader: 1003    Replic
as: 1003,1002   Isr: 1003,1002
```

Log Cleanup Policies (Strategies) — Log Compaction And Retention

Kafka uses log.cleanup.policy configuration property to define **cleanup strategies** (policy) of logs:

- Log Compaction
- Log Retention

Kafka uses log.retention.check.interval.ms configuration property as the interval between regular log checks.

log.cleanup.policy and cleanup.policy Configuration Properties

The cluster-wide log.cleanup.policy and the per-topic cleanup.policy configuration properties are comma-separated lists of cleanup strategies:

- **compact** - enables log compaction
- **delete** - enables log retention

Unless defined, cleanup.policy is exactly log.cleanup.policy.

Log Compaction

Log Compaction is a cleanup strategy in which…FIXME

Kafka brokers use LogCleaner for compact retention strategy.

Log compaction can be reconfigured dynamically at runtime.

Log Retention

Log Retention (*Garbage Collection*) is a cleanup strategy to discard (*delete*) old log segments when their retention time or size limit has been reached.

By default there is only a time limit and no size limit.

Retention time is controlled by the cluster-wide log.retention.ms, log.retention.minutes or log.retention.hours configuration properties (from the highest to the lowest priority) or their per-topic retention.ms configuration property.

Retention size is controlled by the cluster-wide log.retention.bytes or per-topic retention.bytes configuration property.

Enable `ALL` logging level for kafka.log.Log logger to see messages related to log retention.

Kafka brokers schedule kafka-log-retention periodic task for delete retention strategy.

Kafka Controller Election

A Kafka broker can be elected as the controller in the process known as **Kafka Controller Election**.

Kafka Controller Election process relies heavily on the features of Apache ZooKeeper that acts as the source of truth and guarantees that only one broker can ever be elected (due to how **ephemeral nodes** work).

From Nodes and ephemeral nodes in the official documentation of Apache ZooKeeper:

> ZooKeeper also has the notion of ephemeral nodes. These znodes exists as long as the session that created the znode is active. When the session ends the znode is deleted.

From Leader Election in the official documentation of Apache ZooKeeper:

> A simple way of doing leader election with ZooKeeper is to use the SEQUENCE|EPHEMERAL flags when creating znodes that represent "proposals" of clients. The idea is to have a znode, say "/election"

When `ControllerEventThread` is requested to process Startup and Reelect controller events, `KafkaController` (instance that runs on every Kafka broker) is requested to elect.

Tip	Consult Demo: Kafka Controller Election to learn about the process.

Given that all the state is in ZooKeeper use `zookeeper-shell` script to know which broker is the active controller.

```
$ ./bin/zookeeper-shell.sh :2181 get /controller
Connecting to :2181

{"version":1,"brokerid":0,"timestamp":"1543491973573"}
cZxid = 0x48
ctime = Thu Nov 29 12:46:13 CET 2018
mZxid = 0x48
mtime = Thu Nov 29 12:46:13 CET 2018
pZxid = 0x48
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x100073f07ba0001
dataLength = 54
numChildren = 0
```

If you receive `Node does not exist: /controller` error message, that means that no Kafka broker has been elected as the active controller yet (or that you use an incorrect ZooKeeper server to talk to).

You could also use `nc` to talk to ZooKeeper in a more direct way (that allows for `dump` command).

```
$ nc localhost 2181
dump
SessionTracker dump:
Session Sets (2):
0 expire at Fri Jan 02 10:57:03 CET 1970:
1 expire at Fri Jan 02 10:57:06 CET 1970:
  0x100073f07ba0001
ephemeral nodes dump:
Sessions with Ephemerals (1):
0x100073f07ba0001:
  /controller
  /brokers/ids/0
```

From Notable changes in 0.10.1.0 in the official documentation of Apache Kafka:

> The recommended way to detect if a given broker is the controller is via the `kafka.controller:type=KafkaController,name=ActiveControllerCount` metric.

Figure 1. Active KafkaController in jconsole

Controller ID Registered (in ZooKeeper)

The election process stops when there is a controller ID registered in Zookeeper (using `KafkaZkClient` that gets the ID of the active controller and the ID is any number but `-1`).

No Controller ID Registered (in ZooKeeper)

If there is no controller ID registered, every `KafkaController` instance tries to register itself as the controller (in Zookeeper) and increment the controller epoch (using KafkaZkClient).

In the end, the active `KafkaController` is requested to onControllerFailover.

Topic Replication

Topic Replication is the process to offer fail-over capability for a topic.

Replication factor defines the number of copies of a topic in a Kafka cluster.

Replication factor can be defined at topic level.

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

Replicas are distributed evenly among Kafka brokers in a cluster.

Leader replica is…FIXME

Follower replica is…FIXME

Producers always send requests to the broker that is the current leader replica for a topic partition.

Data from producers is first saved to a commit log before consumers can find out that it is available. It will only be visible to consumers when the followers acknowledge that they have got the data and stored in their local logs.

Topic Deletion

Topic Deletion is a feature of Kafka that allows for deleting topics.

TopicDeletionManager is responsible for topic deletion.

Topic deletion is controlled by delete.topic.enable Kafka property that turns it on when `true`.

Start a Kafka broker with broker ID `100`.

```
$ ./bin/kafka-server-start.sh config/server.properties \
  --override delete.topic.enable=true \
  --override broker.id=100 \
  --override log.dirs=/tmp/kafka-logs-100 \
  --override port=9192
```

Create **remove-me** topic.

```
$ ./bin/kafka-topics.sh --zookeeper localhost:2181 \
  --create \
  --topic remove-me \
  --partitions 1 \
  --replication-factor 1
Created topic "remove-me".
```

Use `kafka-topics.sh --list` to list available topics.

```
$ ./bin/kafka-topics.sh --zookeeper localhost:2181 --list
__consumer_offsets
remove-me
```

Use `kafka-topics.sh --describe` to list details for `remove-me` topic.

```
$ ./bin/kafka-topics.sh --zookeeper localhost:2181 --describe --topic remove-me
Topic:remove-me PartitionCount:1        ReplicationFactor:1     Configs:
        Topic: remove-me        Partition: 0    Leader: 100     Replicas: 100   Isr: 1
00
```

Note that the broker `100` is the leader for `remove-me` topic.

Stop the broker `100` and start another with broker ID `200`.

```
$ ./bin/kafka-server-start.sh config/server.properties \
  --override delete.topic.enable=true \
  --override broker.id=200 \
  --override log.dirs=/tmp/kafka-logs-200 \
  --override port=9292
```

Use `kafka-topics.sh --delete` to delete `remove-me` topic.

```
$ ./bin/kafka-topics.sh --zookeeper localhost:2181 --delete --topic remove-me
Topic remove-me is marked for deletion.
Note: This will have no impact if delete.topic.enable is not set to true.
```

List the topics.

```
$ ./bin/kafka-topics.sh --zookeeper localhost:2181 --list
__consumer_offsets
remove-me - marked for deletion
```

As you may have noticed, `kafka-topics.sh --delete` will only delete a topic if the topic's leader broker is available (and can acknowledge the removal). Since the broker 100 is down and currently unavailable the topic deletion has only been recorded in Zookeeper.

```
$ ./bin/zkCli.sh -server localhost:2181
[zk: localhost:2181(CONNECTED) 0] ls /admin/delete_topics
[remove-me]
```

As long as the leader broker `100` is not available, the topic to be deleted remains marked for deletion.

Start the broker `100`.

```
$ ./bin/kafka-server-start.sh config/server.properties \
  --override delete.topic.enable=true \
  --override broker.id=100 \
  --override log.dirs=/tmp/kafka-logs-100 \
  --override port=9192
```

With kafka.controller.KafkaController logger at `DEBUG` level, you should see the following messages in the logs:

```
DEBUG [Controller id=100] Delete topics listener fired for topics remove-me to be dele
ted (kafka.controller.KafkaController)
INFO [Controller id=100] Starting topic deletion for topics remove-me (kafka.controlle
r.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.serv
er.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.fe6d0
39ff884498b9d6113fb22a75264-delete and is scheduled for deletion (kafka.log.LogManager
)
DEBUG [Controller id=100] Delete topic callback invoked for org.apache.kafka.common.re
quests.StopReplicaResponse@8c0f4f0 (kafka.controller.KafkaController)
INFO [Controller id=100] New topics: [Set()], deleted topics: [Set()], new partition r
eplica assignment [Map()] (kafka.controller.KafkaController)
DEBUG [Controller id=100] Delete topics listener fired for topics  to be deleted (kafk
a.controller.KafkaController)
```

The topic is now deleted. Use Zookeeper CLI tool to confirm it.

```
$ ./bin/zkCli.sh -server localhost:2181
[zk: localhost:2181(CONNECTED) 1] ls /admin/delete_topics
[]
```

Transactional Producer

Transactional Producer is a Kafka Producer that uses a TransactionManager for…FIXME

Idempotent Producer

Idempotent Producer is a Kafka Producer that…FIXME

Log Management System

When a Kafka broker is requested to start up, it starts LogManager.

`LogManager` is given the log directories that are configured using log.dirs or log.dir configuration properties (default: /tmp/kafka-logs). The log directories are immediately loaded (on a fixed thread pool with numRecoveryThreadsPerDataDir recovery threads).

`LogManager` uses the num.recovery.threads.per.data.dir dynamic configuration (default: `1`) for the number of threads per log data directory for log recovery at startup and flushing at shutdown.

Every log directory may hold zero, one or more partition log directories. `LogManager` creates a Log for every partition log (that is of the format topic-partition or topic-partition.uniqueId-delete).

`LogManager` handles the enforcement of the flush policy and log compaction and retention.

Log is made up of log segments that are FileRecords (with the data) and an OffsetIndex (for reads by offset on the log).

Use kafka-dump-log.sh shell script for sanity check of log files and printing contents out to the console.

Broker

`Broker` is a data structure that represents a Kafka broker using the following properties:

- Id
- EndPoints
- Rack Id (optional)

`Broker` is created when:

- `KafkaServer` is requested to createBrokerInfo
- `MetadataCache` is requested to update the metadata
- `BrokerIdZNode` utility is used to decode a broker metadata (from JSON format in Zookeeper)

`node` Method

```
node(
  listenerName: ListenerName): Node
```

`node` looks up the node by the `ListenerName`.

`node` throws an `BrokerEndPointNotAvailableException` when the `ListenerName` could not be found:

```
End point with listener name [listenerName] not found for broker [id]
```

Note	`node` is used when: - `ControllerChannelManager` is requested to register a new broker - `AbstractControllerBrokerRequestBatch` is requested to send LeaderAndIsr and UpdateMetadata requests out to all brokers - `KafkaServer` is requested to shut down (and controlledShutdown)

`getNode` Method

```
getNode(
  listenerName: ListenerName): Option[Node]
```

`getNode` returns the `Node` for the `ListenerName` if available in the endPointsMap internal registry. Otherwise, `getNode` returns `None`.

Note	`getNode` is used when: • `Broker` is requested for the Node for a given ListenerName • `KafkaApis` is requested to handle a Metadata request • `MetadataCache` is requested to getPartitionReplicaEndpoints

`toServerInfo` Method

```
toServerInfo(
  clusterId: String,
  config: KafkaConfig): AuthorizerServerInfo
```

`toServerInfo` ...FIXME

Note	`toServerInfo` is used when `KafkaServer` is requested to start up (and start an Authorizer if configured).

EndPoint

`EndPoint` is a data structure that represents an endpoint of a Kafka broker using the following properties:

- Host
- Port
- `ListenerName`
- `SecurityProtocol`

`EndPoint` is created when:

- Broker is created
- `EndPoint` utility is used to create one
- `MetadataCache` is requested to update a metadata
- `BrokerIdZNode` utility is used to encode and decode a broker metadata

`EndPoint` is used when:

- `SocketServer` is requested to create an Acceptor and addDataPlaneProcessors

`connectionString` Method

```
connectionString: String
```

`connectionString` ...FIXME

Note	`connectionString` is used when...FIXME

Creating EndPoint — `createEndPoint` Utility

```
createEndPoint(
  connectionString: String,
  securityProtocolMap: Option[Map[ListenerName, SecurityProtocol]]): EndPoint
```

`createEndPoint` ...FIXME

Note	`createEndPoint` is used when: • `CoreUtils` utility is used to convert listener names to endpoints • `BrokerIdZNode` utility is used to decode a broker metadata (from a JSON format in Zookeeper)

Partition

`Partition` is an internal representation of a TopicPartition for ReplicaManager (to manage all partitions).

`Partition` knows the host broker (by local broker ID) which is one of the **partition replicas** (replicated across brokers of a Kafka cluster).

`Partition` can be the leader of the TopicPartition or a follower.

`Partition` can be elected the partition leader (when `ReplicaManager` is requested to makeLeaders).

`Partition` is **under-replicated** if it is the leader and the number of in-sync replicas is below the number of all replicas. Use the UnderReplicated metric to know it (or UnderReplicatedPartitions metric of the ReplicaManager).

`Partition` uses the remoteReplicasMap internal registry to track remote replicas (and is requested to update partition replicas and the in-sync replicas when…FIXME).

A Kafka topic is spread across a Kafka cluster as a logical group of one or more **partitions**.

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 created when `ReplicaManager` is requested to add a TopicPartition to allPartitions (indirectly using apply) and for the OfflinePartition.

`Partition` uses **[Partition [topicPartition] broker=[localBrokerId]]** as the logging prefix (aka `logIdent`).

Tip	Enable `ALL` logging level for `kafka.cluster.Partition` logger to see what happens inside. Add the following line to `config/log4j.properties`: `log4j.logger.kafka.cluster.Partition=ALL` Refer to Logging.

Performance Metrics

`Partition` is a KafkaMetricsGroup with the following performance metrics.

Table 1. Partition's Performance Metrics

Metric Name	Description
`InSyncReplicasCount`	One of the two possible values: • Number of inSyncReplicas when isLeaderReplicaLocal • `0` otherwise
`LastStableOffsetLag`	
`ReplicasCount`	One of the two possible values: • Number of assignedReplicas when isLeaderReplicaLocal • `0` otherwise
`UnderMinIsr`	One of the two possible values: • `1` when isUnderMinIsr • `0` otherwise
`UnderReplicated`	One of the two possible values: • `1` when isUnderReplicated • `0` otherwise

The performance metrics are registered in **kafka.cluster:type=Partition** group (only when the partition is not offline).

Figure 1. Partition Metrics in JConsole JMX Tool

Creating Partition Instance

`Partition` takes the following to be created:

- `TopicPartition`
- replica.lag.time.max.ms
- inter.broker.protocol.version
- Broker ID (of the local broker)
- `Time`
- PartitionStateStore
- `DelayedOperations`
- MetadataCache
- LogManager

`Partition` initializes the internal properties.

Partition and Partition Log — `log` Registry

```
log: Option[Log] = None
```

`log` is a partition log (of the partition).

`log` is assigned a `Log` when `Partition` is requested to createLogIfNotExists and maybeReplaceCurrentWithFutureReplica.

`log` is available until `Partition` is requested to delete.

`log` is used when:

- `Partition` is requested for the LastStableOffsetLag metric, getLocalLog, leaderLogIfLocal, localLogOrException, maybeReplaceCurrentWithFutureReplica
- `ReplicaManager` is requested to localLog, becomeLeaderOrFollower, checkpointHighWatermarks and handleLogDirFailure.

Partition and PartitionStateStore

When created, a `Partition` is given a PartitionStateStore (that is a ZkPartitionStateStore by default).

`PartitionStateStore` is used when `Partition` is requested for the following:

- createLog
- expandIsr
- shrinkIsr

Creating Partition — `apply` Utility

```
apply(
  topicPartition: TopicPartition,
  time: Time,
  replicaManager: ReplicaManager): Partition
```

`apply` creates a new Partition for the given `TopicPartition`, `Time`, and ReplicaManager with the following:

- Creates a new `ZkPartitionStateStore` for the given `TopicPartition` and ReplicaManager
- Creates a new `DelayedOperations` for the given `TopicPartition` and ReplicaManager
- Uses replica.lag.time.max.ms and inter.broker.protocol.version configuration properties.

Note	`apply` is used when `ReplicaManager` is requested to add a partition to allPartitions Registry.

Read and Write Locks

```
leaderIsrUpdateLock: ReentrantReadWriteLock
```

`leaderIsrUpdateLock` is a Java's ReentrantReadWriteLock for a pair of locks, one for read-only operations and one for writing. (The read lock may be held simultaneously by multiple reader threads, so long as there are no writers. The write lock is exclusive).

A read lock (`inReadLock`) is used for the following:

- futureReplicaDirChanged
- maybeIncrementLeaderHW
- doAppendRecordsToFollowerOrFutureReplica
- appendRecordsToLeader
- readRecords
- fetchOffsetForTimestamp
- fetchOffsetSnapshot
- legacyFetchOffsetsForTimestamp
- logStartOffset
- deleteRecordsOnLeader
- truncateTo
- truncateFullyAndStartAt
- lastOffsetForLeaderEpoch

A write lock (`inWriteLock`) is used for the following:

- maybeCreateFutureReplica
- removeFutureLocalReplica
- maybeReplaceCurrentWithFutureReplica
- delete

- makeLeader
- makeFollower
- maybeExpandIsr
- maybeShrinkIsr

`maybeExpandIsr` Method

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

`maybeExpandIsr` ...FIXME

Note	`maybeExpandIsr` is used when `Partition` is requested to updateFollowerFetchState.

`updateFollowerFetchState` Method

```
updateFollowerFetchState(
  followerId: Int,
  followerFetchOffsetMetadata: LogOffsetMetadata,
  followerStartOffset: Long,
  followerFetchTimeMs: Long,
  leaderEndOffset: Long,
  lastSentHighwatermark: Long): Boolean
```

`updateFollowerFetchState` ...FIXME

Note	`updateFollowerFetchState` is used when `ReplicaManager` is requested to updateFollowerFetchState.

`maybeShrinkIsr` Method

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

`maybeShrinkIsr` ...

Note	`maybeShrinkIsr` is used when `ReplicaManager` is requested to maybeShrinkIsr.

`updateReplicaLogReadResult` Method

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

`updateReplicaLogReadResult` ...FIXME

Note	`updateReplicaLogReadResult` is used exclusively when `ReplicaManager` updateFollowerLogReadResults.

`updateIsr` Internal Method

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

`updateIsr` ...FIXME

Note	`updateIsr` is used when `Partition` is requested to expand or shrink the ISR.

`makeFollower` Method

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

`makeFollower` ...FIXME

Note	`makeFollower` is used exclusively when `ReplicaManager` is requested to makeFollowers.

`leaderReplicaIfLocal` Method

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

`leaderReplicaIfLocal` returns a Replica when the leaderReplicaIdOpt is the localBrokerId. Otherwise, `leaderReplicaIfLocal` returns `None` (i.e. undefined).

Note	`leaderReplicaIfLocal` is used...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

Electing Local Partition Replica as Partition Leader — `makeLeader` Method

```
makeLeader(
  controllerId: Int,
  partitionState: LeaderAndIsrPartitionState,
  correlationId: Int,
  highWatermarkCheckpoints: OffsetCheckpoints): Boolean
```

`makeLeader` returns `true` if this broker has just been elected as the leader for the partition.

Note	`makeLeader` could be executed for a broker that is the partition leader already.

Internally, `makeLeader` starts by acquiring write lock (which makes it a single-thread-exclusive operation).

`makeLeader` changes the internal controller epoch based on the given `LeaderAndIsrPartitionState`.

`makeLeader` requests the given `LeaderAndIsrPartitionState` for the partition replicas (their broker IDs) and the ISR (as a list of broker IDs) and updates internal registries for replica assignments and in-sync replicas.

`makeLeader` creates a Log (unless available already) for the local broker ID and the given `OffsetCheckpoints` (with `isFutureReplica` flag off).

`makeLeader` requests the leader log for the logEndOffset.

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

```
[topicPartition] starts at Leader Epoch [leaderEpoch] from offset [leaderEpochStartOff
set]. Previous Leader Epoch was: [leaderEpoch]
```

`makeLeader` updates the internal registries: leaderEpoch, leaderEpochStartOffsetOpt and zkVersion.

`makeLeader` requests the leader log to maybeAssignEpochStartOffset with the current leaderEpoch and the log end offset (that is now considered the leader epoch's start offset).

`makeLeader` requests the remoteReplicas to resetLastCaughtUpTime. For replicas in inSyncReplicaIds, the last caught-up time is the current time while for the others it is `0`.

When the partition has just been elected a new leader, `makeLeader` updates the leaderReplicaIdOpt internal registry and requests the remoteReplicas to updateFetchState.

`makeLeader` checks if increment the high watermark with the leader log and, if incremented, tryCompleteDelayedRequests.

Note	`makeLeader` is used when `ReplicaManager` is requested to makeLeaders.

Updating Internal Registries for Replica Assignments and In-Sync Replicas — `updateAssignmentAndIsr` Method

```
updateAssignmentAndIsr(
  assignment: Seq[Int],
  isr: Set[Int]): Unit
```

`updateAssignmentAndIsr` uses the remoteReplicasMap internal registry and the given `assignment` to find the broker IDs that are no longer partition replicas (of the partition).

`updateAssignmentAndIsr` creates new partition replicas for the broker IDs that have not been registered in the remoteReplicasMap internal registry before (based on the given `assignment`).

`updateAssignmentAndIsr` updates the allReplicaIds and the inSyncReplicaIds internal registries with the given `assignment` and `isr`, respectively.

Note	`updateAssignmentAndIsr` is used when `Partition` is requested to make the local replica the leader or a follower.

Looking Up or Creating Replica — `getOrCreateReplica` Method

```
getOrCreateReplica(
  replicaId: Int,
  isNew: Boolean = false): Replica
```

`getOrCreateReplica` simply looks up the Replica in the allReplicasMap internal registry (by the given `replicaId`).

If not found, `getOrCreateReplica` ...FIXME

Note	`getOrCreateReplica` is used when: • `Partition` is requested to maybeCreateFutureReplica, makeLeader, and makeFollower • `ReplicaManager` is requested to becomeLeaderOrFollower and makeFollowers

`maybeCreateFutureReplica` Method

```
maybeCreateFutureReplica(logDir: String): Boolean
```

`maybeCreateFutureReplica` ...FIXME

Note	`maybeCreateFutureReplica` is used exclusively when `ReplicaManager` is requested to alterReplicaLogDirs.

`appendRecordsToLeader` Method

```
appendRecordsToLeader(
  records: MemoryRecords,
  isFromClient: Boolean,
  requiredAcks: Int = 0): LogAppendInfo
```

`appendRecordsToLeader` basically requests the `Log` (of the leader Replica) to appendAsLeader.

Internally, `appendRecordsToLeader` ...FIXME

Note	`appendRecordsToLeader` is used when: • `GroupMetadataManager` is requested to cleanupGroupMetadata • `ReplicaManager` is requested to appendToLocalLog

`doAppendRecordsToFollowerOrFutureReplica` Internal Method

```
doAppendRecordsToFollowerOrFutureReplica(
  records: MemoryRecords,
  isFuture: Boolean): Option[LogAppendInfo]
```

`doAppendRecordsToFollowerOrFutureReplica` ...FIXME

Note	`doAppendRecordsToFollowerOrFutureReplica` is used exclusively when `Partition` is requested to appendRecordsToFollowerOrFutureReplica.

`appendRecordsToFollowerOrFutureReplica` Method

```
appendRecordsToFollowerOrFutureReplica(
  records: MemoryRecords,
  isFuture: Boolean): Option[LogAppendInfo]
```

`appendRecordsToFollowerOrFutureReplica` ...FIXME

Note	`appendRecordsToFollowerOrFutureReplica` is used when: • `ReplicaAlterLogDirsThread` is requested to processPartitionData • `ReplicaFetcherThread` is requested to processPartitionData

`truncateTo` Method

```
truncateTo(
  offset: Long,
  isFuture: Boolean): Unit
```

`truncateTo` ...FIXME

Note	`truncateTo` is used when: • `ReplicaAlterLogDirsThread` is requested to truncate • `ReplicaFetcherThread` is requested to truncate

`truncateFullyAndStartAt` Method

```
truncateFullyAndStartAt(newOffset: Long, isFuture: Boolean): Unit
```

`truncateFullyAndStartAt` ...FIXME

Note	`truncateFullyAndStartAt` is used when: • `Partition` is requested to appendRecordsToFollowerOrFutureReplica • `ReplicaAlterLogDirsThread` is requested to truncateFullyAndStartAt • `ReplicaFetcherThread` is requested to truncateFullyAndStartAt

`maybeReplaceCurrentWithFutureReplica` Method

```
maybeReplaceCurrentWithFutureReplica(): Boolean
```

`maybeReplaceCurrentWithFutureReplica` ...FIXME

Note	`maybeReplaceCurrentWithFutureReplica` is used exclusively when `ReplicaAlterLogDirsThread` is requested to processPartitionData.

`delete` Method

```
delete(): Unit
```

`delete` ...FIXME

Note	`delete` is used exclusively when `ReplicaManager` is requested to stopReplica.

`removeFutureLocalReplica` Method

```
removeFutureLocalReplica(deleteFromLogDir: Boolean = true): Unit
```

`removeFutureLocalReplica` ...FIXME

Note	`removeFutureLocalReplica` is used when `ReplicaManager` is requested to alterReplicaLogDirs and handleLogDirFailure.

`isLeaderReplicaLocal` Internal Method

```
isLeaderReplicaLocal: Boolean
```

`isLeaderReplicaLocal` is positive (`true`) when the optional Replica is defined. Otherwise, `false` .

Note	`isLeaderReplicaLocal` is used when `ReplicaManager` is requested for the performance metrics (InSyncReplicasCount and ReplicasCount), isUnderReplicated, and lowWatermarkIfLeader.

`localReplicaOrException` Method

```
localReplicaOrException: Replica
```

`localReplicaOrException` localReplica and returns the local replica if available. Otherwise, `localReplicaOrException` throws a `ReplicaNotAvailableException` :

```
Replica for partition [topicPartition] is not available on broker [localBrokerId]
```

Note	`localReplicaOrException` is used when: • `Partition` is requested to maybeCreateFutureReplica, maybeReplaceCurrentWithFutureReplica, makeLeader, doAppendRecordsToFollowerOrFutureReplica, appendRecordsToFollowerOrFutureReplica • `ReplicaManager` is requested to localReplicaOrException, alterReplicaLogDirs, makeFollowers

`localReplica` Method

```
localReplica: Option[Replica]
```

`localReplica` simply gets the partition replica for the local broker ID.

Note	`localReplica` is used when: • `Partition` is requested to localReplicaOrException and leaderReplicaIfLocal • `ReplicaManager` is requested to localReplica, becomeLeaderOrFollower, checkpointHighWatermarks, and handleLogDirFailure

`getReplica` Method

```
getReplica(replicaId: Int): Option[Replica]
```

`getReplica` returns the replica by the given `replicaId` (in the allReplicasMap registry) or `None`.

Note	`getReplica` is used when: • `Partition` is requested to getReplicaOrException and updateFollowerFetchState • `DelayedFetch` is requested to `tryComplete` • `ReplicaManager` is requested to readFromLocalLog

`addReplicaIfNotExists` Method

```
addReplicaIfNotExists(replica: Replica): Replica
```

`addReplicaIfNotExists` ...FIXME

Note	`addReplicaIfNotExists` is used when...FIXME

`assignedReplicas` Method

```
assignedReplicas: Set[Replica]
```

`assignedReplicas` ...FIXME

Note	`assignedReplicas` is used when...FIXME

`allReplicas` Method

```
allReplicas: Set[Replica]
```

`allReplicas` ...FIXME

Note	`allReplicas` is used when...FIXME

`removeReplica` Internal Method

```
removeReplica(replicaId: Int): Unit
```

`removeReplica` ...FIXME

Note	`removeReplica` is used when...FIXME

String (Textual) Representation — `toString` Method

```
toString(): String
```

Note	`toString` is part of the java.lang.Object Contract for a string representation of the object.

`toString` ...FIXME

`readRecords` Method

```
readRecords(
  fetchOffset: Long,
  currentLeaderEpoch: Optional[Integer],
  maxBytes: Int,
  fetchIsolation: FetchIsolation,
  fetchOnlyFromLeader: Boolean,
  minOneMessage: Boolean): LogReadInfo
```

`readRecords` ...FIXME

Note	`readRecords` is used when...FIXME

`deleteRecordsOnLeader` Method

```
deleteRecordsOnLeader(
  offset: Long): LogDeleteRecordsResult
```

`deleteRecordsOnLeader` ...FIXME

Note	`deleteRecordsOnLeader` is used when...FIXME

Creating Log or Future Log (Unless Available Already) — `createLogIfNotExists` Method

```
createLogIfNotExists(
  replicaId: Int,
  isNew: Boolean,
  isFutureReplica: Boolean,
  offsetCheckpoints: OffsetCheckpoints): Unit
```

`createLogIfNotExists` branches off per the given `isFutureReplica` flag:

- When enabled (`true`) and the futureLog internal registry is undefined, `createLogIfNotExists` createLog and saves it in the futureLog internal registry
- When disabled (`false`) and the log internal registry is undefined, `createLogIfNotExists` createLog and saves it in the log internal registry

For all other cases, `createLogIfNotExists` simply prints out the following TRACE message (with the `Future` prefix for `isFutureReplica` flag enabled):

```
[Future] Log already exists.
```

Note	`createLogIfNotExists` is used when: • `Partition` is requested to maybeCreateFutureReplica (`isFutureReplica` flag is enabled), make the local broker the leader or a follower (`isFutureReplica` flag is disabled) • `ReplicaManager` is requested to becomeLeaderOrFollower (`isFutureReplica` flag is enabled) and makeFollowers (`isFutureReplica` flag is disabled)

`getOutOfSyncReplicas` Method

```
getOutOfSyncReplicas(
  maxLagMs: Long): Set[Int]
```

`getOutOfSyncReplicas` requests the Log for the logEndOffset.

`maybeShrinkIsr` isFollowerOutOfSync for every inSyncReplicaIds (without the local broker ID)

Note	`getOutOfSyncReplicas` is used when `Partition` is requested to maybeShrinkIsr.

`localLogOrException` Method

```
localLogOrException: Log
```

`localLogOrException` gives the Log if defined or throws a `ReplicaNotAvailableException`:

```
Log for partition [topicPartition] is not available on broker [localBrokerId]
```

Note	`localLogOrException` is used when…FIXME

`leaderLogIfLocal` Method

```
leaderLogIfLocal: Option[Log]
```

`leaderLogIfLocal` gives the Log only if the local broker is the leader.

Note	`leaderLogIfLocal` is used when…FIXME

Checking If Broker is Partition Leader — `isLeader` Method

```
isLeader: Boolean
```

`isLeader` is positive (`true`) when the leaderReplicaIdOpt is the local broker ID.

Note	`isLeader` is used when: • `Partition` is requested for the InSyncReplicasCount and ReplicasCount metrics, to isUnderReplicated, getLocalLog, leaderLogIfLocal, makeLeader, lowWatermarkIfLeader • `ReplicaManager` is requested to findPreferredReadReplica

`lastOffsetForLeaderEpoch` Method

```
lastOffsetForLeaderEpoch(
  currentLeaderEpoch: Optional[Integer],
  leaderEpoch: Int,
  fetchOnlyFromLeader: Boolean): EpochEndOffset
```

`lastOffsetForLeaderEpoch` ...FIXME

Note	`lastOffsetForLeaderEpoch` is used when...FIXME

`tryCompleteDelayedRequests` Internal Method

```
tryCompleteDelayedRequests(): Unit
```

`tryCompleteDelayedRequests` ...FIXME

Note	`tryCompleteDelayedRequests` is used when `Partition` is requested to makeLeader, updateReplicaLogReadResult, maybeShrinkIsr, and appendRecordsToLeader (when `leaderHWIncremented`).

`createLog` Internal Method

```
createLog(
  replicaId: Int,
  isNew: Boolean,
  isFutureReplica: Boolean,
  offsetCheckpoints: OffsetCheckpoints): Log
```

`createLog` requests the LogManager to initializingLog for the TopicPartition.

`createLog` requests the LogManager to look up or create a new partition log for the TopicPartition. The `LogConfig` passed in is created by requesting the PartitionStateStore for the fetchTopicConfig to override the currentDefaultConfig of the LogManager.

`createLog` requests the given `OffsetCheckpoints` to fetch the checkpointed high watermark for the TopicPartition. Unless found, `createLog` prints out the following INFO message to the logs and assumes `0` :

```
No checkpointed highwatermark is found for partition [topicPartition]
```

`createLog` requests the `Log` to update the high watermark.

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

```
Log loaded for partition [topicPartition] with initial high watermark [initialHighWate
rmark]
```

In the end, `createLog` requests the LogManager to finishedInitializingLog for the TopicPartition.

Note	`createLog` is used when `Partition` is requested to create a log or future log (unless available already).

`expandIsr` Internal Method

```
expandIsr(
  newIsr: Set[Int]): Unit
```

`expandIsr` ...FIXME

Note	`expandIsr` is used when `Partition` is requested to maybeExpandIsr.

`shrinkIsr` Internal Method

```
shrinkIsr(
  newIsr: Set[Int]): Unit
```

`shrinkIsr` ...FIXME

Note	`shrinkIsr` is used when `Partition` is requested to maybeShrinkIsr.

`isFollowerOutOfSync` Internal Method

```
isFollowerOutOfSync(
  replicaId: Int,
  leaderEndOffset: Long,
  currentTimeMs: Long,
  maxLagMs: Long): Boolean
```

`isFollowerOutOfSync` gets the follower replica for the given `replicaId` (if available or throws an exception).

`isFollowerOutOfSync` is positive (`true`) when the following hold:

1. logEndOffset of the follower replica is different than the given `leaderEndOffset`
2. Time since the lastCaughtUpTimeMs of the follower replica is greater than the given `maxLagMs` (i.e. replica.lag.time.max.ms configuration property).

Note	`isFollowerOutOfSync` is used when `Partition` is requested to getOutOfSyncReplicas.

`getReplicaOrException` Internal Method

```
getReplicaOrException(
  replicaId: Int): Replica
```

`getReplicaOrException` finds the replica for the given replica ID or throws a `ReplicaNotAvailableException`:

```
Replica with id [replicaId] is not available on broker [localBrokerId]
```

Note	`getReplicaOrException` is used when `Partition` is requested to checkEnoughReplicasReachOffset, maybeShrinkIsr, and isFollowerOutOfSync.

`getLocalLog` Internal Method

```
getLocalLog(
  currentLeaderEpoch: Optional[Integer],
  requireLeader: Boolean): Either[Log, Errors]
```

`getLocalLog` checkCurrentLeaderEpoch for the given `currentLeaderEpoch` and branches off per the result:

- For no errors (`Errors.NONE`), if the `requireLeader` flag is on (`true`), but the broker is not the leader, `getLocalLog` returns `Errors.NOT_LEADER_FOR_PARTITION` .
- For no errors (`Errors.NONE`), if the `requireLeader` flag is off (`false`) or the broker is the leader, `getLocalLog` branches off per the optional log. `getLocalLog` gives the log if defined or the errors:
  - `Errors.NOT_LEADER_FOR_PARTITION` when the `requireLeader` flag is on (`true`)
  - `Errors.REPLICA_NOT_AVAILABLE` when the `requireLeader` flag is off (`false`)
- For any error, `getLocalLog` simply returns it

Note	`getLocalLog` is used when `Partition` is requested to localLogWithEpochOrException and lastOffsetForLeaderEpoch.

`checkCurrentLeaderEpoch` Internal Method

```
checkCurrentLeaderEpoch(
  remoteLeaderEpochOpt: Optional[Integer]): Errors
```

`checkCurrentLeaderEpoch` ...FIXME

Note	`checkCurrentLeaderEpoch` is used when `Partition` is requested to getLocalLog.

`localLogWithEpochOrException` Internal Method

```
localLogWithEpochOrException(
  currentLeaderEpoch: Optional[Integer],
  requireLeader: Boolean): Log
```

`localLogWithEpochOrException` ...FIXME

Note	`localLogWithEpochOrException` is used when `Partition` is requested to... FIXME

`isFollowerInSync` Internal Method

```
isFollowerInSync(
  followerReplica: Replica,
  highWatermark: Long): Boolean
```

`isFollowerInSync` ...FIXME

Note	`isFollowerInSync` is used when `Partition` is requested to maybeExpandIsr.

`maybeUpdateIsrAndVersion` Internal Method

```
maybeUpdateIsrAndVersion(
  isr: Set[Int],
  zkVersionOpt: Option[Int]): Unit
```

`maybeUpdateIsrAndVersion` ...FIXME

Note	`maybeUpdateIsrAndVersion` is used when `Partition` is requested to expandIsr and shrinkIsr.

Internal Properties

Name	Description
`controllerEpoch`	Controller epoch Starts as `0` and changes when `Partition` is requested to make the local replica the leader or a follower. Used for expandIsr and shrinkIsr
`inSyncReplicaIds`	In-sync replicas of the topic partition, i.e. broker IDs that are known to be in-sync with the leader Note: `inSyncReplicaIds` contains in-sync replicas only when updateAssignmentAndIsr is executed when the current partition replica is the leader (and empty when a follower or deleted). Updated when maybeUpdateIsrAndVersion, updateAssignmentAndIsr, and delete
`leaderReplicaIdOpt`	Broker ID of the broker that manages the leader replica Default: `None` (undefined) • Assigned a broker ID when makeLeader and makeFollower • `None` when delete
`allReplicaIds`	All replica broker IDs that were assigned to this topic partition
`allReplicasMap`	Replicas by ID • A new entry added in getOrCreateReplica, addReplicaIfNotExists • An entry removed in removeReplica, removeFutureLocalReplica, maybeReplaceCurrentWithFutureReplica, and delete Used in getReplica, assignedReplicas, allReplicas, toString
`futureLog`	

`remoteReplicasMap`	Remote replicas by broker ID (`Pool[Int, Replica]`) that is updated when updateAssignmentAndIsr `remoteReplicasMap` contains broker IDs of the partition replicas (of the partition) without the local broker ID. Used for getReplica, remoteReplicas, maybeIncrementLeaderHW All of the pairs are removed when delete

Replica

`Replica` (aka **partition replica**) is…FIXME

`Replica` is a local partition replica when the replica ID is the local broker ID.

`Replica` is a future local replica when the replica ID is `-3`.

`Replica` is created exclusively when `Partition` is requested to getOrCreateReplica.

`Replica` is **local** when the Log was defined (at the time the replica was created).

Tip	Enable `INFO` or `TRACE` logging level for `kafka.cluster.Replica` logger to see what happens inside. Add the following line to `config/tools-log4j.properties`: `log4j.logger.kafka.cluster.Replica=TRACE` Refer to Logging.

`latestEpoch` Method

```
latestEpoch: Option[Int]
```

`latestEpoch`…FIXME

Note	`latestEpoch` is used when…FIXME

`logEndOffset` Method

```
logEndOffset: Long
```

`logEndOffset` requests the current LogOffsetMetadata for the message offset.

Note	`logEndOffset` is used when: • `ReplicaManager` is requested to findPreferredReadReplica • `Partition` is requested to updateFollowerFetchState and isFollowerInSync

`lastCaughtUpTimeMs` Method

```
lastCaughtUpTimeMs: Long
```

`lastCaughtUpTimeMs` ...FIXME

Note	`lastCaughtUpTimeMs` is used when...FIXME

`logEndOffsetMetadata` Method

```
logEndOffsetMetadata: LogOffsetMetadata
```

`logEndOffsetMetadata` ...FIXME

Note	`logEndOffsetMetadata` is used when...FIXME

`resetLastCaughtUpTime` Method

```
resetLastCaughtUpTime(
  curLeaderLogEndOffset: Long,
  curTimeMs: Long,
  lastCaughtUpTimeMs: Long): Unit
```

`resetLastCaughtUpTime` ...FIXME

Note	`resetLastCaughtUpTime` is used when...FIXME

Creating Replica Instance

`Replica` takes the following to be created:

- Broker ID
- `TopicPartition`
- `Time` (default: `SYSTEM`)
- `initialHighWatermarkValue` (default: `0L`)
- Optional Log (default: `None`)

`Replica` initializes the internal registries and counters.

While being created, `Replica` prints out the following INFO message to the logs and requests the Log to onHighWatermarkIncremented with the given initialHighWatermarkValue.

```
Replica loaded for partition [topicPartition] with initial high watermark [initialHigh
WatermarkValue]
```

`convertHWToLocalOffsetMetadata` Method

```
convertHWToLocalOffsetMetadata(): Unit
```

`convertHWToLocalOffsetMetadata` ...FIXME

Note	`convertHWToLocalOffsetMetadata` is used when...FIXME

`updateFetchState` Method

```
updateFetchState(
  followerFetchOffsetMetadata: LogOffsetMetadata,
  followerStartOffset: Long,
  followerFetchTimeMs: Long,
  leaderEndOffset: Long,
  lastSentHighwatermark: Long): Unit
```

`updateFetchState` ...FIXME

Note	`updateFetchState` is used when...FIXME

`highWatermark_=` Method

```
highWatermark_=(): Unit
```

`highWatermark_=` ...FIXME

Note	`highWatermark_=` is used when...FIXME

PartitionStateStore

`PartitionStateStore` is the abstraction of state stores for Partitions to fetch topic configuration and expand or shrink in-sync replicas.

Note	ZkPartitionStateStore is the default and only known `PartitionStateStore` in Apache Kafka.

Table 1. PartitionStateStore Contract

Method	Description

`expandIsr`	`expandIsr(`
	`controllerEpoch: Int,`
	`leaderAndIsr: LeaderAndIsr): Option[Int]`
	
	Used when `Partition` is requested to expandIsr

`fetchTopicConfig`	`fetchTopicConfig(): Properties` Used when `Partition` is requested to createLog

`shrinkIsr`	`shrinkIsr(`
	`controllerEpoch: Int,`
	`leaderAndIsr: LeaderAndIsr): Option[Int]`
	
	Used when `Partition` is requested to shrinkIsr

ZkPartitionStateStore

`ZkPartitionStateStore` is a state store for a TopicPartition (as Partition).

`ZkPartitionStateStore` uses ZooKeeper as the state storage (using a KafkaZkClient).

`ZkPartitionStateStore` is created for a Partition.

Creating ZkPartitionStateStore Instance

`ZkPartitionStateStore` takes the following to be created:

- `TopicPartition`
- KafkaZkClient
- ReplicaManager

Reading Topic Configuration (/config/topics ZNode) — `fetchTopicConfig` Method

```
fetchTopicConfig(): Properties
```

Note	`fetchTopicConfig` is part of the PartitionStateStore contract to…FIXME.

`fetchTopicConfig` reads the configuration properties from `/config/topics/[topic]` znode in ZooKeeper (deserialized from JSON).

Internally, `fetchTopicConfig` creates a new AdminZkClient with the KafkaZkClient.

`fetchTopicConfig` requests the `AdminZkClient` to read an entity configuration from ZooKeeper for `topics` entity for the `topic` of the TopicPartition.

ReplicationUtils

`ReplicationUtils` ...FIXME

`propagateIsrChanges` Method

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

`propagateIsrChanges` ...FIXME

Note	`propagateIsrChanges` is used exclusively when maybePropagateIsrChanges.

CoreUtils Utility

`CoreUtils` comes with the following utilities:

- listenerListToEndPoints

Converting Listener Names to EndPoints — `listenerListToEndPoints` Utility

```
listenerListToEndPoints(
  listeners: String,
  securityProtocolMap: Map[ListenerName, SecurityProtocol]): Seq[EndPoint]
```

`listenerListToEndPoints` ...FIXME

Note	`listenerListToEndPoints` is used when `KafkaConfig` is requested for the listeners and advertisedListeners

Kafka Security / Communications Security

Communication Security in Kafka is based on the following:

- security.protocol configuration property (for Kafka clients and brokers)

Tip	Consult Demo: Securing Communication Between Clients and Brokers Using SSL.

Demo: Securing Communication Between Clients and Brokers Using SSL

The demo shows how to configure secure communication between Kafka clients (consumers and producers) and brokers using Transport Layer Security (TLS) (and its now-deprecated predecessor, Secure Sockets Layer (SSL)).

Note	TLS is a newer variant of SSL, but because of the term `SSL` is used in many Kafka configuration properties the document will use `SSL` (over `TLS`) to refer to the protocol used for secure communication.

The demo is made up of the following steps:

1. Create Directory of SSL Files
2. Create Certificate Authority (CA)
3. Generate SSL Keys and Certificate for Kafka Brokers
4. Sign Broker Certificate (Using CA)
5. Import Certificates to Broker Keystore
6. Configure SSL on Kafka Broker
7. Import CA Certificate to Client Truststore
8. Configure SSL for Kafka Clients

Create Directory of SSL Files

Create a directory for SSL files.

```
mkdir -p /tmp/kafka-ssl-demo
```

All the following commands are executed in the directory.

```
cd /tmp/kafka-ssl-demo
```

Create Certificate Authority (CA)

> In cryptography, a **certificate authority** or **certification authority (CA)** is an entity that issues digital certificates.
>
> A digital certificate certifies the ownership of a public key by the named subject of the certificate.
>
> A CA acts as a trusted third party — trusted both by the subject (owner) of the certificate and by the party relying upon the certificate. The format of these certificates is specified by the X.509 standard.

— Wikipedia
https://en.wikipedia.org/wiki/Certificate_authority
Simply put, a certificate authority is used to sign certificates (with public keys).

Generate a private key and a self-signed certificate for the CA.

```
// openssl genrsa -out ca.key
// openssl req -new -x509 -key ca.key -out ca.crt
$ openssl req \
  -new \
  -x509 \
  -days 365 \
  -keyout ca.key \
  -out ca.crt \
  -subj "/C=PL/L=Warsaw/CN=Certificate Authority" \
  -passout pass:1234
Generating a RSA private key
.........................+++++
.....................+++++
writing new private key to 'ca.key'
-----
```

You should have the following files in the directory:

- `ca.key` - the private key of the certificate authority
- `ca.crt` - the certificate (with the public key) of the certificate authority

Use the following command to print certificate contents:

```
$ openssl x509 -text -noout -in ca.crt
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            3c:92:ec:a0:fa:31:fc:00:3f:2d:16:2b:ca:3a:a6:72:a8:fb:aa:8c
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = PL, L = Warsaw, CN = Certificate Authority
        Subject: C = PL, L = Warsaw, CN = Certificate Authority
... removed for clarity
```

Since `ca.crt` is a self-signed certificate, the Issuer and Subject sections use the same subject.

Generate SSL Keys and Certificate for Kafka Brokers

Generate the keys and certificate for every Kafka broker in a cluster. This time you'll be using `keytool` utility that comes with Java Platform, Standard Edition (refer to keytool).

```
// openssl genrsa -out server.key
$ keytool \
  -genkey \
  -keystore server.keystore \
  -alias localhost \
  -dname CN=localhost \
  -keyalg RSA \
  -validity 365 \
  -ext san=dns:localhost \
  -storepass 123456
```

Tip	If you want to use a Subject Alternative Name (SAN) in your certificate, add the following options to the `keytool` command line: • Fully-qualified domain name (FQDN): `-ext san=dns:servername.domain.com` • IP address: `-ext san=ip:192.168.10.1` **DistinguishedName** or **SubjectAltNames** should be the fully-qualified domain name of a broker.

You should now have one more file in the directory:

- `server.keystore` - the keystore with the private key and the certificate of the broker

Use `keytool` to print out the content of the keystore.

```
keytool -list -v -keystore server.keystore -storepass 123456
```

The keystore should contain 1 entry for the alias `localhost`.

Sign Broker Certificate (Using CA)

Create a certificate signing request (CSR).

Export the server certificate from `server.keystore`.

```
// openssl req -new -key server.key -out server.csr
$ keytool \
  -certreq \
  -keystore server.keystore \
  -alias localhost \
  -file server.unsigned.crt \
  -storepass 123456
```

Sign the certificate signing request (`server.unsigned.crt`) with the root CA.

```
$ openssl x509 \
  -req \
  -CA ca.crt \
  -CAkey ca.key \
  -in server.unsigned.crt \
  -out server.crt \
  -days 365 \
  -CAcreateserial \
  -passin pass:1234
Signature ok
subject=CN = localhost
Getting CA Private Key
```

You should have the following files in the directory:

- `server.unsigned.crt`
- `ca.srl`
- `server.crt` - the signed certificate of the broker

Import Certificates to Broker Keystore

Create a SSL keystore for the Kafka broker. Each broker gets its own unique keystore.

Import the certificate of the CA into the broker keystore.

```
$ keytool \
  -import \
  -file ca.crt \
  -keystore server.keystore \
  -alias ca \
  -storepass 123456 \
  -noprompt
Certificate was added to keystore
```

Import the signed certificate into the broker keystore. Make sure to use the same `-alias` as you used ealier.

```
$ keytool \
  -import \
  -file server.crt \
  -keystore server.keystore \
  -alias localhost \
  -storepass 123456 \
  -noprompt
Certificate reply was installed in keystore
```

Use `keytool` to print out the certificates in the broker keystore.

```
keytool -list -v -keystore server.keystore -storepass 123456
```

There should be 2 entries (one for the CA and another for the broker itself).

Configure SSL on Kafka Broker

Create `config/server-ssl.properties` (based on `config/server.properties`) and add the following configuration properties to enable SSL:

```
listeners=PLAINTEXT://:9092,SSL://:9093
ssl.keystore.location=/tmp/kafka-ssl-demo/server.keystore
ssl.keystore.password=123456
ssl.key.password=123456
```

Start the broker(s).

```
./bin/kafka-server-start.sh config/server-ssl.properties
```

Tip	Use `export KAFKA_OPTS=-Djavax.net.debug=all` to debug SSL issues.

Verify the SSL configuration of the broker. The following uses the Cryptography and SSL/TLS Toolkit (OpenSSL) and the client tool.

```
openssl s_client -connect localhost:9093
```

The tool should print out the certificate chain of the broker (a chain of the subjects and the issuers). At the end, you should find the following `Verify return code`:

```
Verify return code: 19 (self signed certificate in certificate chain)
```

Enter `Ctrl-C` to close the session.

Use the client tool with `-CAfile` option to trust the CA certificate.

```
openssl s_client -connect localhost:9093 -CAfile /tmp/kafka-ssl-demo/ca.crt
```

With the change, you should find the following `Verify return code`:

```
Verify return code: 0 (ok)
```

Enter `Ctrl-C` to close the session.

Import CA Certificate to Client Truststore

Add the CA certificate `ca.crt` to a client truststore for the clients to trust this CA.

```
$ keytool \
  -import \
  -file ca.crt \
  -keystore client.truststore \
  -alias ca \
  -storepass 123456 \
  -noprompt
Certificate was added to keystore
```

Use `keytool` to print out the certificates in the client keystore.

```
keytool -list -v -keystore client.truststore -storepass 123456
```

There should be 1 entry for the CA.

Configure SSL for Kafka Clients

Use the following `client-ssl.properties` as a minimal configuration of a Kafka client to use SSL:

```
security.protocol=SSL
ssl.truststore.location=/tmp/kafka-ssl-demo/client.truststore
ssl.truststore.password=123456
```

Use `kafka-console-producer.sh` utility to send records to Kafka brokers over SSL:

```
kafka-console-producer.sh \
  --broker-list :9093 \
  --topic ssl \
  --producer.config /tmp/kafka-ssl-demo/client-ssl.properties
```

That's all for the demo.

SslFactory

`SslFactory` is created when:

- `SaslChannelBuilder` is requested to configure for SASL_SSL security protocol (with `none` for the clientAuthConfigOverride text)
- `SslChannelBuilder` is requested to configure (with no clientAuthConfigOverride text)

Creating SslFactory Instance

`SslFactory` takes the following to be created:

- `Mode` (i.e. `CLIENT` or `SERVER`)
- `clientAuthConfigOverride` text
- `keystoreVerifiableUsingTruststore` flag

`SslFactory` initializes the internal properties.

`configure` Method

```
void configure(
  Map<String, ?> configs)
```

Note	`configure` is part of the Configurable contract to pass configurations.

`configure` initializes the endpoint identification algorithm internal registry based on ssl.endpoint.identification.algorithm configuration property.

`configure` ...FIXME

`validateReconfiguration` Method

```
void validateReconfiguration(
  Map<String, ?> newConfigs)
```

Note	`validateReconfiguration` is part of the Reconfigurable to...FIXME.

`validateReconfiguration` ...FIXME

`reconfigure` Method

```
void reconfigure(
  Map<String, ?> newConfigs)
```

Note	`reconfigure` is part of the Reconfigurable to…FIXME.

`reconfigure` …FIXME

`createSslEngine` Method

```
SSLEngine createSslEngine(
  String peerHost,
  int peerPort)
```

`createSslEngine` simply requests the SslEngineBuilder to createSslEngine for the given `peerHost` and `peerPort` (and the mode and the endpointIdentification).

`createSslEngine` throws an `IllegalStateException` if the sslEngineBuilder hasn't been initialized yet (`null`):

```
SslFactory has not been configured.
```

Note	`createSslEngine` is used when SaslChannelBuilder and SslChannelBuilder are requested to `buildTransportLayer` .

`createNewSslEngineBuilder` Internal Method

```
SslEngineBuilder createNewSslEngineBuilder(
  Map<String, ?> newConfigs)
```

`createNewSslEngineBuilder` …FIXME

Note	`createNewSslEngineBuilder` is used when `SslFactory` is requested to validateReconfiguration and reconfigure.

Internal Properties

Name	Description
`endpointIdentification`	Name of the algorithm for endpoint identification (based on ssl.endpoint.identification.algorithm configuration property) Used when `SslFactory` is requested for a SSLEngine.
`sslEngineBuilder`	SslEngineBuilder to create a SSLEngine (for SaslChannelBuilder and SslChannelBuilder) Initialized when `SslFactory` is requested to configure and then later when requested to reconfigure (with a new configuration)

SslEngineBuilder

`SslEngineBuilder` is used to create a SSLEngine (for SslFactory that adds a reconfiguration layer atop).

`SslEngineBuilder` is created when `SslFactory` is requested to configure and reconfigure (indirectly via createNewSslEngineBuilder).

Tip	Enable `ALL` logging level for `org.apache.kafka.common.security.ssl.SslEngineBuilder` logger to see what happens inside. Add the following line to `config/log4j.properties`: `log4j.logger.org.apache.kafka.common.security.ssl.SslEngineBuilder=ALL` Refer to Logging.

Creating SslEngineBuilder Instance

`SslEngineBuilder` takes the following to be created:

- Configuration properties

`SslEngineBuilder` initializes the internal properties.

SSL Key Store — `keystore` Internal Property

```
SecurityStore keystore
```

`keystore` is created (alongside `SslEngineBuilder`) based on the following configuration properties:

- ssl.keystore.type
- ssl.keystore.location
- ssl.keystore.password
- ssl.key.password

`keystore` is used…FIXME

SSL Trust Store — `truststore` Internal Property

```
SecurityStore truststore
```

Used when…FIXME

Creating SSLEngine — `createSslEngine` Method

```
SSLEngine createSslEngine(
  Mode mode,
  String peerHost,
  int peerPort,
  String endpointIdentification)
```

`createSslEngine` requests the SSLContext to create a new Java SSLEngine (for the advisory peer information, i.e. `peerHost` and `peerPort`) that is configured using…FIXME

If enabled cipher suites and enabled protocol versions are configured, `createSslEngine` requests the `SSLEngine` to use them.

For `SERVER` mode, `createSslEngine` requests the `SSLEngine` to be in the server mode and configures it for **client authentication** based on the SslClientAuth:

- `REQUIRED` for client authentication required
- `REQUESTED` for client authentication requested (optional client authentication)
- `NONE` for no client authentication

For `CLIENT` mode, `createSslEngine` requests the `SSLEngine` to be in the client mode and sets the endpoint identification algorithm (to prevent man-in-the-middle attacks) based on the given `endpointIdentification` .

Note	`createSslEngine` is used when: • `SslFactory` is requested to createSslEngine • `SslEngineValidator` is requested to `createSslEngineForValidation`

`createKeystore` Internal Utility

```
SecurityStore createKeystore(
  String type,
  String path,
  Password password,
  Password keyPassword)
```

`createKeystore` ...FIXME

Note	`createKeystore` is used when `SslEngineBuilder` is created.

`createTruststore` Internal Utility

```
SecurityStore createTruststore(
  String type,
  String path,
  Password password)
```

`createTruststore` ...FIXME

Note	`createTruststore` is used when `SslEngineBuilder` is created.

`createSSLContext` Internal Method

```
SSLContext createSSLContext()
```

`createSSLContext` ...FIXME

Note	`createSSLContext` is used when `SslEngineBuilder` is created.

Internal Properties

Name	Description
`cipherSuites`	Enabled cipher suites to use on a SSLEngine.
`enabledProtocols`	Enabled protocol versions to use on a SSLEngine.
`sslClientAuth`	Client authentication (`SslClientAuth`)
`sslContext`	Java's SSLContext

SslPrincipalMapper

`SslPrincipalMapper` is...FIXME

`fromRules` Utility

```
SslPrincipalMapper fromRules(
  String sslPrincipalMappingRules)
```

`fromRules` ...FIXME

Note	`fromRules` is used when `SslChannelBuilder` is requested to configure.

`splitRules` Internal Utility

```
List<String> splitRules(
  String sslPrincipalMappingRules)
```

`splitRules` ...FIXME

Note	`splitRules` is used when `SslPrincipalMapper` is created.

SslConfigs

`SslConfigs` are the configuration properties of SslEngineBuilder.

Table 1. SslConfigs' Configuration Properties

Name	Description
`ssl.keystore.type`	The file format of the key store file. Optional for client. Default: `JKS` Reconfigurable
`ssl.endpoint.identification.algorithm`	The endpoint identification algorithm to validate server hostname using server certificate Default: `https` Non-reconfigurable
`ssl.keystore.location`	The location of the keystore file. Optional for client and can be used for two-way authentication for client. Default: undefined Reconfigurable
`ssl.keystore.password`	The store password for the keystore file. Optional for client and only needed if ssl.keystore.location is configured. Default: undefined Reconfigurable
`ssl.key.password`	The password of the private key in the keystore file. Optional for client. Default: undefined Reconfigurable
`ssl.truststore.location`	
`ssl.truststore.password`	
`ssl.enabled.protocols`	
`ssl.client.auth`	

BrokerSecurityConfigs

Name	Description
`PRINCIPAL_BUILDER_CLASS_CONFIG`	principal.builder.class
`SSL_CLIENT_AUTH_CONFIG`	ssl.client.auth Used when: • SslEngineBuilder is created • `SslFactory` is requested to configure and createNewSslEngineBuilder
`SSL_PRINCIPAL_MAPPING_RULES_CONFIG`	ssl.principal.mapping.rules

SSL Authentication and Authorization

Kafka Authorization is based on the following:

- kafka-acls.sh utility for ACL management
- authorizer.class.name configuration property
- Authorizer
- security.protocol configuration property
- ssl.client.auth configuration property

Note	Apache Kafka 2.4.0 introduced KIP-504 - Add new Java Authorizer Interface for authorization.

For SSL with client authentication enabled, `TransportLayer#handshake()` performs authentication. For SASL, authentication is performed by `Authenticator#authenticate()`.

For SSL authentication, the principal will be derived using the rules defined by ssl.principal.mapping.rules applied on the distinguished name from the client certificate if one is provided. Otherwise, if client authentication is not required, the principal name will be `ANONYMOUS`.

For `PLAINTEXT` listeners or when client authentication is not required, the principal will always be ANONYMOUS.

Tip	Consult Demo: ACL Authorization.

Demo

With Kafka in Docker (and the wurstmeister/kafka Docker image), use the following docker-compose.yml to use AclAuthorizer for authorization.

Caution	Review the demo to use `AclAuthorizer`.

docker-compose.yml

```yaml
version: '2'
services:
  zookeeper:
    image: wurstmeister/zookeeper
    ports:
      - "2181:2181"
  kafka:
    build: .
    ports:
      - "9092"
    environment:
      KAFKA_ADVERTISED_HOST_NAME: 192.168.0.87
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_AUTHORIZER_CLASS_NAME: kafka.security.auth.SimpleAclAuthorizer
      KAFKA_ALLOW_EVERYONE_IF_NO_ACL_FOUND: "true"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
```

Run a 3-node Kafka cluster with authentication. Use the official documentation of kafka-docker for reference.

```
$ docker-compose up -d --scale kafka=3
Starting kafka-docker_kafka_1     ... done
Starting kafka-docker_kafka_2     ... done
Starting kafka-docker_kafka_3     ... done
Starting kafka-docker_zookeeper_1 ... done

$ docker-compose ps
          Name                        Command               State
    Ports
-----------------------------------------------------------------------------------------
--------------------------------
kafka-docker_kafka_1       start-kafka.sh                   Up      0.0.0.0:32792->909
2/tcp
kafka-docker_kafka_2       start-kafka.sh                   Up      0.0.0.0:32790->909
2/tcp
kafka-docker_kafka_3       start-kafka.sh                   Up      0.0.0.0:32791->909
2/tcp
kafka-docker_zookeeper_1   /bin/sh -c /usr/sbin/sshd  ...   Up      0.0.0.0:2181->2181
/tcp, 22/tcp, 2888/tcp, 3888/tcp

// with no -f to check out the status at Kafka level
$ docker logs kafka-docker_kafka_1
...
INFO Kafka version: 2.3.0 (org.apache.kafka.common.utils.AppInfoParser)
INFO Kafka commitId: fc1aaa116b661c8a (org.apache.kafka.common.utils.AppInfoParser)
INFO Kafka startTimeMs: 1568969945089 (org.apache.kafka.common.utils.AppInfoParser)
INFO [KafkaServer id=1001] started (kafka.server.KafkaServer)

// Stop the cluster
$ docker-compose stop
```

Demo: Secure Inter-Broker Communication

The demo shows how to set up a secure communication between brokers (and disable the unsecure plaintext listener altogether). That will make Kafka brokers available via TLS/SSL only.

Note	The demo is a follow-up to Demo: Securing Communication Between Clients and Brokers Using SSL. Please finish it first before this demo.

The demo is made up of the following steps:

1. Configure Broker to Trust Certificate Authority
2. Enable SSL for Inter-Broker Communication
3. Disable Plaintext Unsecure Listener

Configure Broker to Trust Certificate Authority

Import the certificate of the certificate authority (CA) to a broker truststore so the brokers can trust it (when a broker tries to connect using SSL).

```
$ keytool \
  -import \
  -file ca.crt \
  -keystore server.truststore \
  -alias ca \
  -storepass 123456 \
  -noprompt
Certificate was added to keystore
```

Use `keytool` to print out the certificates in the client keystore.

```
keytool -list -v -keystore server.truststore -storepass 123456
```

There should be 1 entry for the CA.

Enable SSL for Inter-Broker Communication

Edit `config/server-ssl.properties` and add the following configuration properties to enable SSL for inter-broker communication:

```
security.inter.broker.protocol=SSL
ssl.truststore.location=/tmp/kafka-ssl-demo/server.truststore
ssl.truststore.password=123456
```

Start the broker(s).

```
./bin/kafka-server-start.sh config/server-ssl.properties
```

Tip	Use `export KAFKA_OPTS=-Djavax.net.debug=all` to debug SSL issues.

Verify the SSL configuration of the broker. The following uses the Cryptography and SSL/TLS Toolkit (OpenSSL) and the client tool.

```
openssl s_client -connect localhost:9093
```

Disable Plaintext Unsecure Listener

Edit `config/server-ssl.properties` and add `listeners` property to use `SSL://:9093` only:

```
listeners=SSL://:9093
```

Restart the broker.

```
./bin/kafka-server-start.sh config/server-ssl.properties
```

Tip	Use `export KAFKA_OPTS=-Djavax.net.debug=all` to debug SSL-related issues.

Verify the SSL configuration of the broker. The following uses the Cryptography and SSL/TLS Toolkit (OpenSSL) and the client tool.

```
openssl s_client -connect localhost:9093
```

Enter `Ctrl-C` to close the session.

That's all for the demo.

Demo: SSL Authentication

The demo shows how to use SSL/TLS for authentication so no connection can be established between Kafka clients (consumers and producers) and brokers unless a valid and trusted certificate is provided.

Note	The demo is a follow-up to Demo: Secure Inter-Broker Communication. Please finish it first before this demo.

The demo is made up of the following steps:

1. Generate Certificate for Client Authentication
2. Sign Client Certificate (Using CA)
3. Import Certificates to Client Keystore
4. Require Client Authorization Using SSL on Kafka Brokers
5. Configure SSL Authentication for Kafka Client

Generate Certificate for Client Authentication

Generate the keys and certificate of a Kafka client to be authenticated as **jacek**.

```
$ keytool \
  -genkey \
  -keystore jacek.keystore \
  -alias jacek \
  -dname CN=jacek \
  -keyalg RSA \
  -validity 365 \
  -storepass 123456
```

You should now have one more file in the directory:

- `jacek.keystore` - the keystore with the private key and the certificate of the user

Use `keytool` to print out the content of the keystore.

```
keytool -list -v -keystore jacek.keystore -storepass 123456
```

The keystore should contain 1 entry for the alias `jacek`.

Sign Client Certificate (Using CA)

Create a certificate signing request (CSR).

Export the client certificate from `jacek.keystore`.

```
$ keytool \
  -certreq \
  -keystore jacek.keystore \
  -alias jacek \
  -file jacek.unsigned.crt \
  -storepass 123456
```

Sign the certificate signing request (`jacek.unsigned.crt`) with the root CA.

```
$ openssl x509 \
  -req \
  -CA ca.crt \
  -CAkey ca.key \
  -in jacek.unsigned.crt \
  -out jacek.crt \
  -days 365 \
  -CAcreateserial \
  -passin pass:1234
Signature ok
subject=CN = jacek
Getting CA Private Key
```

You should have the following file in the directory:

- `jacek.crt` - the signed certificate of the user

Import Certificates to Client Keystore

Create a SSL keystore for the Kafka client. Each client gets its own unique keystore.

Import the certificate of the CA into the client keystore.

```
$ keytool \
  -import \
  -file ca.crt \
  -keystore jacek.keystore \
  -alias ca \
  -storepass 123456 \
  -noprompt
Certificate was added to keystore
```

Import the signed certificate into the client keystore. Make sure to use the same `-alias` as you used ealier.

```
$ keytool \
  -import \
  -file jacek.crt \
  -keystore jacek.keystore \
  -alias jacek \
  -storepass 123456 \
  -noprompt
Certificate reply was installed in keystore
```

Use `keytool` to print out the certificates in the client keystore.

```
keytool -list -v -keystore jacek.keystore -storepass 123456
```

There should be 2 entries (one for the CA and another for the client itself).

Require Client Authorization Using SSL on Kafka Brokers

Enable SSL authentication (require client authentication using SSL certificates).

Edit `config/server-ssl.properties` and add the following configuration property:

```
ssl.client.auth=required
```

Start the broker(s).

```
./bin/kafka-server-start.sh config/server-ssl.properties
```

Tip	Use `export KAFKA_OPTS=-Djavax.net.debug=all` to debug SSL-related issues.

Verify the SSL configuration of the broker. The following uses the Cryptography and SSL/TLS Toolkit (OpenSSL) and the client tool.

```
openssl s_client -connect localhost:9093
```

The client tool will quit immediately since the broker requires clients to provide valid certificates. You should find the following INFO message in the broker logs:

```
[SocketServer brokerId=0] Failed authentication with /0:0:0:0:0:0:0:1 (SSL handshake f
ailed)
```

Configure SSL Authentication for Kafka Client

Use the following `jacek-client.properties` as a minimal configuration of a Kafka client to use SSL authentication:

```
security.protocol=SSL
ssl.truststore.location=/tmp/kafka-ssl-demo/client.truststore
ssl.truststore.password=123456
ssl.keystore.location=/tmp/kafka-ssl-demo/jacek.keystore
ssl.keystore.password=123456
ssl.key.password=123456
```

Use `kafka-console-producer.sh` utility to send records to Kafka brokers over SSL:

```
kafka-console-producer.sh \
  --broker-list :9093 \
  --topic ssl \
  --producer.config /tmp/kafka-ssl-demo/jacek-client.properties
```

Tip	Use `export KAFKA_OPTS=-Djavax.net.debug=all` to debug SSL issues. Consult the source code of Java's SSLLogger.

That's all for the demo.

Demo: ACL Authorization

This demo shows how to use ACL authorization in Apache Kafka (using AclAuthorizer) to restrict topic operations.

In ACL words, the demo shows how to allow `Write` and `Read` operations to `ANY` topic to certain users.

You'll be using there users for different ACLs:

- `CN=root` - a user with all topic operations allowed
- `CN=producer` - a user with Write operation allowed
- `CN=consumer` - a user with Read operation allowed

Note	The demo is a follow-up to Demo: SSL Authentication. Please finish it first before this demo.

The demo is made up of the following steps:

1. Enable ACL Authorization
2. Review Authorization Logs
3. Enable DEBUG Logging Level for kafka.authorizer.logger
4. Create User CN=root
5. Define Super Users
6. (Optional) Allow Everyone If No ACL Found
7. List ACLs
8. Create User CN=producer
9. Create User CN=consumer
10. Restrict Topic Operations — Write for CN=producer
11. Restrict Topic Operations — Read for CN=consumer
12. Send Messages
13. Consume Messages

Enable ACL Authorization

Enable ACL authorization on Kafka brokers.

Add the following configuration property to `config/server-ssl.properties`:

```
authorizer.class.name=kafka.security.authorizer.AclAuthorizer
```

Restart the Kafka broker and observe the logs. You should find the following `ClusterAuthorizationException` at the very end of the logs:

```
org.apache.kafka.common.errors.ClusterAuthorizationException: Request Request(processo
r=0, connectionId=127.0.0.1:9093-127.0.0.1:62402-0, session=Session(User:CN=localhost,
/127.0.0.1), listenerName=ListenerName(SSL), securityProtocol=SSL, buffer=null) is not
 authorized.
```

That's because `User:CN=localhost` user is not authorized to execute an action (since by default no one is allowed to execute any action).

Review Authorization Logs

Access denials are logged at INFO level to `logs/kafka-authorizer.log` by default.

Review `logs/kafka-authorizer.log`. You should find the following INFO message (which corresponds to the `ClusterAuthorizationException` earlier):

```
Principal = User:CN=localhost is Denied Operation = ClusterAction from host = 127.0.0.
1 on resource = Cluster:LITERAL:kafka-cluster for request = UpdateMetadata with resour
ceRefCount = 1
```

Enable DEBUG Logging Level for kafka.authorizer.logger

Enable allowed accesses that are logged at DEBUG level. Edit `config/log4j.properties` and change the logging level of `kafka.authorizer.logger` to DEBUG:

```
log4j.logger.kafka.authorizer.logger=DEBUG, authorizerAppender
log4j.additivity.kafka.authorizer.logger=false
```

Restart the broker.

Create User CN=root

You will now "create" a user identified as `CN=root`.

Generate the keys and certificate of the user.

```
$ keytool \
  -genkey \
  -keystore root.keystore \
  -alias root \
  -dname CN=root \
  -keyalg RSA \
  -validity 365 \
  -storepass 123456
```

Export the certificate of the user from the keystore.

```
$ keytool \
  -certreq \
  -keystore root.keystore \
  -alias root \
  -file root.unsigned.crt \
  -storepass 123456
```

Sign the certificate signing request with the root CA.

```
$ openssl x509 \
  -req \
  -CA ca.crt \
  -CAkey ca.key \
  -in root.unsigned.crt \
  -out root.crt \
  -days 365 \
  -CAcreateserial \
  -passin pass:1234
Signature ok
subject=CN = root
Getting CA Private Key
```

Import the certificate of the CA into the user keystore.

```
$ keytool \
  -importcert \
  -file ca.crt \
  -alias ca \
  -keystore root.keystore \
  -storepass 123456 \
  -noprompt
Certificate was added to keystore
```

Import the signed certificate into the user keystore. Make sure to use the same `-alias` as you used ealier.

```
$ keytool \
  -importcert \
  -file root.crt \
  -alias root \
  -keystore root.keystore \
  -storepass 123456
Certificate reply was installed in keystore
```

Define Super Users

Super users are allowed to perform any operation on any resource in a Kafka cluster.

Define the broker (as `User:CN=localhost`) and `CN=root` as super users.

Add the following configuration property to `config/server-ssl.properties` . Note that the delimiter is semicolon (`;`) since user names may contain comma.

```
super.users=User:CN=localhost;User:CN=root
```

Restart the broker.

There should be no exceptions in the logs.

Moreover, `logs/kafka-authorizer.log` should have the following DEBUG messages:

```
principal = User:CN=localhost is a super user, allowing operation without checking acl
s.
Principal = User:CN=localhost is Allowed Operation = ClusterAction from host = 127.0.0
.1 on resource = Cluster:LITERAL:kafka-cluster for request = UpdateMetadata with resou
rceRefCount = 1
```

(Optional) Allow Everyone If No ACL Found

This step is optional.

For a less-secure broker configuration, you could add the following configuration property to `config/server-ssl.properties` :

```
allow.everyone.if.no.acl.found=true
```

That would make access more open to any client (with a valid and trusted certificate).

For the demo, enable it so you won't run into the following `GroupAuthorizationException` later:

```
org.apache.kafka.common.errors.GroupAuthorizationException: Not authorized to access g
roup: console-consumer-...
```

Edit `config/server-ssl.properties` and restart the broker.

List ACLs

Use kafka-acls utility to list the access control list (ACL). There should be none.

Create `root.properties` as a minimal configuration of a Kafka client to identify itself as `CN=root`.

```
security.protocol=SSL
ssl.truststore.location=/tmp/kafka-ssl-demo/client.truststore
ssl.truststore.password=123456
ssl.keystore.location=/tmp/kafka-ssl-demo/root.keystore
ssl.keystore.password=123456
ssl.key.password=123456
```

Use `--command-config` option to specify the SSL configuration.

```
kafka-acls.sh \
  --bootstrap-server :9093 \
  --list \
  --command-config /tmp/kafka-ssl-demo/root.properties
```

Create User CN=producer

You will now "create" a `CN=producer` user (that will have Write operation allowed).

Generate the keys and certificate of a Kafka client to be authenticated as **CN=producer**.

```
$ keytool \
  -genkey \
  -keystore producer.keystore \
  -alias producer \
  -dname CN=producer \
  -keyalg RSA \
  -validity 365 \
  -storepass 123456
```

Export the user certificate from the keystore.

```
$ keytool \
  -certreq \
  -keystore producer.keystore \
  -alias producer \
  -file producer.unsigned.crt \
  -storepass 123456
```

Sign the certificate signing request with the root CA.

```
$ openssl x509 \
  -req \
  -CA ca.crt \
  -CAkey ca.key \
  -in producer.unsigned.crt \
  -out producer.crt \
  -days 365 \
  -CAcreateserial \
  -passin pass:1234
Signature ok
subject=CN = producer
Getting CA Private Key
```

Import the certificate of the CA into the user keystore.

```
$ keytool \
  -import \
  -file ca.crt \
  -keystore producer.keystore \
  -alias ca \
  -storepass 123456 \
  -noprompt
Certificate was added to keystore
```

Import the signed certificate into the user keystore. Make sure to use the same `-alias` as you used ealier.

```
$ keytool \
  -import \
  -file producer.crt \
  -keystore producer.keystore \
  -alias producer \
  -storepass 123456 \
  -noprompt
Certificate reply was installed in keystore
```

Create User CN=consumer

You will now "create" a `CN=consumer` user (that will have Read operation allowed only).

Generate the keys and certificate of a Kafka client to be authenticated as **CN=consumer**.

```
keytool \
  -genkey \
  -keystore consumer.keystore \
  -alias consumer \
  -dname CN=consumer \
  -keyalg RSA \
  -validity 365 \
  -storepass 123456
```

Export the user certificate from the keystore.

```
keytool \
  -certreq \
  -keystore consumer.keystore \
  -alias consumer \
  -file consumer.unsigned.crt \
  -storepass 123456
```

Sign the certificate signing request with the root CA.

```
$ openssl x509 \
  -req \
  -CA ca.crt \
  -CAkey ca.key \
  -in consumer.unsigned.crt \
  -out consumer.crt \
  -days 365 \
  -CAcreateserial \
  -passin pass:1234
Signature ok
subject=CN = consumer
Getting CA Private Key
```

Import the certificate of the CA into the user keystore.

```
$ keytool \
  -import \
  -file ca.crt \
  -alias ca \
  -keystore consumer.keystore \
  -storepass 123456 \
  -noprompt
Certificate was added to keystore
```

Import the signed certificate into the user keystore. Make sure to use the same `-alias` as you used ealier.

```
$ keytool \
  -import \
  -file consumer.crt \
  -alias consumer \
  -keystore consumer.keystore \
  -storepass 123456
Certificate reply was installed in keystore
```

Restrict Topic Operations — Write for CN=produce

Use kafka-acls utility to restrict `Write` operation on any topic to `CN=produce` user (and super users).

```
kafka-acls.sh \
  --bootstrap-server :9093 \
  --add \
  --allow-principal User:CN=producer \
  --operation Write \
  --topic '*' \
  --command-config /tmp/kafka-ssl-demo/root.properties
```

List the ACLs using `kafka-acls` utility.

```
$ kafka-acls.sh \
  --bootstrap-server :9093 \
  --list \
  --command-config /tmp/kafka-ssl-demo/root.properties
Current ACLs for resource `ResourcePattern(resourceType=TOPIC, name=*, patternType=LIT
ERAL)`:
        (principal=User:CN=producer, host=*, operation=WRITE, permissionType=ALLOW)
```

Restrict Topic Operations — Read for CN=consumer

Use kafka-acls utility to restrict `Write` operation on any topic to `CN=consumer` user (and super users).

```
kafka-acls.sh \
  --bootstrap-server :9093 \
  --add \
  --allow-principal User:CN=consumer \
  --operation Read \
  --topic '*' \
  --command-config /tmp/kafka-ssl-demo/root.properties
```

List the ACLs using `kafka-acls` utility.

```
$ kafka-acls.sh \
  --bootstrap-server :9093 \
  --list \
  --command-config /tmp/kafka-ssl-demo/root.properties
Current ACLs for resource `ResourcePattern(resourceType=TOPIC, name=*, patternType=LIT
ERAL)`:
        (principal=User:CN=producer, host=*, operation=WRITE, permissionType=ALLOW)
        (principal=User:CN=consumer, host=*, operation=READ, permissionType=ALLOW)
```

Send Messages

Create `producer.properties` file as a minimal configuration of a Kafka client to use SSL authentication and identify itself as `CN=producer`:

```
security.protocol=SSL
ssl.truststore.location=/tmp/kafka-ssl-demo/client.truststore
ssl.truststore.password=123456
ssl.keystore.location=/tmp/kafka-ssl-demo/producer.keystore
ssl.keystore.password=123456
ssl.key.password=123456
```

Use `kafka-console-producer.sh` utility to send a message to the Kafka broker as `CN=producer`:

```
kafka-console-producer.sh \
  --broker-list :9093 \
  --topic ssl \
  --producer.config /tmp/kafka-ssl-demo/producer.properties
```

In `logs/kafka-authorizer.log` you should find the following:

```
DEBUG operation = Write on resource = Topic:LITERAL:ssl from host = 127.0.0.1 is Allow
 based on acl = User:CN=producer has Allow permission for operations: Write from hosts
: * (kafka.authorizer.logger)
DEBUG Principal = User:CN=producer is Allowed Operation = Describe from host = 127.0.0
.1 on resource = Topic:LITERAL:ssl for request = Metadata with resourceRefCount = 1 (k
afka.authorizer.logger)
DEBUG operation = Write on resource = Topic:LITERAL:ssl from host = 127.0.0.1 is Allow
 based on acl = User:CN=producer has Allow permission for operations: Write from hosts
: * (kafka.authorizer.logger)
DEBUG Principal = User:CN=producer is Allowed Operation = Write from host = 127.0.0.1
on resource = Topic:LITERAL:ssl for request = Produce with resourceRefCount = 1 (kafka
.authorizer.logger)
```

Consume Messages

Create `consumer.properties` file as a minimal configuration of a Kafka client to use SSL authentication and identify itself as `CN=consumer`:

```
security.protocol=SSL
ssl.truststore.location=/tmp/kafka-ssl-demo/client.truststore
ssl.truststore.password=123456
ssl.keystore.location=/tmp/kafka-ssl-demo/consumer.keystore
ssl.keystore.password=123456
ssl.key.password=123456
```

Use `kafka-console-consumer.sh` utility to consume messages as `CN=consumer`:

```
kafka-console-consumer.sh \
  --bootstrap-server :9093 \
  --topic ssl \
  --consumer.config /tmp/kafka-ssl-demo/consumer.properties
```

In `logs/kafka-authorizer.log` you should find the following:

```
DEBUG operation = Read on resource = Topic:LITERAL:ssl from host = 127.0.0.1 is Allow
based on acl = User:CN=consumer has Allow permission for operations: Read from hosts:
* (kafka.authorizer.logger)
DEBUG Principal = User:CN=consumer is Allowed Operation = Read from host = 127.0.0.1 o
n resource = Topic:LITERAL:ssl for request = Fetch with resourceRefCount = 1 (kafka.au
thorizer.logger)
```

That's all for the demo.

Authorizer

`Authorizer` is the abstraction of Kafka authorizers that Kafka brokers use to authorize actions based on access-control list (ACL).

From Wikipedia's Access-control list:

An **access-control list (ACL)** is a list of permissions attached to an object.

An ACL specifies which users or system processes are granted access to objects, as well as what operations are allowed on given objects.

Each entry in a typical ACL specifies a subject and an operation. For instance, if a file object has an ACL that contains (Alice: read,write; Bob: read), this would give Alice permission to read and write the file and Bob to only read it.

`Authorizer` is configured by authorizer.class.name configuration property.

`Authorizer` is a Configurable.

`Authorizer` is a Java's Closeable to close any resources that may have been opened (at start up).

Note	`Authorizer` abstraction is part of KIP-504 - Add new Java Authorizer Interface.

Table 1. Authorizer Contract

Method	Description
`acls`	`Iterable<AclBinding> acls(AclBindingFilter filter)` Lists ACL bindings for the provided filter (synchronously) Used when: • `JAuthorizerService` is requested to list ACLs for a filter • `KafkaApis` is requested to handle a DescribeAcls request
`authorize`	`List<AuthorizationResult> authorize(AuthorizableRequestContext requestContext, List<Action> actions)` Authorizes the actions performed by the request (synchronously) Used when `KafkaApis` is requested to authorize, filterAuthorized, and authorizedOperations.

`createAcls`	`List<? extends CompletionStage<AclCreateResult>> createAcls(`
	`  AuthorizableRequestContext requestContext,`
	`  List<AclBinding> aclBindings)`
	
	Creates new ACL bindings (asynchronously)
	
	Used when:
	
	• `JAuthorizerService` is requested to add ACLs for a resource
	• `KafkaApis` is requested to handle a CreateAcls request

`deleteAcls`	`List<? extends CompletionStage<AclDeleteResult>> deleteAcls(`
	`  AuthorizableRequestContext requestContext,`
	`  List<AclBindingFilter> aclBindingFilters)`
	
	Deletes all ACL bindings that match the provided filters (asynchronously)
	
	Used when:
	
	• `JAuthorizerService` is requested to remove ACLs for the provided filters
	• `KafkaApis` is requested to handle a DeleteAcls request

`start`	`Map<Endpoint, ? extends CompletionStage<Void>> start(`
	`  AuthorizerServerInfo serverInfo)`
	
	Starts loading authorization metadata (asynchronously)
	
	Returns futures that can be used to wait until metadata for authorizing requests on each listener is available. The future returned for each listener must return only when authorizer is ready to authorize requests on the listener.
	
	Used when `KafkaServer` is requested to start up
	
	Note: Executed after configure (as a Configurable).

Table 2. Authorizers

Authorizer	Description
AclAuthorizer	Uses Apache ZooKeeper to persist ACLs
AuthorizerWrapper	Wrapper of authorizers based on the deprecated `kafka.security.auth.Authorizer` API

AclAuthorizer

`AclAuthorizer` (`kafka.security.authorizer.AclAuthorizer`) is an Authorizer that uses Apache ZooKeeper to persist ACLs.

Note	`AclAuthorizer` is available since Apache Kafka 2.4.0 (KIP-504 - Add new Java Authorizer Interface).

`AclAuthorizer` uses properties for configuration (when `KafkaServer` is requested to start up).

`AclAuthorizer` may use its own dedicated ZooKeeper for ACLs per authorizer.zookeeper.url configuration property (that could be separate from the cluster's ZooKeeper).

`AclAuthorizer` uses a dedicated KafkaZkClient to access ZooKeeper.

Tip	Consult Demo: ACL Authorization.

Table 1. AclAuthorizer's Configuration Properties

Name	Description
`allow.everyone.if.no.acl.found`	Allows all users to access resources with no ACL Default: `false`
`authorizer.zookeeper.connection.timeout.ms`	Default: zkConnectionTimeoutMs
`authorizer.zookeeper.max.in.flight.requests`	Default: zookeeper.max.in.flight.requests
`authorizer.zookeeper.session.timeout.ms`	Default: zookeeper.session.timeout.ms
`authorizer.zookeeper.url`	URL of the dedicated Zookeeper to store ACLs (that can be different from zookeeper.connect) Default: zookeeper.connect
`super.users`	Semicolon-separated list of Kafka principals that are super users and have access to all the resources for all actions from all hosts. Default: (empty)

Tip	Enable `ALL` logging level for `kafka.authorizer.logger` logger to see what happens
	Add the following line to `config/log4j.properties`:
	`log4j.logger.kafka.authorizer.logger=ALL`
	Refer to Logging.
	--- 	
	Please note that Kafka comes with a preconfigured `kafka.authorizer.logger` logger `config/log4j.properties`:
	`log4j.appender.authorizerAppender=org.apache.log4j.DailyRollingFileAppender`
	`log4j.appender.authorizerAppender.DatePattern='.'yyyy-MM-dd-HH`
	`log4j.appender.authorizerAppender.File=${kafka.logs.dir}/kafka-authorizer.log`
	`log4j.appender.authorizerAppender.layout=org.apache.log4j.PatternLayout`
	`log4j.appender.authorizerAppender.layout.ConversionPattern=[%d] %p %m (%c)%n`
	`# Access denials are logged at INFO level, change to DEBUG to also log allowed a`
	`log4j.logger.kafka.authorizer.logger=INFO, authorizerAppender`
	`log4j.additivity.kafka.authorizer.logger=false`
	That means that the logs of `AclAuthorizer` go to `logs/kafka-authorizer.log` file at logging level and are not added to the main logs (per `log4j.additivity` being off).

`configure` Method

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

Note	`configure` is part of the Configurable contract to pass configurations.

`configure` creates a KafkaZkClient (using the properties in the given `javaConfigs`) with `kafka.security` metric group and `AclAuthorizer` metric type, and `ACL authorizer` logger prefix.

Figure 1. AclAuthorizer's KafkaZkClient in jconsole

`configure` requests the KafkaZkClient to createAclPaths.

`configure` startZkChangeListeners and then loadCache.

`start` Method

```
start(
  serverInfo: AuthorizerServerInfo): util.Map[Endpoint, _ <: CompletionStage[Void]]
```

Note	`start` is part of the Authorizer contract to…FIXME.

`start` …FIXME

`authorize` Method

```
authorize(
  requestContext: AuthorizableRequestContext,
  actions: List[Action]): List[AuthorizationResult]
```

Note	`authorize` is part of the Authorizer contract to…FIXME.

`authorize` …FIXME

`createAcls` Method

```
createAcls(
  requestContext: AuthorizableRequestContext,
  aclBindings: util.List[AclBinding]): util.List[_ <: CompletionStage[AclCreateResult]]
```

Note	`createAcls` is part of the Authorizer contract to…FIXME.

`createAcls` …FIXME

`deleteAcls` Method

```
deleteAcls(
  requestContext: AuthorizableRequestContext,
  aclBindingFilters: util.List[AclBindingFilter]): util.List[_ <: CompletionStage[AclD
eleteResult]]
```

Note	`deleteAcls` is part of the Authorizer contract to…FIXME.

`deleteAcls` …FIXME

`acls` Method

```
acls(
  filter: AclBindingFilter): lang.Iterable[AclBinding]
```

Note	`acls` is part of the Authorizer contract to…FIXME.

`acls` …FIXME

`isSuperUser` Method

```
isSuperUser(
  principal: KafkaPrincipal): Boolean
```

`isSuperUser` ...FIXME

Note	`isSuperUser` is used when `AclAuthorizer` is requested to authorizeAction.

`authorizeAction` Internal Method

```
authorizeAction(
  requestContext: AuthorizableRequestContext,
  action: Action): AuthorizationResult
```

`authorizeAction` ...FIXME

Note	`authorizeAction` is used when `AclAuthorizer` is requested to authorize.

`aclsAllowAccess` Internal Helper Method

```
aclsAllowAccess: Boolean
```

`aclsAllowAccess` is positive (`true`) when...FIXME

`isEmptyAclAndAuthorized` Internal Helper Method

```
isEmptyAclAndAuthorized(acls: Set[Acl]): Boolean
```

`isEmptyAclAndAuthorized` is positive (`true`) when the given acls are empty and the shouldAllowEveryoneIfNoAclIsFound internal flag is positive. Otherwise, `isEmptyAclAndAuthorized` is negative (`false`).

`isEmptyAclAndAuthorized` prints out the following DEBUG message when positive:

```
No acl found for resource [resource], authorized = [shouldAllowEveryoneIfNoAclIsFound]
```

AclAuthorizer and KafkaZkClient

```
zkClient: KafkaZkClient
```

`zkClient` is a KafkaZkClient that is created and requested to createAclPaths when `AclAuthorizer` is requested to configure.

`AclAuthorizer` uses the `KafkaZkClient` for the following:

- loadCache and startZkChangeListeners
- updateResourceAcls
- getAclsFromZk
- updateAclChangedFlag

`KafkaZkClient` is requested to close when `AclAuthorizer` is.

`startZkChangeListeners` Internal Method

```
startZkChangeListeners(): Unit
```

`startZkChangeListeners` ...FIXME

Note	`startZkChangeListeners` is used when `AclAuthorizer` is requested to configure.

`loadCache` Internal Method

```
loadCache(): Unit
```

`loadCache` ...FIXME

Note	`loadCache` is used when `AclAuthorizer` is requested to configure.

`updateResourceAcls` Internal Method

```
updateResourceAcls(
  resource: Resource)(
  getNewAcls: Set[Acl] => Set[Acl]): Boolean
```

`updateResourceAcls` ...FIXME

Note	`updateResourceAcls` is used when `AclAuthorizer` is requested to createAcls and deleteAcls.

`updateAclChangedFlag` Internal Method

```
updateAclChangedFlag(
  resource: Resource): Unit
```

`updateAclChangedFlag` ...FIXME

Note	`updateAclChangedFlag` is used when `AclAuthorizer` is requested to updateResourceAcls.

`getAclsFromZk` Internal Method

```
getAclsFromZk(
  resource: Resource): VersionedAcls
```

`getAclsFromZk` ...FIXME

Note	`getAclsFromZk` is used when `AclAuthorizer` is requested to...FIXME

AuthorizerWrapper

`AuthorizerWrapper` is...FIXME

KafkaPrincipal

`KafkaPrincipal` is...FIXME

KafkaPrincipalBuilder

`KafkaPrincipalBuilder` is the abstraction of principal builders that can build a KafkaPrincipal for authentication.

Note	Configurable and `java.io.Closeable` interfaces are respected if implemented.

`KafkaPrincipalBuilder` is configured by the principal.builder.class configuration property.

Table 1. KafkaPrincipalBuilder Contract

Method	Description

`build`	`KafkaPrincipal build(`
	`  AuthenticationContext context)`
	
	Builds a KafkaPrincipal for the `AuthenticationContext`
	
	Used when:
	• `PlaintextAuthenticator` is requested for a `principal`
	• `SaslServerAuthenticator` is requested for a `principal`
	• `SslAuthenticator` is requested for a principal

Note	DefaultKafkaPrincipalBuilder is the default and only known `KafkaPrincipalBuilder` in Apache Kafka.

DefaultKafkaPrincipalBuilder

`DefaultKafkaPrincipalBuilder` is...FIXME

ChannelBuilder

`ChannelBuilder` is the abstraction of channel builders that can build a KafkaChannel.

`ChannelBuilder` is a Configurable.

`ChannelBuilder` is a Java `AutoCloseable`.

`ChannelBuilder` is created using ChannelBuilders utility.

`ChannelBuilder` is used to create a Selector.

Table 1. ChannelBuilder Contract

Method	Description
`buildChannel`	`KafkaChannel buildChannel(String id, SelectionKey key, int maxReceiveSize, MemoryPool memoryPool)` Used when `Selector` is requested to buildAndAttachKafkaChannel

Table 2. ChannelBuilders

ChannelBuilder	Description
PlaintextChannelBuilder	
SaslChannelBuilder	
SslChannelBuilder	

SaslChannelBuilder

`SaslChannelBuilder` is a ChannelBuilder for SSL channels on Kafka clients and brokers.

`SaslChannelBuilder` is a `ListenerReconfigurable`.

`SaslChannelBuilder` is created when `ChannelBuilders` utility is used to create a ChannelBuilder for `SASL_SSL` or `SASL_PLAINTEXT` security protocols.

Creating SaslChannelBuilder Instance

`SaslChannelBuilder` takes the following to be created:

- `Mode` (i.e. `CLIENT` or `SERVER`)
- Lookup table of `JaasContexts` by name
- `SecurityProtocol`
- Listener name (only for `SERVER` channel builder)
- `isInterBrokerListener` flag
- `clientSaslMechanism` text
- `handshakeRequestEnable` flag
- `CredentialCache`
- `DelegationTokenCache`
- `Time`

`SaslChannelBuilder` initializes the internal properties.

`configure` Method

```
void configure(
  Map<String, ?> configs)
```

Note	`configure` is part of the Configurable contract to pass configurations.

`configure` ...FIXME

Creating KafkaChannel — `buildChannel` Method

```
KafkaChannel buildChannel(
  String id,
  SelectionKey key,
  int maxReceiveSize,
  MemoryPool memoryPool)
```

Note	`buildChannel` is part of the ChannelBuilder contract to build a `KafkaChannel`.

`buildChannel` ...FIXME

`buildTransportLayer` Method

```
TransportLayer buildTransportLayer(
  String id,
  SelectionKey key,
  SocketChannel socketChannel)
```

`buildTransportLayer` ...FIXME

Note	`buildTransportLayer` is used when `SaslChannelBuilder` is requested to buildChannel.

SslChannelBuilder

`SslChannelBuilder` is a ChannelBuilder for Kafka clients and brokers when security.protocol configuration property is `SSL`.

`SslChannelBuilder` is a `ListenerReconfigurable`.

`SslChannelBuilder` is created when `ChannelBuilders` utility is used to create a ChannelBuilder for SSL security protocol.

Creating SslChannelBuilder Instance

`SslChannelBuilder` takes the following to be created:

- `Mode` (i.e. `CLIENT` or `SERVER`)
- Listener name (only for `SERVER` channel builder)
- `isInterBrokerListener` flag

`SslChannelBuilder` initializes the internal properties.

SslChannelBuilder and SslPrincipalMapper

```
SslPrincipalMapper sslPrincipalMapper
```

`sslPrincipalMapper` is a SslPrincipalMapper that is used to create a SslAuthenticator when `SslChannelBuilder` is requested to build a channel.

`sslPrincipalMapper` is created based on SSL principal mapping rules when `SslChannelBuilder` is requested to configure with ssl.principal.mapping.rules configuration property defined.

Creating KafkaChannel — `buildChannel` Method

```
KafkaChannel buildChannel(
  String id,
  SelectionKey key,
  int maxReceiveSize,
  MemoryPool memoryPool)
```

Note	`buildChannel` is part of the ChannelBuilder contract to build a `KafkaChannel`.

`buildChannel` ...FIXME

`configure` Method

```
void configure(
  Map<String, ?> configs)
```

Note	`configure` is part of the Configurable contract to pass configurations.

`configure` initializes the sslPrincipalMapper internal registry (using SslPrincipalMapper utility) if ssl.principal.mapping.rules configuration property is defined.

`configure` creates a SslFactory (for the mode and the isInterBrokerListener flag) and requests it to configure (with the given `configs`).

`buildTransportLayer` Method

```
SslTransportLayer buildTransportLayer(
  SslFactory sslFactory,
  String id,
  SelectionKey key,
  String host)
```

`buildTransportLayer` ...FIXME

Note	`buildTransportLayer` is used when `SslChannelBuilder` is requested to buildChannel.

Internal Properties

Name	Description
`configs`	Configuration properties
`sslFactory`	SslFactory

SslAuthenticator

`SslAuthenticator` is an Authenticator for KafkaChannel when security.protocol configuration property is `SSL`.

`SslAuthenticator` is created (lazily as part of `Supplier<Authenticator>`) when `SslChannelBuilder` is requested to build a channel.

`SslAuthenticator` is an internal class of SslChannelBuilder.

Creating SslAuthenticator Instance

`SslAuthenticator` takes the following to be created:

- Configuration properties
- `SslTransportLayer`
- `ListenerName`
- SslPrincipalMapper

`SslAuthenticator` initializes the KafkaPrincipalBuilder.

SslAuthenticator and KafkaPrincipalBuilder

```
KafkaPrincipalBuilder principalBuilder
```

`principalBuilder` is a `KafkaPrincipalBuilder`.

`principalBuilder` is created for `SslAuthenticator` based on the given configuration properties, the SslTransportLayer and the SslPrincipalMapper.

`principal` Method

```
KafkaPrincipal principal()
```

Note	`principal` is part of the Authenticator contract to…FIXME.

`principal` …FIXME

ChannelBuilders Utility

`ChannelBuilders` utility is used for the following:

- Creating a KafkaPrincipalBuilder (per principal.builder.class configuration property)
- ...

Creating ChannelBuilder (for Kafka Clients and Inter-Broker Communication) — `clientChannelBuilder` Method

```
ChannelBuilder clientChannelBuilder(
  SecurityProtocol securityProtocol,
  JaasContext.Type contextType,
  AbstractConfig config,
  ListenerName listenerName,
  String clientSaslMechanism,
  Time time,
  boolean saslHandshakeRequestEnable)
```

`clientChannelBuilder` ...FIXME

Note	`clientChannelBuilder` is used when: • `ClientUtils` utility is used to create a ChannelBuilder (with `CLIENT` JAAS context type and `null` listener name) • `ControllerChannelManager` is requested to register a new broker • `TransactionMarkerChannelManager` is created (for a TransactionCoordinator) • `KafkaServer` is requested to controlledShutdown • ReplicaFetcherBlockingSend is created

`serverChannelBuilder` Method

```
ChannelBuilder serverChannelBuilder(
  ListenerName listenerName,
  boolean isInterBrokerListener,
  SecurityProtocol securityProtocol,
  AbstractConfig config,
  CredentialCache credentialCache,
  DelegationTokenCache tokenCache,
  Time time)
```

`serverChannelBuilder` ...FIXME

Note	`serverChannelBuilder` is used when Processor is created (and creates a Selector).

Creating KafkaPrincipalBuilder Per principal.builder.class Configuration Property — `createPrincipalBuilder` Utility

```
KafkaPrincipalBuilder createPrincipalBuilder(
  Map<String, ?> configs,
  TransportLayer transportLayer,
  Authenticator authenticator,
  KerberosShortNamer kerberosShortNamer,
  SslPrincipalMapper sslPrincipalMapper)
```

`createPrincipalBuilder` reads the principal.builder.class configuration property and creates a KafkaPrincipalBuilder:

- For no class defined (`null`), `createPrincipalBuilder` creates a DefaultKafkaPrincipalBuilder (with the `KerberosShortNamer` and `SslPrincipalMapper`)
- For a KafkaPrincipalBuilder implementation, `createPrincipalBuilder` creates an instance
- For a `PrincipalBuilder` implementation, `createPrincipalBuilder` creates an instance

For a builder that is also a Configurable, `createPrincipalBuilder` requests it to configure (with the given `configs`).

`createPrincipalBuilder` throws an `InvalidConfigurationException` for any other implementation:

```
Type [principalBuilderClassName] is not an instance of
org.apache.kafka.common.security.auth.PrincipalBuilder or
org.apache.kafka.common.security.auth.KafkaPrincipalBuilder
```

Note	`createPrincipalBuilder` is used for creating the Authenticators (i.e. `PlaintextAuthenticator` , SslAuthenticator, and `SaslServerAuthenticator`).

`create` Internal Utility

```
ChannelBuilder create(
  SecurityProtocol securityProtocol,
  Mode mode,
  JaasContext.Type contextType,
  AbstractConfig config,
  ListenerName listenerName,
  boolean isInterBrokerListener,
  String clientSaslMechanism,
  boolean saslHandshakeRequestEnable,
  CredentialCache credentialCache,
  DelegationTokenCache tokenCache,
  Time time)
```

`create` ...FIXME

Note	`create` is used when `ChannelBuilders` utility is requested to clientChannelBuilder (`CLIENT` mode) and serverChannelBuilder (`SERVER` mode).

Authenticator

`Authenticator` is the abstraction of authenticators that can authenticate a KafkaPrincipal.

`Authenticator` is a `java.io.Closeable`.

`Authenticator` is used by KafkaChannel.

Table 1. Authenticator Contract

Method	D
`authenticate`	`void authenticate()` Used when `KafkaChannel` is reque ready)
`clientSessionReauthenticationTimeNanos`	`Long clientSessionReauthenticat` Default: `null` Used when `KafkaChannel` is reque
`complete`	`boolean complete()` Used when `KafkaChannel` is reque
`connectedClientSupportsReauthentication`	`boolean connectedClientSupports` Used when…FIXME
`getAndClearResponsesReceivedDuringReauthentication`	`List<NetworkReceive> getAndClea` Used when…FIXME
`handleAuthenticationFailure`	`void handleAuthenticationFailur` Used when…FIXME
`principal`	`KafkaPrincipal principal()`

	Used when `KafkaChannel` is reque

`reauthenticate`	`void reauthenticate(`
	`ReauthenticationContext reaut`
	
	Used when…FIXME

`reauthenticationLatencyMs`	`Long reauthenticationLatencyMs(` Used when…FIXME
`serverSessionExpirationTimeNanos`	`Long serverSessionExpirationTim` Used when…FIXME

Table 2. Authenticators

Authenticator	Description
PlaintextAuthenticator	
SaslClientAuthenticator	
SaslServerAuthenticator	
SslAuthenticator	

KafkaChannel

`KafkaChannel` is created when ChannelBuilders are requested to build one (i.e. `PlaintextChannelBuilder`, `SaslChannelBuilder`, and SslChannelBuilder).

Creating KafkaChannel Instance

`KafkaChannel` takes the following to be created:

- Identifier
- `TransportLayer`
- (`Supplier` of) Authenticator
- `maxReceiveSize`
- `MemoryPool`

`KafkaChannel` initializes the internal properties.

KafkaChannel and Authenticator

```
Authenticator authenticator
```

`authenticator` is a Authenticator that is created when `KafkaChannel` is and requested to swapAuthenticatorsAndBeginReauthentication.

Used for principal, prepare, completeCloseOnAuthenticationFailure, ready, maybeBeginServerReauthentication, maybeBeginClientReauthentication, reauthenticationLatencyMs, serverAuthenticationSessionExpired, getAndClearResponsesReceivedDuringReauthentication, connectedClientSupportsReauthentication, and swapAuthenticatorsAndBeginReauthentication

Closed when `KafkaChannel` is requested to close.

`prepare` Method

```
void prepare()
```

`prepare` ...FIXME

Note	`prepare` is used when...FIXME

`ready` Method

```
boolean ready()
```

`ready` is positive (`true`) when the TransportLayer is ready and the Authenticator is complete

Note	`ready` is used when...FIXME

`principal` Method

```
KafkaPrincipal principal()
```

`principal` simply requests the Authenticator for the principal.

Note	`principal` is used when...FIXME

`maybeBeginClientReauthentication` Method

```
boolean maybeBeginClientReauthentication(
  Supplier<Long> nowNanosSupplier)
```

`maybeBeginClientReauthentication` ...FIXME

Note	`maybeBeginClientReauthentication` is used when...FIXME

Internal Properties

Name	Description
`transportLayer`	`TransportLayer`

Selectable

`Selectable` is the contract for asynchronous, multi-channel network I/O for NetworkClient.

Note	Selector is the one and only `Selectable`.

```
package org.apache.kafka.common.network;

public interface Selectable {
  void connect(String id, InetSocketAddress address, int sendBufferSize, int receiveBu
fferSize) throws IOException;
  void close();
  void close(String id);

  void send(Send send);
  void poll(long timeout) throws IOException;
  void wakeup();

  List<Send> completedSends();
  List<NetworkReceive> completedReceives();

  Map<String, ChannelState> disconnected();
  List<String> connected();

  void mute(String id);
  void unmute(String id);
  void muteAll();
  void unmuteAll();

  boolean isChannelReady(String id);
}
```

Table 1. Selectable Contract (in alphabetical order)

Method	Description
`connect`	Used exclusively when `NetworkClient` is requested to establish a connection to a broker
`poll`	

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

Tip	Enable `ALL` logging levels for `org.apache.kafka.common.network.Selector` logger to see what happens inside. Add the following line to `config/log4j.properties`: `log4j.logger.org.apache.kafka.common.network.Selector=ALL` Refer to Logging.

`connect` Method

```
void connect(
  String id,
  InetSocketAddress address,
  int sendBufferSize,
  int receiveBufferSize) throws IOException
```

Note	`connect` is a part of Selectable contract that `NetworkClient` uses when requested to establish a connection to a broker.

`connect` ...FIXME

`addToCompletedReceives` Internal Method

```
void addToCompletedReceives()
void addToCompletedReceives(
  KafkaChannel channel,
  Deque<NetworkReceive> stagedDeque)
```

`addToCompletedReceives` ...FIXME

Note	`addToCompletedReceives` is used exclusively when `Selector` is requested to poll.

`poll` Method

```
void poll(
  long timeout)
```

Note	`poll` is part of the Selectable Contract to...FIXME.

`poll` ...FIXME

`register` Method

```
void register(
  String id,
  SocketChannel socketChannel)
```

`register` ...FIXME

Note	`register` is used when `Processor` is requested to configureNewConnections.

`registerChannel` Method

```
SelectionKey registerChannel(
  String id,
  SocketChannel socketChannel,
  int interestedOps)
```

`registerChannel` ...FIXME

Note	`registerChannel` is used when `Selector` is requested to connect and register.

`buildAndAttachKafkaChannel` Internal Method

```
KafkaChannel buildAndAttachKafkaChannel(
  SocketChannel socketChannel,
  String id,
  SelectionKey key)
```

`buildAndAttachKafkaChannel` ...FIXME

Note	`buildAndAttachKafkaChannel` is used when `Selector` is requested to registerChannel

Kafka Tools

Kafka tools includes the following administration tools:

- AclCommand
- ConfigCommand
- ConsumerGroupCommand
- `DelegationTokenCommand`
- `DeleteRecordsCommand`
- DumpLogSegments
- `EndToEndLatency`
- LeaderElectionCommand
- `LogDirsCommand`
- ReassignPartitionsCommand
- ReplicaVerificationTool
- `StreamsResetter`
- TopicCommand

`ConsoleProducer`

`kafka.tools.ConsoleProducer`

```
./bin/kafka-console-producer.sh --broker-list localhost:9092 --topic my-topic
```

`ConsoleConsumer`

`kafka.tools.ConsoleConsumer`

```
./bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic my-topic
```

kafka-configs.sh Shell Script

`kafka-configs.sh` is a shell script that…FIXME

```
./bin/kafka-configs.sh \
  --alter \
  --zookeeper localhost:2181 \
  --entity-type topics \
  --entity-name test \
  --add-config retention.ms=5000
```

```
./bin/kafka-configs.sh \
  --alter \
  --bootstrap-server localhost:9092 \
  --entity-type brokers \
  --entity-name 0 \
  --add-config advertised.listeners=PLAINTEXT://localhost:9092
```

kafka-topics.sh Shell Script

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

kafka-leader-election.sh Shell Script

`kafka-leader-election.sh` is a shell script to launch kafka.admin.LeaderElectionCommand standalone application for Partition Leader Election.

Among the supported options, `kafka-leader-election.sh` accepts accepts a JSON file with partitions to include in election.

```
$ kafka-topics.sh \
    --bootstrap-server :9092 \
    --create \
    --topic t1 \
    --partitions 3 \
    --replication-factor 1

$ cat leader-election.json
{ "partitions":
  [
    { "topic": "t1", "partition": 1 },
    { "topic": "t1", "partition": 2 }
  ]
}

$ ./bin/kafka-leader-election.sh \
  --path-to-json-file leader-election.json \
  --election-type preferred \
  --bootstrap-server :9092
```

When executed with no options, `kafka-leader-election.sh` assumes --help.

```
$ kafka-leader-election.sh
This tool attempts to elect a new leader for a set of topic partitions. The type of el
ections supported are preferred replicas and unclean replicas.
Option                                   Description
------                                   -----------
--admin.config <String: config file>     Configuration properties files to pass
                                           to the admin client
--all-topic-partitions                   Perform election on all of the
                                           eligible topic partitions based on
                                           the type of election (see the --
                                           election-type flag). Not allowed if
                                           --topic or --path-to-json-file is
                                           specified.
--bootstrap-server <String: host:port>   A hostname and port for the broker to
                                           connect to, in the form host:port.
                                           Multiple comma separated URLs can be
                                           given. REQUIRED.
--election-type <[PREFERRED,UNCLEAN]:    Type of election to attempt. Possible
  election type>                           values are "preferred" for preferred
                                           leader election or "unclean" for
                                           unclean leader election. If
                                           preferred election is selection, the
                                           election is only performed if the
                                           current leader is not the preferred
                                           leader for the topic partition. If
                                           unclean election is selected, the
                                           election is only performed if there
                                           are no leader for the topic
                                           partition. REQUIRED.
--help                                   Print usage information.
--partition <Integer: partition id>      Partition id for which to perform an
                                           election. REQUIRED if --topic is
                                           specified.
--path-to-json-file <String: Path to     The JSON file with the list  of
  JSON file>                               partition for which leader elections
                                           should be performed. This is an
                                           example format.
                                         {"partitions":
                                                 [{"topic": "foo", "partition": 1},
                                                  {"topic": "foobar", "partition": 2}]
                                         }
                                         Not allowed if --all-topic-partitions
                                           or --topic flags are specified.
--topic <String: topic name>             Name of topic for which to perform an
                                           election. Not allowed if --path-to-
                                           json-file or --all-topic-partitions
                                           is specified.
--version                                Display Kafka version.
```

LeaderElectionCommand Standalone Application

`kafka.admin.LeaderElectionCommand` is a standalone application for preferred or unclean leader election for all or some partitions.

`LeaderElectionCommand` uses AdminClient internally and is simply a command-line front-end to Admin.electLeaders.

`LeaderElectionCommand` can be executed on command line using kafka-leader-election.sh shell script (that simply uses `kafka-run-class` shell script to launch the command).

Note	`PreferredReplicaLeaderElectionCommand` and the corresponding `kafka-preferred-replica-election.sh` utility have been deprecated and replaced by the more generic `LeaderElectionCommand` and `kafka-leader-election.sh` utility.

`LeaderElectionCommand` supports options:

- --bootstrap-server and --election-type are required
- --topic, --all-topic-partitions and --path-to-json-file are mutually exclusive (only one is required)

Table 1. Options

Option	Description
`admin.config`	File with configuration properties of an admin client • `bootstrap.servers` is --bootstrap-server • `request.timeout.ms` is the timeout argument (default: `30 seconds`)
`all-topic-partitions`	Perform election on all of the eligible topic partitions based on the type of election Not allowed with --topic or --path-to-json-file
`bootstrap-server`	**(required)** Brokers to connect to as a comma-separated list of `host:port` URLs
`election-type`	**(required)** Type of election to attempt: • `preferred` for preferred leader election (only performed if the current leader is not the preferred leader for the topic partition)

	• `unclean` for unclean leader election (only performed if there are no leader for the topic partition)
`help`	Prints usage information
`partition`	Partition id to perform an election for Required and only acceptable with --topic
`path-to-json-file`	JSON file with partitions for leader elections Default: `(empty)` (all partitions) The format of the JSON file should be as follows: `{ "partitions": [ { "topic": "foo", "partition": 1 }, { "topic": "foobar", "partition": 2 } ] }` The JSON file should not be empty and has no duplicate partitions (or `AdminOperationException` is thrown)
`topic`	Name of topic for which to perform an election. Not allowed with --path-to-json-file or --all-topic-partitions
`version`	Displays Kafka version

Tip	Read KIP-183 - Change PreferredReplicaLeaderElectionCommand to use AdminClient.

Executing Standalone Application — `main` Utility

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

`main` is the entry point of the `LeaderElectionCommand` when launched on command line (e.g. using kafka-leader-election.sh shell script).

Internally, `main` simply run with the timeout of `30` seconds.

`run` Utility

```
run(
  args: Array[String],
  timeout: Duration): Unit
```

`run` reads the options from the command line.

`run` prints out the following for no command-line options, --help or --version:

```
This tool helps to causes leadership for each partition to be
transferred back to the 'preferred replica', it can be used to
balance leadership among the servers.
```

`run` creates a AdminClient (with the configuration properties from the admin.config properties file if specified).

In the end, `run` electLeaders followed by requesting the `AdminClient` to close.

Note	`run` is used exclusively when `PreferredReplicaLeaderElectionCommand` command-line application is executed.

`electPreferredLeaders` Method

```
electPreferredLeaders(
  partitionsFromUser: Option[Set[TopicPartition]]): Unit
```

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

```
Calling AdminClient.electPreferredLeaders([partitions])
```

`electPreferredLeaders` creates a new AdminClient and requests to electPreferredLeaders.

Note	`electPreferredLeaders` is used exclusively when `PreferredReplicaLeaderElectionCommand` is requested to run.

Closing AdminClient — `close` Method

```
close(): Unit
```

`close` prints out the following DEBUG message to the logs and requests the `AdminClient` to close.

```
Closing AdminClient
```

Note	`close` is used exclusively when `PreferredReplicaLeaderElectionCommand` is requested to run.

Validating Options — `validate` Internal Utility

```
validate(
  commandOptions: LeaderElectionCommandOptions): Unit
```

`validate` ...FIXME

Note	`validate` is used when `LeaderElectionCommand` is requested to run.

Validating Options — `parseReplicaElectionData` Internal Utility

```
parseReplicaElectionData(
  jsonString: String): Set[TopicPartition]
```

`parseReplicaElectionData` parses the given JSON string that is assumed to have one of more `partitions` top-level field with `topic` and `partition` fields.

```
{ "partitions":
  [
    { "topic": "foo", "partition": 1 },
    { "topic": "foobar", "partition": 2 }
  ]
}
```

`parseReplicaElectionData` throws an `AdminOperationException` for duplicate partitions:

```
Replica election data contains duplicate partitions: [duplicatePartitions]
```

`parseReplicaElectionData` throws an `AdminOperationException` when the JSON string has no `partitions` fields:

```
Replica election data is missing "partitions" field
```

`parseReplicaElectionData` throws an `AdminOperationException` when the JSON string is invalid:

```
Replica election data is empty
```

Note	`parseReplicaElectionData` is used when `LeaderElectionCommand` is requested to run (and handle --path-to-json-file option).

Leader Election — `electLeaders` Internal Utility

```
electLeaders(
  client: Admin,
  electionType: ElectionType,
  topicPartitions: Option[Set[TopicPartition]]): Unit
```

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

```
Calling AdminClient.electLeaders([electionType], [partitions])
```

`electLeaders` requests the `Admin` client to electLeaders.

`electLeaders` splits the election results into three categories: partitions that succeeded, didn't changed (*noops*), and failed.

`electLeaders` prints out the following message to standard output for successful partitions:

```
Successfully completed leader election ([electionType]) for partitions [partitions]
```

`electLeaders` prints out the following message to standard output for partitions that didn't change the leaders (*noops*):

```
Valid replica already elected for partitions [partitions]
```

`electLeaders` prints out the following message to standard output for every failed partition and throws an `AdminCommandFailedException`.

```
Error completing leader election ([electionType]) for partition: [topicPartition]: [ex
ception]
```

Note	`electLeaders` is used when `LeaderElectionCommand` is requested to run.

kafka-acls.sh Shell Script

`kafka-acls.sh` is the **Kafka Authorization management CLI** for ACL management (e.g. add, list, or remove ACLs). Use --help to learn the supported options

`kafka-acls.sh` simply launches kafka.admin.AclCommand administration utility.

Help

When executed with no options or --help option, `kafka-acls.sh` prints out the help.

```
$ kafka-acls.sh
This tool helps to manage acls on kafka.
Option                                   Description
------                                   -----------
--add                                    Indicates you are trying to add ACLs.
--allow-host <String: allow-host>        Host from which principals listed in --
                                           allow-principal will have access. If
                                           you have specified --allow-principal
                                           then the default for this option
                                           will be set to * which allows access
                                           from all hosts.
--allow-principal <String: allow-        principal is in principalType:name
  principal>                               format. Note that principalType must
                                           be supported by the Authorizer being
                                           used. For example, User:* is the
                                           wild card indicating all users.
--authorizer <String: authorizer>        Fully qualified class name of the
                                           authorizer, defaults to kafka.
                                           security.auth.SimpleAclAuthorizer.
...
--remove                                 Indicates you are trying to remove
                                           ACLs.
...
--transactional-id <String:              The transactionalId to which ACLs
  transactional-id>                        should be added or removed. A value
                                           of * indicates the ACLs should apply
                                           to all transactionalIds.
--version                                Display Kafka version.
```

Listing ACLs

```
// Use AuthorizerService / --authorizer-properties
$ kafka-acls.sh --list --authorizer-properties zookeeper.connect=zookeeper:2181
```

```
// FIXME
// Use AdminClientService / --bootstrap-server
$ kafka-acls.sh --list --bootstrap-server kafka:9092
Error while executing ACL command: org.apache.kafka.common.errors.SecurityDisabledExce
ption: No Authorizer is configured on the broker
java.util.concurrent.ExecutionException: org.apache.kafka.common.errors.SecurityDisabl
edException: No Authorizer is configured on the broker
        at org.apache.kafka.common.internals.KafkaFutureImpl.wrapAndThrow(KafkaFutureI
mpl.java:45)
        at org.apache.kafka.common.internals.KafkaFutureImpl.access$000(KafkaFutureImp
l.java:32)
        at org.apache.kafka.common.internals.KafkaFutureImpl$SingleWaiter.await(KafkaF
utureImpl.java:89)
        at org.apache.kafka.common.internals.KafkaFutureImpl.get(KafkaFutureImpl.java:
260)
        at kafka.admin.AclCommand$AdminClientService.getAcls(AclCommand.scala:171)
        at kafka.admin.AclCommand$AdminClientService.$anonfun$listAcls$1(AclCommand.sc
ala:134)
        at kafka.admin.AclCommand$AdminClientService.$anonfun$listAcls$1$adapted(AclCo
mmand.scala:131)
        at kafka.admin.AclCommand$AdminClientService.withAdminClient(AclCommand.scala:
92)
        at kafka.admin.AclCommand$AdminClientService.listAcls(AclCommand.scala:131)
        at kafka.admin.AclCommand$.main(AclCommand.scala:66)
        at kafka.admin.AclCommand.main(AclCommand.scala)
Caused by: org.apache.kafka.common.errors.SecurityDisabledException: No Authorizer is
configured on the broker
```

Adding ACLs

```
// authorize(request.session, Alter, Resource.ClusterResource)
// The principal type is always User for the simple authorizer
$ kafka-acls.sh \
    --authorizer-properties zookeeper.connect=zookeeper:2181 \
    --add \
    --cluster \
    --operation Alter \
    --deny-principal User:ANONYMOUS
Adding ACLs for resource `Cluster:LITERAL:kafka-cluster`:
        User:ANONYMOUS has Deny permission for operations: Alter from hosts: *

Current ACLs for resource `Cluster:LITERAL:kafka-cluster`:
        User:ANONYMOUS has Deny permission for operations: Alter from hosts: *

$ kafka-acls.sh \
    --authorizer-properties zookeeper.connect=zookeeper:2181 \
    --list
Current ACLs for resource `Cluster:LITERAL:kafka-cluster`:
        User:ANONYMOUS has Deny permission for operations: Alter from hosts: *
```

```
// With authorizer.class.name=kafka.security.auth.SimpleAclAuthorizer
$ kafka-acls.sh \
    --bootstrap-server kafka:9092 \
    --list
```

AclCommand Administration Utility

`AclCommand` is an administration utility for ACL management using AdminClient API.

`AclCommand` is executed using kafka-acls.sh shell script.

Table 1. Options

Option	Description
`add`	Adding ACLs
`allow-host`	Host from which principals listed in --allow-principal will have access `*` indicates access allowed from all hosts
`deny-host`	Host from which principals listed in --deny-principal will be denied access `*` indicates access denied from all hosts
`allow-principal`	Principal is in `principalType:name` format
`authorizer`	Fully-qualified class name of the Authorizer Default: kafka.security.authorizer.AclAuthorizer
`authorizer-properties`	**(required)** `key=val` pairs to configure the authorizer
`bootstrap-server`	**(required)** Brokers to connect to as a comma-separated list of `host:port` URLs
`cluster`	Add/Remove cluster ACLs
`command-config`	Property file with configs to be passed to Admin Client
`consumer`	Convenience option to add/remove ACLs for consumer role that generates ACLs that allow READ,DESCRIBE on topic and READ on group
`delegation-token`	Delegation token to which ACLs should be added or removed `*` indicates all tokens
`deny-principal`	Principal is in `principalType:name` format

`force`	Assume Yes to all queries and do not prompt
`group`	Consumer Group to which the ACLs should be added or removed `*` indicates all groups
`help`	Prints usage information
`idempotent`	Enables idempotence for the producer Used in combination with the --producer option Idempotence is enabled automatically if the producer is authorized to a particular transactional-id
`list`	List ACLs for the specified resource Use --topic, --group or --cluster to specify a resource
`operation`	Operation that is being allowed or denied Default: All Supported operations: • `All` • `Alter` • `AlterConfigs` • `ClusterAction` • `Create` • `Delete` • `Describe` • `DescribeConfigs` • `IdempotentWrite` • `Read` • `Write`
`principal`	List ACLs for the specified principal (in `principalType:name` format)
`producer`	Convenience option to add/remove ACLs for producer role that generates ACLs that allow WRITE,DESCRIBE and CREATE on topic
`remove`	Removing ACLs

`resource-pattern-type`	Type of the resource pattern or pattern filter Default: `literal` • When adding acls, this should be a specific pattern type, e.g. 'literal' or 'prefixed'. • When listing or removing acls, a specific pattern type can be used to list or remove acls from specific resource patterns, or use the filter values of 'any' or 'match', where 'any' will match any pattern type, but will match the resource name exactly, where as 'match' will perform pattern matching to list or remove all acls that affect the supplied resource(s). Warning: Use with care with 'match' and --remove option.
`topic`	Topic to which ACLs should be added or removed `*` indicates all topics
`transactional-id`	The transactionalId to which ACLs should be added or removed `*` indicates all transactionalIds
`version`	Displays Kafka version

`AclCommand` uses AdminClientService (when executed with --bootstrap-server option) or `AuthorizerService`.

`kafka-acls.sh` requires Authorizer to be configured on a broker (when executed with --bootstrap-server option) or throws a `SecurityDisabledException`:

```
// kafka-acls.sh --list --bootstrap-server :9092
Error while executing ACL command: org.apache.kafka.common.errors.SecurityDisabledExce
ption: No Authorizer is configured on the broker
```

Executing Standalone Application — `main` Object Method

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

`main` is the entry point of the `AclCommand` when launched on command line (e.g. using kafka-acls.sh shell script).

Internally, `main` reads the options from the command line.

`main` prints out the following for no command-line options, --help or --version:

```
This tool helps to manage acls on kafka
```

`main` selects the command service between `AdminClientService` when --bootstrap-server is used and `AuthorizerService` otherwise.

In the end, `main` requests the command service to add, remove or list ACLs based on --add, --remove or --list command-line options, respectively.

`getFilteredResourceToAcls` Internal Method

```
getFilteredResourceToAcls(
  authorizer: Authorizer,
  filters: Set[ResourcePatternFilter],
  listPrincipal: Option[KafkaPrincipal] = None
): Iterable[(Resource, Set[Acl])]
```

`getFilteredResourceToAcls` ...FIXME

Note	`getFilteredResourceToAcls` is used when...FIXME

`removeAcls` Internal Method

```
removeAcls(
  authorizer: Authorizer,
  acls: Set[Acl],
  filter: ResourcePatternFilter): Unit
```

`removeAcls` ...FIXME

Note	`removeAcls` is used when...FIXME

JAuthorizerService

`JAuthorizerService` is…FIXME

Adding ACLs for Resource — `addAcls` Method

```
addAcls(): Unit
```

Note	`addAcls` is part of the AclCommandService contract to add ACLs for a resource.

`addAcls` …FIXME

Removing ACLs for Given Filters — `removeAcls` Method

```
removeAcls(): Unit
```

Note	`removeAcls` is part of the AclCommandService contract to remove ACLs for the provided filters.

`removeAcls` …FIXME

`removeAcls` Internal Method

```
removeAcls(
  authorizer: JAuthorizer,
  acls: Set[AccessControlEntry],
  filter: ResourcePatternFilter): Unit
```

`removeAcls` …FIXME

Note	`removeAcls` is used when…FIXME

Listing ACLs for Given Filter — `getAcls` Method

```
getAcls(): Unit
```

Note	`getAcls` is part of the AclCommandService contract to list ACLs for the provided filter.

`getAcls` ...FIXME

AclCommandService

`AclCommandService` is...FIXME

AdminClientService — AclCommandService of AclCommand with bootstrap-server Option

`AdminClientService` is the AclCommandService for AclCommand when executed with --bootstrap-server option.

Creating AdminClientService Instance

`AdminClientService` takes the following to be created:

- `AclCommandOptions`

`AdminClientService` initializes the internal properties.

`addAcls` Method

```
addAcls(): Unit
```

Note	`addAcls` is part of the AclCommandService Contract to…FIXME.

`addAcls` …FIXME

`listAcls` Method

```
listAcls(): Unit
```

Note	`listAcls` is part of the AclCommandService Contract to…FIXME.

`listAcls` …FIXME

`removeAcls` Method

```
removeAcls(): Unit
```

Note	`removeAcls` is part of the AclCommandService Contract to…FIXME.

`removeAcls` …FIXME

`withAdminClient` Internal Method

```
withAdminClient(
  opts: AclCommandOptions)(
  f: Admin => Unit): Unit
```

`withAdminClient` loads the property file when specified using --command-config option.

`withAdminClient` creates a new AdminClient that is used as the input argument to execute the given `f` function.

In the end, `withAdminClient` requests the `AdminClient` to close.

Note	`withAdminClient` is used when `AdminClientService` is requested to add, list, and remove ACLs.

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` ...FIXME

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…FIXME

ConsumerGroupService

`ConsumerGroupService` is...FIXME

BrokerApiVersionsCommand

`BrokerApiVersionsCommand` is...FIXME

JmxTool Standalone Application

`kafka.tools.JmxTool` is a standalone application to print JMX metrics of a given endpoint (e.g. Kafka broker) to standard output every reporting-interval (unless one-time option is specified).

`JmxTool` can be executed using `kafka-run-class` shell script.

```
./bin/kafka-run-class.sh kafka.tools.JmxTool \
  --object-name kafka.server:type=BrokerTopicMetrics,name=MessagesInPerSec
Trying to connect to JMX url: service:jmx:rmi:///jndi/rmi://:9999/jmxrmi.
"time","kafka.server:type=BrokerTopicMetrics,name=MessagesInPerSec:Count","kafka.serve
r:type=BrokerTopicMetrics,name=MessagesInPerSec:EventType","kafka.server:type=BrokerTo
picMetrics,name=MessagesInPerSec:FifteenMinuteRate","kafka.server:type=BrokerTopicMetr
ics,name=MessagesInPerSec:FiveMinuteRate","kafka.server:type=BrokerTopicMetrics,name=M
essagesInPerSec:MeanRate","kafka.server:type=BrokerTopicMetrics,name=MessagesInPerSec:
OneMinuteRate","kafka.server:type=BrokerTopicMetrics,name=MessagesInPerSec:RateUnit"
1567588157489,0,messages,0.0,0.0,0.0,0.0,SECONDS
1567588159492,0,messages,0.0,0.0,0.0,0.0,SECONDS
1567588161494,0,messages,0.0,0.0,0.0,0.0,SECONDS
^C
```

Tip	Enable remote JMX of a Kafka broker using `JMX_PORT` environment variable when starting it with `./bin/kafka-server-start.sh`, e.g. `JMX_PORT=9999 ./bin/kafka-server-start.sh config/server.properties`

When executed with no options, `JmxTool` assumes --help.

Table 1. Options

Option	Description
`attributes`	The comma-separated whitelist of attributes to query. If no attributes are specified all objects will be queried.
`date-format`	Date format to use for formatting the time field. See java.text.SimpleDateFormat for options.
`help`	Prints usage information
`jmx-auth-prop`	Mechanism to pass property in the form 'username=password' when enabling remote JMX with password authentication

`jmx-ssl-enable`	Flag to enable remote JMX with SSL Default: `false`
`jmx-url`	URL to connect to poll JMX data Default: `service:jmx:rmi:///jndi/rmi://:9999/jmxrmi` See javax.management.remote.JMXServiceURL for details.
`object-name`	A JMX object name to use as a query. This can contain wildcards, and can be given multiple times to specify more than one query. If no objects are specified all objects will be queried.
`one-time`	Flag to indicate run once only Default: `false`
`report-format`	Output format name. Default: `original` One of the following: • original • properties • csv • tsv
`reporting-interval`	Interval (in millis) to poll jmx stats with Default: `2000` (2 seconds) `-1` is equivalent to one-time
`wait`	Wait for requested JMX objects to become available before starting output. Only supported when the list of objects is non-empty and contains no object name patterns.

Executing Standalone Application — `main` Method

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

`main` is the entry point of the `JmxTool` standalone application when launched on command line (e.g. from `bin/kafka-configs.sh`).

Internally, `main` reads the options from the command line.

`main` prints out the following to standard error:

```
Trying to connect to JMX url: [url].
```

`main` ...FIXME

GetOffsetShell Utility

`kafka.tools.GetOffsetShell` is a standalone application that is used to get offsets of the partitions of a topic.

`GetOffsetShell` can be executed using `kafka-run-class` shell script.

```
./bin/kafka-run-class.sh kafka.tools.GetOffsetShell \
  --broker-list :9092 \
  --topic t1
// topicName:partitionID:offset
t1:0:0
```

When executed with no options, it is equivalent to --help.

Table 1. Options

Option	Description
`broker-list`	**(required)** The comma-separated list of brokers to connect to (e.g. hostname:port,…,hostname:port)
`partitions`	The comma-separated list of partition IDs If not specified (empty), it will find offsets for all partitions Default: `(empty)`
`time`	Timestamp of the offsets before that. No offset is returned, if the timestamp is greater than recently commited record timestamp Possible values: • a timestamp • `-1` (**LATEST_TIMESTAMP**) for the first offsets • `-2` (**EARLIEST_TIMESTAMP**) for the end offsets Default: `-1`
`topic`	**(required)** Topic to get offsets from

`GetOffsetShell` is a Kafka consumer with **GetOffsetShell** client ID.

Executing Standalone Application — `main` Method

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

`main` is the entry point of the GetOffsetShell standalone application when launched on command line (e.g. from `bin/kafka-configs.sh`).

Internally, `main` …FIXME

ReplicaVerificationTool

`ReplicaVerificationTool` is...FIXME

kafka-dump-log.sh Shell Script

`kafka-dump-log.sh` is a shell script to launch kafka.tools.DumpLogSegments standalone application for sanity check of log files and printing contents out to the console.

When executed with no options, `kafka-dump-log.sh` assumes --help.

```
$ ./bin/kafka-dump-log.sh
This tool helps to parse a log file and dump its contents to the console, useful for d
ebugging a seemingly corrupt log segment.
Option                                  Description
------                                  -----------
--deep-iteration                        if set, uses deep instead of shallow
                                          iteration. Automatically set if print-
                                          data-log is enabled.
--files <String: file1, file2, ...>     REQUIRED: The comma separated list of data
                                          and index log files to be dumped.
--help                                  Print usage information.
--index-sanity-check                    if set, just checks the index sanity
                                          without printing its content. This is
                                          the same check that is executed on
                                          broker startup to determine if an index
                                          needs rebuilding or not.
--key-decoder-class [String]            if set, used to deserialize the keys. This
                                          class should implement kafka.serializer.
                                          Decoder trait. Custom jar should be
                                          available in kafka/libs directory.
                                          (default: kafka.serializer.StringDecoder)
--max-message-size <Integer: size>      Size of largest message. (default: 5242880)
--offsets-decoder                       if set, log data will be parsed as offset
                                          data from the __consumer_offsets topic.
--print-data-log                        if set, printing the messages content when
                                          dumping data logs. Automatically set if
                                          any decoder option is specified.
--transaction-log-decoder               if set, log data will be parsed as
                                          transaction metadata from the
                                          __transaction_state topic.
--value-decoder-class [String]          if set, used to deserialize the messages.
                                          This class should implement kafka.
                                          serializer.Decoder trait. Custom jar
                                          should be available in kafka/libs
                                          directory. (default: kafka.serializer.
                                          StringDecoder)
--verify-index-only                     if set, just verify the index log without
                                          printing its content.
--version                               Display Kafka version.
```

DumpLogSegments

`DumpLogSegments` is a standalone application to print-data-log, verify-index-only and index-sanity-check of data and index log files.

`DumpLogSegments` can be executed on command line using kafka-dump-log.sh shell script (that simply uses `kafka-run-class` shell script to launch the command).

`DumpLogSegments` uses custom record parsers to dump records of log files of __consumer_offsets and __transaction_state internal topics. For other topics, `DumpLogSegments` uses key-decoder-class and value-decoder-class decoders (default: `kafka.serializer.StringDecoder`).

Table 1. Options

Option	Description
`deep-iteration`	Uses deep iteration (not shallow). Implied with print-data-log
`files`	**(required)** Comma-separated list of data and index log files to dump contents of
`help`	Prints usage information
`index-sanity-check`	Checks the index sanity without printing content. This is the same check that is executed on broker startup to determine if an index needs rebuilding or not.
`key-decoder-class`	Fully-qualified class name of the key deserializer of messages in the log files. Should implement `kafka.serializer.Decoder` trait, and the custom jar should be available in `kafka/libs` directory. Default: `kafka.serializer.StringDecoder`
`max-message-size`	Size of the largest message Default: `5 * 1024 * 1024`
`offsets-decoder`	Uses OffsetsMessageParser to parse log data as offset data from the `__consumer_offsets` topic
`print-data-log`	Prints message content when dumping data logs. Implied with any decoder option specified.
`transaction-log-decoder`	Uses TransactionLogMessageParser to parse log data as transaction metadata from the `__transaction_state` topic
`value-decoder-class`	Fully-qualified class name of the value deserializer of messages in the log files. Should implement `kafka.serializer.Decoder` trait, and the custom jar should be available in `kafka/libs` directory. Default: `kafka.serializer.StringDecoder`
`verify-index-only`	Verifies the index log without printing content
`version`	Displays Kafka version

Executing DumpLogSegments — `main` Object Method

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

`main` is the entry point of `DumpLogSegments` when launched on command line (e.g. using kafka-dump-log.sh shell script).

Internally, `main` parses the options. For every file (in the required --files option) `main` prints out the following message to the console:

```
Dumping [file]
```

`main` then branches off per file suffix:

- For .log files, dumpLog
- For .index files, dumpIndex
- For .timeindex files, dumpTimeIndex
- For .snapshot files, dumpProducerIdSnapshot
- For .txnindex files, dumpTxnIndex

`main` ...FIXME (`misMatchesForIndexFilesMap`)

`main` ...FIXME (`timeIndexDumpErrors`)

`main` ...FIXME (`nonConsecutivePairsForLogFilesMap`)

`main` simply ignores other file suffices and prints out the following message to standard error output:

```
Ignoring unknown file [file]
```

Dumping Content Of Log File (Of Log Segment) — `dumpLog` Internal Method

```
dumpLog(
  file: File,
  printContents: Boolean,
  nonConsecutivePairsForLogFilesMap: mutable.Map[String, List[(Long, Long)]],
  isDeepIteration: Boolean,
  maxMessageSize: Int,
  parser: MessageParser[_, _]): Unit
```

`dumpLog` reads the starting offset from the name of the given file and prints out the following message to the console:

```
Starting offset: [startOffset]
```

`dumpLog` opens the log file (and creates a `FileRecords`).

For every FileChannelRecordBatch `dumpLog` ...FIXME

Note	`dumpLog` is used when `DumpLogSegments` tool is executed with .log files.

Dumping Content Of Transaction Index (Of Log Segment) — `dumpTxnIndex` Internal Method

```
dumpTxnIndex(
  file: File): Unit
```

`dumpTxnIndex` ...FIXME

Note	`dumpTxnIndex` is used when `DumpLogSegments` tool is executed with .txnindex files.

Dumping Content Of Time Index (Of Log Segment) — `dumpTimeIndex` Internal Method

```
dumpTimeIndex(
  file: File,
  indexSanityOnly: Boolean,
  verifyOnly: Boolean,
  timeIndexDumpErrors: TimeIndexDumpErrors,
  maxMessageSize: Int): Unit
```

`dumpTimeIndex` ...FIXME

Note	`dumpTimeIndex` is used when `DumpLogSegments` tool is executed with .timeindex files.

Dumping Content Of Offset Index (Of Log Segment) — `dumpIndex` Internal Method

```
dumpIndex(
  file: File,
  indexSanityOnly: Boolean,
  verifyOnly: Boolean,
  misMatchesForIndexFilesMap: mutable.Map[String, List[(Long, Long)]],
  maxMessageSize: Int): Unit
```

`dumpIndex` ...FIXME

Note	`dumpIndex` is used when `DumpLogSegments` tool is executed with .index files.

Dumping Content Of Producer Snapshot (Of Log Segment) — `dumpProducerIdSnapshot` Internal Method

```
dumpProducerIdSnapshot(
  file: File): Unit
```

`dumpProducerIdSnapshot` ...FIXME

Note	`dumpProducerIdSnapshot` is used when `DumpLogSegments` tool is executed with .snapshot files.

OffsetsMessageParser

`OffsetsMessageParser` is...FIXME

TransactionLogMessageParser

`TransactionLogMessageParser` is...FIXME

ConfigCommand Command-Line Tool

`kafka.admin.ConfigCommand` is a command-line tool that…FIXME

`ConfigCommand` can be executed using `kafka-configs` shell script (i.e. `bin/kafka-configs.sh` or `bin\windows\kafka-configs.bat`).

```
// Triggers alterBrokerConfig
./bin/kafka-configs.sh \
  --bootstrap-server :9092 \
  --alter \
  --entity-type brokers \
  --entity-name 0 \
  --add-config advertised.listeners=plaintext://:9092
```

```
// Triggers processBrokerConfig
./bin/kafka-configs.sh \
  --bootstrap-server :9092 \
  --describe \
  --entity-type brokers \
  --entity-name 0
```

`alterBrokerConfig` Method

```
alterBrokerConfig(
  adminClient: JAdminClient,
  opts: ConfigCommandOptions,
  entityName: String)
```

`alterBrokerConfig`…FIXME

Note	`alterBrokerConfig` is used exclusively when `ConfigCommand` is requested to processBrokerConfig (with `alter` action).

`processBrokerConfig` Internal Method

```
processBrokerConfig(opts: ConfigCommandOptions): Unit
```

`processBrokerConfig`…FIXME

Note	`processBrokerConfig` is used exclusively when `ConfigCommand` standalone application is executed (without `zookeeper` command-line option).

Executing Standalone Application — `main` Method

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

`main` is the entry point of the ConfigCommand standalone application when launched on command line (e.g. from `bin/kafka-configs.sh`).

`main` ...FIXME

`processCommandWithZk` Internal Method

```
processCommandWithZk(zkConnectString: String, opts: ConfigCommandOptions): Unit
```

`processCommandWithZk` ...FIXME

Note	`processCommandWithZk` is used when...FIXME

`alterConfig` Internal Method

```
alterConfig(
  zkClient: KafkaZkClient,
  opts: ConfigCommandOptions,
  adminZkClient: AdminZkClient): Unit
```

`alterConfig` ...FIXME

Note	`alterConfig` is used when...FIXME

`describeConfig` Internal Method

```
describeConfig(
  zkClient: KafkaZkClient,
  opts: ConfigCommandOptions,
  adminZkClient: AdminZkClient): Unit
```

`describeConfig` ...FIXME

Note	`describeConfig` is used when…FIXME

`describeBrokerConfig` Internal Method

```
describeBrokerConfig(
  adminClient: JAdminClient,
  opts: ConfigCommandOptions,
  entityName: String): Unit
```

`describeBrokerConfig` …FIXME

Note	`describeBrokerConfig` is used exclusively when `ConfigCommand` is requested to processBrokerConfig (with `describe` action).

`brokerConfig` Internal Method

```
brokerConfig(
  adminClient: JAdminClient,
  entityName: String,
  includeSynonyms: Boolean): Seq[ConfigEntry]
```

`brokerConfig` …FIXME

Note	`brokerConfig` is used when `ConfigCommand` is requested to alterBrokerConfig and describeBrokerConfig.

ReassignPartitionsCommand Command-Line Tool — Partition Reassignment on Command Line

`kafka.admin.ReassignPartitionsCommand` is a command-line tool that allows for generating, executing and verifying a custom partition (re)assignment configuration (as specified using a reassignment JSON file).

Table 1. ReassignPartitionsCommand's Actions

Action	Description
--execute	Executes the reassignment as specified by the reassignment-json-file option
--generate	Generates a candidate partition reassignment configuration Note: This only generates a candidate assignment, it does not execute it.
--verify	Verifies if the reassignment completed as specified by the reassignment-json-file option. If there is a throttle engaged for the replicas specified, and the rebalance has completed, the throttle will be removed

`ReassignPartitionsCommand` can be executed using `kafka-reassign-partitions` shell script (i.e. `bin/kafka-reassign-partitions.sh` or `bin\windows\kafka-reassign-partitions.bat`).

```
$ ./bin/kafka-reassign-partitions.sh
This command moves topic partitions between replicas.
```

Table 2. ReassignPartitionsCommand's Options

Option	Description
	A JSON file with a custom partition (re)assignment configuration The format to use is as follows:

`reassignment-json-file`	<pre>{
	  "partitions": [
	  {
	  "topic": "foo",
	  "partition": 1,
	  "replicas": [
	  1,
	  2,
	  3
	  ],
	  "log_dirs": [
	  "dir1",
	  "dir2",
	  "dir3"
	  ]
	  }
	  ],
	  "version": 1
	}</pre>
	Note that "log_dirs" is optional. When specified, its length must equal the length of the replicas list. The value in this list can be either `"any"` or the absolute path of the log directory on the broker.
	
	If absolute log directory path is specified, it is currently required that the replica has not already been created on that broker. The replica will then be created in the specified log directory on the broker later.

`replica-alter-log-dirs-throttle`	The movement of replicas between log directories on the same broker will be throttled to this value (bytes/sec). Default: `-1` Rerunning with this option, whilst a rebalance is in progress, will alter the throttle value. The throttle rate should be at least 1 KB/s.
`throttle`	The movement of partitions between brokers will be throttled to this value (bytes/sec). Default: `-1` Rerunning with this option, whilst a rebalance is in progress, will alter the throttle value. The throttle rate should be at least 1 KB/s.
`timeout`	The maximum time (in ms) allowed to wait for partition reassignment execution to be successfully initiated Default: `10000`

```
$ ./bin/kafka-topics.sh --list --zookeeper :2181
my-topic

$ cat reassign-partitions.json
{
  "partitions": [
    {
      "topic": "my-topic",
      "partition": 1,
      "replicas": [
        1
      ]
    }
  ],
  "version": 1
}

$ ./bin/kafka-reassign-partitions.sh \
  --generate \
  --zookeeper :2181 \
  --topics-to-move-json-file reassign-partitions.json \
  --broker-list 0

$ ./bin/kafka-reassign-partitions.sh \
  --verify \
  --zookeeper :2181 \
  --reassignment-json-file reassign-partitions.json
```

`ReassignPartitionsCommand` is created exclusively when `ReassignPartitionsCommand` is requested to executeAssignment.

Executing Standalone Application — `main` Method

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

`main` ...FIXME

`executeAssignment` Method

```
executeAssignment(
  zkClient: KafkaZkClient,
  adminClientOpt: Option[JAdminClient],
  opts: ReassignPartitionsCommandOptions): Unit (1)
executeAssignment(
  zkClient: KafkaZkClient,
  adminClientOpt: Option[JAdminClient],
  reassignmentJsonString: String,
  throttle: Throttle,
  timeoutMs: Long = 10000L): Unit
```

1. Uses options for `reassignmentJsonString`, `throttle` and `timeoutMs` inputs

`executeAssignment` ...FIXME

Note	`executeAssignment` is used exclusively when `ReassignPartitionsCommand` is executed (with execute action).

`reassignPartitions` Method

```
reassignPartitions(throttle: Throttle = NoThrottle, timeoutMs: Long = 10000L): Boolean
```

`reassignPartitions` ...FIXME

Note	`reassignPartitions` is used exclusively when `ReassignPartitionsCommand` executeAssignment.

`alterReplicaLogDirsIgnoreReplicaNotAvailable` Internal Method

```
alterReplicaLogDirsIgnoreReplicaNotAvailable(
  replicaAssignment: Map[TopicPartitionReplica, String],
  adminClient: JAdminClient,
  timeoutMs: Long): Set[TopicPartitionReplica]
```

`alterReplicaLogDirsIgnoreReplicaNotAvailable` ...FIXME

Note	`alterReplicaLogDirsIgnoreReplicaNotAvailable` is used exclusively when `ReassignPartitionsCommand` reassignPartitions

`generateAssignment` Method

```
generateAssignment(
  zkClient: KafkaZkClient,
  brokerListToReassign: Seq[Int],
  topicsToMoveJsonString: String,
  disableRackAware: Boolean)
: (Map[TopicPartition, Seq[Int]], Map[TopicPartition, Seq[Int]])
```

`generateAssignment` ...FIXME

Note	`generateAssignment` is used when...FIXME

`verifyAssignment` Method

```
verifyAssignment(
  zkClient: KafkaZkClient,
  adminClientOpt: Option[JAdminClient],
  jsonString: String): Unit
```

`verifyAssignment` ...FIXME

Note	`verifyAssignment` is used when...FIXME

`parseAndValidate` Method

```
parseAndValidate(
  zkClient: KafkaZkClient,
  reassignmentJsonString: String)
: (Seq[(TopicPartition, Seq[Int])], Map[TopicPartitionReplica, String])
```

`parseAndValidate` ...FIXME

Note	`parseAndValidate` is used when...FIXME

`removeThrottle` Method

```
removeThrottle(
  zkClient: KafkaZkClient,
  reassignedPartitionsStatus: Map[TopicPartition, ReassignmentStatus],
  replicasReassignmentStatus: Map[TopicPartitionReplica, ReassignmentStatus],
  adminZkClient: AdminZkClient): Unit
```

`removeThrottle` ...FIXME

Note	`removeThrottle` is used when…FIXME

`maybeLimit` Method

```
maybeLimit(throttle: Throttle): Unit
```

`maybeLimit` …FIXME

Note	`maybeLimit` is used when…FIXME

`assignThrottledReplicas` Method

```
assignThrottledReplicas(
  existingPartitionAssignment: Map[TopicPartition, Seq[Int]],
  proposedPartitionAssignment: Map[TopicPartition, Seq[Int]],
  adminZkClient: AdminZkClient): Unit
```

`assignThrottledReplicas` …FIXME

Note	`assignThrottledReplicas` is used when…FIXME

`existingAssignment` Method

```
existingAssignment(): Map[TopicPartition, Seq[Int]]
```

`existingAssignment` takes the topics (from the keys) from the proposedPartitionAssignment and requests the KafkaZkClient to getReplicaAssignmentForTopics.

Note	`existingAssignment` is used when…FIXME

Creating ReassignPartitionsCommand Instance

`ReassignPartitionsCommand` takes the following when created:

- KafkaZkClient
- Optional AdminClient
- Proposed partition assignment (`Map[TopicPartition, Seq[Int]]`)
- Proposed replica assignment (`Map[TopicPartitionReplica, String]`)

- AdminZkClient

TopicCommand Command-Line Tool — Topic Management on Command Line

`kafka.admin.TopicCommand` is a command-line tool that can alter, create, delete, describe and list topics in a Kafka cluster.

Table 1. TopicCommand's Actions

Action	Description
--alter	Alters the number of partitions, replica assignment, and/or configuration of a topic or topics
--create	Creates a new topic
--delete	Deletes a topic or topics
--describe	Describes a topic or topics
--list	Lists available topics

`TopicCommand` can be executed using `kafka-topics` shell script (i.e. `bin/kafka-topics.sh` or `bin\windows\kafka-topics.bat`).

```
$ ./bin/kafka-topics.sh
Create, delete, describe, or change a topic.
```

Table 2. TopicCommand's Options

Option	Description
`config`	A topic configuration override for the topic being worked on (as `name=value`)
`disable-rack-aware`	Disable rack aware replica assignment
`if-not-exists`	When creating a topic(s), the action will only execute if the topic does not already exist
`partitions`	The number of partitions of the topic being created or altered **Warning**: If partitions are increased for a topic that has a key, the partition logic or ordering of the messages will be affected.
`replica-assignment`	A list of manual partition-to-broker assignments for the topic being created or altered
`replication-factor`	The replication factor for each partition of the topic being created
`topic`	A topic or topics to be altered, created, deleted, or listed and described (through getTopics)

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

$ ./bin/kafka-topics.sh --list --zookeeper :2181
my-topic

$ ./bin/kafka-topics.sh \
  --zookeeper :2181 \
  --describe \
  --topic my-topic
Topic:my-topic  PartitionCount:1  ReplicationFactor:1  Configs:
  Topic: my-topic  Partition: 0  Leader: 0  Replicas: 0  Isr: 0
```

`alterTopic` Method

```
alterTopic(zkClient: KafkaZkClient, opts: TopicCommandOptions): Unit
```

`alterTopic` ...FIXME

Note	`alterTopic` is used when...FIXME

Creating Topic — `createTopic` Method

```
createTopic(zkClient: KafkaZkClient, opts: TopicCommandOptions): Unit
```

`createTopic` takes the values of topic, config and if-not-exists options.

`createTopic` creates a new AdminZkClient (with the input KafkaZkClient).

`createTopic` branches off per replica-assignment option's existence:

- With replica-assignment, `createTopic` requests the `AdminZkClient` to createOrUpdateTopicPartitionAssignmentPathInZK (with the `update` flag off)
- With no replica-assignment, `createTopic` checks the required arguments, i.e. partitions and replication-factor, and together with disable-rack-aware flag requests the `AdminZkClient` to create the topic

In the end, `createTopic` prints out the following:

```
Created topic "[topic]"
```

Note	`createTopic` is used exclusively when `TopicCommand` is executed with --create action.

`deleteTopic` Method

```
deleteTopic(zkClient: KafkaZkClient, opts: TopicCommandOptions): Unit
```

`deleteTopic` ...FIXME

Note	`deleteTopic` is used when...FIXME

`describeTopic` Method

```
describeTopic(
  zkClient: KafkaZkClient,
  opts: TopicCommandOptions): Unit
```

`describeTopic` ...FIXME

Note	`describeTopic` is used when...FIXME

`listTopics` Method

```
listTopics(zkClient: KafkaZkClient, opts: TopicCommandOptions): Unit
```

`listTopics` ...FIXME

Note	`listTopics` is used when...FIXME

Executing Standalone Application — `main` Method

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

`main` is the entry point of the TopicCommand standalone application when launched on command line (e.g. from `bin/kafka-topics.sh`).

`main` creates a TopicCommandOptions.

`main` prints out the following message and exits when no command-line arguments were given:

```
Create, delete, describe, or change a topic.
```

`main` prints out the following message and exits when no action was specified:

```
Command must include exactly one action: --list, --describe, --create, --alter or --de
lete
```

`main` makes sure the action and the options were all valid.

`main` creates a new KafkaZkClient (with the `zookeeper` option).

`main` branches off per the action.

In case of any exception, `main` prints out the following and exits with `1` exit code:

```
Error while executing topic command : [message]
```

`getTopics` Internal Method

```
getTopics(zkClient: KafkaZkClient, opts: TopicCommandOptions): Seq[String]
```

`getTopics` ...FIXME

Note	`getTopics` is used when `TopicCommand` is requested to alterTopic, listTopics, deleteTopic, describeTopic.

KafkaAdminClient

`KafkaAdminClient` is the default and only known AdminClient that is used in Kafka administration utilities.

Tip	Enable `ALL` logging level for `org.apache.kafka.clients.admin.KafkaAdminClient` logger to see what happens inside. Add the following line to `config/log4j.properties`: `log4j.logger.org.apache.kafka.clients.admin.KafkaAdminClient=ALL` Refer to Logging.

Creating KafkaAdminClient Instance

`KafkaAdminClient` takes the following to be created:

- `AdminClientConfig`
- Client ID
- `Time`
- AdminMetadataManager
- Metrics
- KafkaClient
- `TimeoutProcessorFactory`
- `LogContext`

`KafkaAdminClient` initializes the internal properties.

Creating KafkaAdminClient Instance — `createInternal` Factory Method

```
KafkaAdminClient createInternal(
  AdminClientConfig config,
  TimeoutProcessorFactory timeoutProcessorFactory)
```

`createInternal` ...FIXME

Note	`createInternal` is used exclusively when `AdminClient` object is requested to create an AdminClient (for Kafka administration utilities).

Triggering Preferred Replica Leader Election — `electPreferredLeaders` Method

```
ElectPreferredLeadersResult electPreferredLeaders(
  Collection<TopicPartition> partitions,
  ElectPreferredLeadersOptions options)
```

Note	`electPreferredLeaders` is part of the AdminClient Contract to trigger preferred replica leader election.

`electPreferredLeaders` creates a **electPreferredLeaders** call that simply uses ElectPreferredLeadersRequest.Builder to serialize partitions into a ElectPreferredLeadersRequest and, when a response comes in, requests the `ElectPreferredLeadersRequest` to deserialize it.

In the end, `electPreferredLeaders` requests the AdminClientRunnable to enqueue the `electPreferredLeaders` call.

`describeTopics` Method

```
DescribeTopicsResult describeTopics(
  Collection<String> topicNames,
  DescribeTopicsOptions options)
```

`describeTopics` ...FIXME

Note	`describeTopics` is used when...FIXME

`alterReplicaLogDirs` Method

```
AlterReplicaLogDirsResult alterReplicaLogDirs(
  Map<TopicPartitionReplica, String> replicaAssignment,
  AlterReplicaLogDirsOptions options)
```

`alterReplicaLogDirs` ...FIXME

Note	`alterReplicaLogDirs` is used when...FIXME

`alterConfigs` Method

```
AlterConfigsResult alterConfigs(
  Map<ConfigResource, Config> configs,
  AlterConfigsOptions options)
// Private API
Map<ConfigResource, KafkaFutureImpl<Void>> alterConfigs(
  Map<ConfigResource, Config> configs,
  AlterConfigsOptions options,
  Collection<ConfigResource> resources,
  NodeProvider nodeProvider)
```

Note	`alterConfigs` is part of the AdminClient Contract to…FIXME.

`alterConfigs` …FIXME

`describeCluster` Method

```
DescribeClusterResult describeCluster(
  DescribeClusterOptions options)
```

Note	`describeCluster` is part of the AdminClient Contract to…FIXME.

`describeCluster` …FIXME

`deleteRecords` Method

```
DeleteRecordsResult deleteRecords(
  Map<TopicPartition, RecordsToDelete> recordsToDelete,
  DeleteRecordsOptions options)
```

Note	`deleteRecords` is part of the AdminClient Contract to…FIXME.

`deleteRecords` …FIXME

`listConsumerGroups` Method

```
ListConsumerGroupsResult listConsumerGroups(
  ListConsumerGroupsOptions options)
```

Note	`listConsumerGroups` is part of the AdminClient Contract to…FIXME.

`listConsumerGroups` ...FIXME

`listTopics` Method

```
ListTopicsResult listTopics(
  ListTopicsOptions options)
```

Note	`listTopics` is part of the AdminClient Contract to...FIXME.

`listTopics` ...FIXME

`describeConsumerGroups` Method

```
DescribeConsumerGroupsResult describeConsumerGroups(
  Collection<String> groupIds,
  DescribeConsumerGroupsOptions options)
```

Note	`describeConsumerGroups` is part of the AdminClient Contract to...FIXME.

`describeConsumerGroups` ...FIXME

`listConsumerGroupOffsets` Method

```
ListConsumerGroupOffsetsResult listConsumerGroupOffsets(
  String groupId,
  ListConsumerGroupOffsetsOptions options)
```

Note	`listConsumerGroupOffsets` is part of the AdminClient Contract to...FIXME.

`listConsumerGroupOffsets` ...FIXME

`deleteConsumerGroups` Method

```
DeleteConsumerGroupsResult deleteConsumerGroups(
  Collection<String> groupIds,
  DeleteConsumerGroupsOptions options)
```

Note	`deleteConsumerGroups` is part of the AdminClient Contract to...FIXME.

`deleteConsumerGroups` ...FIXME

`createTopics` Method

```
CreateTopicsResult createTopics(
  Collection<NewTopic> newTopics,
  CreateTopicsOptions options)
```

Note	`createTopics` is part of the AdminClient Contract to…FIXME.

`createTopics` …FIXME

`deleteTopics` Method

```
DeleteTopicsResult deleteTopics(
  Collection<String> topicNames,
  DeleteTopicsOptions options)
```

Note	`deleteTopics` is part of the AdminClient Contract to…FIXME.

`deleteTopics` …FIXME

`createPartitions` Method

```
CreatePartitionsResult createPartitions(
  Map<String, NewPartitions> newPartitions,
  CreatePartitionsOptions options)
```

Note	`createPartitions` is part of the AdminClient Contract to…FIXME.

`createPartitions` …FIXME

Describing Configuration Of Specified Resources (Topics and Brokers) — `describeConfigs` Method

```
DescribeConfigsResult describeConfigs(
  Collection<ConfigResource> configResources,
  DescribeConfigsOptions options)
```

Note	`describeConfigs` is part of the AdminClient Contract to describe the configuration of the specified resources.

`describeConfigs` …FIXME

Internal Properties

Name	Description
`runnable`	AdminClientRunnable Used to create the daemon KafkaThread
`thread`	Java's Thread Used when…FIXME

AdminClient

`AdminClient` is an extension of the Admin contract to create AdminClients.

Note	KafkaAdminClient is the default and only known implementation of the AdminClient Contract in Apache Kafka.

Creating AdminClient Instance — `create` Object Method

```
AdminClient create(
  Properties props)
AdminClient create(
  Map<String, Object> conf)
```

`create` simply requests the `Admin` to create an AdminClient.

Note	`create` is used in Kafka administration utilities.

AdminMetadataManager

`AdminMetadataManager` is…FIXME

`clearController` Method

```
void clearController()
```

`clearController` …FIXME

Note	`clearController` is used when `KafkaAdminClient` is requested to createTopics, deleteTopics, and createPartitions (and `Errors.NOT_CONTROLLER` error is received).

AdminClientRunnable

`AdminClientRunnable` is a java.lang.Runnable that is used exclusively by KafkaAdminClient for…FIXME

Note	`AdminClientRunnable` is an internal part of and with full access to the internals of KafkaAdminClient (`private final class`).

Tip	`AdminClientRunnable` uses the org.apache.kafka.clients.admin.KafkaAdminClient logger so you should configure it properly to see what happens inside.

`sendEligibleCalls` Internal Method

```
long sendEligibleCalls(long now)
```

`sendEligibleCalls` …FIXME

Note	`sendEligibleCalls` is used exclusively when `AdminClientRunnable` is requested to run.

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

```
void run()
```

Note	`run` is a part of java.lang.Runnable that is executed when the thread is started.

`run` …FIXME

`timeoutCallsInFlight` Internal Method

```
void timeoutCallsInFlight(TimeoutProcessor processor)
```

`timeoutCallsInFlight` …FIXME

Note	`timeoutCallsInFlight` is used when…FIXME

`handleResponses` Internal Method

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

`handleResponses` ...FIXME

Note	`handleResponses` is used when...FIXME

`call` Method

```
void call(
  Call call,
  long now)
```

`call` ...FIXME

Note	`call` is used when...FIXME

Enqueueing Call — `enqueue` Method

```
void enqueue(
  Call call,
  long now)
```

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

```
Queueing [call] with a timeout [timeout] ms from now.
```

`enqueue` ...FIXME

Note	`enqueue` is used when...FIXME

Admin Contract — Administrative Clients

`Admin` is the abstraction of administrative clients for managing and inspecting resources of a Kafka cluster (e.g. topics, brokers, configurations, ACLs).

Note	AdminClient is the only known direct extension of the Admin Contract in Apache Kafka.

Table 1. Admin Contract

Method	Description

`alterPartitionReassignments`	`AlterPartitionReassignmentsResult alterPartitionRea`
	`  Map<TopicPartition, Optional<NewPartitionReassign`
	`AlterPartitionReassignmentsResult alterPartitionRea`
	`  Map<TopicPartition, Optional<NewPartitionReassign`
	`  AlterPartitionReassignmentsOptions options)`
	
	1. Uses a new `AlterPartitionReassignmentsOptions`

`alterReplicaLogDirs`	`AlterReplicaLogDirsResult alterReplicaLogDirs(`
	`  Map<TopicPartitionReplica, String> replicaAssignm`
	`AlterReplicaLogDirsResult alterReplicaLogDirs(`
	`  Map<TopicPartitionReplica, String> replicaAssignm`
	`  AlterReplicaLogDirsOptions options)`
	
	1. Uses a new `AlterReplicaLogDirsOptions` with the

`close`	`void close()  (1)`
	`void close(`
	`  Duration timeout)`
	
	1. Uses an infinite timeout

`create`	`Admin create(`
	`  Map<String, Object> conf)`
	`Admin create(`
	`  Properties props)`

`createAcls`	`CreateAclsResult createAcls(`
	`  Collection<AclBinding> acls) (1)`
	`CreateAclsResult createAcls(`
	`  Collection<AclBinding> acls,`
	`  CreateAclsOptions options)`
	
	1. Uses a new `CreateAclsOptions` with the defaults

`createDelegationToken`	`CreateDelegationTokenResult createDelegationToken()`
	`CreateDelegationTokenResult createDelegationToken(`
	`  CreateDelegationTokenOptions options)`
	
	1. Uses a new `CreateDelegationTokenOptions` with th

`createPartitions`	`CreatePartitionsResult createPartitions(`
	`  Map<String, NewPartitions> newPartitions) (1)`
	`CreatePartitionsResult createPartitions(`
	`  Map<String, NewPartitions> newPartitions,`
	`  CreatePartitionsOptions options)`
	
	1. Uses a new `CreatePartitionsOptions` with the def

`createTopics`	`CreateTopicsResult createTopics(`
	`  Collection<NewTopic> newTopics) (1)`
	`CreateTopicsResult createTopics(`
	`  Collection<NewTopic> newTopics,`
	`  CreateTopicsOptions options)`
	
	1. Uses a `CreateTopicsOptions` with the defaults

`deleteAcls`	`DeleteAclsResult deleteAcls(`
	`  Collection<AclBindingFilter> filters) (1)`
	`DeleteAclsResult deleteAcls(`
	`  Collection<AclBindingFilter> filters,`
	`  DeleteAclsOptions options)`
	
	1. Uses a `DeleteAclsOptions` with the defaults

`deleteConsumerGroupOffsets`	`DeleteConsumerGroupOffsetsResult deleteConsumerGrou`
	`  String groupId,`
	`  Set<TopicPartition> partitions) (1)`
	`DeleteConsumerGroupOffsetsResult deleteConsumerGrou`
	`  String groupId,`
	`  Set<TopicPartition> partitions,`
	`  DeleteConsumerGroupOffsetsOptions options)`
	
	1. Uses a `DeleteConsumerGroupOffsetsOptions` with th

`deleteConsumerGroups`	`DeleteConsumerGroupsResult deleteConsumerGroups(`
	`  Collection<String> groupIds) (1)`
	`DeleteConsumerGroupsResult deleteConsumerGroups(`
	`  Collection<String> groupIds,`
	`  DeleteConsumerGroupsOptions options)`
	
	1. Uses a `DeleteConsumerGroupsOptions` with the defa

`deleteRecords`	`DeleteRecordsResult deleteRecords(`
	`  Map<TopicPartition, RecordsToDelete> recordsToDel`
	`DeleteRecordsResult deleteRecords(`
	`  Map<TopicPartition, RecordsToDelete> recordsToDel`
	`  DeleteRecordsOptions options)`
	
	1. Uses a `DeleteRecordsOptions` with the defaults

`deleteTopics`	`DeleteTopicsResult deleteTopics(`
	`  Collection<String> topics) (1)`
	`DeleteTopicsResult deleteTopics(`
	`  Collection<String> topics,`
	`  DeleteTopicsOptions options)`
	
	1. Uses a `DeleteTopicsOptions` with the defaults

`describeAcls`	`DescribeAclsResult describeAcls(`
	`  AclBindingFilter filter) (1)`
	`DescribeAclsResult describeAcls(`
	`  AclBindingFilter filter,`
	`  DescribeAclsOptions options)`
	
	1. Uses a `DescribeAclsOptions` with the defaults

`describeCluster`	`DescribeClusterResult describeCluster() (1)`
	`DescribeClusterResult describeCluster(`
	`  DescribeClusterOptions options)`
	
	1. Uses a `DescribeClusterOptions` with the defaults

`describeConfigs`	`DescribeConfigsResult describeConfigs(`
	`  Collection<ConfigResource> resources) (1)`
	`DescribeConfigsResult describeConfigs(`
	`  Collection<ConfigResource> resources,`
	`  DescribeConfigsOptions options)`
	
	1. Uses a `DescribeConfigsOptions` with the defaults

`describeConsumerGroups`	`DescribeConsumerGroupsResult describeConsumerGroups`
	`  Collection<String> groupIds) (1)`
	`DescribeConsumerGroupsResult describeConsumerGroups`
	`  Collection<String> groupIds,`
	`  DescribeConsumerGroupsOptions options)`
	
	1. Uses a `DescribeConsumerGroupsOptions` with the de

`describeDelegationToken`	`DescribeDelegationTokenResult describeDelegationTok`
	`DescribeDelegationTokenResult describeDelegationTok`
	`  DescribeDelegationTokenOptions options)`

	1. Uses a `DescribeDelegationTokenOptions` with the c

`describeLogDirs`	`DescribeLogDirsResult describeLogDirs(`
	`  Collection<Integer> brokers) (1)`
	`DescribeLogDirsResult describeLogDirs(`
	`  Collection<Integer> brokers,`
	`  DescribeLogDirsOptions options)`
	
	1. Uses a `DescribeLogDirsOptions` with the defaults

`describeReplicaLogDirs`	`DescribeReplicaLogDirsResult describeReplicaLogDirs`
	`  Collection<TopicPartitionReplica> replicas) (1)`
	`DescribeReplicaLogDirsResult describeReplicaLogDirs`
	`  Collection<TopicPartitionReplica> replicas,`
	`  DescribeReplicaLogDirsOptions options)`
	
	1. Uses a `DescribeReplicaLogDirsOptions` with the de

`describeTopics`	`DescribeTopicsResult describeTopics(`
	`  Collection<String> topicNames) (1)`
	`DescribeTopicsResult describeTopics(`
	`  Collection<String> topicNames,`
	`  DescribeTopicsOptions options)`
	
	1. Uses a `DescribeTopicsOptions` with the defaults

`electLeaders`	`ElectLeadersResult electLeaders(`
	`  ElectionType electionType,`
	`  Set<TopicPartition> partitions) (1)`
	`ElectLeadersResult electLeaders(`
	`  ElectionType electionType,`
	`  Set<TopicPartition> partitions,`
	`  ElectLeadersOptions options)`
	
	1. Uses an `ElectLeadersOptions` with the defaults
	
	Elects a replica as the leader for the given topic partitic
	
	Used for kafka-leader-election.sh utility (via LeaderEle

`expireDelegationToken`	`ExpireDelegationTokenResult expireDelegationToken(`
	`  byte[] hmac) (1)`
	`ExpireDelegationTokenResult expireDelegationToken(`
	`  byte[] hmac,`
	`  ExpireDelegationTokenOptions options)`
	
	1. Uses a new `ExpireDelegationTokenOptions` with th

`incrementalAlterConfigs`	`AlterConfigsResult incrementalAlterConfigs(`
	`  Map<ConfigResource, Collection<AlterConfigOp>> co`
	`AlterConfigsResult incrementalAlterConfigs(`
	`  Map<ConfigResource, Collection<AlterConfigOp>> co`
	`  AlterConfigsOptions options)`
	
	1. Uses a new `AlterConfigsOptions` with the defaults

`listConsumerGroupOffsets`	`ListConsumerGroupOffsetsResult listConsumerGroupOff`
	`  String groupId) (1)`
	`ListConsumerGroupOffsetsResult listConsumerGroupOff`
	`  String groupId,`
	`  ListConsumerGroupOffsetsOptions options)`
	
	1. Uses a new `ListConsumerGroupOffsetsOptions` wit
	
	Used for ConsumerGroupCommand (when executed v

`listConsumerGroups`	`ListConsumerGroupsResult listConsumerGroups() (1)`
	`ListConsumerGroupsResult listConsumerGroups(`
	`  ListConsumerGroupsOptions options)`
	
	1. Uses a new `ListConsumerGroupsOptions` with the d
	
	Used for ConsumerGroupCommand (when executed v

`listPartitionReassignments`	`ListPartitionReassignmentsResult listPartitionReass`
	`ListPartitionReassignmentsResult listPartitionReass`
	`  ListPartitionReassignmentsOptions options)`
	`ListPartitionReassignmentsResult listPartitionReass`
	`  Set<TopicPartition> partitions)`
	`ListPartitionReassignmentsResult listPartitionReass`
	`  Set<TopicPartition> partitions,`
	`  ListPartitionReassignmentsOptions options)`
	`ListPartitionReassignmentsResult listPartitionReass`
	`  Optional<Set<TopicPartition>> partitions,`
	`  ListPartitionReassignmentsOptions options)`
	
	1. Uses a new `ListPartitionReassignmentsOptions` w

`listTopics`	`ListTopicsResult listTopics() (1)`
	`ListTopicsResult listTopics(`
	`  ListTopicsOptions options)`
	
	1. Uses a new `ListTopicsOptions` with the defaults

`metrics`	`Map<MetricName, ? extends Metric> metrics()` Used for `StreamThread` (Kafka Streams)

`removeMembersFromConsumerGroup`	`RemoveMembersFromConsumerGroupResult removeMembersF`
	`String groupId,`
	`RemoveMembersFromConsumerGroupOptions options)`
	
	Removes members from the consumer group by given

`renewDelegationToken`	`RenewDelegationTokenResult renewDelegationToken(`
	`byte[] hmac) (1)`
	`RenewDelegationTokenResult renewDelegationToken(`
	`byte[] hmac,`
	`RenewDelegationTokenOptions options)`
	
	1. Uses a new `RenewDelegationTokenOptions` with the

AdminUtils Helper Object

`AdminUtils` is an utility (a Scala object) with methods for…FIXME

Table 1. AdminUtils API

Name	Description
`assignReplicasToBrokers`	`assignReplicasToBrokers(brokerMetadatas: Seq[BrokerMetadata], nPartitions: Int, replicationFactor: Int, fixedStartIndex: Int = -1, startPartitionId: Int = -1): Map[Int, Seq[Int]]`

`AdminUtils` uses **__admin_client** for…FIXME

`assignReplicasToBrokers` Method

```
assignReplicasToBrokers(
  brokerMetadatas: Seq[BrokerMetadata],
  nPartitions: Int,
  replicationFactor: Int,
  fixedStartIndex: Int = -1,
  startPartitionId: Int = -1): Map[Int, Seq[Int]]
```

`assignReplicasToBrokers` branches off per whether all the brokers have rack information or not:

- `assignReplicasToBrokers` assignReplicasToBrokersRackUnaware when all the brokers (in `brokerMetadatas`) have no rack information available.
- `assignReplicasToBrokers` assignReplicasToBrokersRackAware when all the brokers (in `brokerMetadatas`) have rack information available.

`assignReplicasToBrokers` throws a `InvalidPartitionsException` if the given `nPartitions` is `0` or less.

```
Number of partitions must be larger than 0.
```

`assignReplicasToBrokers` throws a `InvalidReplicationFactorException` if the given `replicationFactor` is `0` or less.

```
Replication factor must be larger than 0.
```

`assignReplicasToBrokers` throws a `InvalidReplicationFactorException` if the given `replicationFactor` is greater than the number of all brokers in the cluster.

```
Replication factor: [replicationFactor] larger than available brokers: [brokerMetadata
s.size].
```

`assignReplicasToBrokers` throws an `AdminOperationException` if there is at least one broker (in `brokerMetadatas`) with no rack information (when it is assumed either all brokers have it or none).

```
Not all brokers have rack information for replica rack aware assignment.
```

Note	`assignReplicasToBrokers` is used when: • `ReassignPartitionsCommand` is requested to generateAssignment • `AdminManager` is requested to createTopics • `AdminZkClient` is requested to create a topic and addPartitions

`assignReplicasToBrokersRackUnaware` Internal Method

```
assignReplicasToBrokersRackUnaware(
  nPartitions: Int,
  replicationFactor: Int,
  brokerList: Seq[Int],
  fixedStartIndex: Int,
  startPartitionId: Int): Map[Int, Seq[Int]]
```

`assignReplicasToBrokersRackUnaware` performs the replicas to brokers assignment in a fairly random manner (i.e. includes two random numbers). No additional information is used to make the decision (except the input parameters).

```
import kafka.admin.{AdminUtils, BrokerMetadata}
// assignReplicasToBrokersRackUnaware is a private method
// Using assignReplicasToBrokers instead as the entry point
val brokerMetadatas = Seq(
  BrokerMetadata(0, None),
  BrokerMetadata(1, None),
  BrokerMetadata(2, None))
val assignment = AdminUtils.assignReplicasToBrokers(
  brokerMetadatas,
  nPartitions = 3,
  replicationFactor = 2)
val output = assignment.toSeq.sortBy(_._1).map { case (brokerId, replicas) => s"$broke
rId => $replicas" }
scala> output.foreach(println)
0 => ArrayBuffer(2, 1)
1 => ArrayBuffer(0, 2)
2 => ArrayBuffer(1, 0)
```

`assignReplicasToBrokersRackUnaware` ...FIXME

Note	`assignReplicasToBrokersRackUnaware` is used exclusively when `AdminUtils` is requested to assignReplicasToBrokers (when all the brokers in a cluster have no rack information assigned).

`assignReplicasToBrokersRackAware` Internal Method

```
assignReplicasToBrokersRackAware(
  nPartitions: Int,
  replicationFactor: Int,
  brokerMetadatas: Seq[BrokerMetadata],
  fixedStartIndex: Int,
  startPartitionId: Int): Map[Int, Seq[Int]]
```

`assignReplicasToBrokersRackAware` ...FIXME

Note	`assignReplicasToBrokersRackAware` is used exclusively when `AdminUtils` is requested to assignReplicasToBrokers (when all the brokers in a cluster have rack information assigned).

Controller Broker

Controller Broker (KafkaController) is a Kafka service that runs on every broker in a Kafka cluster, but only one can be active (*elected*) at any point in time.

The process of promoting a broker to be the active controller is called Kafka Controller Election.

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.

Kafka Controller registers handlers to be notified about changes in Zookeeper and propagate them across brokers in a Kafka cluster.

Lifecycle

When started, `KafkaController` emits Startup controller event. That starts controller election (on the controller-event-thread).

During controller election, one `KafkaController` becomes active (*elected*) and onControllerFailover. The ControllerContext is built on what is available in Zookeeper.

While in initializeControllerContext, `KafkaController` updateLeaderAndIsrCache (and reads partition state from `/brokers/topics/[topic]/partitions/[partition]/state` paths in Zookeeper that is then stored as partitionLeadershipInfo of the ControllerContext).

KafkaController

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

Figure 1. KafkaController

`KafkaController` is the default and only known ControllerEventProcessor to process and preempt controller events.

`KafkaController` is in one of the ControllerStates (that is the state of the ControllerEventManager).

`KafkaController` uses the KafkaZkClient to be notified about changes in the state of a Kafka cluster (that are reflected in changes in znodes of Apache Zookeeper) and propagate the state changes to other brokers.

`KafkaController` uses listeners as a notification system to monitor znodes in Zookeeper and react accordingly.

`KafkaController` emulates a state machine using controller events.

Table 1. KafkaController's Controller Events

Event	ControllerState	
ApiPartitionReassignment		processApiPartiti
AutoPreferredReplicaLeaderElection		processAutoPrefe
BrokerChange	BrokerChange	processBrokerCh
BrokerModifications		processBrokerMo
ControlledShutdown		

• `ID` • `controlledShutdownCallback` : Try[Set[TopicAndPartition]] ⇒ Unit	ControlledShutdown	processControlle
ControllerChange • `newControllerId` : Int	ControllerChange	processControlle 1. Assigns the `newControlle` 2. (only when th controller) Re Note: Similar differen electio
Expire		processExpire
IsrChangeNotification		processIsrChang
LeaderAndIsrResponseReceived		processLeaderAr
ListPartitionReassignments		processListPartiti
LogDirEventNotification		processLogDirEv
PartitionModifications		processPartitionM
PartitionReassignmentIsrChange		processPartitionR
Reelect	ControllerChange	processReelect 1. Assigns the activeContro 2. (only when th controller) Re 3. elect
RegisterBrokerAndReelect		processRegisterB
ReplicaLeaderElection		processReplicaLe
ShutdownEventThread		
Startup	ControllerChange	processStartup 1. registerSessi

Startup	ControllerChange	2. registerContr 3. elect
TopicChange		processTopicCha
TopicDeletion		processTopicDele
TopicDeletionStopReplicaResponseReceived		processTopicDele
TopicUncleanLeaderElectionEnable		processTopicUnc
UncleanLeaderElectionEnable		processUncleanL
ZkPartitionReassignment		processZkPartitio

Table 2. KafkaController's ZNodeChangeHandler and ZNodeChildChangeHandlers

Name	Description
`brokerChangeHandler`	`ZNodeChildChangeHandler` of `/brokers/ids` path On `handleChildChange`, `brokerChangeHandler` simply sends BrokerChange event to the ControllerEventManager.
`isrChangeNotificationHandler`	`ZNodeChildChangeHandler` of `/isr_change_notification` path On `handleChildChange`, `isrChangeNotificationHandler` simply sends IsrChangeNotification event to the ControllerEventManager.
`logDirEventNotificationHandler`	`ZNodeChildChangeHandler` of `/log_dir_event_notification` path On `handleChildChange`, `logDirEventNotificationHandler` simply sends LogDirEventNotification event to the ControllerEventManager.
`partitionModificationsHandlers`	`ZNodeChangeHandlers` per topic of `/brokers/topics/[topic]` path On `handleDataChange`, `partitionModificationsHandlers` simply send PartitionModifications event to the ControllerEventManager.

`partitionReassignmentHandler`	`ZNodeChangeHandler` of `/admin/reassign_partitions` path On `handleCreation`, `partitionReassignmentHandler` simply sends PartitionReassignment event to the ControllerEventManager.
`preferredReplicaElectionHandler`	`ZNodeChangeHandler` of `/admin/preferred_replica_election` path On `handleCreation`, `preferredReplicaElectionHandler` simply sends PreferredReplicaLeaderElection event to the ControllerEventManager.
`topicChangeHandler`	`ZNodeChildChangeHandler` of `/brokers/topics` path On `handleChildChange`, `topicChangeHandler` simply sends TopicChange event to the ControllerEventManager.
`topicDeletionHandler`	`ZNodeChildChangeHandler` of `/admin/delete_topics` path On `handleChildChange`, `topicDeletionHandler` simply sends TopicDeletion event to the ControllerEventManager.

Table 3. KafkaController's Listeners

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

`KafkaController` uses **[Controller id=[brokerId]]** as the logging prefix (aka `logIdent`).

Tip	Enable `ALL` logging level for `kafka.controller.KafkaController` logger to see what happens inside.
	
	Add the following line to `config/log4j.properties`:
	
	`log4j.logger.kafka.controller.KafkaController=ALL`
	
	Refer to Logging.
	
	--- 	
	
	Please note that Kafka comes with a preconfigured `kafka.controller` logger in `config/log4j.properties`:
	
	`log4j.appender.controllerAppender=org.apache.log4j.DailyRollingFileAppender`
	`log4j.appender.controllerAppender.DatePattern='.'yyyy-MM-dd-HH`
	`log4j.appender.controllerAppender.File=${kafka.logs.dir}/controller.log`
	`log4j.appender.controllerAppender.layout=org.apache.log4j.PatternLayout`
	`log4j.appender.controllerAppender.layout.ConversionPattern=[%d] %p %m (%c)%n`
	
	`log4j.logger.kafka.controller=TRACE, controllerAppender`
	`log4j.additivity.kafka.controller=false`
	
	That means that the logs of `KafkaController` go to `logs/controller.log` file at `TRACE` logging level and are not added to the main logs (per `log4j.additivity` being off).

Creating KafkaController Instance

`KafkaController` takes the following to be created:

- KafkaConfig
- KafkaZkClient
- `Time`
- Metrics
- `BrokerInfo`
- `initialBrokerEpoch`
- DelegationTokenManager
- Thread name prefix (default: undefined)

`KafkaController` initializes the internal properties.

KafkaController and ControllerContext

```
controllerContext: ControllerContext
```

When created, `KafkaController` creates a new ControllerContext.

KafkaController and ControllerChannelManager

```
controllerChannelManager: ControllerChannelManager
```

When created, `KafkaController` creates a new ControllerChannelManager.

`ControllerChannelManager` is used to create separate ControllerBrokerRequestBatches of the KafkaController itself, the ZkReplicaStateMachine and ZkPartitionStateMachine.

`ControllerChannelManager` is requested to start up when `KafkaController` is requested to start controller election (and a broker is successfully elected as the active controller).

`KafkaController` uses the `ControllerChannelManager` to add or remove brokers when processing broker changes in Zookeeper (a new or updated znode under `/brokers/ids` path).

`ControllerChannelManager` is requested to shut down when `KafkaController` is requested to resign as the active controller.

KafkaController and ReplicaStateMachine (ZkReplicaStateMachine)

```
replicaStateMachine: ReplicaStateMachine
```

When created, `KafkaController` creates a new ZkReplicaStateMachine.

`ZkReplicaStateMachine` is requested to start up at onControllerFailover (when a broker is successfully elected as the controller) and shut down at controller resignation.

`ZkReplicaStateMachine` is requested to handle state changes of partition replicas at the following events:

- onBrokerLogDirFailure to transition replicas to `OnlineReplica` state
- onBrokerStartup to transition replicas to `OnlineReplica` state
- onReplicasBecomeOffline to transition replicas to `OfflineReplica` state

- onNewPartitionCreation to transition replicas to `NewReplica` state first and then to `OnlineReplica`
- onPartitionReassignment to transition replicas to `OnlineReplica` state
- stopOldReplicasOfReassignedPartition to transition replicas to `OfflineReplica` state first and then to `ReplicaDeletionStarted`, `ReplicaDeletionSuccessful`, and `NonExistentReplica` in the end
- startNewReplicasForReassignedPartition to transition replicas to `NewReplica` state
- doControlledShutdown to transition replicas to `OfflineReplica` state

`KafkaController` uses the `ZkReplicaStateMachine` to create the TopicDeletionManager.

KafkaController and PartitionStateMachine (ZkPartitionStateMachine)

```
partitionStateMachine: PartitionStateMachine
```

When created, `KafkaController` creates a new ZkPartitionStateMachine.

`ZkPartitionStateMachine` is requested to start up at onControllerFailover (when a broker is successfully elected as the controller) and shut down at controller resignation.

`ZkPartitionStateMachine` is requested to triggerOnlinePartitionStateChange at the following events:

- onBrokerStartup
- onReplicasBecomeOffline
- processUncleanLeaderElectionEnable
- processTopicUncleanLeaderElectionEnable (with a topic)

`ZkPartitionStateMachine` is requested to handleStateChanges at the following events:

- onReplicasBecomeOffline
- onNewPartitionCreation to transition partitions to `NewPartition` state first and then to `OnlinePartition`
- onPreferredReplicaElection to transition partitions to `OnlinePartition`
- moveReassignedPartitionLeaderIfRequired to transition partitions to `OnlinePartition`
- doControlledShutdown to transition partitions to `OnlinePartition` state

`KafkaController` uses the `ZkPartitionStateMachine` to create the TopicDeletionManager.

Preempting Controller Events — `preempt` Method

```
preempt(event: ControllerEvent): Unit
```

Note	`preempt` is part of the ControllerEventProcessor Contract to preempt controller events.

`preempt` ...FIXME

Processing Controller Events — `process` Method

```
process(event: ControllerEvent): Unit
```

Note	`process` is part of the ControllerEventProcessor Contract to process controller events.

`process` handles the ControllerEvent using ControllerEvent handlers.

Table 4. ControllerEvent Handlers

Name	Description
`ShutdownEventThread`	
`AutoPreferredReplicaLeaderElection`	processAutoPreferredReplicaLeaderElectior
`ReplicaLeaderElection`	processReplicaLeaderElection
`UncleanLeaderElectionEnable`	processUncleanLeaderElectionEnable
`TopicUncleanLeaderElectionEnable`	processTopicUncleanLeaderElectionEnable
`ControlledShutdown`	processControlledShutdown
`LeaderAndIsrResponseReceived`	processLeaderAndIsrResponseReceived
`TopicDeletionStopReplicaResponseReceived`	processTopicDeletionStopReplicaResponse
`BrokerChange`	processBrokerChange
`BrokerModifications`	processBrokerModification

`ControllerChange`	processControllerChange
`Reelect`	processReelect
`RegisterBrokerAndReelect`	processRegisterBrokerAndReelect
`Expire`	processExpire
`TopicChange`	processTopicChange
`LogDirEventNotification`	processLogDirEventNotification
`PartitionModifications`	processPartitionModifications
`TopicDeletion`	processTopicDeletion
`ApiPartitionReassignment`	processApiPartitionReassignment
`ZkPartitionReassignment`	processZkPartitionReassignment
`ListPartitionReassignments`	processListPartitionReassignments
`PartitionReassignmentIsrChange`	processPartitionReassignmentIsrChange
`IsrChangeNotification`	processIsrChangeNotification
`Startup`	processStartup

In the end, `process` updateMetrics.

In case of a `ControllerMovedException`, `process` prints out the following INFO message to the logs and maybeResign.

```
Controller moved to another broker when processing [event].
```

In case of any error (`Throwable`), `process` simply prints out the following ERROR message to the logs:

```
Error processing event [event]
```

`initiateReassignReplicasForTopicPartition` Method

```
initiateReassignReplicasForTopicPartition
```

`initiateReassignReplicasForTopicPartition` ...FIXME

Note	`initiateReassignReplicasForTopicPartition` is used when...FIXME

`deregisterPartitionReassignmentIsrChangeListeners` Method

```
deregisterPartitionReassignmentIsrChangeListeners
```

`deregisterPartitionReassignmentIsrChangeListeners` ...FIXME

Note	`deregisterPartitionReassignmentIsrChangeListeners` is used when...FIXME

`resetControllerContext` Method

```
resetControllerContext
```

`resetControllerContext` ...FIXME

Note	`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: • `ControllerEventThread` is requested to process controller events, i.e. ControllerChange and Reelect • triggerControllerMove • `KafkaController` is requested to shut down

Unsubscribing from Child Changes to /isr_change_notification ZNode — `deregisterIsrChangeNotificationListener` Internal Method

```
deregisterIsrChangeNotificationListener(): Unit
```

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

```
De-registering IsrChangeNotificationListener
```

`deregisterIsrChangeNotificationListener` requests ZkUtils to unsubscribe from intercepting changes to `/isr_change_notification` znode with IsrChangeNotificationListener.

Note	`deregisterIsrChangeNotificationListener` is used exclusively when `KafkaController` resigns as the active controller.

Unsubscribing from Child Changes to /log_dir_event_notification ZNode — `deregisterLogDirEventNotificationListener` Internal Method

```
deregisterLogDirEventNotificationListener(): Unit
```

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

```
De-registering logDirEventNotificationListener
```

`deregisterLogDirEventNotificationListener` requests ZkUtils to unsubscribe from intercepting changes to `/log_dir_event_notification` znode with LogDirEventNotificationListener.

Note	`deregisterLogDirEventNotificationListener` is used exclusively when `KafkaController` resigns as the active controller.

Unsubscribing from Data Changes to /admin/preferred_replica_election ZNode — `deregisterPreferredReplicaElectionListener` Method

```
deregisterPreferredReplicaElectionListener(): Unit
```

`deregisterPreferredReplicaElectionListener` requests ZkUtils to unsubscribe from intercepting data changes to `/admin/preferred_replica_election` znode with PreferredReplicaElectionListener.

Note	`deregisterPreferredReplicaElectionListener` is used exclusively when `KafkaController` resigns as the active controller.

Unsubscribing from Data Changes to /admin/reassign_partitions ZNode — `deregisterPartitionReassignmentListener` Method

```
deregisterPartitionReassignmentListener(): Unit
```

`deregisterPartitionReassignmentListener` requests ZkUtils to unsubscribe from intercepting data changes to `/admin/reassign_partitions` znode with PartitionReassignmentListener.

Note	`deregisterPartitionReassignmentListener` is used exclusively when `KafkaController` resigns as the active controller.

`triggerControllerMove` Internal Method

```
triggerControllerMove(): Unit
```

`triggerControllerMove` ...FIXME

Note	`triggerControllerMove` is used when `KafkaController` is requested to handleIllegalState and elect an active controller (and failed).

`handleIllegalState` Internal Method

```
handleIllegalState(e: IllegalStateException): Nothing
```

`handleIllegalState` ...FIXME

Note	`handleIllegalState` is used when `KafkaController` catches an `IllegalStateException` in updateLeaderEpochAndSendRequest, sendUpdateMetadataRequest and when processing a ControlledShutdown event.

`sendUpdateMetadataRequest` Method

```
sendUpdateMetadataRequest(): Unit
```

`sendUpdateMetadataRequest` requests the ControllerBrokerRequestBatch to newBatch and addUpdateMetadataRequestForBrokers.

In the end, `sendUpdateMetadataRequest` requests the ControllerBrokerRequestBatch to sendRequestsToBrokers with the current epoch.

In case of `IllegalStateException`, `sendUpdateMetadataRequest` handleIllegalState (that triggers controller movement).

Note	`sendUpdateMetadataRequest` is used when: • `KafkaController` is requested to onControllerFailover, onBrokerStartup, onBrokerUpdate, onReplicasBecomeOffline, onPartitionReassignment, process a IsrChangeNotification controller event • `TopicDeletionManager` is requested to onTopicDeletion

`updateLeaderEpochAndSendRequest` Internal Method

```
updateLeaderEpochAndSendRequest(
  partition: TopicPartition,
  replicasToReceiveRequest: Seq[Int],
  newAssignedReplicas: Seq[Int]): Unit
```

`updateLeaderEpochAndSendRequest` updates leader epoch for the partition and branches off per result: a LeaderIsrAndControllerEpoch or none at all.

`updateLeaderEpochAndSendRequest` and LeaderIsrAndControllerEpoch

When updating leader epoch for the partition returns a `LeaderIsrAndControllerEpoch`, `updateLeaderEpochAndSendRequest` requests the ControllerBrokerRequestBatch to prepare a new batch. `updateLeaderEpochAndSendRequest` requests the ControllerBrokerRequestBatch to addLeaderAndIsrRequestForBrokers followed by sendRequestsToBrokers.

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

```
Sent LeaderAndIsr request [updatedLeaderIsrAndControllerEpoch]
with new assigned replica list [newAssignedReplicas] to leader
[leader] for partition being reassigned [partition]
```

`updateLeaderEpochAndSendRequest` and No LeaderIsrAndControllerEpoch

When updating leader epoch for the partition returns `None`, `updateLeaderEpochAndSendRequest` prints out the following ERROR message to the logs:

```
Failed to send LeaderAndIsr request with new assigned replica
list [newAssignedReplicas] to leader for partition being
reassigned [partition]
```

Note	`updateLeaderEpochAndSendRequest` is used when `KafkaController` is requested to onPartitionReassignment and moveReassignedPartitionLeaderIfRequired.

Shutting Down — `shutdown` Method

```
shutdown(): Unit
```

`shutdown` requests the ControllerEventManager to close followed by onControllerResignation.

Note	`shutdown` is used exclusively when `KafkaServer` is requested to shutdown.

`updateMetrics` Internal Method

```
updateMetrics(): Unit
```

`updateMetrics` ...FIXME

Note	`updateMetrics` is used exclusively when `KafkaController` is created (and creates the ControllerEventManager).

`onBrokerStartup` Method

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

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

```
New broker startup callback for [newBrokers]
```

`onBrokerStartup` requests the ControllerContext for the replicasOnOfflineDirs and removes the given broker IDs (in `newBrokers`).

`onBrokerStartup` sendUpdateMetadataRequest to the liveOrShuttingDownBrokerIds (of the ControllerContext).

`onBrokerStartup` requests the ControllerContext for the replicas on the given `newBrokers`.

`onBrokerStartup` requests the ReplicaStateMachine to handleStateChanges for the replicas on the new brokers and `OnlineReplica` target state.

`onBrokerStartup` requests the PartitionStateMachine to triggerOnlinePartitionStateChange.

`onBrokerStartup` requests the ControllerContext for the partitionsBeingReassigned and collects the partitions that have replicas on the new brokers. For every partition with a replica on the new brokers, `onBrokerStartup` onPartitionReassignment.

`onBrokerStartup` collects replicas (on the new brokers) that are scheduled to be deleted by requesting the TopicDeletionManager to see whether isTopicQueuedUpForDeletion. If there are any, `onBrokerStartup` prints out the following INFO message to the logs and requests

the TopicDeletionManager to resumeDeletionForTopics.

```
Some replicas [replicasForTopicsToBeDeleted] for topics scheduled for deletion [topics
ToBeDeleted] are on the newly restarted brokers [newBrokers]. Signaling restart of top
ic deletion for these topics
```

In the end, `onBrokerStartup` registerBrokerModificationsHandler for the new brokers.

Note	`onBrokerStartup` is used exclusively when `KafkaController` is requested to process a BrokerChange controller event.

Controller Election — `elect` Method

```
elect(): Unit
```

`elect` requests the KafkaZkClient for the active controller ID (or assumes `-1` if not available) and saves it to the activeControllerId internal registry.

`elect` stops the controller election if there is an active controller ID available and prints out the following DEBUG message to the logs:

```
Broker [activeControllerId] has been elected as the controller, so stopping the electi
on process.
```

Otherwise, with no active controller, `elect` requests the KafkaZkClient to registerControllerAndIncrementControllerEpoch (with the broker ID).

`elect` saves the controller epoch and the zookeeper epoch as the epoch and epochZkVersion of the ControllerContext, respectively.

`elect` saves the broker ID as the activeControllerId internal registry.

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

```
[brokerId] successfully elected as the controller. Epoch
incremented to [epoch] and epoch zk version is now
[epochZkVersion]
```

In the end, `elect` onControllerFailover.

Note	`elect` is used when `ControllerEventThread` is requested to process Startup and Reelect controller events.

`elect` and ControllerMovedException

In case of a `ControllerMovedException`, `elect` maybeResign and prints out either DEBUG or WARN message to the logs per the activeControllerId internal registry:

```
Broker [activeControllerId] was elected as controller instead of broker [brokerId]
```

```
A controller has been elected but just resigned, this will result in another round of
election
```

`elect` and Throwable

In case of a `Throwable`, `elect` prints out the following ERROR message to the logs and triggerControllerMove.

```
Error while electing or becoming controller on broker
[brokerId]. Trigger controller movement immediately
```

Is Broker The Active Controller? — `isActive` Method

```
isActive: Boolean
```

`isActive` indicates whether the current broker (by the broker ID) hosts the active `KafkaController` (given the activeControllerId) or not.

Note	`isActive` is on (`true`) after the `KafkaController` of a Kafka broker has been elected.

Note	`isActive` is used (as a valve to stop processing early) when: • `ControllerEventThread` is requested to process controller events (that should only be processed on the active controller, e.g. `AutoPreferredReplicaLeaderElection`, `UncleanLeaderElectionEnable`, `ControlledShutdown`, `LeaderAndIsrResponseReceived`, `TopicDeletionStopReplicaResponseReceived`, `BrokerChange`, `BrokerModifications`, `TopicChange`) • `KafkaController` is requested to updateMetrics • `KafkaApis` is requested to handleCreateTopicsRequest, handleCreatePartitionsRequest and handleDeleteTopicsRequest

`registerIsrChangeNotificationListener` Internal Method

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

`registerIsrChangeNotificationListener` ...FIXME

Note	`registerIsrChangeNotificationListener` is used when...FIXME

`deregisterIsrChangeNotificationListener` Internal Method

```
deregisterIsrChangeNotificationListener(): Unit
```

`deregisterIsrChangeNotificationListener` ...FIXME

Note	`deregisterIsrChangeNotificationListener` is used when...FIXME

Starting Up — `startup` Method

```
startup(): Unit
```

`startup` requests the KafkaZkClient to register a StateChangeHandler (under the name **controller-state-change-handler**) that is does the following:

- On `afterInitializingSession`, the `StateChangeHandler` simply puts `RegisterBrokerAndReelect` event on the event queue of the ControllerEventManager
- On `beforeInitializingSession`, the `StateChangeHandler` simply puts `Expire` event on the event queue of the ControllerEventManager

`startup` then puts `Startup` event at the end of the event queue of the ControllerEventManager and immediately requests it to start.

Note	`startup` is used exclusively when `KafkaServer` is requested to start.

Registering SessionExpirationListener To Control Session Recreation — `registerSessionExpirationListener` Internal Method

```
registerSessionExpirationListener(): Unit
```

`registerSessionExpirationListener` requests ZkUtils to subscribe to state changes with a `SessionExpirationListener` (with the `KafkaController` and ControllerEventManager).

Note	`SessionExpirationListener` puts Reelect event on the event queue of `ControllerEventManager` every time the Zookeeper session has expired and a new session has been created.

Note	`registerSessionExpirationListener` is used exclusively when Startup event is processed (after `ControllerEventThread` is started).

Registering ControllerChangeListener for /controller ZNode Changes — `registerControllerChangeListener` Internal Method

```
registerControllerChangeListener(): Unit
```

`registerControllerChangeListener` requests ZkUtils to subscribe to data changes for `/controller` znode with a `ControllerChangeListener` (with the `KafkaController` and ControllerEventManager).

Note	`ControllerChangeListener` emits: 1. ControllerChange event with the current controller ID (on the event queue of `ControllerEventManager`) every time the data of a znode changes 2. Reelect event when the data associated with a znode has been deleted

Note	`registerControllerChangeListener` is used exclusively when Startup event is processed (after `ControllerEventThread` is started).

`registerBrokerChangeListener` Internal Method

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

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

Note	`registerBrokerChangeListener` is used exclusively when `KafkaController` does onControllerFailover.

Getting Active Controller ID (from JSON under /controller znode) — `getControllerID` Method

```
getControllerID(): Int
```

`getControllerID` returns the ID of the active Kafka controller that is associated with `/controller` znode in JSON format or `-1` otherwise.

Internally, `getControllerID` requests ZkUtils for data associated with `/controller` znode.

If available, `getControllerID` parses the data (being the current controller info in JSON format) to extract `brokerid` field.

```
$ ./bin/zookeeper-shell.sh :2181 get /controller

{"version":1,"brokerid":0,"timestamp":"1543499076007"}
cZxid = 0x60
ctime = Thu Nov 29 14:44:36 CET 2018
mZxid = 0x60
mtime = Thu Nov 29 14:44:36 CET 2018
pZxid = 0x60
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x100073f07ba0003
dataLength = 54
numChildren = 0
```

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

Note	`getControllerID` is used when: 1. Processing `Reelect` controller event 2. elect

Registering TopicDeletionListener for Child Changes to /admin/delete_topics ZNode — `registerTopicDeletionListener` Internal Method

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

`registerTopicDeletionListener` requests ZkUtils to subscribeChildChanges to `/admin/delete_topics` znode with TopicDeletionListener.

Note	`registerTopicDeletionListener` is used exclusively when `KafkaController` does onControllerFailover.

De-Registering TopicDeletionListener for Child Changes to /admin/delete_topics ZNode — `deregisterTopicDeletionListener` Internal Method

```
deregisterTopicDeletionListener(): Unit
```

`deregisterTopicDeletionListener` requests ZkUtils to unsubscribeChildChanges to `/admin/delete_topics` znode with TopicDeletionListener.

Note	`deregisterTopicDeletionListener` is used exclusively when `KafkaController` resigns as the active controller.

`onReplicasBecomeOffline` Internal Method

```
onReplicasBecomeOffline(newOfflineReplicas: Set[PartitionAndReplica]): Unit
```

`onReplicasBecomeOffline` ...FIXME

Note	`onReplicasBecomeOffline` is used when...FIXME

`onPartitionReassignment` Internal Method

```
onPartitionReassignment(
  topicPartition: TopicPartition,
  reassignedPartitionContext: ReassignedPartitionsContext): Unit
```

`onPartitionReassignment` ...FIXME

Note	`onPartitionReassignment` is used when `KafkaController` is requested to onBrokerStartup, maybeTriggerPartitionReassignment and process a PartitionReassignmentIsrChange event.

`onBrokerUpdate` Internal Method

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

`onBrokerUpdate` ...FIXME

Note	`onBrokerUpdate` is used when...FIXME

`updateBrokerInfo` Internal Method

```
updateBrokerInfo(newBrokerInfo: BrokerInfo): Unit
```

`updateBrokerInfo` ...FIXME

Note	`updateBrokerInfo` is used exclusively when `DynamicListenerConfig` is requested to reconfigure.

`registerBrokerModificationsHandler` Internal Method

```
registerBrokerModificationsHandler(brokerIds: Iterable[Int]): Unit
```

`registerBrokerModificationsHandler` ...FIXME

Note	`registerBrokerModificationsHandler` is used when `KafkaController` is requested to onBrokerStartup and onControllerFailover (indirectly through initializeControllerContext).

Initializing ControllerContext — `initializeControllerContext` Internal Method

```
initializeControllerContext(): Unit
```

`initializeControllerContext` ...FIXME

In the end, `initializeControllerContext` prints out the following INFO messages to the logs (with the current state based on the ControllerContext):

```
Currently active brokers in the cluster: [liveBrokerIds]
Currently shutting brokers in the cluster:
[shuttingDownBrokerIds]
Current list of topics in the cluster: [allTopics]
```

Note	`initializeControllerContext` is used exclusively when `KafkaController` is requested to onControllerFailover.

`unregisterBrokerModificationsHandler` Internal Method

```
unregisterBrokerModificationsHandler(brokerIds: Iterable[Int]): Unit
```

`unregisterBrokerModificationsHandler` ...FIXME

Note	`unregisterBrokerModificationsHandler` is used when `KafkaController` is requested to onControllerResignation and onBrokerFailure.

`onBrokerFailure` Internal Method

```
onBrokerFailure(deadBrokers: Seq[Int]): Unit
```

`onBrokerFailure` ...FIXME

Note	`onBrokerFailure` is used exclusively when `KafkaController` is requested to handle a BrokerChange controller event.

`maybeTriggerPartitionReassignment` Internal Method

```
maybeTriggerPartitionReassignment(topicPartitions: Set[TopicPartition]): Unit
```

`maybeTriggerPartitionReassignment` ...FIXME

Note	`maybeTriggerPartitionReassignment` is used when `KafkaController` is requested to onControllerFailover and process the PartitionReassignment controller event.

`incrementControllerEpoch` Internal Method

```
incrementControllerEpoch(): Unit
```

`incrementControllerEpoch` ...FIXME

Note	`incrementControllerEpoch` is used exclusively when `KafkaController` is requested to onControllerFailover.

`fetchPendingPreferredReplicaElections` Internal Method

```
fetchPendingPreferredReplicaElections(): Set[TopicPartition]
```

`fetchPendingPreferredReplicaElections` ...FIXME

Note	`fetchPendingPreferredReplicaElections` is used exclusively when `KafkaController` is requested to onControllerFailover.

`initializePartitionReassignment` Internal Method

```
initializePartitionReassignment(): Unit
```

`initializePartitionReassignment` ...FIXME

Note	`initializePartitionReassignment` is used exclusively when `KafkaController` is requested to initializeControllerContext.

`fetchTopicDeletionsInProgress` Internal Method

```
fetchTopicDeletionsInProgress(): (Set[String], Set[String])
```

`fetchTopicDeletionsInProgress` ...FIXME

Note	`fetchTopicDeletionsInProgress` is used exclusively when `KafkaController` is requested to onControllerFailover.

`updateLeaderAndIsrCache` Internal Method

```
updateLeaderAndIsrCache(partitions: Seq[TopicPartition]
```

Unless given, `updateLeaderAndIsrCache` defaults to allPartitions of the ControllerContext for the partitions.

`updateLeaderAndIsrCache` requests the KafkaZkClient to getTopicPartitionStates (with the given partitions) and updates the partitionLeadershipInfo of the ControllerContext.

Note	`updateLeaderAndIsrCache` is used when `KafkaController` is requested to initializeControllerContext (with no partitions) and process an IsrChangeNotification controller event (with partitions given).

`areReplicasInIsr` Internal Method

```
areReplicasInIsr(partition: TopicPartition, replicas: Seq[Int]): Boolean
```

`areReplicasInIsr` ...FIXME

Note	`areReplicasInIsr` is used exclusively when `KafkaController` is requested to onPartitionReassignment.

`updateAssignedReplicasForPartition` Internal Method

```
updateAssignedReplicasForPartition(
  partition: TopicPartition,
  replicas: Seq[Int]): Unit
```

`updateAssignedReplicasForPartition` ...FIXME

Note	`updateAssignedReplicasForPartition` is used exclusively when `KafkaController` is requested to onPartitionReassignment.

`registerPartitionModificationsHandlers` Internal Method

```
registerPartitionModificationsHandlers(topics: Seq[String]): Unit
```

`registerPartitionModificationsHandlers` ...FIXME

Note	`registerPartitionModificationsHandlers` is used when `KafkaController` is requested to initializeControllerContext and a TopicChange controller event is processed.

`unregisterPartitionModificationsHandlers` Internal Method

```
unregisterPartitionModificationsHandlers(topics: Seq[String]): Unit
```

`unregisterPartitionModificationsHandlers` ...FIXME

Note	`unregisterPartitionModificationsHandlers` is used when: • `KafkaController` is requested to onControllerResignation • `TopicDeletionManager` is requested to completeDeleteTopic

`unregisterPartitionReassignmentIsrChangeHandlers` Internal Method

```
unregisterPartitionReassignmentIsrChangeHandlers(): Unit
```

`unregisterPartitionReassignmentIsrChangeHandlers` ...FIXME

Note	`unregisterPartitionReassignmentIsrChangeHandlers` is used exclusively when `KafkaController` is requested to onControllerResignation.

`readControllerEpochFromZooKeeper` Internal Method

```
readControllerEpochFromZooKeeper(): Unit
```

`readControllerEpochFromZooKeeper` ...FIXME

Note	`readControllerEpochFromZooKeeper` is used exclusively when `KafkaController` is requested to onControllerFailover.

`removePartitionsFromReassignedPartitions` Internal Method

```
removePartitionsFromReassignedPartitions(partitionsToBeRemoved: Set[TopicPartition]):
Unit
```

`removePartitionsFromReassignedPartitions` ...FIXME

Note	`removePartitionsFromReassignedPartitions` is used when `KafkaController` is requested to onPartitionReassignment and maybeTriggerPartitionReassignment.

`removePartitionsFromPreferredReplicaElection` Internal Method

```
removePartitionsFromPreferredReplicaElection(
  partitionsToBeRemoved: Set[TopicPartition],
  isTriggeredByAutoRebalance : Boolean): Unit
```

`removePartitionsFromPreferredReplicaElection` ...FIXME

Note	`removePartitionsFromPreferredReplicaElection` is used exclusively when `KafkaController` is requested to onPreferredReplicaElection (and the election type is not AdminClientTriggered).

Preferred Replica Leader Election — `onPreferredReplicaElection` Internal Method

```
onPreferredReplicaElection(
  partitions: Set[TopicPartition],
  electionType: ElectionType): Map[TopicPartition, Throwable]
```

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

```
Starting preferred replica leader election for partitions [partitions]
```

`onPreferredReplicaElection` requests the PartitionStateMachine to handle partition state changes for the partitions (with `OnlinePartition` target state and PreferredReplicaPartitionLeaderElectionStrategy).

(only for election types that are not AdminClientTriggered) In the end, `onPreferredReplicaElection` removePartitionsFromPreferredReplicaElection.

(only for election types that are not AdminClientTriggered) In case of an error

Note	`onPreferredReplicaElection` is used when `KafkaController` is requested for the following: • onControllerFailover (with ZkTriggered election type) • checkAndTriggerAutoLeaderRebalance (with AutoTriggered election type) • Process a PreferredReplicaLeaderElection controller event (any election type with ZkTriggered the default)

`updateLeaderEpoch` Internal Method

```
updateLeaderEpoch(partition: TopicPartition): Option[LeaderIsrAndControllerEpoch]
```

`updateLeaderEpoch` ...FIXME

Note	`updateLeaderEpoch` is used exclusively when `KafkaController` is requested to updateLeaderEpochAndSendRequest.

`moveReassignedPartitionLeaderIfRequired` Internal Method

```
moveReassignedPartitionLeaderIfRequired(
  topicPartition: TopicPartition,
  reassignedPartitionContext: ReassignedPartitionsContext): Unit
```

`moveReassignedPartitionLeaderIfRequired` ...FIXME

Note	`moveReassignedPartitionLeaderIfRequired` is used exclusively when `KafkaController` is requested to onPartitionReassignment.

`onControllerFailover` Internal Method

```
onControllerFailover(): Unit
```

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

```
Registering handlers
```

`onControllerFailover` requests the KafkaZkClient to registerZNodeChildChangeHandlers:

- brokerChangeHandler
- topicChangeHandler
- topicDeletionHandler
- logDirEventNotificationHandler
- isrChangeNotificationHandler

`onControllerFailover` requests the KafkaZkClient to registerZNodeChangeHandlerAndCheckExistence:

- preferredReplicaElectionHandler
- partitionReassignmentHandler

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

```
Deleting log dir event notifications
```

`onControllerFailover` requests the KafkaZkClient to deleteLogDirEventNotifications (with the epochZkVersion of the ControllerContext).

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

```
Deleting isr change notifications
```

`onControllerFailover` requests the KafkaZkClient to deleteIsrChangeNotifications (with the epochZkVersion of the ControllerContext).

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

```
Initializing controller context
```

`onControllerFailover` initializeControllerContext.

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

```
Fetching topic deletions in progress
```

`onControllerFailover` fetchTopicDeletionsInProgress.

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

```
Initializing topic deletion manager
```

`onControllerFailover` requests the TopicDeletionManager to initialize (with the topics to be deleted and ineligible for deletion).

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

```
Sending update metadata request
```

`onControllerFailover` sendUpdateMetadataRequest (with the liveOrShuttingDownBrokerIds of the ControllerContext).

`onControllerFailover` requests the ReplicaStateMachine to start up.

`onControllerFailover` requests the PartitionStateMachine to start up.

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

```
Ready to serve as the new controller with epoch [epoch]
```

`onControllerFailover` maybeTriggerPartitionReassignment (with the partitionsBeingReassigned of the ControllerContext).

`onControllerFailover` requests the TopicDeletionManager to tryTopicDeletion.

`onControllerFailover` onPreferredReplicaElection with the fetchPendingPreferredReplicaElections.

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

```
Starting the controller scheduler
```

`onControllerFailover` requests the KafkaScheduler to startup.

With auto.leader.rebalance.enable enabled (default: `true`), `onControllerFailover` scheduleAutoLeaderRebalanceTask with the initial delay of 5 seconds.

With delegation.token.master.key password set (default: `(empty)`), `onControllerFailover` prints out the following INFO message to the logs:

```
starting the token expiry check scheduler
```

`onControllerFailover` requests the tokenCleanScheduler KafkaScheduler to startup and requests it to schedule the **delete-expired-tokens** task (FIXME).

Note	`onControllerFailover` is used when `KafkaController` is requested to elect (and a broker is successfully elected as the active controller).

`scheduleAutoLeaderRebalanceTask` Internal Method

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

`scheduleAutoLeaderRebalanceTask` simply requests the KafkaScheduler to schedule a task called **auto-leader-rebalance-task** with the given initial delay.

The `auto-leader-rebalance-task` simply requests the ControllerEventManager to enqueue an AutoPreferredReplicaLeaderElection controller event.

Note	`scheduleAutoLeaderRebalanceTask` is used when `KafkaController` is requested to onControllerFailover and processAutoPreferredReplicaLeaderElection

`processAutoPreferredReplicaLeaderElection` Internal Method

```
processAutoPreferredReplicaLeaderElection(): Unit
```

Note	`processAutoPreferredReplicaLeaderElection` does nothing (and simply returns) unless the Kafka broker (`KafkaController`) is an active controller.

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

```
Processing automatic preferred replica leader election
```

`processAutoPreferredReplicaLeaderElection` checkAndTriggerAutoLeaderRebalance.

In the end, `processAutoPreferredReplicaLeaderElection` scheduleAutoLeaderRebalanceTask with the initial delay based on leader.imbalance.check.interval.seconds configuration property (default: `300` seconds).

Note	`processAutoPreferredReplicaLeaderElection` is used exclusively when `KafkaController` is requested to process a AutoPreferredReplicaLeaderElection event.

`checkAndTriggerAutoLeaderRebalance` Internal Method

```
checkAndTriggerAutoLeaderRebalance(): Unit
```

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

```
Checking need to trigger auto leader balancing
```

`checkAndTriggerAutoLeaderRebalance` ...FIXME

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

```
Preferred replicas by broker [preferredReplicasForTopicsByBrokers]
```

For every broker with one or more partition leaders, `checkAndTriggerAutoLeaderRebalance` ... FIXME

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

```
Topics not in preferred replica for broker [leaderBroker] [topicsNotInPreferredReplica]
```

`checkAndTriggerAutoLeaderRebalance` calculates an imbalance ratio of the broker which is the number of `topicsNotInPreferredReplica` divided by the total number of partitions (`topicPartitionsForBroker`).

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

```
Leader imbalance ratio for broker [leaderBroker] is [imbalanceRatio]
```

With the imbalance ratio greater than the desired ratio (per leader.imbalance.per.broker.percentage configuration property with the default: `10%`), `checkAndTriggerAutoLeaderRebalance` onPreferredReplicaElection for...FIXME (with AutoTriggered election type).

Note	`checkAndTriggerAutoLeaderRebalance` is used exclusively when `KafkaController` is requested to processAutoPreferredReplicaLeaderElection.

`onNewPartitionCreation` Internal Method

```
onNewPartitionCreation(
  newPartitions: Set[TopicPartition]): Unit
```

`onNewPartitionCreation` ...FIXME

Note	`onNewPartitionCreation` is used when TopicChange and PartitionModifications controller events are processed.

`onReplicaElection` Internal Method

```
onReplicaElection(
  partitions: Set[TopicPartition],
  electionType: ElectionType,
  electionTrigger: ElectionTrigger): Map[TopicPartition, Either[Throwable, LeaderAndIsr
]]
```

`onReplicaElection` ...FIXME

Note	`onReplicaElection` is used when...FIXME

Handling Log Directory Failures for Brokers — `onBrokerLogDirFailure` Internal Method

```
onBrokerLogDirFailure(
  brokerIds: Seq[Int]): Unit
```

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

```
Handling log directory failure for brokers [brokerIds]
```

`onBrokerLogDirFailure` requests the ControllerContext for the replicas on the brokers and then requests the ReplicaStateMachine to handle state changes for the replicas to enter `OnlineReplica` state.

Note	`onBrokerLogDirFailure` is used exclusively when `KafkaController` is requested to process a LogDirEventNotification controller event (on controller-event-thread).

`stopOldReplicasOfReassignedPartition` Internal Method

```
stopOldReplicasOfReassignedPartition(
  topicPartition: TopicPartition,
  reassignedPartitionContext: ReassignedPartitionsContext,
  oldReplicas: Set[Int]): Unit
```

`stopOldReplicasOfReassignedPartition` ...FIXME

Note	`stopOldReplicasOfReassignedPartition` is used when...FIXME

`startNewReplicasForReassignedPartition` Internal Method

```
startNewReplicasForReassignedPartition(
  topicPartition: TopicPartition,
  reassignedPartitionContext: ReassignedPartitionsContext,
  newReplicas: Set[Int]): Unit
```

`startNewReplicasForReassignedPartition` ...FIXME

Note	`startNewReplicasForReassignedPartition` is used when...FIXME

`enableDefaultUncleanLeaderElection` Method

```
enableDefaultUncleanLeaderElection(): Unit
```

Note	`enableDefaultUncleanLeaderElection` does nothing (and simply returns) unless the Kafka broker (`KafkaController`) is an active controller.

`enableDefaultUncleanLeaderElection` simply requests the ControllerEventManager to enqueue a UncleanLeaderElectionEnable event.

Note	`enableDefaultUncleanLeaderElection` is used when `DynamicLogConfig` is requested to reconfigure (for unclean.leader.election.enable configuration property).

Preferred Replica Leader Election — `electPreferredLeaders` Method

```
electPreferredLeaders(
  partitions: Set[TopicPartition],
  callback: ElectPreferredLeadersCallback = { (_, _) => }): Unit
```

`electPreferredLeaders` simply requests the ControllerEventManager to enqueue an PreferredReplicaLeaderElection event (with AdminClientTriggered election type)

Note	`electPreferredLeaders` is used exclusively when `ReplicaManager` is requested to trigger preferred replica leader election.

`maybeResign` Internal Method

```
maybeResign(): Unit
```

`maybeResign` ...FIXME

Note	`maybeResign` is used when `KafkaController` is requested to...FIXME

`processControlledShutdown` Internal Method

```
processControlledShutdown(
  id: Int,
  brokerEpoch: Long,
  controlledShutdownCallback: Try[Set[TopicPartition]] => Unit): Unit
```

`processControlledShutdown` ...FIXME

Note	`processControlledShutdown` is used exclusively when `KafkaController` is requested to process a ControlledShutdown controller event.

`doControlledShutdown` Internal Method

```
doControlledShutdown(
  id: Int,
  brokerEpoch: Long): Set[TopicPartition]
```

`doControlledShutdown` ...FIXME

Note	`doControlledShutdown` is used exclusively when `KafkaController` is requested to processControlledShutdown.

`processUncleanLeaderElectionEnable` Internal Method

```
processUncleanLeaderElectionEnable(): Unit
```

Note	`processUncleanLeaderElectionEnable` does nothing (and simply returns) unless the Kafka broker (`KafkaController`) is an active controller.

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

```
Unclean leader election has been enabled by default
```

`processUncleanLeaderElectionEnable` requests the PartitionStateMachine to triggerOnlinePartitionStateChange.

Note	`processUncleanLeaderElectionEnable` is used exclusively when `KafkaController` is requested to process a UncleanLeaderElectionEnable controller event.

`processTopicUncleanLeaderElectionEnable` Internal Method

```
processTopicUncleanLeaderElectionEnable(topic: String): Unit
```

`processTopicUncleanLeaderElectionEnable` ...FIXME

Note	`processTopicUncleanLeaderElectionEnable` is used when `KafkaController` is requested to process a TopicUncleanLeaderElectionEnable controller event.

Processing LeaderAndIsrResponseReceived Controller Event (On controller-event-thread) — `processLeaderAndIsrResponseReceived` Internal Method

```
processLeaderAndIsrResponseReceived(
  leaderAndIsrResponseObj: AbstractResponse,
  brokerId: Int): Unit
```

`processLeaderAndIsrResponseReceived` ...FIXME

Note	`processLeaderAndIsrResponseReceived` is used exclusively when `KafkaController` is requested to process a LeaderAndIsrResponseReceived controller event (on the controller-event-thread).

`processTopicDeletionStopReplicaResponseReceived` Internal Method

```
processTopicDeletionStopReplicaResponseReceived(
  replicaId: Int,
  requestError: Errors,
  partitionErrors: Map[TopicPartition, Errors]): Unit
```

`processTopicDeletionStopReplicaResponseReceived` ...FIXME

Note	`processTopicDeletionStopReplicaResponseReceived` is used exclusively when `KafkaController` is requested to process a TopicDeletionStopReplicaResponseReceived controller event.

Processing BrokerChange Controller Event (On controller-event-thread) — `processBrokerChange` Internal Method

```
processBrokerChange(): Unit
```

Note	`processBrokerChange` does nothing (and simply returns) unless the Kafka broker (`KafkaController`) is an active controller.

`processBrokerChange` requests the KafkaZkClient for the brokers in a Kafka cluster and compares the broker list with the current brokers (of the ControllerContext).

At this point in time, `processBrokerChange` knows what brokers are new, dead or bounced.

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

```
Newly added brokers: [ids], deleted brokers: [ids], bounced
brokers: [ids], all live brokers: [ids]
```

`processBrokerChange` notifies (*updates*) the ControllerChannelManager:

- For every newly-added broker, `processBrokerChange` requests to register it
- For bounced brokers, `processBrokerChange` requests to deregister them all first followed by registering them
- For every deleted broker, `processBrokerChange` requests to deregister it

`processBrokerChange` updates the ControllerContext:

- For newly-added brokers (if there were any), `processBrokerChange` requests to addLiveBrokersAndEpochs followed by onBrokerStartup
- For bounced brokers (if there were any), `processBrokerChange` first requests to remove them followed by onBrokerFailure and then requests to addLiveBrokersAndEpochs followed by onBrokerStartup
- For deleted brokers (if there were any), `processBrokerChange` requests to remove them followed by onBrokerFailure

In the end, only when they were any updates (new, dead or bounced brokers), `processBrokerChange` prints out the following INFO message to the logs:

```
Updated broker epochs cache: [liveBrokerIdAndEpochs]
```

Note	`processBrokerChange` is used exclusively when `KafkaController` is requested to process a BrokerChange controller event (on the controller-event-thread).

`processBrokerModification` Internal Method

```
processBrokerModification(
  brokerId: Int): Unit
```

`processBrokerModification` ...FIXME

Note	`processBrokerModification` is used exclusively when `KafkaController` is requested to process a BrokerModifications controller event.

`processControllerChange` Internal Method

```
processControllerChange(): Unit
```

`processControllerChange` ...FIXME

Note	`processControllerChange` is used exclusively when `KafkaController` is requested to process a ControllerChange controller event.

`processReelect` Internal Method

```
processReelect(): Unit
```

`processReelect` ...FIXME

Note	`processReelect` is used when `KafkaController` is requested to processRegisterBrokerAndReelect and process a Reelect controller event.

`processRegisterBrokerAndReelect` Internal Method

```
processRegisterBrokerAndReelect(): Unit
```

`processRegisterBrokerAndReelect` ...FIXME

Note	`processRegisterBrokerAndReelect` is used exclusively when `KafkaController` is requested to process a RegisterBrokerAndReelect controller event.

`processExpire` Internal Method

```
processExpire(): Unit
```

`processExpire` ...FIXME

Note	`processExpire` is used exclusively when `KafkaController` is requested to process a Expire controller event.

`processTopicChange` Internal Method

```
processTopicChange(): Unit
```

`processTopicChange` ...FIXME

Note	`processTopicChange` is used exclusively when `KafkaController` is requested to process a TopicChange controller event.

Processing LogDirEventNotification Controller Event (On controller-event-thread) — `processLogDirEventNotification` Internal Method

```
processLogDirEventNotification(): Unit
```

Note	`processLogDirEventNotification` does nothing (and simply returns) unless the Kafka broker (`KafkaController`) is an active controller.

`processLogDirEventNotification` requests the KafkaZkClient for the LogDirEvent notifications (from Zookeeper) (*sequence numbers*).

`processLogDirEventNotification` requests the KafkaZkClient for the convert the LogDirEvent notifications to broker IDs and then handles the log directory failures for the brokers.

In the end, `processLogDirEventNotification` requests the KafkaZkClient to delete LogDirEvent notifications (from Zookeeper).

Note	`processLogDirEventNotification` is used exclusively when `KafkaController` is requested to process a LogDirEventNotification controller event.

`processPartitionModifications` Internal Method

```
processPartitionModifications(
  topic: String): Unit
```

`processPartitionModifications` ...FIXME

Note	`processPartitionModifications` is used exclusively when `KafkaController` is requested to process a PartitionModifications controller event.

`processTopicDeletion` Internal Method

```
processTopicDeletion(): Unit
```

`processTopicDeletion` ...FIXME

Note	`processTopicDeletion` is used exclusively when `KafkaController` is requested to process a TopicDeletion controller event.

`processPartitionReassignmentIsrChange` Internal Method

```
processPartitionReassignmentIsrChange(
  partition: TopicPartition): Unit
```

`processPartitionReassignmentIsrChange` ...FIXME

Note	`processPartitionReassignmentIsrChange` is used exclusively when `KafkaController` is requested to process a PartitionReassignmentIsrChange controller event.

`processIsrChangeNotification` Internal Method

```
processIsrChangeNotification(): Unit
```

`processIsrChangeNotification` ...FIXME

Note	`processIsrChangeNotification` is used exclusively when `KafkaController` is requested to process a IsrChangeNotification controller event.

`processStartup` Internal Method

```
processStartup(): Unit
```

`processStartup` requests the KafkaZkClient to registerZNodeChangeHandlerAndCheckExistence (with the ControllerChangeHandler).

In the end, `processStartup` starts controller election.

Note	`processStartup` is used exclusively when `KafkaController` is requested to process a Startup controller event (on the controller-event-thread).

`processZkPartitionReassignment` Internal Method

```
processZkPartitionReassignment(): Unit
```

`processZkPartitionReassignment` ...FIXME

Note	`processZkPartitionReassignment` is used when `KafkaController` is requested to process a ZkPartitionReassignment controller event and initializePartitionReassignments (for `KafkaController` to process Startup and Reelect controller events).

`initializePartitionReassignments` Internal Method

```
initializePartitionReassignments(): Unit
```

`initializePartitionReassignments` ...FIXME

Note	`initializePartitionReassignments` is used when `KafkaController` is requested to onControllerFailover (for `KafkaController` to process Startup and Reelect controller events).

`processReplicaLeaderElection` Internal Method

```
processReplicaLeaderElection(
  partitionsFromAdminClientOpt: Option[Set[TopicPartition]],
  electionType: ElectionType,
  electionTrigger: ElectionTrigger,
  callback: ElectLeadersCallback): Unit
```

`processReplicaLeaderElection` ...FIXME

Note	`processReplicaLeaderElection` is used when `KafkaController` is requested to process a ReplicaLeaderElection controller event.

`processListPartitionReassignments` Internal Method

```
processListPartitionReassignments(
  partitionsOpt: Option[Set[TopicPartition]],
  callback: ListReassignmentsCallback): Unit
```

`processListPartitionReassignments` ...FIXME

Note	`processListPartitionReassignments` is used when `KafkaController` is requested to process a ListPartitionReassignments controller event.

`processApiPartitionReassignment` Internal Method

```
processApiPartitionReassignment(
  reassignments: Map[TopicPartition, Option[Seq[Int]]],
  callback: AlterReassignmentsCallback): Unit
```

`processApiPartitionReassignment` ...FIXME

Note	`processApiPartitionReassignment` is used when `KafkaController` is requested to process a ApiPartitionReassignment controller event.

Preempting PreferredReplicaLeaderElection — `preemptPreferredReplicaLeaderElection` Internal Method

```
preemptPreferredReplicaLeaderElection(
  partitionsFromAdminClientOpt: Option[Set[TopicPartition]],
  callback: ElectPreferredLeadersCallback = (_, _) => {}): Unit
```

`preemptPreferredReplicaLeaderElection` ...FIXME

Note	`preemptPreferredReplicaLeaderElection` is used exclusively when `KafkaController` is requested to preempt a PreferredReplicaLeaderElection controller event.

Internal Properties

Name	Description
`activeControllerId`	The ID of the active `KafkaController` • Initialized to `-1`
`brokerRequestBatch`	ControllerBrokerRequestBatch (with the ControllerChannelManager, ControllerEventManager, and ControllerContext)
`controllerChangeHandler`	A `ZNodeChangeHandler` (for the `KafkaController` and the ControllerEventManager) that listens to change events on `/controller` znode. `controllerChangeHandler` emits controller events as follows: • `ControllerChange` when the znode is created or the znode data changed • `Reelect` when the znode is deleted
`eventManager`	ControllerEventManager (with rateAndTimeMetrics of the ControllerContext, the updateMetrics as the eventProcessedListener` and the maybeResign as the controllerMovedListener) `eventManager` is used to create other internal components to allow them for emitting controller events at state changes: • TopicDeletionManager • ControllerChangeHandler • BrokerChangeHandler • TopicChangeHandler • TopicDeletionHandler • PartitionReassignmentHandler • PreferredReplicaElectionHandler • IsrChangeNotificationHandler

	• LogDirEventNotificationHandler • BrokerModificationsHandlers • PartitionReassignmentIsrChangeHandlers • PartitionModificationsHandlers • PartitionReassignmentIsrChangeHandlers `eventManager` is started when `KafkaController` is requested to start. `eventManager` is closed when `KafkaController` is requested to shutdown.
`kafkaScheduler`	KafkaScheduler with 1 daemon thread with **kafka-scheduler** prefix
`stateChangeLogger`	StateChangeLogger with the broker ID and `inControllerContext` flag enabled
`tokenCleanScheduler`	KafkaScheduler with 1 daemon thread with **delegation-token-cleaner** prefix
`topicDeletionManager`	TopicDeletionManager

StateChangeLogger

`StateChangeLogger` is...FIXME

ControllerContext

`ControllerContext` is the context of an active KafkaController (and is created right when `KafkaController` is created).

`ControllerContext` takes no input arguments to be created.

`allPartitions` Method

```
allPartitions: Set[TopicPartition]
```

`allPartitions` converts the partitionReplicaAssignmentUnderlying into `TopicPartitions`, i.e. `allPartitions` takes the partitions for the topics and simply creates new `TopicPartitions`.

Note	`allPartitions` is used when: • `KafkaController` is requested to updateLeaderAndIsrCache, checkAndTriggerAutoLeaderRebalance, and updateMetrics • `PartitionStateMachine` is requested to initializePartitionState • `ReplicaStateMachine` is requested to initializeReplicaState

`updatePartitionReplicaAssignment` Method

```
updatePartitionReplicaAssignment(topicPartition: TopicPartition, newReplicas: Seq[Int]
): Unit
```

`updatePartitionReplicaAssignment` simply updates the partitionReplicaAssignmentUnderlying registry with `newReplicas` for the topic and the partition (of a given `TopicPartition`).

Note	`updatePartitionReplicaAssignment` is used when: • `KafkaController` is requested to initializeControllerContext, moveReassignedPartitionLeaderIfRequired, updateAssignedReplicasForPartition, and at TopicChange and PartitionModifications controller events • `ReplicaStateMachine` is requested to doHandleStateChanges

`partitionsOnBroker` Method

```
partitionsOnBroker(brokerId: Int): Set[TopicPartition]
```

`partitionsOnBroker` ...FIXME

Note	`partitionsOnBroker` is used when...FIXME

`replicasOnBrokers` Method

```
replicasOnBrokers(
  brokerIds: Set[Int]): Set[PartitionAndReplica]
```

`replicasOnBrokers` ...FIXME

Note	`replicasOnBrokers` is used when...FIXME

`replicasForTopic` Method

```
replicasForTopic(topic: String): Set[PartitionAndReplica]
```

`replicasForTopic` ...FIXME

Note	`replicasForTopic` is used when...FIXME

`partitionsForTopic` Method

```
partitionsForTopic(topic: String): collection.Set[TopicPartition]
```

`partitionsForTopic` ...FIXME

Note	`partitionsForTopic` is used when...FIXME

`removeTopic` Method

```
removeTopic(topic: String): Unit
```

`removeTopic` ...FIXME

Note	`removeTopic` is used when…FIXME

`isReplicaOnline` Method

```
isReplicaOnline(
  brokerId: Int,
  topicPartition: TopicPartition,
  includeShuttingDownBrokers: Boolean = false): Boolean
```

`isReplicaOnline` …FIXME

Note	`isReplicaOnline` is used when…FIXME

`allLiveReplicas` Method

```
allLiveReplicas(): Set[PartitionAndReplica]
```

`allLiveReplicas` …FIXME

Note	`allLiveReplicas` is used when…FIXME

`onlineAndOfflineReplicas` Method

```
onlineAndOfflineReplicas: (Set[PartitionAndReplica], Set[PartitionAndReplica])
```

`onlineAndOfflineReplicas` …FIXME

Note	`onlineAndOfflineReplicas` is used exclusively when `ReplicaStateMachine` is requested to start up.

`addLiveBrokersAndEpochs` Method

```
addLiveBrokersAndEpochs(
  brokerAndEpochs: Map[Broker, Long]): Unit
```

`addLiveBrokersAndEpochs` …FIXME

Note	`addLiveBrokersAndEpochs` is used when…FIXME

`removeLiveBrokers` Method

```
removeLiveBrokers(
  brokerIds: Set[Int]): Unit
```

`removeLiveBrokers` ...FIXME

Note	`removeLiveBrokers` is used when...FIXME

`partitionReplicaAssignment` Method

```
partitionReplicaAssignment(
  topicPartition: TopicPartition): Seq[Int]
```

`partitionReplicaAssignment` finds the brokers with the replicas of the given partition (aka *partition replica assignment*).

Internally, `partitionReplicaAssignment` finds broker IDs of the replicas of the given partition (`TopicPartition`) in the partitionAssignments internal registry.

`partitionReplicaAssignment` returns an empty collection when no topic or partition are found.

Note	`partitionReplicaAssignment` is used when...FIXME

`putReplicaStateIfNotExists` Method

```
putReplicaStateIfNotExists(
  replica: PartitionAndReplica,
  state: ReplicaState): Unit
```

`putReplicaStateIfNotExists` simply adds the replica to the replicaStates internal registry unless available already.

Note	`putReplicaStateIfNotExists` is used exclusively when `ZkReplicaStateMachine` is requested to handle state changes of partition replicas.

`checkValidReplicaStateChange` Method

```
checkValidReplicaStateChange(
  replicas: Seq[PartitionAndReplica],
  targetState: ReplicaState
): (Seq[PartitionAndReplica], Seq[PartitionAndReplica])
```

For every replica (in the given replicas), `checkValidReplicaStateChange` isValidReplicaStateTransition with the target state (`ReplicaState`).

Note	`checkValidReplicaStateChange` is used exclusively when `ZkReplicaStateMachine` is requested to handle replica state changes.

`checkValidPartitionStateChange` Method

```
checkValidPartitionStateChange(
  partitions: Seq[TopicPartition],
  targetState: PartitionState
): (Seq[TopicPartition], Seq[TopicPartition])
```

For every replica (in the given replicas), `checkValidPartitionStateChange` isValidPartitionStateTransition with the target state (`PartitionState`).

Note	`checkValidPartitionStateChange` is used exclusively when `ZkReplicaStateMachine` is requested to handle partition state changes.

`putPartitionStateIfNotExists` Method

```
putPartitionStateIfNotExists(
  partition: TopicPartition,
  state: PartitionState): Unit
```

`putPartitionStateIfNotExists` …FIXME

Note	`putPartitionStateIfNotExists` is used when…FIXME

`isValidReplicaStateTransition` Internal Method

```
isValidReplicaStateTransition(
  replica: PartitionAndReplica,
  targetState: ReplicaState): Boolean
```

`isValidReplicaStateTransition` …FIXME

Note	`isValidReplicaStateTransition` is used when…FIXME

`isValidPartitionStateTransition` Internal Method

```
isValidPartitionStateTransition(
  partition: TopicPartition,
  targetState: PartitionState): Boolean
```

`isValidPartitionStateTransition` …FIXME

Note	`isValidPartitionStateTransition` is used when…FIXME

`putPartitionState` Method

```
putPartitionState(
  partition: TopicPartition,
  targetState: PartitionState): Unit
```

`putPartitionState` …FIXME

Note	`putPartitionState` is used when…FIXME

Finding Partitions by Given States — `partitionsInStates` Method

```
partitionsInStates(
  states: Set[PartitionState]): Set[TopicPartition]
partitionsInStates(
  topic: String, states: Set[PartitionState]): Set[TopicPartition]
```

`partitionsInStates` uses the partitionStates internal registry to find all of the `TopicPartitions` (of the topic if defined) in the given `PartitionStates`.

Note	`partitionsInStates` is used when `PartitionStateMachine` is requested to triggerOnlinePartitionStateChange.

Internal Properties

Name	Description
`allTopics`	
`epoch`	Current controller epoch
`epochZkVersion`	
`liveBrokerEpochs`	(Live and shutting down) Broker IDs and the current epoc `liveBrokerEpochs: Map[Int, Long]`
`liveBrokers`	
`offlinePartitionCount`	
`partitionAssignments`	Broker IDs of partition replicas (`Map[String, Map[Int, Seq`
`partitionLeadershipInfo`	`partitionLeadershipInfo: mutable.Map[TopicPartition, L`
`partitionsBeingReassigned`	
`partitionStates`	
`replicasOnOfflineDirs`	
`replicaStates`	`ReplicaStates` of `PartitionAndReplica` (`Map[PartitionAn`
`shuttingDownBrokerIds`	
`stats`	`ControllerStats` with `UncleanLeaderElectionsPerSec` met for every ControllerState (except Idle state) `stats` is used exclusively to create the ControllerEventM that is then used to collect the times (metrics) of processi (except ShutdownEventThread) • Every `ControllerState` has the RateAndTimeMs me state) The timer metric name pattern is **kafka.controller:type=**
`topicsIneligibleForDeletion`	
`topicsToBeDeleted`	
`topicsWithDeletionStarted`	

ControllerState Contract

`ControllerState` is the contract of the states that KafkaController can be in.

Every `ControllerEvent` has an associated state. When a `ControllerEvent` is processed, it triggers a state transition to the requested state. These state transitions are used for ControllerEventManager and the owning KafkaController.

`ControllerState` has the rateAndTimeMetricName (except Idle state). ControllerContext uses them to build a registry of `KafkaTimers` for every `ControllerState`. The timer metric name pattern is **kafka.controller:type=ControllerStats,name=**.

Figure 1. ControllerStates and RateAndTimeMs Timer Metrics in JConsole

Table 1. ControllerState Contract

Method	Description
`value`	`value: Byte` Internal identifier
`rateAndTimeMetricName`	`rateAndTimeMetricName: Option[String]` `rateAndTimeMetricName` depends on the hasRateAndTimeMetric flag: • **[ControllerState_name]RateAndTimeMs** when enabled • Undefined when disabled
`hasRateAndTimeMetric`	`hasRateAndTimeMetric: Boolean` Controls whether the `ControllerState` has a RateAndTime metric (`true`) or not (`false`) Default: `true`

Table 2. ControllerStates (and Triggering ControllerEvents)

ControllerState	Description
`AutoLeaderBalance`	
`BrokerChange`	rateAndTimeMetricName is **LeaderElectionRateAndTimeMs** ControllerEvents: BrokerChange, BrokerModifications
`ControlledShutdown`	
`ControllerChange`	ControllerEvents: RegisterBrokerAndReelect
`ControllerShutdown`	
`Idle`	The initial state of ControllerEventManager (and indirectly KafkaController) `Idle` has no rateAndTime metric.
`IsrChange`	ControllerEvents: IsrChangeNotification
`LeaderAndIsrResponseReceived`	
`LogDirChange`	ControllerEvents: LogDirEventNotification
`ManualLeaderBalance`	ControllerEvents: PreferredReplicaLeaderElection
`PartitionReassignment`	ControllerEvents: PartitionReassignment, PartitionReassignmentIsrChange
`TopicChange`	ControllerEvents: TopicChange, PartitionModifications
`TopicDeletion`	ControllerEvents: TopicDeletion
`TopicUncleanLeaderElectionEnable`	
`UncleanLeaderElectionEnable`	

Note	`ControllerState` is a Scala sealed abstract class and so all the possible controller states are in a single compilation unit (i.e. a file).

PartitionStateMachine

`PartitionStateMachine` is the base of Partition State Machines that can handleStateChanges, be started up and shut down in the end.

Table 1. PartitionStateMachine Contract (Abstract Methods Only)

Method	Description

`handleStateChanges`	`handleStateChanges(`
	`  partitions: Seq[TopicPartition],`
	`  targetState: PartitionState,`
	`  leaderElectionStrategy: Option[PartitionLeaderElectio`
	`nStrategy]`
	`): Map[TopicPartition, Throwable]`
	
	Handles state changes of partitions (*partition state changes*)
	
	PartitionLeaderElectionStrategy is undefined (`None`) by default
	
	Used when:
	
	• `KafkaController` is requested to onReplicasBecomeOffline, onNewPartitionCreation, onNewPartitionCreation, onPreferredReplicaElection, moveReassignedPartitionLeaderIfRequired, and doControlledShutdown
	• `PartitionStateMachine` is requested to triggerOnlineStateChangeForPartitions
	• `TopicDeletionManager` is requested to onTopicDeletion

Note	ZkPartitionStateMachine is the default and only known implementation of the PartitionStateMachine Contract in Apache Kafka.

`PartitionStateMachine` takes a single ControllerContext to be created.

Note	`PartitionStateMachine` is a Scala abstract class and cannot be created directly. It is created indirectly for the concrete PartitionStateMachines.

PartitionLeaderElectionStrategy

Table 2. PartitionLeaderElectionStrategies

Name	Description
`ControlledShutdownPartitionLeaderElectionStrategy`	
`OfflinePartitionLeaderElectionStrategy`	Accepts `allowUnclean` flag Handled by `ZkPartitionStateMachir` requested to doElectLeaderForPar Used when: • `KafkaController` is requested onNewPartitionCreation (with t `allowUnclean` flag off), onReplicaElection (with the `allowUnclean` flag on for the a client) • `PartitionStateMachine` is requ triggerOnlineStateChangeForF (with the `allowUnclean` flag off
`PreferredReplicaPartitionLeaderElectionStrategy`	`KafkaController` is requested for p replica leader election that in turn tr `ZkPartitionStateMachine` to leaderForPreferredReplica
`ReassignPartitionLeaderElectionStrategy`	

`triggerOnlinePartitionStateChange` Method

```
triggerOnlinePartitionStateChange(): Unit
triggerOnlinePartitionStateChange(
  topic: String): Unit (1)
```

1. Uses the partitions of the given topic only

`triggerOnlinePartitionStateChange` requests the ControllerContext for the partitions in the states: `OfflinePartition` and `NewPartition`.

With the topic specified, `triggerOnlinePartitionStateChange` requests the ControllerContext for the partitions of the topic in the states: `OfflinePartition` and `NewPartition`.

In the end, `triggerOnlinePartitionStateChange` triggers online state change for the partitions.

Note	`triggerOnlinePartitionStateChange` is used when: • `KafkaController` is requested to onBrokerStartup, onReplicasBecomeOffline, processUncleanLeaderElectionEnable, processTopicUncleanLeaderElectionEnable • `PartitionStateMachine` is requested to start up

`triggerOnlineStateChangeForPartitions` Internal Method

```
triggerOnlineStateChangeForPartitions(
  partitions: collection.Set[TopicPartition]): Unit
```

`triggerOnlineStateChangeForPartitions` filters out the partitions of the topics to be deleted and tries to move the partitions to `OnlinePartition` state with OfflinePartitionLeaderElectionStrategy (with `allowUnclean` flag off).

Note	`triggerOnlineStateChangeForPartitions` is used when `PartitionStateMachine` is requested to triggerOnlinePartitionStateChange.

Shutting Down — `shutdown` Method

```
shutdown(): Unit
```

`shutdown` simply prints out the following INFO message to the logs:

```
Stopped partition state machine
```

Note	`shutdown` is used exclusively when is requested to resign as the active controller

Starting Up (On Active Controller) — `startup` Method

```
startup(): Unit
```

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

```
Initializing partition state
```

`startup` initializePartitionState.

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

```
Triggering online partition state changes
```

`startup` triggerOnlinePartitionStateChange.

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

```
Started partition state machine with initial state -> [partitionStates]
```

Note	`startup` is used exclusively when `KafkaController` is requested to onControllerFailover (when a broker is successfully elected as the controller).

`initializePartitionState` Internal Method

```
initializePartitionState(): Unit
```

`initializePartitionState` requests the ControllerContext for all partitions (across all the brokers in the Kafka cluster).

For every `TopicPartition`, `initializePartitionState` requests the ControllerContext for the `LeaderIsrAndControllerEpoch` metadata (using the partitionLeadershipInfo internal registry).

`initializePartitionState` changeStateTo of a `TopicPartition` as follows:

- `OnlinePartition` when the ControllerContext says that the replica is online (for the leader ISR and the `TopicPartition`)
- `OfflinePartition` when the ControllerContext says that the replica is not online (for the leader ISR and the `TopicPartition`)
- `NewPartition` when the ControllerContext has no metadata about the `TopicPartition`

Note	`initializePartitionState` is used exclusively when `PartitionStateMachine` is requested to start up on the active controller.

ZkPartitionStateMachine

`ZkPartitionStateMachine` is a PartitionStateMachine (with a given ControllerContext) for KafkaController (for every KafkaServer).

Figure 1. ZkPartitionStateMachine and KafkaController

When requested to handle partition state changes, `ZkPartitionStateMachine` uses the ControllerBrokerRequestBatch to propagate them to all brokers in a cluster.

`ZkPartitionStateMachine` uses **[PartitionStateMachine controllerId=[brokerId]]** as the logging prefix (aka `logIdent`).

Tip	Enable `ALL` logging levels for `kafka.controller.ZkPartitionStateMachine` logger to see what happens inside. Add the following line to `config/log4j.properties`: `log4j.logger.kafka.controller.ZkPartitionStateMachine=ALL` Refer to Logging.

Creating ZkPartitionStateMachine Instance

`ZkPartitionStateMachine` takes the following to be created:

- KafkaConfig
- StateChangeLogger

- ControllerContext
- KafkaZkClient
- ControllerBrokerRequestBatch

Handling State Changes of Partitions — `handleStateChanges` Method

```
handleStateChanges(
  partitions: Seq[TopicPartition],
  targetState: PartitionState,
  partitionLeaderElectionStrategyOpt: Option[PartitionLeaderElectionStrategy]
): Map[TopicPartition, Throwable]
```

Note	`handleStateChanges` is part of the PartitionStateMachine Contract to handle state changes of partitions (*partition state changes*).

`handleStateChanges` requests the ControllerBrokerRequestBatch to prepare a new batch.

`handleStateChanges` doHandleStateChanges (that may give some errors that are returned in the end).

In the end, `handleStateChanges` requests the ControllerBrokerRequestBatch to send controller requests to brokers and returns the errors.

In case of `ControllerMovedException`, `handleStateChanges` prints out the following ERROR message to the logs:

```
Controller moved to another broker when moving some partitions to [targetState] state
```

In case of any other error (`Throwable`), `handleStateChanges` prints out the following ERROR message to the logs:

```
Error while moving some partitions to [targetState] state
```

`doHandleStateChanges` Internal Method

```
doHandleStateChanges(
  partitions: Seq[TopicPartition],
  targetState: PartitionState,
  partitionLeaderElectionStrategyOpt: Option[PartitionLeaderElectionStrategy]
): Map[TopicPartition, Throwable]
```

For every partition (in `partitions`), `doHandleStateChanges` requests the ControllerContext to putPartitionStateIfNotExists to `NonExistentPartition`.

`doHandleStateChanges` requests the ControllerContext to checkValidPartitionStateChange (that gives valid and invalid partitions).

For every invalid partition, `doHandleStateChanges` logInvalidTransition.

`doHandleStateChanges` branches off per the target state: NewPartition, OnlinePartition, OfflinePartition, and NonExistentPartition.

In the end, `doHandleStateChanges` returns the partitions with election failed (for `OnlinePartition` target state and the valid partitions in `OfflinePartition` or `OnlinePartition` states).

Note	`doHandleStateChanges` is used exclusively when `ZkPartitionStateMachine` is requested to handle partition state changes.

NewPartition

For `NewPartition` target state, `doHandleStateChanges` goes over the valid partitions and for every partition prints out the following TRACE message to the logs and requests the ControllerContext to putPartitionState to `NewPartition` state.

```
Changed partition [partition] state from [state] to NewPartition
with assigned replicas [partitionReplicaAssignment]
```

OnlinePartition (and Partition Leader Election)

For `OnlinePartition` target state, `doHandleStateChanges` splits valid partitions into two collections. One partition set (`uninitializedPartitions`) with partitions in `NewPartition` state while the other with partitions in `OfflinePartition` or `OnlinePartition` states (`partitionsToElectLeader`).

For the partitions in `NewPartition` state (uninitialized valid partitions), `doHandleStateChanges` initializeLeaderAndIsrForPartitions (that gives partitions successfully initialized, aka *successfulInitializations*). For every partition successfully initialized, `doHandleStateChanges` prints out the following TRACE message to the logs and requests the ControllerContext to putPartitionState to `OnlinePartition` state.

```
Changed partition [partition] from [state] to OnlinePartition
with state [leaderAndIsr]
```

For partitions to elect leader (valid partitions in `OfflinePartition` or `OnlinePartition` states), `doHandleStateChanges` electLeaderForPartitions (with the partitions and the `PartitionLeaderElectionStrategy`). That gives two sets of partitions with election successful and failed. For every partition with election successful, `doHandleStateChanges` prints out the following TRACE message to the logs and requests the ControllerContext to putPartitionState to `OnlinePartition` state.

```
Changed partition [partition] from [state] to OnlinePartition
with state [leaderAndIsr]
```

In the end, `doHandleStateChanges` returns the partitions with election failed.

OfflinePartition

For `OfflinePartition` target state, `doHandleStateChanges` goes over the valid partitions and for every partition prints out the following TRACE message to the logs and requests the ControllerContext to putPartitionState to `OfflinePartition` state.

```
Changed partition [partition] state from [state] to OfflinePartition
```

NonExistentPartition

For `NonExistentPartition` target state, `doHandleStateChanges` goes over the valid partitions and for every partition prints out the following TRACE message to the logs and requests the ControllerContext to putPartitionState to `NonExistentPartition` state.

```
Changed partition [partition] state from [state] to NonExistentPartition
```

`initializeLeaderAndIsrForPartitions` Internal Method

```
initializeLeaderAndIsrForPartitions(
  partitions: Seq[TopicPartition]): Seq[TopicPartition]
```

`initializeLeaderAndIsrForPartitions` starts by requesting the ControllerContext for the partition replica assignment for every partition (in the given `partitions`).

From the partition replica assignments, `initializeLeaderAndIsrForPartitions` makes sure that the replicas are all online only (per the ControllerContext) so all other partitions are filtered out (*excluded*).

`initializeLeaderAndIsrForPartitions` splits the partitions (with online replicas only) into two sets with and without replicas (`partitionsWithLiveReplicas` and `partitionsWithoutLiveReplicas` , respectively).

For every partition without live (online) replicas, `initializeLeaderAndIsrForPartitions` prints out the following ERROR message and the StateChangeFailedException to the logs:

```
Controller [controllerId] epoch [epoch] failed to change state
for partition [partition] from NewPartition to OnlinePartition
```

```
Controller [controllerId] epoch [epoch] encountered error during
state change of partition [partition] from New to Online,
assigned replicas are [[replicas]], live brokers are
[[liveBrokerIds]]. No assigned replica is alive.
```

`initializeLeaderAndIsrForPartitions` converts the partitions with live (online) replicas into `leaderIsrAndControllerEpochs` (`LeaderIsrAndControllerEpoch` with `LeaderAndIsr`) and for every pair `initializeLeaderAndIsrForPartitions` requests the KafkaZkClient to create state znodes for the partitions.

For every successful response (from creating partition state znodes), `initializeLeaderAndIsrForPartitions` requests the following:

1. The ControllerContext to record the `leaderIsrAndControllerEpoch` for the partition (in the partitionLeadershipInfo registry)
2. The ControllerBrokerRequestBatch to addLeaderAndIsrRequestForBrokers (with `isNew` flag on)

In the end, `initializeLeaderAndIsrForPartitions` returns the partitions that were successfully initialized.

In case of `ControllerMovedException` (while…FIXME), `initializeLeaderAndIsrForPartitions` …FIXME

In case of any other error (`Exception`) (while…FIXME), `initializeLeaderAndIsrForPartitions` …FIXME

Note	`initializeLeaderAndIsrForPartitions` is used exclusively when `ZkPartitionStateMachine` is requested to handle partition state changes (for partitions in NewPartition state that are transitioned to OnlinePartition target state).

`electLeaderForPartitions` Internal Method

```
electLeaderForPartitions(
  partitions: Seq[TopicPartition],
  partitionLeaderElectionStrategy: PartitionLeaderElectionStrategy
): (Seq[TopicPartition], Map[TopicPartition, Throwable])
```

`electLeaderForPartitions` simply doElectLeaderForPartitions until all the given partitions have partition leaders elected successfully or not.

For every failed election, `electLeaderForPartitions` prints out the following ERROR message (with an exception) to the logs:

```
Controller [controllerId] epoch [epoch] failed to change state
for partition [partition] from [state] to OnlinePartition
```

In the end, `electLeaderForPartitions` returns the partitions with leader election successful and failed.

Note	`electLeaderForPartitions` is used when `ZkPartitionStateMachine` is requested to handle partition state changes (when the partitions are expected in `OnlinePartition` target state).

`doElectLeaderForPartitions` Internal Method

```
doElectLeaderForPartitions(
  partitions: Seq[TopicPartition],
  partitionLeaderElectionStrategy: PartitionLeaderElectionStrategy
): (Seq[TopicPartition], Seq[TopicPartition], Map[TopicPartition, Exception])
```

`doElectLeaderForPartitions` requests the KafkaZkClient for the partition states and converts them to `LeaderIsrAndControllerEpoch` per partition.

`doElectLeaderForPartitions` collects the partitions that are eligible for partition election (that have controller epoch older than the current controller epoch per the ControllerContext).

`doElectLeaderForPartitions` branches off per the PartitionLeaderElectionStrategy and ends up with partitions with and without leaders:

- For OfflinePartitionLeaderElectionStrategy, `doElectLeaderForPartitions` first collectUncleanLeaderElectionState with the valid partitions for election followed by leaderForOffline.
- For ReassignPartitionLeaderElectionStrategy, `doElectLeaderForPartitions` leaderForReassign
- For PreferredReplicaPartitionLeaderElectionStrategy, `doElectLeaderForPartitions` leaderForPreferredReplica
- For ControlledShutdownPartitionLeaderElectionStrategy, `doElectLeaderForPartitions` leaderForControlledShutdown

`doElectLeaderForPartitions` requests the KafkaZkClient to updateLeaderAndIsr (with the adjusted leader and ISRs).

For every successfully-updated partition (in Zookeeper), `doElectLeaderForPartitions` requests the following:

1. The ControllerContext to record the `leaderIsrAndControllerEpoch` for the partition (in the partitionLeadershipInfo registry)
2. The ControllerBrokerRequestBatch to addLeaderAndIsrRequestForBrokers to every live replica broker (with `isNew` flag off)

In the end, `doElectLeaderForPartitions` returns the partitions that were successfully updated, to be updated again and failed (during election and update).

Note	`doElectLeaderForPartitions` is used when `ZkPartitionStateMachine` is requested to electLeaderForPartitions.

`collectUncleanLeaderElectionState` Internal Method

```
collectUncleanLeaderElectionState(
  leaderIsrAndControllerEpochs: Seq[(TopicPartition, LeaderIsrAndControllerEpoch)]
): Seq[(TopicPartition, Option[LeaderIsrAndControllerEpoch], Boolean)]
```

`collectUncleanLeaderElectionState` ...FIXME

Note	`collectUncleanLeaderElectionState` is used when `ZkPartitionStateMachine` is requested to doElectLeaderForPartitions (for OfflinePartitionLeaderElectionStrategy).

`logInvalidTransition` Internal Method

```
logInvalidTransition(
  partition: TopicPartition,
  targetState: PartitionState): Unit
```

`logInvalidTransition` ...FIXME

Note	`logInvalidTransition` is used exclusively when `ZkPartitionStateMachine` is requested to doHandleStateChanges (for invalid partitions).

Printing Out ERROR Message to Logs — `logFailedStateChange` Internal Method

```
logFailedStateChange(
  partition: TopicPartition,
  currState: PartitionState,
  targetState: PartitionState,
  code: Code): Unit (1)
logFailedStateChange(
  partition: TopicPartition,
  currState: PartitionState,
  targetState: PartitionState,
  t: Throwable): Unit
```

1. Converts the code to a `KeeperException`

`logFailedStateChange` simply prints out the following ERROR message to the logs:

```
Controller [controllerId] epoch [epoch] failed to change state
for partition [partition] from [currState] to [targetState]
```

Note	`logFailedStateChange` is used when `ZkPartitionStateMachine` is requested to initializeLeaderAndIsrForPartitions, electLeaderForPartitions, collectUncleanLeaderElectionState, and logInvalidTransition.

`partitionState` Internal Method

```
partitionState(
  partition: TopicPartition): PartitionState
```

`partitionState` ...FIXME

Note	`partitionState` is used when...FIXME

`leaderForOffline` Method

```
leaderForOffline(
  controllerContext: ControllerContext,
  partitionsWithUncleanLeaderElectionState: Seq[(TopicPartition, Option[LeaderIsrAndCo
ntrollerEpoch], Boolean)]
): Seq[ElectionResult]
// Private API
leaderForOffline(
  partition: TopicPartition,
  leaderIsrAndControllerEpochOpt: Option[LeaderIsrAndControllerEpoch],
  uncleanLeaderElectionEnabled: Boolean,
  controllerContext: ControllerContext
): ElectionResult
```

`leaderForOffline` ...FIXME

Note	`leaderForOffline` is used when...FIXME

`leaderForReassign` Method

```
leaderForReassign(
  controllerContext: ControllerContext,
  leaderIsrAndControllerEpochs: Seq[(TopicPartition, LeaderIsrAndControllerEpoch)]
): Seq[ElectionResult]
// Private API
leaderForReassign(
  partition: TopicPartition,
  leaderIsrAndControllerEpoch: LeaderIsrAndControllerEpoch,
  controllerContext: ControllerContext
): ElectionResult
```

`leaderForReassign` ...FIXME

Note	`leaderForReassign` is used when...FIXME

`leaderForPreferredReplica` Method

```
leaderForPreferredReplica(
  controllerContext: ControllerContext,
  leaderIsrAndControllerEpochs: Seq[(TopicPartition, LeaderIsrAndControllerEpoch)]
): Seq[ElectionResult]
// Private API
leaderForPreferredReplica(
  partition: TopicPartition,
  leaderIsrAndControllerEpoch: LeaderIsrAndControllerEpoch,
  controllerContext: ControllerContext
): ElectionResult
```

`leaderForPreferredReplica` ...FIXME

Note	`leaderForPreferredReplica` is used when...FIXME

`leaderForControlledShutdown` Method

```
leaderForControlledShutdown(
  controllerContext: ControllerContext,
  leaderIsrAndControllerEpochs: Seq[(TopicPartition, LeaderIsrAndControllerEpoch)]
): Seq[ElectionResult]
// Private API
leaderForControlledShutdown(
  partition: TopicPartition,
  leaderIsrAndControllerEpoch: LeaderIsrAndControllerEpoch,
  shuttingDownBrokerIds: Set[Int],
  controllerContext: ControllerContext
): ElectionResult
```

`leaderForControlledShutdown` ...FIXME

Note	`leaderForControlledShutdown` is used when...FIXME

Election Utility

`Election` is a utility with the algorithms for partition leader election:

- leaderForControlledShutdown
- leaderForOffline
- leaderForPreferredReplica
- leaderForReassign

`leaderForControlledShutdown` Utility

```
leaderForControlledShutdown(
  controllerContext: ControllerContext,
  leaderAndIsrs: Seq[(TopicPartition, LeaderAndIsr)]): Seq[ElectionResult]
```

`leaderForControlledShutdown` ...FIXME

Note	`leaderForControlledShutdown` is used when `ZkPartitionStateMachine` is requested to doElectLeaderForPartitions for ControlledShutdownPartitionLeaderElectionStrategy.

`leaderForControlledShutdown` Internal Method

```
leaderForControlledShutdown(
  partition: TopicPartition,
  leaderAndIsr: LeaderAndIsr,
  shuttingDownBrokerIds: Set[Int],
  controllerContext: ControllerContext): ElectionResult
```

`leaderForControlledShutdown` ...FIXME

Note	`leaderForControlledShutdown` is used when `Election` utility is used to leaderForControlledShutdown.

`leaderForOffline` Utility

```
leaderForOffline(
  controllerContext: ControllerContext,
  partitionsWithUncleanLeaderElectionState: Seq[(TopicPartition, Option[LeaderAndIsr],
Boolean)]): Seq[ElectionResult]
```

`leaderForOffline` ...FIXME

Note	`leaderForOffline` is used when `ZkPartitionStateMachine` is requested to doElectLeaderForPartitions for OfflinePartitionLeaderElectionStrategy.

`leaderForOffline` Internal Method

```
leaderForOffline(
  partition: TopicPartition,
  leaderAndIsrOpt: Option[LeaderAndIsr],
  uncleanLeaderElectionEnabled: Boolean,
  controllerContext: ControllerContext): ElectionResult
```

`leaderForOffline` ...FIXME

Note	`leaderForOffline` is used when `Election` utility is used to leaderForOffline.

`leaderForPreferredReplica` Utility

```
leaderForPreferredReplica(
  controllerContext: ControllerContext,
  leaderAndIsrs: Seq[(TopicPartition, LeaderAndIsr)]): Seq[ElectionResult]
```

`leaderForPreferredReplica` ...FIXME

Note	`leaderForPreferredReplica` is used when `ZkPartitionStateMachine` is requested to doElectLeaderForPartitions for PreferredReplicaPartitionLeaderElectionStrategy.

`leaderForPreferredReplica` Internal Method

```
leaderForPreferredReplica(
  partition: TopicPartition,
  leaderAndIsr: LeaderAndIsr,
  controllerContext: ControllerContext): ElectionResult
```

`leaderForPreferredReplica` ...FIXME

Note	`leaderForPreferredReplica` is used when `Election` utility is used to leaderForPreferredReplica.

`leaderForReassign` Utility

```
leaderForReassign(
  controllerContext: ControllerContext,
  leaderAndIsrs: Seq[(TopicPartition, LeaderAndIsr)]): Seq[ElectionResult]
```

`leaderForReassign` ...FIXME

Note	`leaderForReassign` is used when `ZkPartitionStateMachine` is requested to doElectLeaderForPartitions for ReassignPartitionLeaderElectionStrategy.

`leaderForReassign` Internal Method

```
leaderForReassign(
  partition: TopicPartition,
  leaderAndIsr: LeaderAndIsr,
  controllerContext: ControllerContext): ElectionResult
```

`leaderForReassign` ...FIXME

Note	`leaderForReassign` is used when `Election` utility is used to leaderForReassign.

PartitionLeaderElectionAlgorithms Utility

`PartitionLeaderElectionAlgorithms` is a utility with the algorithms for partition leader election:

- controlledShutdownPartitionLeaderElection
- offlinePartitionLeaderElection
- preferredReplicaPartitionLeaderElection
- reassignPartitionLeaderElection

`controlledShutdownPartitionLeaderElection` Utility

```
controlledShutdownPartitionLeaderElection(
  assignment: Seq[Int],
  isr: Seq[Int],
  liveReplicas: Set[Int],
  shuttingDownBrokers: Set[Int]): Option[Int]
```

`controlledShutdownPartitionLeaderElection` ...FIXME

Note	`controlledShutdownPartitionLeaderElection` is used when `Election` utility is used to leaderForControlledShutdown.

`offlinePartitionLeaderElection` Utility

```
offlinePartitionLeaderElection(
  assignment: Seq[Int],
  isr: Seq[Int],
  liveReplicas: Set[Int],
  uncleanLeaderElectionEnabled: Boolean,
  controllerContext: ControllerContext): Option[Int]
```

`offlinePartitionLeaderElection` ...FIXME

Note	`offlinePartitionLeaderElection` is used when `Election` utility is used to leaderForOffline.

`preferredReplicaPartitionLeaderElection` Utility

```
preferredReplicaPartitionLeaderElection(
  assignment: Seq[Int],
  isr: Seq[Int],
  liveReplicas: Set[Int]): Option[Int]
```

`preferredReplicaPartitionLeaderElection` ...FIXME

Note	`preferredReplicaPartitionLeaderElection` is used when `Election` utility is used to leaderForPreferredReplica.

`reassignPartitionLeaderElection` Utility

```
reassignPartitionLeaderElection(
  reassignment: Seq[Int],
  isr: Seq[Int],
  liveReplicas: Set[Int]): Option[Int]
```

`reassignPartitionLeaderElection` ...FIXME

Note	`reassignPartitionLeaderElection` is used when `Election` utility is used to leaderForReassign.

ReplicaStateMachine

`ReplicaStateMachine` is the base of Replica State Machines that can handleStateChanges, be started up and shut down in the end.

Table 1. ReplicaStateMachine Contract (Abstract Methods Only)

Method	Description
`handleStateChanges`	`handleStateChanges(replicas: Seq[PartitionAndReplica], targetState: ReplicaState): Unit` Handles state changes of replicas (*replica state changes*) Used when: • `KafkaController` is requested to onBrokerLogDirFailure, onBrokerStartup, onReplicasBecomeOffline, onNewPartitionCreation, onPartitionReassignment, stopOldReplicasOfReassignedPartition, startNewReplicasForReassignedPartition, and doControlledShutdown • `ReplicaStateMachine` is requested to start up (when `KafkaController` is requested to onControllerFailover on a broker elected as the controller) • `TopicDeletionManager` is requested to failReplicaDeletion, completeReplicaDeletion, retryDeletionForIneligibleReplicas, completeDeleteTopic, and startReplicaDeletion

Note	ZkReplicaStateMachine is the default and only known implementation of the ReplicaStateMachine Contract in Apache Kafka.

`ReplicaStateMachine` takes a single ControllerContext to be created.

Note	`ReplicaStateMachine` is a Scala abstract class and cannot be created directly. It is created indirectly for the concrete ReplicaStateMachines.

Shutting Down — `shutdown` Method

```
shutdown(): Unit
```

`shutdown` simply prints out the following INFO message to the logs:

```
Stopped replica state machine
```

Note	`shutdown` is used exclusively when `KafkaController` is requested to resign as the active controller.

Starting Up (On Active Controller) — `startup` Method

```
startup(): Unit
```

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

```
Initializing replica state
```

`startup` initializeReplicaState.

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

```
Triggering online replica state changes
```

`startup` requests the ControllerContext for online and offline replicas.

`startup` handleStateChanges (with the online replicas and `OnlineReplica` target state).

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

```
Triggering offline replica state changes
```

`startup` handleStateChanges (with the offline replicas and `OfflineReplica` target state).

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

```
Started replica state machine with initial state -> [replicaState]
```

Note	`startup` is used exclusively when `KafkaController` is requested to onControllerFailover (when a broker is successfully elected as the controller).

Initializing Partition Replica States using ControllerContext — `initializeReplicaState` Internal Method

```
initializeReplicaState(): Unit
```

`initializeReplicaState` requests the ControllerContext for all partitions.

For every partition, `initializeReplicaState` requests the ControllerContext for the partition replica assignment (the broker IDs of partition replicas).

For every partition replica, `initializeReplicaState` requests the ControllerContext to putReplicaState to `OnlineReplica` or `ReplicaDeletionIneligible` per isReplicaOnline.

Note	`initializeReplicaState` is used exclusively when `ReplicaStateMachine` is requested to start up.

ZkReplicaStateMachine

`ZkReplicaStateMachine` is a concrete ReplicaStateMachine to handle changes of the state of partition replicas (*replica state changes*).

`ZkReplicaStateMachine` uses ControllerBrokerRequestBatch to propagate the changes to all brokers in a cluster.

`ZkReplicaStateMachine` is created exclusively for KafkaController.

`ZkReplicaStateMachine` uses **[ReplicaStateMachine controllerId=[brokerId]]** as the logging prefix (aka `logIdent`).

Tip	Enable `ALL` logging levels for `kafka.controller.ZkReplicaStateMachine` logger to see what happens inside. Add the following line to `config/log4j.properties`: `log4j.logger.kafka.controller.ZkReplicaStateMachine=ALL` Refer to Logging.

Creating ZkReplicaStateMachine Instance

`ZkReplicaStateMachine` takes the following to be created:

- KafkaConfig
- StateChangeLogger
- ControllerContext
- KafkaZkClient
- ControllerBrokerRequestBatch

Handling State Changes of Replicas — `handleStateChanges` Method

```
handleStateChanges(
  replicas: Seq[PartitionAndReplica],
  targetState: ReplicaState): Unit
```

Note	`handleStateChanges` is part of the ReplicaStateMachine Contract to handle state changes of replicas (*replica state changes*).

`handleStateChanges` requests the ControllerBrokerRequestBatch to prepare a new batch.

`handleStateChanges` groups the replicas (`PartitionAndReplicas`) by ID and for every group doHandleStateChanges.

In the end, `handleStateChanges` requests the ControllerBrokerRequestBatch to send controller requests to brokers.

In case of a `ControllerMovedException` , `handleStateChanges` prints out the following ERROR message to the logs and re-throws the exception.

```
Controller moved to another broker when moving some replicas to [targetState] state
```

In case of any other error (`Throwable`), `handleStateChanges` simply prints out the following ERROR message to the logs:

```
Error while moving some replicas to [targetState] state
```

`handleStateChanges` does nothing and returns when there are no `PartitionAndReplica` to work on (i.e. the given replicas is empty).

`doHandleStateChanges` Internal Method

```
doHandleStateChanges(
  replicaId: Int,
  replicas: Seq[PartitionAndReplica],
  targetState: ReplicaState): Unit
```

For every replica (in the `replicas`), `doHandleStateChanges` requests the ControllerContext to putReplicaStateIfNotExists.

`doHandleStateChanges` requests the ControllerContext to checkValidReplicaStateChange (that gives valid and invalid replicas).

For every invalid replica, `doHandleStateChanges` logInvalidTransition.

`doHandleStateChanges` branches off per the target state (`ReplicaState`): OfflineReplica,… FIXME

Note	`doHandleStateChanges` is used exclusively when `ZkReplicaStateMachine` is requested to handle replica state changes.

OfflineReplica

For `OfflineReplica` target state, `doHandleStateChanges` ...FIXME

`doRemoveReplicasFromIsr` Internal Method

```
doRemoveReplicasFromIsr(
  replicaId: Int,
  partitions: Seq[TopicPartition]):
(Map[TopicPartition, LeaderIsrAndControllerEpoch],
 Seq[TopicPartition],
 Map[TopicPartition, Exception])
```

`doRemoveReplicasFromIsr` ...FIXME

Note	`doRemoveReplicasFromIsr` is used exclusively when `ZkReplicaStateMachine` is requested to removeReplicasFromIsr.

`getTopicPartitionStatesFromZk` Internal Method

```
getTopicPartitionStatesFromZk(partitions: Seq[TopicPartition]):
  (Map[TopicPartition, LeaderAndIsr],
   Seq[TopicPartition],
   Map[TopicPartition, Exception])
```

`getTopicPartitionStatesFromZk` ...FIXME

Note	`getTopicPartitionStatesFromZk` is used exclusively when `ZkReplicaStateMachine` is requested to doRemoveReplicasFromIsr.

`logFailedStateChange` Internal Method

```
logFailedStateChange(
  replica: PartitionAndReplica,
  currState: ReplicaState,
  targetState: ReplicaState,
  t: Throwable): Unit
```

`logFailedStateChange` ...FIXME

Note	`logFailedStateChange` is used when `ZkReplicaStateMachine` is requested to… FIXME

`logInvalidTransition` Internal Method

```
logInvalidTransition(
  replica: PartitionAndReplica,
  targetState: ReplicaState): Unit
```

`logInvalidTransition` …FIXME

Note	`logInvalidTransition` is used exclusively when `ZkReplicaStateMachine` is requested to doHandleStateChanges.

`logSuccessfulTransition` Internal Method

```
logSuccessfulTransition(
  replicaId: Int,
  partition: TopicPartition,
  currState: ReplicaState,
  targetState: ReplicaState): Unit
```

`logSuccessfulTransition` …FIXME

Note	`logSuccessfulTransition` is used when `ZkReplicaStateMachine` is requested to…FIXME

`removeReplicasFromIsr` Internal Method

```
removeReplicasFromIsr(
  replicaId: Int,
  partitions: Seq[TopicPartition]):
Map[TopicPartition, LeaderIsrAndControllerEpoch]
```

`removeReplicasFromIsr` …FIXME

Note	`removeReplicasFromIsr` is used exclusively when `ZkReplicaStateMachine` is requested to doHandleStateChanges.

ControllerEventManager

`ControllerEventManager` manages the event queue (of QueuedEvents) and the controller-event-thread for event processing.

`ControllerEventManager` allows for emitting ControllerEvents (with optional preempting the queue first).

`ControllerEventManager` is created exclusively for the KafkaController (when started up).

When started, `ControllerEventManager` simply requests the ControllerEventThread to start processing ControllerEvents.

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

Figure 1. ControllerEventManager is Created and Started With KafkaController

`ControllerEventManager` is in one of the possible ControllerStates.

Note	The state of `ControllerEventManager` is exactly the state of the owning KafkaController.

Creating ControllerEventManager Instance

`ControllerEventManager` takes the following to be created:

- Controller ID (i.e. the broker.id of the broker)
- ControllerEventProcessor
- `Time`
- `KafkaTimers` per ControllerState (`Map[ControllerState, KafkaTimer]`)

`ControllerEventManager` initializes the internal properties.

Performance Metrics

`ControllerEventManager` is a KafkaMetricsGroup with the following performance metrics.

Table 1. ControllerEventManager's Performance Metrics

Metric Name	Description
`EventQueueTimeMs`	
`EventQueueSize`	Number of QueuedEvents in the event queue

The performance metrics are registered in **kafka.controller:type=ControllerEventManager** group.

Figure 2. Partition in jconsole

Emitting (Enqueuing) Controller Event — `put` Method

```
put(event: ControllerEvent): QueuedEvent
```

`put` creates a new QueuedEvent with the ControllerEvent and adds it at the end of the event queue.

Note	`put` is used when: • `ControllerBrokerRequestBatch` is requested to sendEvent • `ControllerEventManager` is requested to clearAndPut • `KafkaController` is requested to do its operation and emits various events

Starting ControllerEventManager (and ControllerEventThread) — `start` Method

```
start(): Unit
```

`start` simply requests the ControllerEventThread to start processing ControllerEvents.

Note	`ControllerEventThread` is a `ShutdownableThread` that triggers `doWork()` method when started.

Note	`start` is used exclusively when `KafkaController` is requested to start up (when `KafkaServer` is requested to start).

`clearAndPut` Method

```
clearAndPut(
  event: ControllerEvent): QueuedEvent
```

`clearAndPut` ...FIXME

Note	`clearAndPut` is used when: • `ControllerEventManager` is requested to close (and emit a `ShutdownEventThread` event) • `KafkaController` is requested to startup (and registers a `StateChangeHandler` that emits an `Expire` event when `beforeInitializingSession`)

Closing Up — `close` Method

```
close(): Unit
```

`close` simply clears the event queue and emits a `ShutdownEventThread` event.

In the end, `close` waits until the shutdown is complete. You should see the following INFO message in the logs:

```
Shutdown completed
```

Note	`close` is used exclusively when `KafkaController` is requested to shutdown.

Internal Properties

Name	Description
`queue`	Event queue of QueuedEvents (`LinkedBlockingQueue[QueuedEvent]`) that is a Java java.util.concurrent.LinkedBlockingQueue (i.e. an optionally-bounded blocking queue based on linked nodes that orders elements in first-in-first-out fashion)
`_state`	ControllerState: • Idle when `ControllerEventManager` is created and right after `ControllerEventThread` has finished processing a controller event • State transitions happen per the requested state of the `ControllerEvent` being processed (while `ControllerEventThread` is processing controller events)
`thread`	ControllerEventThread with **controller-event-thread** thread name

ControllerEventThread

`ControllerEventThread` is a ShutdownableThread that is created for ControllerEventManager (with the name being **controller-event-thread**).

```
// jstack [brokerPid]
"controller-event-thread" #42 prio=5 os_prio=31 cpu=387,10ms elapsed=82679,68s tid=0x0
0007f920e489800 nid=0x14703 waiting on condition  [0x000070000fcea000]
   java.lang.Thread.State: WAITING (parking)
        at jdk.internal.misc.Unsafe.park(java.base@12.0.2/Native Method)
        - parking to wait for  <0x00000007c07f0298> (a java.util.concurrent.locks.Abst
ractQueuedSynchronizer$ConditionObject)
        at java.util.concurrent.locks.LockSupport.park(java.base@12.0.2/LockSupport.ja
va:194)
        at java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.await
(java.base@12.0.2/AbstractQueuedSynchronizer.java:2081)
        at java.util.concurrent.LinkedBlockingQueue.take(java.base@12.0.2/LinkedBlocki
ngQueue.java:433)
        at kafka.controller.ControllerEventManager$ControllerEventThread.doWork(Contro
llerEventManager.scala:127)
        at kafka.utils.ShutdownableThread.run(ShutdownableThread.scala:89)
```

`ControllerEventThread` is started with ControllerEventManager.

Figure 1. ControllerEventThread is Started Alongside ControllerEventManager

`ControllerEventThread` takes the name of the thread to be created.

`ControllerEventThread` uses **[ControllerEventThread controllerId=[controllerId]]** as the logging prefix (aka `logIdent`).

Processing Controller Events — `doWork` Method

```
doWork(): Unit
```

`doWork` takes (and removes) a QueuedEvent from the head of the event queue (waiting for a `QueuedEvent` to be available if the queue is empty).

Note	The very first event in the event queue is `Startup` that `KafkaController` puts when it is started.

`doWork` sets _state (of `ControllerEventManager`) to be the state of the controller event.

`doWork` requests the EventQueueTimeMs histogram to `update` (with the time between the current time and the enqueue time of the event).

`doWork` finds the `KafkaTimer` for the current controller state (in rateAndTimeMetrics lookup table) to measure and record the time to requests the event to process using the ControllerEventProcessor.

In the end, `doWork` sets the _state (of `ControllerEventManager`) as `Idle`.

In case of any error (`Throwable`), `doWork` simply prints out the following ERROR message to the logs:

```
Uncaught error processing event [controllerEvent]
```

QueuedEvent — ControllerEvent with Enqueue Time

`QueuedEvent` is a ControllerEvent with the time it was enqueued (*enqueue time*) to ControllerEventManager.

`QueuedEvent` is created every time `ControllerEventManager` is requested to enqueue a controller event.

Processing ControllerEvent Using ControllerEventProcessor — `process` Method

```
process(
  processor: ControllerEventProcessor): Unit
```

`process` atomically turns the spent internal registry on and simply exits when it was on already.

`process` decrements the count of the processingStarted internal latch.

In the end, `process` simply requests the given ControllerEventProcessor to process the controller event.

Note	`process` is used exclusively when `ControllerEventThread` is requested to process controller events.

Preempting ControllerEvent Using ControllerEventProcessor — `preempt` Method

```
preempt(
  processor: ControllerEventProcessor): Unit
```

`preempt` atomically turns the spent internal registry on and simply exits when it was on already.

In the end, `preempt` simply requests the given ControllerEventProcessor to preempt the controller event.

Note	`preempt` is used exclusively when `ControllerEventManager` is requested to clearAndPut.

`awaitProcessing` Method

```
awaitProcessing(): Unit
```

`awaitProcessing` simply pauses the current thread until the processingStarted internal latch has counted down to zero (which happens only after processing the ControllerEvent using a ControllerEventProcessor).

Note	`awaitProcessing` is used exclusively when `KafkaController` is requested to start up.

String (Textual) Representation — `toString` Method

```
toString: String
```

Note	`toString` is part of the java.lang.Object Contract for a string representation of the object.

`toString` returns the following string representation:

```
QueuedEvent(event=[event], enqueueTimeMs=[enqueueTimeMs])
```

Internal Properties

Name	Description
`processingStarted`	Java's CountDownLatch (with the count of `1`) to wait until…FIXME Used when…FIXME
`spent`	Java's AtomicBoolean (with the initial value of `false`) to be updated atomically…FIXME Used when…FIXME

ControllerEventProcessor

`ControllerEventProcessor` is the abstraction of processors that can process and preempt controller events.

Table 1. ControllerEventProcessor Contract

Method	Description

`preempt`	`preempt(`
	`  event: ControllerEvent): Unit`
	
	Preempts a controller event
	
	Used exclusively when `QueuedEvent` is requested to preempt a ControllerEventProcessor

`process`	`process(`
	`  event: ControllerEvent): Unit`
	
	Processes a controller event
	
	Used exclusively when `QueuedEvent` is requested to process a ControllerEventProcessor

Note	KafkaController is the default and only known implementation of the ControllerEventProcessor Contract in Apache Kafka.

ControllerEvent

`ControllerEvent` is the contract of the events in the lifecycle of KafkaController (state machine) that trigger state change.

`ControllerEvent` events are managed by ControllerEventManager.

Table 1. ControllerEvent Contract

Method	Description
`state`	`state: ControllerState` Requested ControllerState of the ControllerEventManager (while processing a controller event)

Table 2. ControllerEvents

ControllerEvent	ControllerState	
AutoPreferredReplicaLeaderElection	AutoLeaderBalance	
BrokerChange	BrokerChange	
BrokerModifications	BrokerChange	
ControlledShutdown	ControlledShutdown	
ControllerChange	ControllerChange	
Expire	ControllerChange	
IsrChangeNotification	IsrChange	Er ZI be ch
LeaderAndIsrResponseReceived	LeaderAndIsrResponseReceived	
LogDirEventNotification	LogDirChange	
PartitionModifications	TopicChange	Er ZI no
PartitionReassignment	PartitionReassignment	

PartitionReassignmentIsrChange	PartitionReassignment	
PreferredReplicaLeaderElection	ManualLeaderBalance	
Reelect	ControllerChange	
RegisterBrokerAndReelect	ControllerChange	
ShutdownEventThread	ControllerShutdown	
Startup	ControllerChange	
TopicChange	TopicChange	
TopicDeletion	TopicDeletion	
TopicDeletionStopReplicaResponseReceived	TopicDeletion	
TopicUncleanLeaderElectionEnable	TopicUncleanLeaderElectionEnable	
UncleanLeaderElectionEnable	UncleanLeaderElectionEnable	Er re er

Note	`ControllerEvent` is a Scala sealed trait and so all the possible controller events are in a single compilation unit (i.e. a file).

AutoPreferredReplicaLeaderElection Controller Event

`AutoPreferredReplicaLeaderElection` is a controller event that transition the KafkaController to `AutoLeaderBalance` state.

`AutoPreferredReplicaLeaderElection` is enqueued (to the ControllerEventManager) exclusively from the auto-leader-rebalance-task.

When processed on a controller broker, `AutoPreferredReplicaLeaderElection` event checkAndTriggerAutoLeaderRebalance and in the end scheduleAutoLeaderRebalanceTask with the delay as configured using the leader.imbalance.check.interval.seconds property.

Note	`AutoPreferredReplicaLeaderElection` event is ignored (skipped) when processed on any broker but controller broker.

BrokerChange Controller Event

`BrokerChange` is a controller event that transition the KafkaController to ControllerChange state.

`BrokerChange` is enqueued exclusively when BrokerChangeHandler is notified about a new or updated node under `/brokers/ids` path in Zookeeper (`handleChildChange`).

When processed (on the controller-event-thread), `KafkaController` is requested to processBrokerChange (that in turn starts controller election).

LeaderAndIsrResponseReceived Controller Event

`LeaderAndIsrResponseReceived` is a controller event that transition the KafkaController to LeaderAndIsrResponseReceived state.

`LeaderAndIsrResponseReceived` takes the following to be created:

- `AbstractResponse` (which is assumed to be `LeaderAndIsrResponse`)
- Broker ID

`LeaderAndIsrResponseReceived` is enqueued exclusively when `AbstractControllerBrokerRequestBatch` (ControllerBrokerRequestBatch actually) is requested to sendLeaderAndIsrRequest (and a response has arrived).

Processing LeaderAndIsrResponseReceived

When processed (on the controller-event-thread), `KafkaController` is requested to processLeaderAndIsrResponseReceived.

PreferredReplicaLeaderElection Controller Event

`PreferredReplicaLeaderElection` is a controller event that transition the KafkaController to ManualLeaderBalance state.

`PreferredReplicaLeaderElection` takes the following to be created:

- Partitions (`Option[Set[TopicPartition]]`)
- Election type (default: ZkTriggered)
- Elect Preferred Leaders Callback (default: `(_, _) ⇒ {}`)

`PreferredReplicaLeaderElection` is enqueued when `KafkaController` is requested for the following:

- removePartitionsFromPreferredReplicaElection and `isTriggeredByAutoRebalance` flag is off
- Preferred Replica Leader Election

Note	FIXME When removePartitionsFromPreferredReplicaElection and `isTriggeredByAutoRebalance` flag is off?

When created, `PreferredReplicaLeaderElection` is given the optional partitions and the electionType as follows:

- No partitions (`None`) and the default `ZkTriggered` when:
  - `KafkaController` is requested to removePartitionsFromPreferredReplicaElection and `isTriggeredByAutoRebalance` flag is off
  - PreferredReplicaElectionHandler is notified about a new node under `/admin/preferred_replica_election` path in Zookeeper (`handleCreation`).
- Some partitions, `AdminClientTriggered` election type and a custom callback when `KafkaController` is requested to electPreferredLeaders

Processing PreferredReplicaLeaderElection

When processed (on the controller-event-thread), `KafkaController` is requested to processPreferredReplicaLeaderElection.

Preempting PreferredReplicaLeaderElection

When preempted, `KafkaController` is requested to preemptPreferredReplicaLeaderElection.

Election Type

Election type can be one of the following:

- `AutoTriggered`
- `ZkTriggered`
- `AdminClientTriggered`

Reelect

`Reelect` is…FIXME

Startup Controller Event

`Startup` is a controller event that transition the KafkaController to ControllerChange state.

`Startup` is enqueued when `KafkaController` is requested to start up.

Processing Startup

When processed (on the controller-event-thread), `KafkaController` is requested to processStartup (that in turn starts controller election).

TopicDeletion Controller Event

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

Note	`TopicDeletion` uses delete.topic.enable Kafka property.

Note	Topics to be deleted are created in `/admin/delete_topics` path in Zookeeper.

`state` is `TopicDeletion`.

`process` Method

```
process(): Unit
```

Note	`process` is a part of ControllerEvent Contract.

Note	`process` is executed on the active controller only (and does nothing otherwise).

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

```
Delete topics listener fired for topics [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).

TopicUncleanLeaderElectionEnable

`TopicUncleanLeaderElectionEnable` is...FIXME

UncleanLeaderElectionEnable Controller Event

`UncleanLeaderElectionEnable` is a controller event that transition the KafkaController to UncleanLeaderElectionEnable state.

`UncleanLeaderElectionEnable` is enqueued when `KafkaController` is requested to enableDefaultUncleanLeaderElection.

Processing UncleanLeaderElectionEnable

When processed (on the controller-event-thread), `KafkaController` is requested to processUncleanLeaderElectionEnable.

ControllerChannelManager

`ControllerChannelManager` manages connections to all brokers for KafkaController.

`ControllerChannelManager` is created when KafkaController is created (to add or remove brokers as they "announce" themselves in Zookeeper) for ControllerBrokerRequestBatches of the KafkaController itself, the ZkReplicaStateMachine and ZkPartitionStateMachine (so they can send broker and partition changes out to all brokers in a Kafka cluster).

When started, `ControllerChannelManager` establishes connection to every broker and starts a corresponding RequestSendThread to keep sending queued controller requests.

`ControllerChannelManager` uses **[Channel manager on controller [brokerId]]** as the logging prefix (aka `logIdent`).

Tip	Enable `ALL` logging levels for `kafka.controller.ControllerChannelManager` logger to see what happens inside. Add the following line to `config/log4j.properties`: `log4j.logger.kafka.controller.ControllerChannelManager=ALL` Refer to Logging.

Connection Metadata of All Brokers — `brokerStateInfo` Internal Registry

```
brokerStateInfo: HashMap[Int, ControllerBrokerStateInfo]
```

`brokerStateInfo` is connection metadata by broker ID.

Request threads for brokers are all started when `ControllerChannelManager` is requested to start up.

A new broker is added when `ControllerChannelManager` is requested to add a broker (when addBroker).

A broker is removed when `ControllerChannelManager` is requested to remove a broker (when removeBroker or shutting down).

Use the TotalQueueSize gauge metric for the queue depth (i.e. how many controller requests are waiting to be sent out to all brokers).

Performance Metrics

`ControllerChannelManager` is a KafkaMetricsGroup with the following performance metrics.

Table 1. ControllerChannelManager's Performance Metrics

Metric Name	Description
`QueueSize`	Controller requests (AbstractControlRequests) queue size (per broker)
`RequestRateAndQueueTimeMs`	For every broker
`TotalQueueSize`	Total number of controller requests (AbstractControlRequests) to be sent out to brokers

The performance metrics are registered in **kafka.controller:type=ControllerChannelManager** group.

Figure 1. ControllerChannelManager in jconsole

Creating ControllerChannelManager Instance

`ControllerChannelManager` takes the following to be created:

- ControllerContext
- KafkaConfig
- `Time`
- Metrics
- StateChangeLogger
- Thread name prefix (default: `(empty)`)

Registering New Broker — `addNewBroker` Internal Method

```
addNewBroker(
  broker: Broker): Unit
```

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

```
Controller [brokerId] trying to connect to broker [id]
```

`addNewBroker` finds the name of the listener to use for communication with the broker based on control.plane.listener.name configuration property (if defined) or inter.broker.listener.name.

`addNewBroker` finds the security protocol to use for communication with the broker based on control.plane.listener.name and listener.security.protocol.map configuration properties (if defined) or security.inter.broker.protocol.

`addNewBroker` requests the `Broker` for the node for the listener name.

`addNewBroker` creates a new `LogContext` to use the prefix:

```
[Controller id=[brokerId], targetBrokerId=[brokerNode]]
```

`addNewBroker` creates a NetworkClient. Firstly, `addNewBroker` creates a ChannelBuilder (for the security protocol, the listener name, `SERVER` JAAS context type and SASL-related properties) and, if it is a Reconfigurable, adds it to the KafkaConfig as reconfigurable.

`addNewBroker` then creates a Selector with `controller-channel` metric group (and `broker-id` of the broker node).

`addNewBroker` builds a thread name per the optional threadNamePrefix:

```
[threadNamePrefix]:Controller-[brokerId]-to-broker-[id]-send-thread
```

`addNewBroker` creates a new RequestRateAndQueueTimeMs timer metric with the id of the broker to connect to.

`addNewBroker` creates a daemon RequestSendThread for the broker ID (of the controller broker), the ControllerContext, the `NetworkClient`, the `RequestRateAndQueueTimeMs` metric, the StateChangeLogger, and the thread name.

`addNewBroker` creates a QueueSize gauge metric (with the id of the broker to connect) that is the number of the AbstractControlRequest messages in the queue.

In the end, `addNewBroker` registers (*adds*) the id of the broker to connect with a new ControllerBrokerStateInfo to the brokerStateInfo internal registry.

Note	`addNewBroker` is used when `ControllerChannelManager` is requested to start up (and connect to brokers) and add a broker.

Registering Newly-Added Broker — `addBroker` Method

```
addBroker(broker: Broker): Unit
```

`addBroker` ...FIXME

Note	`addBroker` is used when `KafkaController` is requested to process a BrokerChange controller event.

Deregistering Broker — `removeBroker` Method

```
removeBroker(brokerId: Int): Unit
```

`removeBroker` finds the broker metadata in the brokerStateInfo internal registry that is then used to removeExistingBroker.

Note	`removeBroker` is used exclusively when `KafkaController` is requested to process a BrokerChange controller event.

Starting Up — `startup` Method

```
startup(): Unit
```

`startup` ...FIXME

Note	`startup` is used when `KafkaController` is requested to initializeControllerContext.

Shutting Down — `shutdown` Method

```
shutdown(): Unit
```

`shutdown` ...FIXME

Note	`shutdown` is used when...FIXME

Sending AbstractControlRequest Out to Broker — `sendRequest` Method

```
sendRequest(
  brokerId: Int,
  request: AbstractControlRequest.Builder[_ <: AbstractControlRequest],
  callback: AbstractResponse => Unit = null)
```

`sendRequest` ...FIXME

Note	`sendRequest` is used exclusively when `ControllerBrokerRequestBatch` is requested to send a controller request to a broker.

`removeExistingBroker` Internal Method

```
removeExistingBroker(
  brokerState: ControllerBrokerStateInfo): Unit
```

`removeExistingBroker` ...FIXME

Note	`removeExistingBroker` is used when...FIXME

Starting RequestSendThread — `startRequestSendThread` Internal Method

```
startRequestSendThread(
  brokerId: Int): Unit
```

`startRequestSendThread` finds the `RequestSendThread` in the broker metadata in the brokerStateInfo internal registry and, if the thread has not started yet, `startRequestSendThread` starts it.

Note	`startRequestSendThread` is used when `ControllerChannelManager` is requested to start up and addBroker.

ControllerBrokerStateInfo

`ControllerBrokerStateInfo` is a broker metadata that holds the following:

- Non-Blocking Network KafkaClient
- Broker Node
- Message Queue (`BlockingQueue[QueueItem]`)
- `RequestSendThread`
- Queue Size (`Gauge[Int]`)
- RequestRateAndTime Metrics
- Reconfigurable

RequestSendThread

`RequestSendThread` is...FIXME

ControllerBrokerRequestBatch

`ControllerBrokerRequestBatch` is a concrete AbstractControllerBrokerRequestBatch.

`ControllerBrokerRequestBatch` uses a ControllerChannelManager to send an controller request out to a broker.

`ControllerBrokerRequestBatch` uses a ControllerEventManager to enqueue a ControllerEvent (to ControllerEventManager).

`ControllerBrokerRequestBatch` is created exclusively for KafkaController (for brokerRequestBatch, ReplicaStateMachine, and PartitionStateMachine).

Every time `ControllerBrokerRequestBatch` is used it is first requested to prepare a new batch (of controller requests) that is then send out to selected brokers.

Creating ControllerBrokerRequestBatch Instance

`ControllerBrokerRequestBatch` takes the following to be created:

- KafkaConfig
- ControllerChannelManager
- ControllerEventManager
- ControllerContext
- StateChangeLogger

Emitting (Enqueuing) ControllerEvent — `sendEvent` Method

```
sendEvent(event: ControllerEvent): Unit
```

Note	`sendEvent` is part of the AbstractControllerBrokerRequestBatch Contract to send a ControllerEvent.

`sendEvent` simply requests the ControllerEventManager to enqueue the controller event.

Sending AbstractControlRequest Out to Broker — `sendRequest` Method

```
sendRequest(
  brokerId: Int,
  request: AbstractControlRequest.Builder[_ <: AbstractControlRequest],
  callback: AbstractResponse => Unit = null): Unit
```

Note	`sendRequest` is part of the AbstractControllerBrokerRequestBatch Contract to send a AbstractControlRequest to a broker.

`sendRequest` simply requests the ControllerChannelManager to send the request to the broker.

AbstractControllerBrokerRequestBatch

`AbstractControllerBrokerRequestBatch` is the base of ControllerBrokerRequestBatches that sendRequestsToBrokers.

Note	ControllerBrokerRequestBatch is the default and only known implementation of the AbstractControllerBrokerRequestBatch Contract in Apache Kafka.

Table 1. AbstractControllerBrokerRequestBatch Contract (Abstract Methods Only)

Method	Description

`sendEvent`	`sendEvent(`
	`  event: ControllerEvent): Unit`
	
	Sends a ControllerEvent
	
	Used when `AbstractControllerBrokerRequestBatch` is requested to sendLeaderAndIsrRequest and sendStopReplicaRequests

`sendRequest`	`sendRequest(`
	`  brokerId: Int,`
	`  request: AbstractControlRequest.Builder[_ <: AbstractControlRequest],`
	`  callback: AbstractResponse => Unit = null): Unit`
	
	Sends an controller request out to the given broker with optional callback to handle a response
	
	Used when `AbstractControllerBrokerRequestBatch` is requested to sendLeaderAndIsrRequest, sendUpdateMetadataRequests, and sendStopReplicaRequests

Creating AbstractControllerBrokerRequestBatch Instance

`AbstractControllerBrokerRequestBatch` takes the following to be created:

- KafkaConfig
- ControllerContext
- StateChangeLogger

`AbstractControllerBrokerRequestBatch` initializes the internal properties.

Note	`AbstractControllerBrokerRequestBatch` is a Scala abstract class and cannot be created directly. It is created indirectly for the concrete AbstractControllerBrokerRequestBatches.

Sending Controller Requests To Brokers — `sendRequestsToBrokers` Method

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

`sendRequestsToBrokers` sends LeaderAndIsr requests out to brokers.

`sendRequestsToBrokers` sends UpdateMetadata requests out to brokers.

`sendRequestsToBrokers` sends StopReplica requests out to brokers.

In case of any error (`Throwable`), `sendRequestsToBrokers` ...FIXME

Note	`sendRequestsToBrokers` is used when: • `KafkaController` is requested to updateLeaderEpochAndSendRequest, sendUpdateMetadataRequest and at ControlledShutdown controller event • `ZkPartitionStateMachine` is requested to handleStateChanges • `ReplicaStateMachine` is requested to handleStateChanges

Sending LeaderAndIsr Requests Out to All Brokers — `sendLeaderAndIsrRequest` Internal Method

```
sendLeaderAndIsrRequest(
  controllerEpoch: Int,
  stateChangeLog: StateChangeLogger): Unit
```

`sendLeaderAndIsrRequest` ...FIXME

Note	`sendLeaderAndIsrRequest` is used exclusively when `AbstractControllerBrokerRequestBatch` is requested to sendRequestsToBrokers.

Sending UpdateMetadata Requests Out to All Brokers — `sendUpdateMetadataRequests` Internal Method

```
sendUpdateMetadataRequests(
  controllerEpoch: Int,
  stateChangeLog: StateChangeLogger): Unit
```

`sendUpdateMetadataRequests` prints out the following TRACE message to the logs for every pair in the updateMetadataRequestPartitionInfoMap internal registry:

```
Sending UpdateMetadata request [partitionState] to brokers [updateMetadataRequestBroke
rSet] for partition [tp]
```

`sendUpdateMetadataRequests` prepares metadata for UpdateMetadataRequests.

For every broker (in the updateMetadataRequestBrokerSet internal registry) that is live or shutting down `sendUpdateMetadataRequests` requests the ControllerContext for the broker epoch, creates a new UpdateMetadataRequest (with the updateMetadataRequestPartitionInfoMap among others) and then sends the request out to the broker.

In the end, `sendUpdateMetadataRequests` removes all elements from (*clears*) the updateMetadataRequestBrokerSet and updateMetadataRequestPartitionInfoMap internal registries.

Note	`sendUpdateMetadataRequests` is used exclusively when `AbstractControllerBrokerRequestBatch` is requested to send controller requests to all brokers in a cluster.

Sending StopReplica Requests Out to All Brokers — `sendStopReplicaRequests` Internal Method

```
sendStopReplicaRequests(
  controllerEpoch: Int): Unit
```

`sendStopReplicaRequests` ...FIXME

Note	`sendStopReplicaRequests` is used exclusively when `AbstractControllerBrokerRequestBatch` is requested to sendRequestsToBrokers.

Preparing New Batch — `newBatch` Method

```
newBatch(): Unit
```

`newBatch` simply throws an `IllegalStateException` when any of the following internal registries are not empty: leaderAndIsrRequestMap, stopReplicaRequestMap and updateMetadataRequestBrokerSet.

```
Controller to broker state change requests batch is not empty while creating a new one
. Some LeaderAndIsr state changes [leaderAndIsrRequestMap] might be lost
```

```
Controller to broker state change requests batch is not empty while creating a new one
. Some StopReplica state changes [stopReplicaRequestMap] might be lost
```

```
Controller to broker state change requests batch is not empty while creating a new one
. Some UpdateMetadata state changes to brokers [updateMetadataRequestBrokerSet] with p
artition info [updateMetadataRequestPartitionInfoMap] might be lost
```

Note	`newBatch` is used when: • `KafkaController` is requested to updateLeaderEpochAndSendRequest, sendUpdateMetadataRequest, and process a ControlledShutdown controller event • `PartitionStateMachine` is requested to handleStateChanges • `PartitionStateMachine` is requested to handleStateChanges

`addLeaderAndIsrRequestForBrokers` Method

```
addLeaderAndIsrRequestForBrokers(
  brokerIds: Seq[Int],
  topicPartition: TopicPartition,
  leaderIsrAndControllerEpoch: LeaderIsrAndControllerEpoch,
  replicas: Seq[Int],
  isNew: Boolean): Unit
```

`addLeaderAndIsrRequestForBrokers` looks up the `LeaderAndIsrPartitionStates` of `TopicPartitions` on a broker in the leaderAndIsrRequestMap internal registry. `addLeaderAndIsrRequestForBrokers` uses IDs from the given `brokerIds` that are `0` or higher.

`addLeaderAndIsrRequestForBrokers` adds (or replaces) the given `topicPartition` in the result with a new `LeaderAndIsrPartitionState` based on the input arguments.

In the end, `addLeaderAndIsrRequestForBrokers` addUpdateMetadataRequestForBrokers to the live or shutting down brokers for the requested partition (`topicPartition`).

Note	`addLeaderAndIsrRequestForBrokers` is used when: • `KafkaController` is requested to updateLeaderEpochAndSendRequest • `ZkPartitionStateMachine` is requested to initializeLeaderAndIsrForPartitions and doElectLeaderForPartitions • `ZkReplicaStateMachine` is requested to \

`addUpdateMetadataRequestForBrokers` Method

```
addUpdateMetadataRequestForBrokers(
  brokerIds: Seq[Int],
  partitions: collection.Set[TopicPartition]): Unit
```

`addUpdateMetadataRequestForBrokers` ...FIXME

Note	`addUpdateMetadataRequestForBrokers` is used when: • `AbstractControllerBrokerRequestBatch` is requested to addLeaderAndIsrRequestForBrokers • `KafkaController` is requested to sendUpdateMetadataRequest • `ZkReplicaStateMachine` is requested to doHandleStateChanges

Internal Properties

Name	Description
`leaderAndIsrRequestMap`	`LeaderAndIsrPartitionState` of `TopicPartitions` by br `leaderAndIsrRequestMap: Map[ Int, Map[TopicPartition, LeaderAndIsrRequestData.Leade ]]` Used when…FIXME
`stopReplicaRequestMap`	(`Map[Int, ListBuffer[StopReplicaRequestInfo]]`) Used when…FIXME
`updateMetadataRequestBrokerSet`	Broker IDs to send UpdateMetadata requests out to • Broker IDs added in addUpdateMetadataRequestI • Cleared (*emptied*) in clear and after successful sendUpdateMetadataRequests

TopicDeletionManager

`TopicDeletionManager` is…FIXME

`TopicDeletionManager` is created exclusively when `KafkaController` is created.

Figure 1. TopicDeletionManager is Created Alongside KafkaController

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

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

Name	Description
`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)

`TopicDeletionManager` uses **[Topic Deletion Manager [brokerId]]** as the logging prefix (aka `logIdent`).

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

```
startReplicaDeletion(replicasForTopicsToBeDeleted: Set[PartitionAndReplica]): Unit
```

`startReplicaDeletion` ...FIXME

Note	`startReplicaDeletion` is used when...FIXME

`enqueueTopicsForDeletion` Method

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

`enqueueTopicsForDeletion` ...FIXME

Note	`enqueueTopicsForDeletion` is used when...FIXME

`failReplicaDeletion` Method

```
failReplicaDeletion(replicas: Set[PartitionAndReplica]): Unit
```

`failReplicaDeletion` ...FIXME

Note	`failReplicaDeletion` is used when...FIXME

Creating TopicDeletionManager Instance

`TopicDeletionManager` takes the following when created:

- KafkaController
- ControllerEventManager

`TopicDeletionManager` initializes the internal registries and counters.

`markTopicIneligibleForDeletion` Method

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

(only with delete.topic.enable Kafka property enabled) `markTopicIneligibleForDeletion` computes the intersection between topicsToBeDeleted and the input `topics` sets and adds the intersection to topicsIneligibleForDeletion set.

If there are any topics in the intersection, `markTopicIneligibleForDeletion` prints out the following INFO message to the logs:

```
Halted deletion of topics [newTopicsToHaltDeletion]
```

Note	`markTopicIneligibleForDeletion` is used when: • `KafkaController` initiateReassignReplicasForTopicPartition • `TopicDeletion` controller event is processed (for topics to be deleted with partitions in `controllerContext.partitionsBeingReassigned` list) • `TopicDeletionManager` does startReplicaDeletion and failReplicaDeletion

Reseting — `reset` Method

```
reset(): Unit
```

(only with delete.topic.enable Kafka property enabled) `reset` removes all elements from the following internal registries:

- topicsToBeDeleted
- partitionsToBeDeleted
- topicsIneligibleForDeletion

Note	`reset` does nothing when delete.topic.enable Kafka property is `false`.

Note	`reset` is used exclusively when `KafkaController` resigns as the active controller.

`onTopicDeletion` Internal Method

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

`onTopicDeletion` ...FIXME

Note	`onTopicDeletion` is used when...FIXME

`completeDeleteTopic` Internal Method

```
completeDeleteTopic(topic: String): Unit
```

`completeDeleteTopic` ...FIXME

Note	`completeDeleteTopic` is used when...FIXME

Initializing — `init` Method

```
init(
  initialTopicsToBeDeleted: Set[String],
  initialTopicsIneligibleForDeletion: Set[String]): Unit
```

`init` ...FIXME

Note	`init` is used when...FIXME

`tryTopicDeletion` Method

```
tryTopicDeletion(): Unit
```

`tryTopicDeletion` ...FIXME

Note	`tryTopicDeletion` is used when...FIXME

`isTopicQueuedUpForDeletion` Method

```
isTopicQueuedUpForDeletion(topic: String): Boolean
```

`isTopicQueuedUpForDeletion` ...FIXME

Note	`isTopicQueuedUpForDeletion` is used when...FIXME

`resumeDeletionForTopics` Method

```
resumeDeletionForTopics(topics: Set[String] = Set.empty): Unit
```

`resumeDeletionForTopics` ...FIXME

Note	`resumeDeletionForTopics` is used when...FIXME

`completeReplicaDeletion` Method

```
completeReplicaDeletion(replicas: Set[PartitionAndReplica]): Unit
```

`completeReplicaDeletion` ...FIXME

Note	`completeReplicaDeletion` is used when...FIXME

`markTopicForDeletionRetry` Internal Method

```
markTopicForDeletionRetry(topic: String): Unit
```

`markTopicForDeletionRetry` ...FIXME

Note	`markTopicForDeletionRetry` is used when...FIXME

`retryDeletionForIneligibleReplicas` Internal Method

```
retryDeletionForIneligibleReplicas(topic: String): Unit
```

`retryDeletionForIneligibleReplicas` ...FIXME

Note	`retryDeletionForIneligibleReplicas` is used when...FIXME

AbstractControlRequest — Base of Controller Requests

`AbstractControlRequest` is the extension of the AbstractRequest contract for controller requests that KafkaController uses to propage broker and partition state changes to brokers.

Table 1. AbstractControlRequests

AbstractControlRequest	Description
LeaderAndIsrRequest	
StopReplicaRequest	
UpdateMetadataRequest	

Creating AbstractControlRequest Instance

`AbstractControlRequest` takes the following to be created:

- `ApiKeys`
- Version
- Controller ID
- Controller Epoch
- Broker Epoch

`AbstractControlRequest` initializes the internal properties.

Note	`AbstractControlRequest` is a Java abstract class and cannot be created directly. It is created indirectly for the concrete AbstractControlRequests.

LeaderAndIsrRequest

`LeaderAndIsrRequest` is a controller request with **LeaderAndIsr** API key and the following properties:

- Controller ID
- Controller Epoch
- Broker Epoch
- `PartitionStates` by `TopicPartition` (`Map<TopicPartition, PartitionState>`)
- **Live leaders** (brokers with leaders)
- Version

`LeaderAndIsrRequest` is created (via build factory method) when:

- `AbstractRequest` is requested to parse a request (with the LeaderAndIsr API key)
- `LeaderAndIsrRequest` is requested to parse a byte buffer
- `LeaderAndIsrRequest.Builder` is requested to build a LeaderAndIsrRequest (when `ControllerBrokerRequestBatch` is requested to sendRequestsToBrokers)

`LeaderAndIsrRequest.Builder` Factory Object

`LeaderAndIsrRequest` comes with a concrete AbstractRequest.Builder factory object that can build a LeaderAndIsrRequest.

```
LeaderAndIsrRequest build(short version)
```

`LeaderAndIsrRequest.Builder` is used exclusively when `AbstractControllerBrokerRequestBatch` is requested to send LeaderAndIsr requests out to all brokers.

Creating LeaderAndIsrRequest from Byte Buffer — `parse` Factory Method

```
LeaderAndIsrRequest parse(
  ByteBuffer buffer,
  short version)
```

`parse` simply creates a LeaderAndIsrRequest.

Note	`parse` is used when…FIXME

StopReplicaRequest

`StopReplicaRequest` is a controller request with **STOP_REPLICA** API key and the following properties:

- Controller ID
- Controller Epoch
- Broker Epoch
- `deletePartitions` Flag
- Partitions (`Collection<TopicPartition>`)
- Version

`StopReplicaRequest` is created (indirectly via build factory method) when:

- FIXME

`StopReplicaRequest.Builder` Factory Object

`StopReplicaRequest` comes with a concrete AbstractRequest.Builder factory object that can build a StopReplicaRequest.

```
StopReplicaRequest build(short version)
```

`StopReplicaRequest.Builder` is used exclusively when `AbstractControllerBrokerRequestBatch` is requested to send UpdateMetadata requests out to all brokers.

UpdateMetadataRequest

`UpdateMetadataRequest` is a controller request with **UPDATE_METADATA** API key and the following properties:

- Version
- Controller ID
- Controller Epoch
- Broker Epoch
- `PartitionStates` by `TopicPartition` (`Map<TopicPartition, PartitionState>`)
- **Live leaders** (brokers with leaders)

KafkaApis and UpdateMetadataRequest

When received, `UpdateMetadataRequest` is handled by handleUpdateMetadataRequest.

`UpdateMetadataRequest.Builder` Factory Object

`UpdateMetadataRequest` comes with a concrete AbstractRequest.Builder factory object that can build a UpdateMetadataRequest.

```
UpdateMetadataRequest build(short version)
```

`UpdateMetadataRequest.Builder` is used exclusively when `AbstractControllerBrokerRequestBatch` is requested to send UpdateMetadata requests out to all brokers.

KafkaProducer — Main Class For Kafka Producers

`KafkaProducer` is the default Producer client in Apache Kafka.

`KafkaProducer` is the class that a Kafka developer uses to send messages to a Kafka cluster.

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

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

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

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

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

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

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

Figure 1. KafkaProducer

`KafkaProducer` uses **producer-metrics** for…FIXME

Tip	Enable `ALL` logging levels for `org.apache.kafka.clients.producer.KafkaProducer` logger to see what happens inside. Add the following line to `config/log4j.properties`: `log4j.logger.org.apache.kafka.clients.producer.KafkaProducer=ALL` Refer to Logging.

Performance Metrics

`KafkaProducer` exposes performance metrics.

Table 1. KafkaProducer's Metrics

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

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

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

1. Uses no callback (`null`)

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

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

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

Sending ProducerRecord to Kafka Cluster Asynchronously — `doSend` Internal Method

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

`doSend` ...FIXME

`doSend` requests the RecordAccumulator to append the serialized key and value.

`doSend` ...FIXME

Note	`doSend` is used exclusively when `KafkaProducer` is requested to send a ProducerRecord to a Kafka cluster asynchronously with producer interceptors.

Configuring ClusterResourceListeners — `configureClusterResourceListeners` Internal Method

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

`configureClusterResourceListeners` creates a ClusterResourceListeners and registers `ClusterResourceListener` instances from the input `candidateLists`, `keySerializer` and `valueSerializer`.

Note	`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...[partitionsC
ount]).
```

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

Creating KafkaProducer Instance

`KafkaProducer` takes the following when created:

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

`KafkaProducer` initializes the internal registries and counters.

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

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

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

```
Starting the Kafka producer
```

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

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

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

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

`KafkaProducer` sets the partitioner as ProducerConfig.PARTITIONER_CLASS_CONFIG.

`KafkaProducer` sets the keySerializer as follows:

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

`KafkaProducer` sets the valueSerializer as follows:

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

`KafkaProducer` sets the interceptors as ProducerConfig.INTERCEPTOR_CLASSES_CONFIG.

`KafkaProducer` sets the following:

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

`KafkaProducer` creates a new `ApiVersions` for the apiVersions.

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

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

`KafkaProducer` sets the metadata as follows:

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

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

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

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

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

`KafkaProducer` sets the transactionManager as configureTransactionState.

`KafkaProducer` configureRetries for the number of retries.

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

`KafkaProducer` configureAcks for acks.

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

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

`KafkaProducer` requests the `ProducerConfig` to logUnused.

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

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

```
Kafka producer started
```

In case of any errors, `KafkaProducer` closes itself with `0` millis timeout and throws a `KafkaException`:

```
Failed to construct kafka producer
```

Computing Partition For ProducerRecord — `partition` Internal Method

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

`partition` requests the `ProducerRecord` for the partition and return it when specified. Otherwise, `partition` requests the Partitioner for the partition.

Note	`partition` is used exclusively when `KafkaProducer` is requested to send a ProducerRecord to a Kafka cluster asynchronously.

`beginTransaction` Method

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

Note	`beginTransaction` is part of the Producer Contract to…FIXME.

`beginTransaction` simply requests the internal TransactionManager to beginTransaction.

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

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

`configureTransactionState` Static Internal Method

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

`configureTransactionState` ...FIXME

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

Closing Kafka Producer — `close` Method

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

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

Note	`close` is a part of Producer Contract.

`close` ...FIXME

Flushing Accumulated Records — `flush` Method

```
void flush()
```

Note	`flush` is a part of Producer Contract.

`flush` ...FIXME

`initTransactions` Method

```
void initTransactions()
```

Note	`initTransactions` is a part of Producer Contract.

`initTransactions` ...FIXME

`sendOffsetsToTransaction` Method

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

Note	`sendOffsetsToTransaction` is a part of Producer Contract.

`sendOffsetsToTransaction` ...FIXME

`commitTransaction` Method

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

Note	`commitTransaction` is a part of Producer Contract.

`commitTransaction` ...FIXME

`abortTransaction` Method

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

Note	`abortTransaction` is a part of Producer Contract.

`abortTransaction` ...FIXME

`newSender` Method

```
Sender newSender(
  LogContext logContext,
  KafkaClient kafkaClient,
  ProducerMetadata metadata)
```

`newSender` ...FIXME

Note	`newSender` is used when `KafkaProducer` is created.

Internal Properties

Name	Description
`accumulator`	RecordAccumulator Used when: • Sending a ProducerRecord to a Kafka cluster asynchronously • Creating a new Sender • Flushing accumulated records
`clientId`	Client ID per CLIENT_ID_CONFIG (if defined) or `producer-[number]` Used when: • `KafkaProducer` is requested for logging • NetworkClient is created
`interceptors`	ProducerInterceptors that are notified (about successes and failures) when `KafkaProducer` is requested to send a record to a topic The `ProducerInterceptors` are initialized when the `KafkaProducer` is created using ProducerConfig.INTERCEPTOR_CLASSES_CONFIG configuration property.
`ioThread`	**kafka-producer-network-thread** daemon thread of execution for the sender
`maxBlockTimeMs`	Time `KafkaProducer` uses to block when requesting partitions for a topic. Note: Use max.block.ms Kafka property to set the value.

`metadata`	Metadata
	
	• Created when `KafkaProducer` is created with the following properties:
	  ◦ retry.backoff.ms for refreshBackoffMs
	  ◦ metadata.max.age.ms for metadataExpireMs
	  ◦ allowAutoTopicCreation flag enabled
	  ◦ topicExpiryEnabled flag enabled

	• Updated with a bootstrap cluster when `KafkaProducer` is created • Used in waitOnMetadata
`partitioner`	Partitioner that is used (to compute the partition for a record) when `KafkaProducer` is requested to partition (when requested to send a ProducerRecord to a Kafka cluster asynchronously) The `Partitioner` is initialized when the `KafkaProducer` is created using ProducerConfig.PARTITIONER_CLASS_CONFIG configuration property.
`producerConfig`	ProducerConfig
`sender`	Kafka producer I/O thread (aka `Sender`) that is started when `KafkaProducer` is created.
`time`	Time abstraction (with `SYSTEM` being the default).
`transactionManager`	TransactionManager Used when: • `KafkaProducer` is created (and creates a RecordAccumulator and a Sender) • For transactional public methods: initTransactions, beginTransaction, sendOffsetsToTransaction, commitTransaction, abortTransaction • Sending a ProducerRecord to a Kafka cluster asynchronously

Producer Contract

`Producer` is the contract of Kafka producers.

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

Table 1. Producer Contract

Method	Description
`abortTransaction`	`void abortTransaction() throws ProducerFencedException` Used when…FIXME
`beginTransaction`	`void beginTransaction() throws ProducerFencedException` Used when…FIXME

`close`	`void close()`
	`void close(long timeout, TimeUnit unit)`
	
	Used when…FIXME

`commitTransaction`	`void commitTransaction() throws ProducerFencedException` Used when…FIXME
`flush`	`void flush()` Used when…FIXME
`initTransactions`	`void initTransactions()` Initializes transactions (when the `transactional.id` is configured)
`metrics`	`Map<MetricName, ? extends Metric> metrics()`

	The performance metrics
`partitionsFor`	`List<PartitionInfo> partitionsFor(String topic)` The partition metadata for the given topic. This can be used for custom partitioning. Used when…FIXME

`send`	`Future<RecordMetadata> send(`
	`  ProducerRecord<K, V> record)`
	`Future<RecordMetadata> send(`
	`  ProducerRecord<K, V> record,`
	`  Callback callback)`
	
	Sending a ProducerRecord to a Kafka cluster asynchronously
	
	Used when…FIXME

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

ProducerConfig

`ProducerConfig` is the configuration of a Kafka Producer.

`ProducerConfig` is created when:

- …

Table 1. ProducerConfig's Configuration Properties

Name / Default Value / Property	Description
`ACKS_CONFIG`	The number of acknowledgments a producer requires the leader to have received before considering a request complete. This controls tl durability of records that are sent. Default: `1` Property: `acks` The following settings are allowed: • `0` - a producer will not wait for any acknowledgment from the server at all. The record will be immediately added to the socket buffer and considered sent. No guarantee can be made that the server has received the record in this case, and the retries configuration will not take effect (as the client won't generally know of any failures). The offset given back for each record will always be set to `-1`. • `1` (default) - the leader will write the recor to its local log but will respond without awaiting full acknowledgement from all followers. In this case should the leader fai immediately after acknowledging the recor but before the followers have replicated it then the record will be lost. • `all` or `-1` - the leader will wait for the ful set of in-sync replicas to acknowledge the record. This guarantees that the record will not be lost as long as at least one in-sync replica remains alive. This is the strongest available guarantee.
	Host:port pairs to use to establish the initial connection to a Kafka cluster. The client will make use of all servers irrespective of which

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

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

`DELIVERY_TIMEOUT_MS_CONFIG`	A Kafka producer may report failure to send a record earlier than this config if either an unrecoverable error is encountered, the retries have been exhausted, or the record is added to batch which reached an earlier delivery expiration deadline. The value of this config should be greater than equal to the sum of request.timeout.ms and linger.ms. Must be at least `0`
`ENABLE_IDEMPOTENCE_CONFIG`	When enabled (`true`), a producer will ensure that exactly one copy of a message is written to the stream. Default: `false` Property: `enable.idempotence` When disabled (`false`), it is acceptable that a producer may write duplicates of a message to the stream (e.g. due to broker failures and retries). Enabling idempotence requires: • max.in.flight.requests.per.connection to be less than or equal to `5` • retries to be greater than `0` • acks must be `all` If these values are not explicitly set, suitable values will be chosen. If incompatible values are set, a `ConfigException` will be thrown.
`INTERCEPTOR_CLASSES_CONFIG`	ProducerInterceptors to use to intercept (and possibly mutate) the records sent out by the producer before they are published to the Kafka cluster. Default: (empty) Property: `interceptor.classes`
`KEY_SERIALIZER_CLASS_CONFIG`	Serializer class for keys that implements the org.apache.kafka.common.serialization.Serializ interface. Default: (empty) Property: `key.serializer`

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

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

`METADATA_MAX_AGE_CONFIG`	The period of time (in milliseconds) after which we force a refresh of metadata even if we haven't seen any partition leadership changes t proactively discover any new brokers or partitions. Default: `5 * 60 * 1000` Property: `metadata.max.age.ms` Must be at least `0`
`METRICS_NUM_SAMPLES_CONFIG`	The number of samples maintained to compute metrics (for Kafka producers). Must be at least `1`. Property: `metrics.num.samples`
`METRICS_RECORDING_LEVEL_CONFIG`	The name of highest recording level for metrics Property: `metrics.recording.level` Must be one of the following: `INFO` or `DEBUG`.
`METRIC_REPORTER_CLASSES_CONFIG`	The class names of the MetricsReporters that will be notified of new metric creation. Property: `metric.reporters` The JmxReporter is always included to register JMX statistics.
`METRICS_SAMPLE_WINDOW_MS_CONFIG`	The window of time a metrics sample is computed over (for Kafka producers). Property: `metrics.sample.window.ms`
`PARTITIONER_CLASS_CONFIG`	The Partitioner to compute the partition for a record when `KafkaProducer` is requested to ser a record to topic. Default: DefaultPartitioner Property: `partitioner.class`
`RECONNECT_BACKOFF_MAX_MS_CONFIG`	
`RECONNECT_BACKOFF_MS_CONFIG`	
`RECEIVE_BUFFER_CONFIG`	
	Maximum amount of time the client will wait for the response of a request. If the response is no received before the timeout elapses the client w

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

`TRANSACTIONAL_ID_CONFIG`	With no Transactional ID provided, the produce is limited to idempotent delivery. enable.idempotence must be enabled (`true`) i a Transactional ID is configured. The default means transactions cannot be usec Note that transactions requires a cluster of at least 3 brokers by default what is the recommended setting for production; for development you can change this, by adjusting broker setting `transaction.state.log.replication.factor` .

`logUnused` Method

```
void logUnused()
```

`logUnused` ...FIXME

Note	`logUnused` is used when...FIXME

ProducerRecord

`ProducerRecord` is...FIXME

Callback Contract

`Callback` is…FIXME

Partitioner Contract — Partitioning Strategy

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

`Partitioner` is also a Configurable that can be created by reflection and need to take configuration parameters.

Note	DefaultPartitioner is the one and only known implementation of the Partitioner Contract in Apache Kafka.

Table 1. Partitioner Contract

Method	Description
`close`	`void close()` Closes the partitioner

`partition`	`int partition(`
	`  String topic,`
	`  Object key,`
	`  byte[] keyBytes,`
	`  Object value,`
	`  byte[] valueBytes,`
	`  Cluster cluster)`
	
	Computes the partition ID for a record (described by a topic, a key, a value, the serialized key and value, and the Cluster metadata)
	
	Used when `KafkaProducer` is requested to compute the partition for a ProducerRecord

ProducerInterceptor

`ProducerInterceptor` is...FIXME

Sender — Kafka Producer Network Thread

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

Figure 1. Sender and KafkaProducer

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

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

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

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

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

Creating Sender Instance

`Sender` takes the following to be created:

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

`Sender` initializes the internal properties.

Starting Thread of Execution — `run` Method

```
void run()
```

Note	`run` is a part of java.lang.Runnable that is executed when the thread is started.

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

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

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

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

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

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

In the end, `run` requests the KafkaClient to close and prints out the following DEBUG message to the logs:

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

Running Single Iteration of Sending — `runOnce` Method

```
void runOnce()
```

`runOnce` simply sendProducerData followed by requesting the KafkaClient to poll.

With a transactionManager assigned, `runOnce` …FIXME

Note	`run` is used exclusively when `Sender` is requested to run (as a thread of execution).

`sendProducerData` Internal Method

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

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

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

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

`sendProducerData` adds the batches to inflight batches.

`sendProducerData` …FIXME (`guaranteeMessageOrder`)

`sendProducerData` requests the SenderMetrics to updateProduceRequestMetrics.

`sendProducerData` …FIXME

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

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

Sending ProduceRequests — `sendProduceRequests` Internal Method

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

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

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

Sending ProduceRequest to Broker — `sendProduceRequest` Internal Method

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

`sendProduceRequest` ...FIXME

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

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

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

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

`sendProduceRequest` requests the KafkaClient to send the ClientRequest.

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

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

Note	`sendProduceRequest` is used exclusively when `Sender` is requested to sendProduceRequests.

Handling ProduceResponse — `handleProduceResponse` Internal Method

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

`handleProduceResponse` ...FIXME

Note	`handleProduceResponse` is used exclusively when `Sender` is requested to send a ProduceRequest.

`completeBatch` Internal Method

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

`completeBatch` ...FIXME

Note	`completeBatch` is used exclusively when `Sender` is requested to handle a ProduceResponse.

`maybeSendAndPollTransactionalRequest` Internal Method

```
boolean maybeSendAndPollTransactionalRequest()
```

`maybeSendAndPollTransactionalRequest` ...FIXME

Note	`maybeSendAndPollTransactionalRequest` is used when `Sender` is requested to run a single iteration of sending.

`addToInflightBatches` Method

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

`addToInflightBatches` ...FIXME

Note	`addToInflightBatches` is used exclusively when `Sender` is requested to sendProducerData.

`maybeSendTransactionalRequest` Internal Method

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

`maybeSendTransactionalRequest` ...FIXME

Note	`maybeSendTransactionalRequest` is used exclusively when `Sender` is running.

`maybeWaitForProducerId` Internal Method

```
void maybeWaitForProducerId()
```

`maybeWaitForProducerId` ...FIXME

Note	`maybeWaitForProducerId` is used exclusively when `Sender` is running.

`awaitLeastLoadedNodeReady` Internal Method

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

`awaitLeastLoadedNodeReady` ...FIXME

Note	`awaitLeastLoadedNodeReady` is used when `Sender` is requsted to maybeSendTransactionalRequest and maybeWaitForProducerId.

`initiateClose` Method

```
void initiateClose()
```

`initiateClose` requests the RecordAccumulator to close.

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

Note	`initiateClose` is used when: • `KafkaProducer` is requested to close • `Sender` is requested to forceClose

`wakeup` Method

```
void wakeup()
```

`wakeup` merely requests the KafkaClient to wakeup.

Note	`wakeup` is used when: • `KafkaProducer` is requested to initTransactions, sendOffsetsToTransaction, commitTransaction, abortTransaction, doSend, waitOnMetadata, and flush • `Sender` is requested to initiateClose

`forceClose` Method

```
void forceClose()
```

`forceClose` ...FIXME

Note	`forceClose` is used exclusively when `KafkaProducer` is requested to close.

`reenqueueBatch` Internal Method

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

`reenqueueBatch` ...FIXME

Note	`reenqueueBatch` is used when...FIXME

Internal Properties

Name	Description
`running`	Flag that controls whether run should stop (`false`) or not (`true`) • Enabled (`true`) by default when `Sender` is created • Disabled (`false`) when `Sender` is requested to initiateClose

RecordAccumulator

`RecordAccumulator` is created exclusively when `KafkaProducer` is created.

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

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

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

Creating RecordAccumulator Instance

`RecordAccumulator` takes the following to be created:

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

- `BufferPool`

`RecordAccumulator` initializes the internal properties.

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

`registerMetrics` Method

```
void registerMetrics()
```

`registerMetrics` ...FIXME

Note	`registerMetrics` is used exclusively when `RecordAccumulator` is created.

`close` Method

```
void close()
```

`close` ...FIXME

Note	`close` is used when...FIXME

`append` Method

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

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

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

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

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

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

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

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

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

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

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

Note	`append` is used exclusively when `KafkaProducer` is requested to send a ProducerRecord to a Kafka Cluster asynchronously.

`tryAppend` Internal Method

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

`tryAppend` ...FIXME

Note	`tryAppend` is used exclusively when `RecordAccumulator` is requested to append.

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

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

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

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

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

Note	`recordsBuilder` is used exclusively when `RecordAccumulator` is requested to append.

Aborting Incomplete Batches — `abortIncompleteBatches` Method

```
void abortIncompleteBatches()
```

`abortIncompleteBatches` ...FIXME

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

`reenqueue` Method

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

`reenqueue` ...FIXME

Note	`reenqueue` is used exclusively when `Sender` is requested to reenqueueBatch.

`splitAndReenqueue` Method

```
int splitAndReenqueue(ProducerBatch bigBatch)
```

`splitAndReenqueue` ...FIXME

Note	`splitAndReenqueue` is used exclusively when `Sender` is requested to completeBatch.

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

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

`getOrCreateDeque` ...FIXME

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

`expiredBatches` Method

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

`expiredBatches` ...FIXME

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

`ready` Method

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

`ready` ...FIXME

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

`drain` Method

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

`drain` ...FIXME

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

`drainBatchesForOneNode` Internal Method

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

`drainBatchesForOneNode` ...FIXME

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

Internal Properties

Name	Description
`appendsInProgress`	java.util.concurrent.atomic.AtomicInteger to keep track of the number of appending threads • Starts at `0` when `RecordAccumulator` is created • Increments and decrements when `RecordAccumulator` is requested to append (just after it is requested and right before it finishes) Used exclusively when `RecordAccumulator` is requested to abort incomplete batches
`batches`	ProducerBatches per `TopicPartition` (`ConcurrentMap<TopicPartition, Deque<ProducerBatch>>`)
`incomplete`	`IncompleteBatches`

ProducerBatch

`ProducerBatch` is…FIXME

`ProducerBatch` is created when:

- `RecordAccumulator` is requested to append
- `ProducerBatch` is requested to split

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

Creating ProducerBatch Instance

`ProducerBatch` takes the following to be created:

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

`ProducerBatch` initializes the internal properties.

`split` Method

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

`split`…FIXME

Note	`split` is used exclusively when `RecordAccumulator` is requested to splitAndReenqueue.

`createBatchOffAccumulatorForRecord` Internal Method

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

`createBatchOffAccumulatorForRecord` ...FIXME

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

ProducerInterceptors

`ProducerInterceptors` is…FIXME

Passing ProducerRecord Through Chain Of Registered Producer Interceptors — `onSend` Method

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

`onSend` …FIXME

Note	`onSend` is used when…FIXME

DefaultPartitioner — Default Partitioning Strategy

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

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

`DefaultPartitioner` is a Configurable that needs no configuration while being created.

`DefaultPartitioner` does nothing when requested to close.

Computing Partition For Record — `partition` Method

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

Note	`partition` is part of the Partitioner Contract to compute the partition for a record.

`partition` requests the input `Cluster` for the partitions for the given topic.

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

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

`nextValue` Internal Method

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

`nextValue` ...FIXME

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

TransactionManager

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

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

Tip	You should see one of the following INFO messages in the logs with a transactional KafkaProducer:
	`Instantiated a transactional producer.`
	or
	`Instantiated an idempotent producer.`

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

// Observe the logs
```

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

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

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

`TransactionManager` is in one of the following states:

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

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

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

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

`beginTransaction` Method

```
void beginTransaction()
```

`beginTransaction` …FIXME

Note	`beginTransaction` is used when…FIXME

Creating TransactionManager Instance

`TransactionManager` takes the following to be created:

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

`TransactionManager` initializes the internal registries and counters.

`initializeTransactions` Method

```
TransactionalRequestResult initializeTransactions()
```

`initializeTransactions` …FIXME

Note	`initializeTransactions` is used when…FIXME

`sendOffsetsToTransaction` Method

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

`sendOffsetsToTransaction` ...FIXME

Note	`sendOffsetsToTransaction` is used when...FIXME

`beginCommit` Method

```
TransactionalRequestResult beginCommit()
```

`beginCommit` ...FIXME

Note	`beginCommit` is used when...FIXME

`beginAbort` Method

```
TransactionalRequestResult beginAbort()
```

`beginAbort` ...FIXME

Note	`beginAbort` is used when...FIXME

`maybeAddPartitionToTransaction` Method

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

`maybeAddPartitionToTransaction` ...FIXME

Note	`maybeAddPartitionToTransaction` is used when...FIXME

KafkaServer — Kafka Broker

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

Figure 1. KafkaServer's Startup and Auxiliary Services

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

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

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 the Kafka broker.

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

```
starting
```

`startup` sets BrokerState as `Starting` .

`startup` requests KafkaScheduler to start.

`startup` connects to Zookeeper (and initializes ZkUtils).

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

You should see the following INFO message in the logs:

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

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

`startup` requests the KafkaConfig to set the brokerId to the broker ID.

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

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

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

`startup` creates and configures metrics.

Caution	FIXME

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

`startup` initializes QuotaManagers.

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

`startup` creates the LogDirFailureChannel

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

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

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

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

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

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

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

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

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

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

`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 `KafkaServer` may start processing requests.

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

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

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

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

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

`startup` checkpoints the broker ID.

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

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

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

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

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

Note	`startup` is used exclusively when `KafkaServerStartable` is requested to starts up.

Authorizer

```
authorizer: Option[Authorizer] = None
```

`authorizer` is an Authorizer based on authorizer.class.name configuration property (default: `(empty)`).

`Authorizer` is used to create the data-plane and control-plane `KafkaApis` (for authorizing operations).

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

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

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

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

Creating ReplicaManager — `createReplicaManager` Internal Method

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

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

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

`getOrGenerateClusterId` Internal Method

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

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

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

Shutting Down — `shutdown` Method

```
shutdown(): Unit
```

`shutdown` ...FIXME

Note	`shutdown` is used when: • `KafkaServer` is requested to start up (and there was an exception) • `KafkaServerStartable` is requested to shut down

`initZkClient` Internal Method

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

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

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

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

`initZkClient` ...FIXME (secureAclsEnabled)

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

In the end, `initZkClient` requests the KafkaZkClient to createTopLevelPaths.

Note	`initZkClient` is used exclusively when `KafkaServer` is requested to start up.

`controlledShutdown` Internal Method

```
controlledShutdown(): Unit
```

`controlledShutdown` ...FIXME

Note	`controlledShutdown` is used when `KafkaServer` is requested to shut down.

Checkpointing Broker — `checkpointBrokerId` Internal Method

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

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

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

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

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

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

`getBrokerIdAndOfflineDirs` ...FIXME

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

`generateBrokerId` Internal Method

```
generateBrokerId: Int
```

`generateBrokerId` ...FIXME

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

`createBrokerInfo` Internal Method

```
createBrokerInfo: BrokerInfo
```

`createBrokerInfo` ...FIXME

Note	`createBrokerInfo` is used when: • `KafkaServer` is requested to start up • `DynamicListenerConfig` is requested to reconfigure

Cluster ID — `_clusterId` Internal Property

```
_clusterId: String
```

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

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

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

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

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

Cluster ID is used for the following:

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

Default Configuration Properties of Logs (for LogManager and AdminManager) — `copyKafkaConfigToLog` Internal Utility

```
copyKafkaConfigToLog(
  kafkaConfig: KafkaConfig): Map[String, Object]
```

`copyKafkaConfigToLog` sets the topic-level configuration properties based on the given KafkaConfig.

Note	`copyKafkaConfigToLog` uses the same configuration properties as TopicConfigSynonyms. The keys of the configuration properties of `LogConfig` are simply aliases of TopicConfig.

Table 1. Topic-Level Configuration Properties

LogConfig	KafkaConfig
CleanupPolicyProp	logCleanupPolicy
CompressionTypeProp	compressionType
DeleteRetentionMsProp	logCleanerDeleteRetentionMs
FileDeleteDelayMsProp	logDeleteDelayMs
FlushMessagesProp	logFlushIntervalMessages
FlushMsProp	logFlushIntervalMs
IndexIntervalBytesProp	logIndexIntervalBytes
MaxCompactionLagMsProp	logCleanerMaxCompactionLagMs
MaxMessageBytesProp	messageMaxBytes
MessageDownConversionEnableProp	logMessageDownConversionEnable
MessageFormatVersionProp	logMessageFormatVersion
MessageTimestampDifferenceMaxMsProp	logMessageTimestampDifferenceMaxMs
MessageTimestampTypeProp	logMessageTimestampType
MinCleanableDirtyRatioProp	logCleanerMinCleanRatio
MinCompactionLagMsProp	logCleanerMinCompactionLagMs
MinInSyncReplicasProp	minInSyncReplicas
PreAllocateEnableProp	logPreAllocateEnable
RetentionBytesProp	logRetentionBytes
RetentionMsProp	logRetentionTimeMillis
SegmentBytesProp	logSegmentBytes
SegmentIndexBytesProp	logIndexSizeMaxBytes
SegmentJitterMsProp	logRollTimeJitterMillis

SegmentMsProp	logRollTimeMillis
UncleanLeaderElectionEnableProp	uncleanLeaderElectionEnable

Note	`copyKafkaConfigToLog` is used when: • `LogManager` utility is requested to create an instance of LogManager (for `KafkaServer` when requested to start up) • `AdminManager` is requested to describe the configuration of a topic and create topics

Internal Properties

Name	Description
`adminManager`	AdminManager
`apis`	KafkaApis
`brokerMetadataCheckpoints`	
`brokerState`	`BrokerState`
`_brokerTopicStats`	BrokerTopicStats Created when `KafkaServer` is requested to start up Used (as `brokerTopicStats` method) to create the data-plane KafkaApis, the control-plane KafkaApis, the ReplicaManager, the LogManager when `KafkaServer` is requested to start up
`controlPlaneRequestHandlerPool`	Control-plane KafkaRequestHandlerPool for the control-plane KafkaApis
`controlPlaneRequestProcessor`	Control-plane KafkaApis request processor (handler) for the optional control-plane KafkaRequestHandlerPool of the SocketServer
`credentialProvider`	`CredentialProvider`
`dataPlaneRequestProcessor`	Data-plane KafkaApis request handler for the data-plane RequestChannel of the SocketServer
`dataPlaneRequestHandlerPool`	Data-plane KafkaRequestHandlerPool for the data-plane KafkaApis

`dynamicConfigHandlers`	ConfigHandlers by name: • TopicConfigHandler as **topics** • ClientIdConfigHandler as **clients** • UserConfigHandler as **users** • BrokerConfigHandler as **brokers** Initialized when `KafkaServer` is requested to start up for the only purpose of creating the DynamicConfigManager.
`dynamicConfigManager`	DynamicConfigManager
`groupCoordinator`	GroupCoordinator (for the only purpose of creating the KafkaApis) Created and immediately started up when `KafkaServer` is requested to start up Shut down when `KafkaServer` is requested to shut down
`isStartingUp`	Flag for…FIXME
`kafkaController`	KafkaController
`kafkaHealthcheck`	KafkaHealthcheck
`kafkaScheduler`	KafkaScheduler with the number of daemon threads as configured using background.threads configuration property (default: `10`)
`logContext`	`LogContext`
`logDirFailureChannel`	`LogDirFailureChannel`
`logManager`	LogManager Used when: • checkpointBrokerId (when `KafkaServer` is requested to start up) • `DynamicBrokerConfig` is requested to addReconfigurables • `KafkaServer` is requested to start up (and creates a `TopicConfigHandler` for topics for dynamicConfigHandlers and a ReplicaManager)

	Created and immediately started when `KafkaServer` is requested to start up. Shut down when `KafkaServer` is requested to shut down.
`metadataCache`	MetadataCache that is created for the sake of creating the following services (at startup): • AdminManager • KafkaApis • ReplicaManager • TransactionCoordinator
`replicaManager`	ReplicaManager used to create: • KafkaApis • GroupCoordinator • TransactionCoordinator • Created (and started immmediately) when `KafkaServer` is requested to start up • Shut down when `KafkaServer` shuts down
`reporters`	Collection of MetricsReporter Used when…FIXME
`requestHandlerPool`	KafkaRequestHandlerPool
`socketServer`	SocketServer
`transactionCoordinator`	TransactionCoordinator
`quotaManagers`	QuotaManagers
`shutdownLatch`	java.util.concurrent.CountDownLatch
`startupComplete`	Flag for…FIXME
`zkUtils`	ZkUtils

Kafka Server and Periodic Tasks

Apache Kafka uses the Scheduler to schedule periodic tasks. Kafka internal services use the interface to schedule maintenance tasks.

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

Table 1. Tasks

Kafka Service	Task Name	Task
KafkaController	`delete-expired-tokens`	expireTokens
	`auto-leader-rebalance-task`	Enqueues AutoPreferredReplicaLe controller event to ControllerEventManage `KafkaController` is req scheduleAutoLeaderRe
GroupMetadataManager	`delete-expired-group-metadata`	cleanupGroupMetadata
	`__consumer_offsets-[offsetsPartition]`	loadGroupsAndOffsets
	`__consumer_offsets-[offsetsPartition]`	removeGroupsAndOffs
	`handleTxnCompletion-[producerId]`	handleTxnCompletion
TransactionCoordinator	`transaction-abort`	abortTimedOutTransact
TransactionStateManager	`transactionalId-expiration`	enableTransactionalIdE
	`load-txns-for-partition-[topicPartition]`	loadTransactions
	`remove-txns-for-partition-[topicPartition]`	removeTransactions
Log	`PeriodicProducerExpirationCheck`	removeExpiredProduce
	`flush-log`	flush
	`delete-file`	deleteSeg

LogManager	`kafka-log-retention`	cleanupLogs
	`kafka-log-flusher`	flushDirtyLogs
	`kafka-recovery-point-checkpoint`	checkpointLogRecovery
	`kafka-log-start-offset-checkpoint`	checkpointLogStartOffs
	`kafka-delete-logs`	deleteLogs
ReplicaManager	`highwatermark-checkpoint`	checkpointHighWaterm
	`isr-expiration`	maybeShrinkIsr
	`isr-change-propagation`	maybePropagateIsrCha
	`shutdown-idle-replica-alter-log-dirs-thread`	shutdownIdleReplicaAlt

AdminManager

`AdminManager` is the administration interface of KafkaServer.

`AdminManager` is created exclusively when `KafkaServer` is started.

Figure 1. AdminManager

`AdminManager` uses the MetadataCache for the following:

- createTopics (to get the getAliveBrokers)
- Describing the configuration of topic resources (to check whether a topic is available or not)

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

Tip	Enable `ALL` 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=ALL` Refer to Logging.

Creating AdminManager Instance

`AdminManager` takes the following when created:

- KafkaConfig
- Metrics
- MetadataCache

- KafkaZkClient

`AdminManager` initializes the internal registries and counters.

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` is requested to handle a CREATE_TOPICS request.

Describing Configuration Of Specified Resources (Topics and Brokers) — `describeConfigs` Method

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

For `TOPIC` resources, `describeConfigs` does the following:

1. Requests the AdminZkClient to fetch configuration of the given topics (from Zookeeper)
2. Uses `KafkaServer` utility to create the default log properties based on the KafkaConfig
3. Uses `LogConfig` utility to fromProps with the default log properties and the topic configuration (from Zookeeper)

For `BROKER` resources, `describeConfigs` does the following…FIXME

Note	`describeConfigs` is used exclusively when `KafkaApis` is requested to handle a DescribeConfigs request.

`alterConfigs` Method

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

`alterConfigs` ...FIXME

Note	`alterConfigs` is used exclusively when `KafkaApis` is requested to handleAlterConfigsRequest.

`configSynonyms` Internal Method

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

`configSynonyms` ...FIXME

Note	`configSynonyms` is used when...FIXME

`createPartitions` Method

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

`createPartitions` ...FIXME

Note	`createPartitions` is used when...FIXME

`tryCompleteDelayedTopicOperations` Method

```
tryCompleteDelayedTopicOperations(topic: String): Unit
```

`tryCompleteDelayedTopicOperations` ...FIXME

Note	`tryCompleteDelayedTopicOperations` is used when...FIXME

`incrementalAlterConfigs` Method

```
incrementalAlterConfigs(
  configs: Map[ConfigResource,
  List[AlterConfigOp]],
  validateOnly: Boolean): Map[ConfigResource, ApiError]
```

`incrementalAlterConfigs` ...FIXME

Note	`incrementalAlterConfigs` is used when...FIXME

`createTopicConfigEntry` Internal Method

```
createTopicConfigEntry(
  logConfig: LogConfig,
  topicProps: Properties,
  includeSynonyms: Boolean)(
    name: String,
    value: Any): DescribeConfigsResponse.ConfigEntry
```

`createTopicConfigEntry` ...FIXME

Note	`createTopicConfigEntry` is used when `AdminManager` is requested to describe the configuration of a topic and create topics.

`createBrokerConfigEntry` Internal Method

```
createBrokerConfigEntry(
  perBrokerConfig: Boolean,
  includeSynonyms: Boolean)(
    name: String,
    value: Any): DescribeConfigsResponse.ConfigEntry
```

`createBrokerConfigEntry` ...FIXME

Note	`createBrokerConfigEntry` is used when `AdminManager` is requested to...FIXME

Internal Properties

Name	Description
`adminZkClient`	
`alterConfigPolicy`	
`createTopicPolicy`	
`topicPurgatory`	

DelegationTokenManager

`DelegationTokenManager` is…FIXME

`DynamicConfigManager` uses the TokenChangedNotificationHandler that…FIXME

`expireTokens` Method

```
expireTokens(): Unit
```

`expireTokens` …FIXME

Note	`expireTokens` is used when…FIXME

TokenChangedNotificationHandler Object

`TokenChangedNotificationHandler` is a NotificationHandler that processNotification.

`processNotification` Method

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

Note	`processNotification` is part of the NotificationHandler Contract to…FIXME.

`processNotification` …FIXME

DynamicConfigManager — Dynamic Configuration Management

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

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

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

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

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

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

Creating DynamicConfigManager Instance

`DynamicConfigManager` takes the following to be created:

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

Note	changeExpirationMs seems never used.

`DynamicConfigManager` initializes the internal properties.

`startup` Method

```
startup
```

`startup` ...FIXME

Note	`startup` is used exclusively when `KafkaServer` is requested to start up.

ConfigChangedNotificationHandler Object

`ConfigChangedNotificationHandler` is a NotificationHandler that processNotification.

Processing Config Change Notifications — `processNotification` Method

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

Note	`processNotification` is part of the NotificationHandler Contract to handle a change notification.

`processNotification` ...FIXME

`processEntityConfigChangeVersion1` Internal Method

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

`processEntityConfigChangeVersion1` ...FIXME

Note	`processEntityConfigChangeVersion1` is used when...FIXME

`processEntityConfigChangeVersion2` Internal Method

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

`processEntityConfigChangeVersion2` ...FIXME

Note	`processEntityConfigChangeVersion2` is used when...FIXME

Shutting Down — `shutdown` Method

```
shutdown(): Unit
```

`shutdown` simply requests the ZkNodeChangeNotificationListener to close.

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

Internal Properties

Name	Description
`adminZkClient`	AdminZkClient (for the KafkaZkClient) Used when: • `ConfigChangedNotificationHandler` is requested to processEntityConfigChangeVersion1 and processEntityConfigChangeVersion2 • `DynamicConfigManager` is requested to start up
`configChangeListener`	ZkNodeChangeNotificationListener (for the KafkaZkClient and /config/changes path in Zookeeper, config_change sequence number prefix and the ConfigChangedNotificationHandler) Initialized when `DynamicConfigManager` is requested to start up Closed when `DynamicConfigManager` is requested to shut down

ConfigHandler Contract — Configuration Change Handlers

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

Table 1. ConfigHandler Contract

Method	Description

`processConfigChanges`	`processConfigChanges(`
	`  entityName: String,`
	`  value: Properties)`
	
	Processes configuration changes
	
	Used when:
	
	• `ConfigChangedNotificationHandler` is requested to `processEntityConfigChangeVersion1` and `processEntityConfigChangeVersion2` (that is a notification handler registered when `DynamicConfigManager` is created)
	
	• `DynamicConfigManager` is requested to start up

Table 2. ConfigHandlers

ConfigHandler	Description
BrokerConfigHandler	
ClientIdConfigHandler	
TopicConfigHandler	
UserConfigHandler	

BrokerConfigHandler

`BrokerConfigHandler` is a ConfigHandler that can process configuration changes for **brokers** entity type.

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

`BrokerConfigHandler` takes the following to be created:

- KafkaConfig
- QuotaManagers

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

Processing Configuration Changes — `processConfigChanges` Method

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

Note	`processConfigChanges` is part of the ConfigHandler Contract to process configuration changes.

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

All Brokers

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

Current Broker

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

`processConfigChanges` requests the QuotaManagers for the leader ReplicationQuotaManager that is in turn requested to updateQuota.

`processConfigChanges` requests the QuotaManagers for the follower ReplicationQuotaManager that is in turn requested to updateQuota.

`processConfigChanges` requests the QuotaManagers for the alterLogDirs ReplicationQuotaManager that is in turn requested to updateQuota.

ClientIdConfigHandler

`ClientIdConfigHandler` is...FIXME

Processing Configuration Changes — `processConfigChanges` Method

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

Note	`processConfigChanges` is part of the ConfigHandler Contract to process configuration changes.

`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.

Processing Configuration Changes — `processConfigChanges` Method

```
processConfigChanges(
  topic: String,
  topicConfig: Properties): Unit
```

Note	`processConfigChanges` is part of the ConfigHandler Contract to process configuration changes.

`processConfigChanges`…FIXME

UserConfigHandler

`UserConfigHandler` is...FIXME

Processing Configuration Changes — `processConfigChanges` Method

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

Note	`processConfigChanges` is part of the ConfigHandler Contract to process configuration changes.

`processConfigChanges` ...FIXME

DynamicBrokerConfig

`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.

Tip	Enable `ALL` logging levels for `kafka.server.DynamicBrokerConfig` logger to see what happens inside. Add the following line to `config/log4j.properties`: `log4j.logger.kafka.server.DynamicBrokerConfig=ALL` Refer to Logging.

AllDynamicConfigs

`AllDynamicConfigs` are…FIXME

Note	`AllDynamicConfigs` is used when: • `AdminManager` is requested to createBrokerConfigEntry • `DynamicBrokerConfig` utility is requested for the DynamicPasswordConfigs, to addDynamicConfigs and for the dynamicConfigUpdateModes

DynamicPasswordConfigs

`DynamicPasswordConfigs` are…FIXME

Note	`DynamicPasswordConfigs` is used when…FIXME

`addDynamicConfigs` Internal Utility

```
addDynamicConfigs(
  configDef: ConfigDef): Unit
```

`addDynamicConfigs` ...FIXME

Note	`addDynamicConfigs` is used when...FIXME

`dynamicConfigUpdateModes` Internal Utility

```
dynamicConfigUpdateModes: util.Map[String, String]
```

`dynamicConfigUpdateModes` ...FIXME

Note	`dynamicConfigUpdateModes` is used when...FIXME

`addBrokerReconfigurable` Method

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

`addBrokerReconfigurable` ...FIXME

Note	`addBrokerReconfigurable` is used when...FIXME

`processReconfiguration` Internal Method

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

`processReconfiguration` ...FIXME

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

`updateCurrentConfig` Internal Method

```
updateCurrentConfig(): Unit
```

`updateCurrentConfig` ...FIXME

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

`currentDynamicDefaultConfigs` Method

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

`currentDynamicDefaultConfigs` simply clones the dynamicDefaultConfigs.

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

Initializing (Fetching Broker Configuration From Zookeeper) — `initialize` Method

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

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

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

`initialize` requests the `AdminZkClient` to fetch the default configuration for **brokers** entities and updateDefaultConfig

`initialize` requests the `AdminZkClient` to fetch the configuration for the broker by the broker.id.

`initialize` maybeReEncodePasswords in the broker configuration and updateBrokerConfig with the broker.id.

Note	`initialize` is used exclusively when `KafkaServer` is requested to start up.

`updateDefaultConfig` Method

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

`updateDefaultConfig` ...FIXME

Note	`updateDefaultConfig` is used when: • `BrokerConfigHandler` is requested to processConfigChanges • `DynamicBrokerConfig` is requested to initialize

`updateBrokerConfig` Method

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

`updateBrokerConfig` ...FIXME

Note	`updateBrokerConfig` is used when: • `BrokerConfigHandler` is requested to processConfigChanges • `DynamicBrokerConfig` is requested to initialize

`maybeReEncodePasswords` Internal Method

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

`maybeReEncodePasswords` ...FIXME

Note	`maybeReEncodePasswords` is used when...FIXME

Registering (Adding) Reconfigurables For KafkaServer — `addReconfigurables` Method

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

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

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

`addReconfigurables` addBrokerReconfigurable with the LogCleaner (if configured).

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

`addReconfigurables` creates a DynamicMetricsReporters for the broker and addReconfigurable.

`addReconfigurables` creates a DynamicClientQuotaCallback for the broker and addReconfigurable.

`addReconfigurables` creates a DynamicListenerConfig with the input KafkaServer and addBrokerReconfigurable.

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

`validate` Method

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

`validate` ...FIXME

Note	`validate` is used exclusively when `AdminManager` is requested to alterConfigs (when `KafkaApis` is requested to handleAlterConfigsRequest).

`maybeReconfigure` Method

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

`maybeReconfigure` ...FIXME

Note	`maybeReconfigure` is used when: • `DynamicMetricsReporters` is requested to reconfigure • `DynamicClientQuotaCallback` is requested to reconfigure

`processReconfigurable` Method

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

`processReconfigurable` ...FIXME

Note	`processReconfigurable` is used when `DynamicBrokerConfig` is requested to processReconfiguration and processListenerReconfigurable.

`processListenerReconfigurable` Internal Method

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

`processListenerReconfigurable` ...FIXME

Note	`processListenerReconfigurable` is used when `DynamicBrokerConfig` is requested to reloadUpdatedFilesWithoutConfigChange and processReconfiguration.

`reloadUpdatedFilesWithoutConfigChange` Internal Method

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

`reloadUpdatedFilesWithoutConfigChange` ...FIXME

Note	`reloadUpdatedFilesWithoutConfigChange` is used exclusively when `AdminManager` is requested to alterConfigs (when `KafkaApis` is requested to handle a AlterConfigs request).

`clear` Method

```
clear(): Unit
```

`clear` ...FIXME

Note	`clear` is used exclusively when `KafkaServer` is requested to shut down.

`validatedKafkaProps` Method

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

`validatedKafkaProps` ...FIXME

Note	`validatedKafkaProps` is used when…FIXME

Registering Reconfigurable — `addReconfigurable` Method

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

`addReconfigurable` verifyReconfigurableConfigs of the Reconfigurable.

In the end, `addReconfigurable` adds the `Reconfigurable` to the reconfigurables internal registry.

Note	`addReconfigurable` is used when: • `DynamicBrokerConfig` is requested to addReconfigurables • `KafkaConfig` is requested to add a Reconfigurable

`verifyReconfigurableConfigs` Internal Method

```
verifyReconfigurableConfigs(
  configNames: Set[String]): Unit
```

`verifyReconfigurableConfigs` …FIXME

Note	`verifyReconfigurableConfigs` is used when…FIXME

Internal Properties

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

BrokerReconfigurable Contract — Reconfigurable Broker Services

`BrokerReconfigurable` is the contract of reconfigurable broker services that allow for self-reconfiguration.

`BrokerReconfigurable` interface is used exclusively by DynamicBrokerConfig.

Table 1. BrokerReconfigurable Contract

Property	Description
`reconfigurableConfigs`	`reconfigurableConfigs: Set[String]` Dynamic configurations Used when `DynamicBrokerConfig` is requested to addBrokerReconfigurable and processReconfiguration

`reconfigure`	`reconfigure(`
	`  oldConfig: KafkaConfig,`
	`  newConfig: KafkaConfig): Unit`
	
	Used when `DynamicBrokerConfig` is requested to updateCurrentConfig

`validateReconfiguration`	`validateReconfiguration(newConfig: KafkaConfig): Unit` Used exclusively when `DynamicBrokerConfig` is requested to processReconfiguration

Table 2. BrokerReconfigurables

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

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

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

DynamicConnectionQuota

`DynamicConnectionQuota` is a BrokerReconfigurable for the following dynamic configurations:

- max.connections.per.ip
- max.connections.per.ip.overrides

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

`DynamicConnectionQuota` takes a single KafkaServer to be created.

Reconfiguring — `reconfigure` Method

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

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

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

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

DynamicListenerConfig

`DynamicListenerConfig` is a BrokerReconfigurable for the following dynamic configurations:

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

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

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

`DynamicListenerConfig` takes a single KafkaServer to be created.

Reconfiguring — `reconfigure` Method

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

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

`reconfigure` ...FIXME

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

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

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

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

`validateReconfiguration` Method

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

Note	`validateReconfiguration` is part of the BrokerReconfigurable Contract to… FIXME.

`validateReconfiguration` …FIXME

DynamicThreadPool

`DynamicThreadPool` is a BrokerReconfigurable for the following dynamic configurations:

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

Tip	Use `kafka-configs` shell script to alter a dynamic configuration: `$ ./bin/kafka-configs.sh \ --bootstrap-server :9092 \ --alter \ --entity-type brokers \ --entity-name 0 \ --add-config num.recovery.threads.per.data.dir=2`

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

`DynamicThreadPool` takes a single KafkaServer to be created.

Reconfiguring — `reconfigure` Method

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

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

`reconfigure` ...FIXME

DynamicClientQuotaCallback

`DynamicClientQuotaCallback` is...FIXME

Reconfiguring — `reconfigure` Method

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

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

`reconfigure` ...FIXME

DynamicLogConfig

`DynamicLogConfig` is a Reconfigurable that…FIXME

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

`DynamicLogConfig` defines `KafkaConfigToLogConfigName` that are…FIXME…that is used for reconfigure.

Creating DynamicLogConfig Instance

`DynamicLogConfig` takes the following when created:

- LogManager
- KafkaServer

`DynamicLogConfig` initializes the internal registries and counters.

`reconfigurableConfigs` Method and `ReconfigurableConfigs` Value

```
reconfigurableConfigs: Set[String]
```

Note	`reconfigurableConfigs` is part of the BrokerReconfigurable contract to…FIXME.

`reconfigurableConfigs` simply returns the `ReconfigurableConfigs` that are the values of the TopicConfigSynonyms (of `LogConfig`) except log.message.format.version.

Note	`ReconfigurableConfigs` is used when: • `DynamicBrokerConfig` utility is requested for the AllDynamicConfigs • `DynamicLogConfig` is requested for the reconfigurableConfigs and to reconfigure

Reconfiguring — `reconfigure` Method

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

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

`reconfigure` ...FIXME

DynamicMetricsReporters

`DynamicMetricsReporters` is…FIXME

Creating DynamicMetricsReporters Instance

`DynamicMetricsReporters` takes the following to be created:

- Broker ID
- KafkaServer

`DynamicMetricsReporters` initializes the internal properties.

`reconfigure` Method

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

Note	`reconfigure` is part of the Reconfigurable Contract to reconfigure the Reconfigurable.

`reconfigure` …FIXME

`createReporters` Internal Method

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

`createReporters` …FIXME

Note	`createReporters` is used when `DynamicMetricsReporters` is created and reconfigure.

Internal Properties

Name	Description
`dynamicConfig`	DynamicBrokerConfig Used when…FIXME

FetchManager

`FetchManager` is…FIXME

GroupCoordinator

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

`GroupCoordinator` is also used for partitionFor.

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

Figure 1. GroupCoordinator's Startup

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

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

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

Creating GroupCoordinator Instance — `apply` Factory Method

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

1. Creates a `DelayedOperationPurgatory` and a `DelayedOperationPurgatory`

`apply` ...FIXME

Note	`apply` is used exclusively when `KafkaServer` is requested to start up.

Creating GroupCoordinator Instance

`GroupCoordinator` takes the following to be created:

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

`GroupCoordinator` initializes the internal registries and counters.

Starting Up — `startup` Method

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

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

```
Starting up.
```

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

`startup` turns the isActive internal flag on.

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

```
Startup complete.
```

Note	`startup` is used exclusively when `KafkaServer` is requested to start up.

`partitionFor` Method

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

`partitionFor` simply requests the GroupMetadataManager to partitionFor the given `group` ID.

Note	`partitionFor` is used when: • `KafkaApis` is requested to handle an AddOffsetsToTxn and FindCoordinatorRequest requests • `GroupCoordinator` is requested to prepareRebalance and onCompleteJoin

`handleCommitOffsets` Method

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

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

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

If the GroupMetadataManager could not getGroup, `handleCommitOffsets` …FIXME

In case of an error while validateGroupStatus, `handleCommitOffsets` …FIXME

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

`doCommitOffsets` Internal Method

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

`doCommitOffsets` ...FIXME

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

`onCompleteJoin` Method

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

`onCompleteJoin` ...FIXME

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

`doSyncGroup` Internal Method

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

`doSyncGroup` ...FIXME

Note	`doSyncGroup` is used when...FIXME

`handleDescribeGroup` Method

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

`handleDescribeGroup` ...FIXME

Note	`handleDescribeGroup` is used exclusively when `KafkaApis` is requested to handleDescribeGroupRequest.

`handleGroupImmigration` Method

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

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

Note	`handleGroupImmigration` is used exclusively when `KafkaApis` is requested to handle a LeaderAndIsrRequest.

`handleGroupEmigration` Method

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

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

Note	`handleGroupEmigration` is used when `KafkaApis` is requested to handle a LeaderAndIsrRequest and a StopReplicaRequest.

Handling Deleted Partitions — `handleDeletedPartitions` Method

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

`handleDeletedPartitions` simply requests the GroupMetadataManager to cleanupGroupMetadata and...FIXME

Note	`handleDeletedPartitions` is used when...FIXME

`handleFetchOffsets` Method

```
handleFetchOffsets(
  groupId: String,
  partitions: Option[Seq[TopicPartition]] = None):
(Errors, Map[TopicPartition, OffsetFetchResponse.PartitionData])
```

`handleFetchOffsets` ...FIXME

Note	`handleFetchOffsets` is used when...FIXME

`handleListGroups` Method

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

`handleListGroups` ...FIXME

Note	`handleListGroups` is used when...FIXME

`handleJoinGroup` Method

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

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

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

`handleJoinGroup` requests the GroupMetadataManager to getGroup by the given `groupId`.

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

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

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

`handleSyncGroup` Method

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

`handleSyncGroup` ...FIXME

Note	`handleSyncGroup` is used when...FIXME

`handleDeleteGroups` Method

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

`handleDeleteGroups` ...FIXME

Note	`handleDeleteGroups` is used when...FIXME

`handleHeartbeat` Method

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

`handleHeartbeat` ...FIXME

Note	`handleHeartbeat` is used when...FIXME

`handleLeaveGroup` Method

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

`handleLeaveGroup` ...FIXME

Note	`handleLeaveGroup` is used when...FIXME

`scheduleHandleTxnCompletion` Method

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

`scheduleHandleTxnCompletion` ...FIXME

Note	`scheduleHandleTxnCompletion` is used exclusively when `KafkaApis` is requested to handleWriteTxnMarkersRequest.

`handleTxnCommitOffsets` Method

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

`handleTxnCommitOffsets` ...FIXME

Note	`handleTxnCommitOffsets` is used when...FIXME

`onGroupLoaded` Internal Callback

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

`onGroupLoaded` ...FIXME

Note	`onGroupLoaded` is used when...FIXME

`onGroupUnloaded` Internal Callback

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

`onGroupUnloaded` ...FIXME

Note	`onGroupUnloaded` is used when...FIXME

Validating Group Status — `validateGroupStatus` Internal Method

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

`validateGroupStatus` ...FIXME

Note	`validateGroupStatus` is used when...FIXME

`doJoinGroup` Internal Method

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

`doJoinGroup` ...FIXME

Note	`doJoinGroup` is used exclusively when `GroupCoordinator` is requested to handleJoinGroup.

`prepareRebalance` Internal Method

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

`prepareRebalance` ...FIXME

Note	`prepareRebalance` is used exclusively when `GroupCoordinator` is requested to maybePrepareRebalance.

`maybePrepareRebalance` Internal Method

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

`maybePrepareRebalance` ...FIXME

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

`addMemberAndRebalance` Internal Method

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

`addMemberAndRebalance` ...FIXME

Note	`addMemberAndRebalance` is used exclusively when `GroupCoordinator` is requested to doJoinGroup.

`removeMemberAndUpdateGroup` Internal Method

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

`removeMemberAndUpdateGroup` ...FIXME

Note	`removeMemberAndUpdateGroup` is used when `GroupCoordinator` is requested to handleLeaveGroup and onExpireHeartbeat.

`onExpireHeartbeat` Method

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

`onExpireHeartbeat` ...FIXME

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

`offsetsTopicConfigs` Method

```
offsetsTopicConfigs: Properties
```

`offsetsTopicConfigs` ...FIXME

Note	`offsetsTopicConfigs` is used when...FIXME

GroupMetadataManager

`GroupMetadataManager` is created exclusively when `GroupCoordinator` is created.

Figure 1. Creating GroupMetadataManager

`GroupMetadataManager` is a KafkaMetricsGroup and registers performance metrics.

Table 1. GroupMetadataManager's Performance Metrics

Metric Name	Description
`NumGroups`	
`NumGroupsCompletingRebalance`	
`NumGroupsDead`	
`NumGroupsEmpty`	
`NumGroupsPreparingRebalance`	
`NumGroupsStable`	
`NumOffsets`	

The performance metrics are registered in **kafka.coordinator.group:type=GroupMetadataManager** group.

Figure 2. GroupMetadataManager in jconsole

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

Name	Description
`compressionType`	`CompressionType` (default: `NoCompressionCodec`)
`groupMetadataCache`	GroupMetadata by group ID
`groupMetadataTopicPartitionCount`	Number of partitions for the `__consumer_offsets` consumer group metadata topic (default: `50`)
`scheduler`	KafkaScheduler with 1 daemon thread with **group-metadata-manager-** prefix Started when `GroupMetadataManager` is requested to start up. Shut down when `GroupMetadataManager` is requested to shut down. Used when: • startup (with `enableMetadataExpiration` flag on) that schedules the **delete-expired-group-metadata** task that cleanupGroupMetadata every `600000L` millis • scheduleLoadGroupAndOffsets that loads offsets and group metadata from __consumer_offsets topic • removeGroupsForPartition that unloads offsets and group metadata from __consumer_offsets topic • scheduleHandleTxnCompletion that schedules the **handleTxnCompletion-[producerId]** task that handleTxnCompletion

`enableMetadataExpiration` Method

```
enableMetadataExpiration(): Unit
```

`enableMetadataExpiration` requests KafkaScheduler to start.

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

Note	`enableMetadataExpiration` is used exclusively when `GroupCoordinator` is started.

Creating GroupMetadataManager Instance

`GroupMetadataManager` takes the following when created:

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

`GroupMetadataManager` initializes the internal registries and counters.

`cleanupGroupMetadata` Internal Method

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

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

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

1. FIXME

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

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

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

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

```
getGroupMetadataTopicPartitionCount: Int
```

`getGroupMetadataTopicPartitionCount` requests the KafkaZkClient for getTopicPartitionCount of `__consumer_offsets` consumer group metadata topic.

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

Note	`getGroupMetadataTopicPartitionCount` is used exclusively when `GroupMetadataManager` is requested for groupMetadataTopicPartitionCount.

`shutdown` Method

```
shutdown(): Unit
```

`shutdown` ...FIXME

Note	`shutdown` is used when...FIXME

`startup` Method

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

`startup` ...FIXME

Note	`startup` is used when...FIXME

`scheduleLoadGroupAndOffsets` Method

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

`scheduleLoadGroupAndOffsets` ...FIXME

Note	`scheduleLoadGroupAndOffsets` is used when...FIXME

`removeGroupsForPartition` Method

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

`removeGroupsForPartition` ...FIXME

Note	`removeGroupsForPartition` is used when...FIXME

`scheduleHandleTxnCompletion` Method

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

`scheduleHandleTxnCompletion` ...FIXME

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

`loadGroupsAndOffsets` Method

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

`loadGroupsAndOffsets` ...FIXME

Note	`loadGroupsAndOffsets` is used when...FIXME

`removeGroupsAndOffsets` Internal Method

```
removeGroupsAndOffsets(): Unit
```

`removeGroupsAndOffsets` ...FIXME

Note	`removeGroupsAndOffsets` is used when...FIXME

`handleTxnCompletion` Method

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

`handleTxnCompletion` ...FIXME

Note	`handleTxnCompletion` is used exclusively when `GroupMetadataManager` is requested to scheduleHandleTxnCompletion.

`partitionFor` Method

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

`partitionFor` ...FIXME

Note	`partitionFor` is used when...FIXME

`storeOffsets` Method

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

`storeOffsets` ...FIXME

Note	`storeOffsets` is used exclusively when `GroupCoordinator` is requested to doCommitOffsets.

`storeGroup` Method

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

`storeGroup` ...FIXME

Note	`storeGroup` is used exclusively when `GroupCoordinator` is requested to doSyncGroup and onCompleteJoin.

Requesting ReplicaManager to Append Records — `appendForGroup` Internal Method

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

`appendForGroup` simply requests the ReplicaManager to append records.

Note	`appendForGroup` is used exclusively when `GroupMetadataManager` is requested to storeGroup and storeOffsets.

`addGroup` Method

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

`addGroup` ...FIXME

Note	`addGroup` is used when...FIXME

Getting Metadata of Group by Group ID — `getGroup` Method

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

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

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

Note	`getGroup` is used when: • `GroupCoordinator` is requested to handleJoinGroup, handleSyncGroup, handleLeaveGroup, handleDeleteGroups, handleHeartbeat, handleTxnCommitOffsets, handleCommitOffsets, and handleDescribeGroup • `GroupMetadataManager` is requested to groupNotExists (when `GroupCoordinator` is requested to handleDeleteGroups) and handleTxnCompletion (when `GroupCoordinator` is requested to scheduleHandleTxnCompletion)

`readGroupMessageValue` Method

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

`readGroupMessageValue` ...FIXME

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

`doLoadGroupsAndOffsets` Internal Method

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

`doLoadGroupsAndOffsets` ...FIXME

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

`groupNotExists` Method

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

`groupNotExists` ...FIXME

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

GroupMetadata — Metadata of Consumer Group

`GroupMetadata` is the metadata of a consumer group.

`GroupMetadata` takes the following to be created:

- Group ID
- GroupState
- `Time`

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

Table 1. States of Consumer Group

State	Description
`CompletingRebalance`	Transitioned to when: • initNextGeneration (and there are any members)
`Dead`	Transitioned to when: • `GroupCoordinator` is requested to handleDeleteGroups (when in Empty state) • `GroupCoordinator` is requested to onGroupUnloaded • `GroupMetadataManager` is requested to cleanupGroupMetadata
`Empty`	Transitioned to when: • initNextGeneration (and there are no members) • `GroupMetadataManager` is requested to cleanupGroupMetadata (when in Empty state with no offsets) • `GroupMetadataManager` is requested to readGroupMessageValue (and there are no members)
`PreparingRebalance`	Transitioned to when: • `GroupCoordinator` is requested to prepareRebalance
`Stable`	Transitioned to when: • `GroupCoordinator` is requested to doSyncGroup (when in CompletingRebalance state and the generationId did not change) • `GroupMetadataManager` is requested to readGroupMessageValue (and there is at least one member)

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

Table 2. Valid Previous States

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

`GroupMetadata` is created when:

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

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

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

Registering Consumer with Consumer Group — `add` Method

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

`add` ...FIXME

Note	`add` is used when: • `GroupCoordinator` is requested to addMemberAndRebalance • `GroupMetadata` is requested to loadGroup

Loading Consumer Group Metadata — `loadGroup` Factory Method

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

`loadGroup` ...FIXME

Note	`loadGroup` is used exclusively when `GroupMetadataManager` is requested to readGroupMessageValue.

Unregistering Consumer from Consumer Group — `remove` Method

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

`remove` ...FIXME

Note	`remove` is used when `GroupCoordinator` is requested to removeMemberAndUpdateGroup and onCompleteJoin.

Changing State of Consumer Group — `transitionTo` Method

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

`transitionTo` ...FIXME

Note	`transitionTo` is used when...FIXME

`initNextGeneration` Method

```
initNextGeneration(): Unit
```

`initNextGeneration` ...FIXME

Note	`initNextGeneration` is used exclusively when `GroupCoordinator` is requested to onCompleteJoin.

Asserting No Offets in Use — `hasOffsets` Method

```
hasOffsets: Boolean
```

`hasOffsets` ...FIXME

Note	`hasOffsets` is used exclusively when `GroupMetadataManager` is requested to cleanupGroupMetadata

Kafka — Standalone Command-Line Application

`kafka.Kafka` is a standalone command-line application that starts a Kafka broker.

`kafka.Kafka` is started using kafka-server-start.sh shell script.

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

`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	`main` uses Java's Runtime.addShutdownHook to register the shutdown hook.

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

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

Note	`main` uses Java's System.exit to terminate a JVM.

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

```
registerLoggingSignalHandler(): Unit
```

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

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

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

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

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

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

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

KafkaApis — API Request Handler

`KafkaApis` is responsible for handling requests to a Kafka broker (by means of handlers).

`KafkaApis` is created when `KafkaServer` is requested to start (for KafkaRequestHandlerPool).

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

When created, `KafkaApis` is given broker services (from the owning KafkaServer) that are used to handle requests (from producers, consumers, brokers, administrative clients, etc.)

Some requests are meant for the controller broker and simply do nothing (*no-ops*) when received by a regular non-controller broker.

Table 1. KafkaApis's API Keys and Handlers

API Key	Handler
`AddOffsetsToTxn`	handleAddOffsetsToTxnRequest
`AddPartitionsToTxn`	handleAddPartitionToTxnRequest
`AlterConfigs`	handleAlterConfigsRequest
`AlterPartitionReassignments`	handleAlterPartitionReassignmentsRequest
`AlterReplicaLogDirs`	handleAlterReplicaLogDirsRequest
`ApiVersions`	handleApiVersionsRequest
`ControlledShutdown`	handleControlledShutdownRequest

`CreateAcls`	handleCreateAcls
`CreateDelegationToken`	handleCreateTokenRequest
`CreatePartitions`	handleCreatePartitionsRequest
`CreateTopics`	handleCreateTopicsRequest
`DeleteAcls`	handleDeleteAcls
`DeleteGroups`	handleDeleteGroupsRequest
`DeleteRecords`	handleDeleteRecordsRequest
`DeleteTopics`	handleDeleteTopicsRequest
`DescribeConfigs`	handleDescribeConfigsRequest
`DescribeGroups`	handleDescribeGroupRequest
`DescribeLogDirs`	handleDescribeLogDirsRequest
`DescribeAcls`	handleDescribeAcls
`DescribeDelegationToken`	handleDescribeTokensRequest
`ElectLeaders`	handleElectReplicaLeader
`EndTxn`	handleEndTxnRequest
`Fetch`	handleFetchRequest
`ExpireDelegationToken`	handleExpireTokenRequest
`FindCoordinator`	handleFindCoordinatorRequest
`Heartbeat`	handleHeartbeatRequest
`InitProducerId`	handleInitProducerIdRequest
`IncrementalAlterConfigs`	handleIncrementalAlterConfigsRequest
`JoinGroup`	handleJoinGroupRequest

`LeaderAndIsr`	Cluster action / handleLeaderAndIsrRequest
`LeaveGroup`	handleLeaveGroupRequest
`ListGroups`	handleListGroupsRequest
`ListOffsets`	handleListOffsetRequest
`ListPartitionReassignments`	handleListPartitionReassignmentsRequest
`Metadata`	handleTopicMetadataRequest
`OffsetCommit`	handleOffsetCommitRequest
`OffsetDelete`	handleOffsetDeleteRequest
`OffsetFetch`	handleOffsetFetchRequest
`OffsetForLeaderEpoch`	handleOffsetForLeaderEpochRequest
`Produce`	handleProduceRequest
`RenewDelegationToken`	handleRenewTokenRequest
`SaslAuthenticate`	handleSaslAuthenticateRequest
`SaslHandshake`	handleSaslHandshakeRequest
`StopReplica`	Cluster action / handleStopReplicaRequest
`SyncGroup`	handleSyncGroupRequest
`TxnOffsetCommit`	handleTxnOffsetCommitRequest
`UpdateMetadata`	Cluster action / handleUpdateMetadataRequest
`WriteTxnMarkers`	handleWriteTxnMarkersRequest

Tip	Enable `ALL` logging levels for `kafka.server.KafkaApis` logger to see what happens inside.
	
	Add the following line to `config/log4j.properties`:
	
	`log4j.logger.kafka.server.KafkaApis=ALL`
	
	Refer to Logging.
	
	--- 	
	
	Please note that Kafka comes with a preconfigured `kafka.server.KafkaApis` logger in `config/log4j.properties`:
	
	`log4j.appender.requestAppender=org.apache.log4j.DailyRollingFileAppender`
	`log4j.appender.requestAppender.DatePattern='.'yyyy-MM-dd-HH`
	`log4j.appender.requestAppender.File=${kafka.logs.dir}/kafka-request.log`
	`log4j.appender.requestAppender.layout=org.apache.log4j.PatternLayout`
	`log4j.appender.requestAppender.layout.ConversionPattern=[%d] %p %m (%c)%n`
	
	`log4j.logger.kafka.server.KafkaApis=TRACE, requestAppender`
	`log4j.additivity.kafka.server.KafkaApis=false`
	
	That means that the logs of `KafkaApis` go to `logs/kafka-request.log` file at `TRACE` logging level and are not added to the main logs (per `log4j.additivity` being off).

Creating KafkaApis Instance

`KafkaApis` takes the following to be created:

- RequestChannel
- ReplicaManager
- AdminManager
- GroupCoordinator
- TransactionCoordinator
- KafkaController
- KafkaZkClient
- Broker ID
- KafkaConfig
- MetadataCache

- Metrics
- Optional Authorizer
- QuotaManagers
- FetchManager
- BrokerTopicStats
- Cluster ID
- `Time`
- DelegationTokenManager

KafkaApis and Authorizer

`KafkaApis` may be given an Authorizer when created.

The `Authorizer` is used for the following:

- handleCreateAcls
- handleDeleteAcls
- handleDescribeAcls
- authorize, filterAuthorized, and authorizedOperations

KafkaApis and AdminZkClient

```
adminZkClient: AdminZkClient
```

When created, `KafkaApis` creates a AdminZkClient that is used to create a topic for the following handlers:

- handleFindCoordinatorRequest (for `consumer_offsets` and `transaction_state` internal topics)
- handleTopicMetadataRequest (for `consumer_offsets` and `transaction_state` internal topics or any other topic with auto-creation enabled)

Routing API Requests to Respective Handlers — `handle` Method

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

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

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

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

Note	`handle` is used exclusively when `KafkaRequestHandler` thread is requested to run.

Handling ElectLeaders Request — `handleElectReplicaLeader` Handler

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

Caution	FIXME Describe me again

In summary, `handleElectPreferredReplicaLeader` requests the ReplicaManager to electPreferredLeaders.

Internally, `handleElectPreferredReplicaLeader` …FIXME

Note	`handleElectPreferredReplicaLeader` is used exclusively when `KafkaApis` is requested to handle a ElectPreferredLeaders request.

Handling LeaderAndIsr Request — `handleLeaderAndIsrRequest` Handler

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

In summary, `handleLeaderAndIsrRequest` requests the ReplicaManager to become the leader or a follower (of partitions).

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

When authorized for cluster action and not isBrokerEpochStale, `handleLeaderAndIsrRequest` requests the ReplicaManager to become the leader or a follower (of partitions) (with the onLeadershipChange callback that gets the Partitions of which the broker is a new leader or a follower).

`handleLeaderAndIsrRequest` isBrokerEpochStale and…FIXME

Note	`handleLeaderAndIsrRequest` is used exclusively when `KafkaApis` is requested to handle a LeaderAndIsr request.

`onLeadershipChange` Callback

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

`onLeadershipChange` …FIXME

Handling AlterReplicaLogDirs Request — `handleAlterReplicaLogDirsRequest` Handler

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

In summary, `handleAlterReplicaLogDirsRequest` requests the ReplicaManager to alterReplicaLogDirs.

`handleAlterReplicaLogDirsRequest` …FIXME

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

Handling CreateTopics Request — `handleCreateTopicsRequest` Handler

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

`handleCreateTopicsRequest` …FIXME

`handleCreateTopicsRequest` checks whether KafkaController is active…FIXME

`handleCreateTopicsRequest` authorizes the `Create` operation for `ClusterResource` …FIXME

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

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

Handling OffsetFetch Request — `handleOffsetFetchRequest` Handler

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

`handleOffsetFetchRequest` ...FIXME

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

Handling Fetch Request — `handleFetchRequest` Handler

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

In summary, `handleFetchRequest` requests the ReplicaManager to fetch messages.

`handleFetchRequest` ...FIXME

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

Handling CreateAcls Request — `handleCreateAcls` Handler

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

`handleCreateAcls` ...FIXME

Note	`handleCreateAcls` is used when `KafkaApis` is requested to handle a CreateAcls request.

Handling Metadata Request — `handleTopicMetadataRequest` Handler

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

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

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

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

For every authorized topic, `handleTopicMetadataRequest` ...FIXME

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

`handleTopicMetadataRequest` getTopicMetadata if there are `authorizedTopics`.

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

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

In the end, `handleTopicMetadataRequest` sendResponseMaybeThrottle with a new MetadataResponse.

Note	`handleTopicMetadataRequest` is used exclusively when `KafkaApis` is requested to handle a Metadata request.

Authorizing Request for Operation on Resource — `authorize` Internal Method

```
authorize(
  request: RequestChannel.Request,
  operation: AclOperation,
  resourceType: ResourceType,
  resourceName: String,
  logIfAllowed: Boolean = true,
  logIfDenied: Boolean = true,
  refCount: Int = 1): Boolean
```

`authorize` simply requests the Authorizer (when defined) to authorize the given `AclOperation` on a broker resource (described by the `ResourceType` and `resourceName`).

`authorize` is positive (`true`) when the `Authorizer` returned `ALLOWED` .

Note	The Authorizer is created in KafkaServer (when the `KafkaApis` is created). It is configured using authorizer.class.name configuration property which is empty by default and so all operations are authorized.

Request	AclOperation	Resource Type	Resource Name
OffsetCommit	`READ`	`GROUP`	groupId
Produce	`WRITE`	`TRANSACTIONAL_ID`	transactionalId
Produce	`IDEMPOTENT_WRITE`	`CLUSTER`	kafka-cluster
Fetch (from followers)	`CLUSTER_ACTION`	`CLUSTER`	kafka-cluster
Metadata (for auto-create topics)	`CREATE`	`CLUSTER`	kafka-cluster
Metadata	`DESCRIBE`	`CLUSTER`	kafka-cluster
OffsetFetch	`DESCRIBE`	`CLUSTER`	kafka-cluster
FindCoordinator	`DESCRIBE`	`GROUP`	Coordinator key
FindCoordinator	`DESCRIBE`	`TRANSACTIONAL_ID`	Coordinator key
DescribeGroups	`DESCRIBE`	`GROUP`	Group ID
ListGroups	`DESCRIBE`	`CLUSTER`	kafka-cluster
ListGroups	`DESCRIBE`	`GROUP`	Group ID

JoinGroup	`READ`	`GROUP`	Group ID
SyncGroup	`READ`	`GROUP`	Group ID
DeleteGroups	`DELETE`	`GROUP`	Group ID
Heartbeat	`READ`	`GROUP`	Group ID
LeaveGroup	`READ`	`GROUP`	Group ID
CreateTopics	`CREATE`	`CLUSTER`	kafka-cluster
InitProducerId	`WRITE`	`TRANSACTIONAL_ID`	Transactional ID
InitProducerId	`IDEMPOTENT_WRITE`	`CLUSTER`	kafka-cluster
EndTxn	`WRITE`	`TRANSACTIONAL_ID`	Transactional ID
AddPartitionsToTxn	`WRITE`	`TRANSACTIONAL_ID`	Transactional ID
AddOffsetsToTxn	`WRITE`	`TRANSACTIONAL_ID`	Transactional ID
AddOffsetsToTxn	`READ`	`GROUP`	Group ID
TxnOffsetCommit	`WRITE`	`TRANSACTIONAL_ID`	Transactional ID
TxnOffsetCommit	`READ`	`GROUP`	Group ID
OffsetForLeaderEpoch	`CLUSTER_ACTION`	`CLUSTER`	kafka-cluster
AlterConfigs (for brokers)	`ALTER_CONFIGS`	`CLUSTER`	kafka-cluster
AlterConfigs (for topics)	`ALTER_CONFIGS`	`TOPIC`	Topic name
IncrementalAlterConfigs (for brokers)	`ALTER_CONFIGS`	`CLUSTER`	kafka-cluster
IncrementalAlterConfigs (for topics)	`ALTER_CONFIGS`	`TOPIC`	Topic name

DescribeConfigs (for brokers)	`DESCRIBE_CONFIGS`	`CLUSTER`	kafka-cluster
DescribeConfigs (for topics)	`DESCRIBE_CONFIGS`	`TOPIC`	Topic name
AlterReplicaLogDirs	`ALTER`	`CLUSTER`	kafka-cluster
DescribeLogDirs	`DESCRIBE`	`CLUSTER`	kafka-cluster
DescribeDelegationToken	`DESCRIBE`	`DELEGATION_TOKEN`	Token ID
ElectLeaders	`ALTER`	`CLUSTER`	kafka-cluster
OffsetDelete	`DELETE`	`GROUP`	Group ID

Handling CreatePartitions Request — `handleCreatePartitionsRequest` Handler

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

`handleCreatePartitionsRequest` ...FIXME

Note	`handleCreatePartitionsRequest` is used when...FIXME

Handling DeleteTopics Request — `handleDeleteTopicsRequest` Handler

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

`handleDeleteTopicsRequest` ...FIXME

Note	`handleDeleteTopicsRequest` is used when...FIXME

Handling TxnOffsetCommit Request — `handleTxnOffsetCommitRequest` Handler

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

`handleTxnOffsetCommitRequest` ...FIXME

Note	`handleTxnOffsetCommitRequest` is used when...FIXME

Handling SyncGroup Request — `handleSyncGroupRequest` Handler

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

`handleSyncGroupRequest` ...FIXME

Note	`handleSyncGroupRequest` is used when...FIXME

Handling SaslHandshake Request — `handleSaslHandshakeRequest` Handler

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

`handleSaslHandshakeRequest` ...FIXME

Note	`handleSaslHandshakeRequest` is used when...FIXME

Handling SaslAuthenticate Request — `handleSaslAuthenticateRequest` Handler

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

`handleSaslAuthenticateRequest` ...FIXME

Note	`handleSaslAuthenticateRequest` is used when...FIXME

Handling AddPartitionToTxn Request — `handleAddPartitionToTxnRequest` Handler

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

`handleAddPartitionToTxnRequest` ...FIXME

Note	`handleAddPartitionToTxnRequest` is used when...FIXME

Handling ApiVersions Request — `handleApiVersionsRequest` Handler

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

`handleApiVersionsRequest` ...FIXME

Note	`handleApiVersionsRequest` is used when...FIXME

Handling CreateToken Request — `handleCreateTokenRequest` Handler

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

`handleCreateTokenRequest` ...FIXME

Note	`handleCreateTokenRequest` is used when...FIXME

Handling DeleteAcls Request — `handleDeleteAcls` Handler

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

`handleDeleteAcls` ...FIXME

Note	`handleDeleteAcls` is used when...FIXME

Handling DeleteGroups Request — `handleDeleteGroupsRequest` Handler

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

`handleDeleteGroupsRequest` ...FIXME

Note	`handleDeleteGroupsRequest` is used when...FIXME

Handling DescribeConfigs Request — `handleDescribeConfigsRequest` Handler

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

`handleDescribeConfigsRequest` takes the DescribeConfigsRequest from (the body of) the given RequestChannel.Request.

`handleDescribeConfigsRequest` authorizes the DescribeConfigs operation on the broker and topic resources (of the DescribeConfigsRequest).

For every authorized operation, `handleDescribeConfigsRequest` requests the AdminManager to describeConfigs.

In the end, `handleDescribeConfigsRequest` sendResponseMaybeThrottle with a new `DescribeConfigsResponse`.

Note	`handleDescribeConfigsRequest` is used exclusively when `KafkaApis` is requested to handle a DescribeConfigs request.

Handling DescribeAcls Request — `handleDescribeAcls` Handler

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

`handleDescribeAcls` authorizeClusterOperation for the `DESCRIBE` operation.

`handleDescribeAcls` ...FIXME

Note	`handleDescribeAcls` is used when...FIXME

Handling DescribeTokens Request — `handleDescribeTokensRequest` Handler

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

`handleDescribeTokensRequest` ...FIXME

Note	`handleDescribeTokensRequest` is used when...FIXME

Handling EndTxn Request — `handleEndTxnRequest` Handler

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

`handleEndTxnRequest` ...FIXME

Note	`handleEndTxnRequest` is used when...FIXME

Handling ExpireToken Request — `handleExpireTokenRequest` Handler

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

`handleExpireTokenRequest` ...FIXME

Note	`handleExpireTokenRequest` is used when...FIXME

Handling Heartbeat Request — `handleHeartbeatRequest` Handler

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

`handleHeartbeatRequest` ...FIXME

Note	`handleHeartbeatRequest` is used when...FIXME

Handling InitProducerId Request — `handleInitProducerIdRequest` Handler

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

`handleInitProducerIdRequest` ...FIXME

Note	`handleInitProducerIdRequest` is used when...FIXME

Handling IncrementalAlterConfigs Request — `handleIncrementalAlterConfigsRequest` Handler

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

`handleIncrementalAlterConfigsRequest` ...FIXME

Note	`handleIncrementalAlterConfigsRequest` is used when...FIXME

Handling ListGroups Request — `handleListGroupsRequest` Handler

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

`handleListGroupsRequest` ...FIXME

Note	`handleListGroupsRequest` is used when...FIXME

Handling LeaveGroup Request — `handleLeaveGroupRequest` Handler

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

`handleLeaveGroupRequest` ...FIXME

Note	`handleLeaveGroupRequest` is used when...FIXME

Handling RenewToken Request — `handleRenewTokenRequest` Handler

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

`handleRenewTokenRequest` ...FIXME

Note	`handleRenewTokenRequest` is used when...FIXME

Handling ControlledShutdown Request — `handleControlledShutdownRequest` Handler

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

`handleControlledShutdownRequest` ...FIXME

Note	`handleControlledShutdownRequest` is used when...FIXME

`fetchOffsetForTimestamp` Internal Method

```
fetchOffsetForTimestamp(topicPartition: TopicPartition, timestamp: Long): Option[Times
tampOffset]
```

`fetchOffsetForTimestamp` ...FIXME

Note	`fetchOffsetForTimestamp` is used exclusively when `KafkaApis` is requested to handleListOffsetRequestV1AndAbove.

`handleListOffsetRequestV0` Internal Method

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

`handleListOffsetRequestV0` ...FIXME

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

`handleListOffsetRequestV1AndAbove` Internal Method

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

`handleListOffsetRequestV1AndAbove` ...FIXME

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

Handling DescribeLogDirs Request — `handleDescribeLogDirsRequest` Handler

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

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

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

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

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

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

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

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

Note	`handleDescribeLogDirsRequest` is used exclusively when `KafkaApis` is requested to handle a DescribeLogDirs request.

`sendResponseMaybeThrottle` Internal Method

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

`sendResponseMaybeThrottle` ...FIXME

Note	`sendResponseMaybeThrottle` is used when...FIXME

`fetchOffsetsBefore` Method

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

`fetchOffsetsBefore` ...FIXME

Note	`fetchOffsetsBefore` is used exclusively when `KafkaApis` is requested to fetchOffsets.

`fetchOffsets` Method

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

`fetchOffsets` ...FIXME

Note	`fetchOffsets` is used exclusively when `KafkaApis` is requested to handleListOffsetRequestV0.

Handling StopReplica Request — `handleStopReplicaRequest` Handler

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

In summary, `handleStopReplicaRequest` requests the ReplicaManager to stopReplicas.

`handleStopReplicaRequest` ...FIXME

Note	`handleStopReplicaRequest` is used exclusively when `KafkaApis` is requested to handle a StopReplica request.

Handling UpdateMetadata Request (From Kafka Controller) — `handleUpdateMetadataRequest` Handler

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

`handleUpdateMetadataRequest` takes the UpdateMetadataRequest from (the body of) the RequestChannel.Request.

When authorized for cluster action and not isBrokerEpochStale, `handleUpdateMetadataRequest` requests the following:

- ReplicaManager to maybeUpdateMetadataCache (that gives deleted partitions)
- GroupCoordinator to handle the deleted partitions
- AdminManager to tryCompleteDelayedTopicOperations for all the topics (based on the partitionStates of the `UpdateMetadataRequest`)

`handleUpdateMetadataRequest` updates quotas…FIXME

`handleUpdateMetadataRequest` requests the ReplicaManager to tryCompleteElection for every partition (based on the partitionStates of the `UpdateMetadataRequest`).

In the end, `handleUpdateMetadataRequest` sendResponseExemptThrottle (with a no-error `UpdateMetadataResponse`).

Note	`handleUpdateMetadataRequest` is used exclusively when `KafkaApis` is requested to handle a UpdateMetadata request.

Handling OffsetCommitRequest — `handleOffsetCommitRequest` Handler

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

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

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

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

If not authorized, `handleOffsetCommitRequest` ...FIXME

Note	`handleOffsetCommitRequest` is used exclusively when `KafkaApis` is requested to handle an OffsetCommit request.

`sendResponseCallback` Method

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

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

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

In the end, `sendResponseCallback` sendResponseMaybeThrottle a new `OffsetCommitResponse`.

`createInternalTopic` Internal Method

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

`createInternalTopic` ...FIXME

Note	`createInternalTopic` is used when `KafkaApis` is requested to getOrCreateInternalTopic and getTopicMetadata (for metadata of `consumer_offsets` and `transaction_state` internal topics).

`sendResponseExemptThrottle` Internal Method

```
sendResponseExemptThrottle(
  request: RequestChannel.Request,
  response: AbstractResponse,
  onComplete: Option[Send => Unit] = None): Unit
```

`sendResponseExemptThrottle` ...FIXME

Note	`sendResponseExemptThrottle` is used when `KafkaApis` is requested to…FIXME

`getOrCreateInternalTopic` Internal Method

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

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

In the end, `getOrCreateInternalTopic` returns the `TopicMetadata` if available or createInternalTopic.

Note	`getOrCreateInternalTopic` is used exclusively when `KafkaApis` is requested to handle a FindCoordinator request (and requests the metadata of `consumer_offsets` and `transaction_state` internal topics).

`getTopicMetadata` Internal Method

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

`getTopicMetadata`…FIXME

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

Handling DescribeGroups Request — `handleDescribeGroupRequest` Handler

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

`handleDescribeGroupRequest`…FIXME

Note	`handleDescribeGroupRequest` is used exclusively when `KafkaApis` is requested to handle a DescribeGroups request.

Handling AlterConfigs Request — `handleAlterConfigsRequest` Handler

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

`handleAlterConfigsRequest` ...FIXME

Note	`handleAlterConfigsRequest` is used exclusively when `KafkaApis` is requested to handle a AlterConfigs request.

`createTopic` Internal Method

```
createTopic(
  topic: String,
  numPartitions: Int,
  replicationFactor: Int,
  properties: Properties = new Properties()): MetadataResponse.TopicMetadata
```

`createTopic` ...FIXME

Note	`createTopic` is used when `KafkaApis` is requested to createInternalTopic and getTopicMetadata.

Handling FindCoordinatorRequest — `handleFindCoordinatorRequest` Handler

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

`handleFindCoordinatorRequest` takes the FindCoordinatorRequest from the body (of the RequestChannel.Request).

`handleFindCoordinatorRequest` checks permissions...FIXME

For an authorized request, `handleFindCoordinatorRequest` branches off per CoordinatorType, i.e. GROUP or TRANSACTION.

For `GROUP` coordinator type, `handleFindCoordinatorRequest` does the following:

1. Requests the GroupCoordinator for partitionFor the coordinator key (of the `FindCoordinatorRequest`)
2. getOrCreateInternalTopic for __consumer_offsets topic

For `TRANSACTION` coordinator type, `handleFindCoordinatorRequest` does the following:

1. Requests the TransactionCoordinator for partitionFor (for the `coordinatorKey` of the `FindCoordinatorRequest`)
2. getOrCreateInternalTopic for __transaction_state topic

In the end, `handleFindCoordinatorRequest` sendResponseMaybeThrottle with a new FindCoordinatorResponse.

You should see the following TRACE message in the logs:

```
Sending FindCoordinator response [body] for correlation id [correlationId] to client [
clientId].
```

Note	`handleFindCoordinatorRequest` is used exclusively when `KafkaApis` is requested to handle a FindCoordinator request.

Handling JoinGroupRequest — `handleJoinGroupRequest` Handler

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

`handleJoinGroupRequest` takes the JoinGroupRequest from the body (of the RequestChannel.Request) and simply requests the GroupCoordinator to handleJoinGroup (with sendResponseCallback to handle the response).

Note	`handleJoinGroupRequest` is used exclusively when `KafkaApis` is requested to handle a JoinGroup request.

Handling JoinGroup Response — `sendResponseCallback` Method

```
sendResponseCallback(joinResult: JoinGroupResult): Unit
```

`sendResponseCallback` creates a new JoinGroupResponse for the given `JoinGroupResult` and prints out the following TRACE message to the logs:

```
Sending join group response [responseBody] for correlation id [correlationId] to clien
t [clientId].
```

In the end, `sendResponseCallback` sendResponseMaybeThrottle with the new JoinGroupResponse.

Handling AddOffsetsToTxn Request — `handleAddOffsetsToTxnRequest` Handler

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

`handleAddOffsetsToTxnRequest` ...FIXME

Note	`handleAddOffsetsToTxnRequest` is used exclusively when `KafkaApis` is requested to handle a AddOffsetsToTxn request.

Handling Produce Request — `handleProduceRequest` Handler

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

In summary, `handleProduceRequest` takes the ProduceRequest from the body (of the RequestChannel.Request) and requests the ReplicaManager to appendRecords (with `isFromClient` flag enabled).

Note	`internalTopicsAllowed` flag (when the ReplicaManager is requested to appendRecords) is enabled (`true`) only when the client ID is __admin_client.

`handleProduceRequest` ...FIXME

`handleProduceRequest` filterAuthorized the request for `WRITE` operation on a `TOPIC` resource to the topics (from the `ProduceRequest`).

`handleProduceRequest` ...FIXME

Note	`handleProduceRequest` is used when `KafkaApis` is requested to handle a Produce request.

`handleWriteTxnMarkersRequest` Handler

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

In summary, `handleWriteTxnMarkersRequest` requests the ReplicaManager to getMagic followed by appendRecords (with `isFromClient` flag disabled).

`handleWriteTxnMarkersRequest` ...FIXME

Note	`handleWriteTxnMarkersRequest` is used exclusively when `KafkaApis` is requested to handle a WriteTxnMarkers request.

`handleDeleteRecordsRequest` Handler

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

In summary, `handleDeleteRecordsRequest` requests the ReplicaManager to deleteRecords.

`handleDeleteRecordsRequest` ...FIXME

Note	`handleDeleteRecordsRequest` is used exclusively when `KafkaApis` is requested to handle a DeleteRecords request.

`handleOffsetForLeaderEpochRequest` Handler

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

In summary, `handleOffsetForLeaderEpochRequest` requests the ReplicaManager to lastOffsetForLeaderEpoch.

`handleOffsetForLeaderEpochRequest` ...FIXME

Note	`handleOffsetForLeaderEpochRequest` is used exclusively when `KafkaApis` is requested to handle a OffsetForLeaderEpoch request.

`handleListOffsetRequest` Handler

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

In summary, `handleListOffsetRequest` requests the ReplicaManager to fetchOffsetForTimestamp.

`handleListOffsetRequest` ...FIXME

Note	`handleListOffsetRequest` is used exclusively when `KafkaApis` is requested to handle a ListOffsets request.

`handleOffsetDeleteRequest` Handler

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

In summary, `handleOffsetDeleteRequest` ...FIXME

`handleOffsetDeleteRequest` ...FIXME

Note	`handleOffsetDeleteRequest` is used when `KafkaApis` is requested to handle a OffsetDelete request.

`handleAlterPartitionReassignmentsRequest` Handler

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

In summary, `handleAlterPartitionReassignmentsRequest` ...FIXME

`handleAlterPartitionReassignmentsRequest` ...FIXME

Note	`handleAlterPartitionReassignmentsRequest` is used when `KafkaApis` is requested to handle a AlterPartitionReassignments request.

`handleListPartitionReassignmentsRequest` Handler

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

In summary, `handleListPartitionReassignmentsRequest` ...FIXME

`handleListPartitionReassignmentsRequest` ...FIXME

Note	`handleListPartitionReassignmentsRequest` is used when `KafkaApis` is requested to handle a ListPartitionReassignments request.

`isAuthorizedClusterAction` Internal Method

```
isAuthorizedClusterAction(request: RequestChannel.Request): Boolean
```

`isAuthorizedClusterAction` simply authorize with `ClusterAction` operation and `ClusterResource` resource.

Note	`isAuthorizedClusterAction` is used when…FIXME

`updateRecordConversionStats` Internal Method

```
updateRecordConversionStats(
  request: RequestChannel.Request,
  tp: TopicPartition,
  conversionStats: RecordConversionStats): Unit
```

`updateRecordConversionStats` …FIXME

Note	`updateRecordConversionStats` is used when…FIXME

Asserting Permissions for Cluster Action — `authorizeClusterAction` Method

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

`authorizeClusterAction` simply asserts that the RequestChannel.Request is authorized to execute `ClusterAction` on a `ClusterResource`. If so, `authorizeClusterAction` does nothing and returns.

If not authorized, `authorizeClusterAction` throws a `ClusterAuthorizationException`:

```
Request [request] is not authorized.
```

Note	`authorizeClusterAction` is used when `KafkaApis` is requested to handleLeaderAndIsrRequest, handleStopReplicaRequest, handleUpdateMetadataRequest, handleControlledShutdownRequest, and handleWriteTxnMarkersRequest.

`isBrokerEpochStale` Internal Method

```
isBrokerEpochStale(
  brokerEpochInRequest: Long): Boolean
```

`isBrokerEpochStale` …FIXME

Note	`isBrokerEpochStale` is used when `KafkaApis` is requested to handleLeaderAndIsrRequest, handleStopReplicaRequest, and handleUpdateMetadataRequest.

`filterAuthorized` Internal Method

```
filterAuthorized(
  request: RequestChannel.Request,
  operation: AclOperation,
  resourceType: ResourceType,
  resourceNames: Seq[String],
  logIfAllowed: Boolean = true,
  logIfDenied: Boolean = true): Set[String]
```

`filterAuthorized` ...FIXME

Note	`filterAuthorized` is used when...FIXME

`authorizedOperations` Internal Method

```
authorizedOperations(
  request: RequestChannel.Request,
  resource: Resource): Int
```

`authorizedOperations` ...FIXME

Request	Resource Type	Resource Name
Metadata	`CLUSTER`	kafka-cluster
Metadata	`TOPIC`	Topic name
DescribeGroups	`GROUP`	Group ID

Throwing ClusterAuthorizationException for Unauthorized Cluster Operation — `authorizeClusterOperation` Internal Method

```
authorizeClusterOperation(
  request: RequestChannel.Request,
  operation: AclOperation): Unit
```

`authorizeClusterOperation` simply throws a `ClusterAuthorizationException` when the given request is not authorized for the given operation on `CLUSTER` resource and the name as `kafka-cluster`.

```
Request [request] is not authorized.
```

Request	AclOperation
LeaderAndIsr	`CLUSTER_ACTION`
StopReplica	`CLUSTER_ACTION`
UpdateMetadata	`CLUSTER_ACTION`
ControlledShutdown	`CLUSTER_ACTION`
WriteTxnMarkers	`CLUSTER_ACTION`
DescribeAcls	`DESCRIBE`
CreateAcls	`ALTER`
DeleteAcls	`ALTER`
AlterPartitionReassignments	`ALTER`
ListPartitionReassignments	`DESCRIBE`

KafkaHealthcheck

`KafkaHealthcheck` registers the broker it runs on with Zookeeper (which in turn makes the broker visible to other brokers that together can form a Kafka cluster).

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

Figure 1. KafkaHealthcheck

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

Name	Description
`sessionExpireListener`	`SessionExpireListener`

Creating KafkaHealthcheck Instance

`KafkaHealthcheck` takes the following when created:

- Broker ID
- Advertised endpoints
- ZkUtils
- Optional rack name
- `ApiVersion`

`KafkaHealthcheck` initializes the internal registries and counters.

Starting Up — `startup` Method

```
startup
```

`startup` requests ZkUtils to subscribeStateChanges with sessionExpireListener.

In the end, `startup` registers the broker with Zookeeper.

Note	`startup` is used exclusively when `KafkaServer` 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 `EndPoint` with `PLAINTEXT` 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	`register` makes a broker visible for other brokers to form a Kafka cluster.

Note	`register` is used when `KafkaHealthcheck` 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
`server`	KafkaServer instance. Created when `KafkaServerStartable` is created.

`awaitShutdown` Method

Caution	FIXME

`shutdown` Method

```
shutdown(): Unit
```

`shutdown` ...FIXME

Note	`shutdown` is used exclusively when `Kafka` standalone application is requested to "shut down" (as part of the kafka-shutdown-hook shutdown hook).

Creating KafkaServerStartable Instance

`KafkaServerStartable` takes the following when created:

1. KafkaConfig
2. Collection of KafkaMetricsReporters

`KafkaServerStartable` initializes the internal registries and counters.

Creating KafkaServerStartable From Java Properties — `fromProps` Method

```
fromProps(serverProps: Properties): KafkaServerStartable
```

`fromProps` creates a `KafkaServerStartable` with a custom `serverProps` properties file.

Caution	FIXME

Note	`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 System.exit to terminate a JVM.

Note	`startup` is used when `kafka.Kafka` command-line application (`kafka-server-start`) is executed.

KafkaConfig

`KafkaConfig` is the configuration of a Kafka broker and the services.

`KafkaConfig` is created when:

- `KafkaConfig` is requested to fromProps and apply
- `DynamicBrokerConfig` is requested to initialize and processReconfiguration

```
import kafka.server.KafkaConfig
assert(KafkaConfig.LogCleanupPolicyProp == "log.cleanup.policy")

import java.util.Properties
val props = new Properties()
props.put(KafkaConfig.ZkConnectProp, ":9092") // required property
val kafkaConfig = KafkaConfig.fromProps(props, doLog = false)

import scala.collection.JavaConverters._
assert(kafkaConfig.logCleanupPolicy.asScala.head == "delete")
```

Table 1. KafkaConfig's Configuration Values

Value	Kafka Property
`advertisedListeners`	advertised.listeners (see advertisedListeners) Used when: - `DynamicListenerConfig` is requested to validate reconfiguration - `KafkaConfig` is requested to validateValu - `KafkaServer` is requested to createBroke (when started)
`authorizerClassName`	authorizer.class.name Used exclusively for KafkaServer
`autoCreateTopicsEnable`	auto.create.topics.enable Used when `KafkaApis` is requested to getTopicMetadata and handleTopicMetadataRequest
`autoLeaderRebalanceEnable`	auto.leader.rebalance.enable Used exclusively when `KafkaController` is requested to onControllerFailover

`backgroundThreads`	background.threads
`brokerId`	broker.id
`brokerIdGenerationEnable`	broker.id.generation.enable Used exclusively when `KafkaServer` is reque to getBrokerIdAndOfflineDirs
`compressionType`	compression.type
`connectionsMaxIdleMs`	connections.max.idle.ms Used when: • `TransactionMarkerChannelManager` is crea • `SocketServer` is requested to create a ne network processor thread • `ReplicaFetcherBlockingSend` is created
`deleteTopicEnable`	delete.topic.enable
`failedAuthenticationDelayMs`	connection.failed.authentication.delay.ms Used exclusively when `SocketServer` is requ to create a new network processor thread
`hostName`	host.name
`interBrokerListenerName`	inter.broker.listener.name (based on getInterBrokerListenerNameAndSecurityProto Used when: • `Broker` is requested for toServerInfo • `ControllerChannelManager` is requested t register a new broker • `AbstractControllerBrokerRequestBatch` is requested to send LeaderAndIsr request to all brokers • `TransactionMarkerChannelManager` utility is requested to create a TransactionMarkerChannelManager • `SocketServer` is requested to startDataPlaneProcessors

	• Processor is created (and creates the Selector) • `ConnectionQuotas` is requested to `protectedListener` • `DynamicListenerConfig` is requested to validateReconfiguration • `KafkaConfig` is requested to validateValu • `KafkaServer` is requested to controlledShutdown • ReplicaFetcherBlockingSend is created • `ReplicaManager` is requested to makeFollowers
`interBrokerProtocolVersion`	inter.broker.protocol.version
`interBrokerSecurityProtocol`	security.inter.broker.protocol (based on getInterBrokerListenerNameAndSecurityProto Used when: • `ControllerChannelManager` is requested t a new broker (unless control.plane.listener.name is defined) • `TransactionMarkerChannelManager` utility is requested to create a TransactionMarkerChannelManager (witl inter.broker.listener.name) • `KafkaServer` is requested to controlledShutdown (with inter.broker.listener.name) • ReplicaFetcherBlockingSend is created (inter.broker.listener.name)
`controlPlaneListenerName`	control.plane.listener.name (based on getControlPlaneListenerNameAndSecurityPro Used when: • SocketServer is created (and creates a dedicated control-plane request handler controller-brokers communication for `KafkaServer`) • `ControllerChannelManager` is requested t a new broker • `KafkaConfig` is requested for controlPlaneListener and to validateValu

`leaderImbalanceCheckIntervalSeconds`	leader.imbalance.check.interval.seconds Used exclusively when `KafkaController` is requested to processAutoPreferredReplicaLeaderElection
`leaderImbalancePerBrokerPercentage`	leader.imbalance.per.broker.percentage Used exclusively when `KafkaController` is requested to checkAndTriggerAutoLeaderRebalance
`listeners`	listeners (see listeners)
`listenerSecurityProtocolMap`	listener.security.protocol.map (see listenerSecurityProtocolMap)
`logCleanerThreads`	log.cleaner.threads
`logCleanerDedupeBufferSize`	log.cleaner.dedupe.buffer.size
`logCleanerDedupeBufferLoadFactor`	log.cleaner.io.buffer.load.factor
`logCleanerIoBufferSize`	log.cleaner.io.buffer.size
`logCleanerIoMaxBytesPerSecond`	log.cleaner.io.max.bytes.per.second
`logCleanerBackoffMs`	log.cleaner.backoff.ms
`logCleanerDeleteRetentionMs`	log.cleaner.delete.retention.ms
`logCleanerEnable`	log.cleaner.enable Used exclusively when `LogCleaner` utility is requested to build a CleanerConfig from KafkaConfig.
`logCleanerMaxCompactionLagMs`	log.cleaner.max.compaction.lag.ms
`logCleanerMinCleanRatio`	log.cleaner.min.cleanable.ratio
`logCleanerMinCompactionLagMs`	log.cleaner.min.compaction.lag.ms
`logCleanupIntervalMs`	log.retention.check.interval.ms Used for LogManager (for the kafka-log-reten task)

`logCleanupPolicy`	log.cleanup.policy Used exclusively when `KafkaServer` is reque to copyKafkaConfigToLog
`logDeleteDelayMs`	log.segment.delete.delay.ms
`logDirs`	log.dirs or log.dir At least one log directory must be defined via log.dirs or log.dir properties. Used when: • `LogManager` is created • `LogDirFailureChannel` is created (when `KafkaServer` is requested to start up) • `KafkaConfig` is requested to getNumReplicaAlterLogDirsThreads • `KafkaServer` is created and is requested getBrokerIdAndOfflineDirs and checkpointBrokerId • `ReplicaManager` is requested to describeLogDirs
`logFlushIntervalMessages`	log.flush.interval.messages Used exclusively when `Log` is requested to append records
`logFlushIntervalMs`	log.flush.interval.ms Used exclusively when `KafkaServer` is reque for the default log properties (and sets the val LogConfig.FlushMsProp)
`logIndexIntervalBytes`	log.index.interval.bytes
`logIndexSizeMaxBytes`	log.index.size.max.bytes Used exclusively when `KafkaServer` is reque for the default log properties (and sets the val LogConfig.SegmentIndexBytesProp)
`logMessageDownConversionEnable`	log.message.downconversion.enable
`logMessageFormatVersion`	log.message.format.version
`logMessageTimestampDifferenceMaxMs`	log.message.timestamp.difference.max.ms

`logMessageTimestampType`	log.message.timestamp.type
`logPreAllocateEnable`	log.preallocate
`logRetentionBytes`	log.retention.bytes
`logRetentionTimeMillis`	log.retention.ms, log.retention.minutes, or log.retention.hours (from the highest to the lo priority)
`logRollTimeJitterMillis`	
`LogRollTimeHoursProp`	
`LogRollTimeJitterMillisProp`	log.roll.jitter.ms
`LogRollTimeMillisProp`	log.roll.ms Default: `null` Topic-level property (`TopicConfigSynonyms`): segment.ms Used when: • …
`logSegmentBytes`	log.segment.bytes Used exclusively when `KafkaServer` is reque for the default log properties (and sets the val LogConfig.SegmentBytesProp)
`maxConnectionsPerIp`	max.connections.per.ip Used exclusively when `SocketServer` is requ to start up
`maxConnectionsPerIpOverrides`	max.connections.per.ip.overrides Used exclusively when `SocketServer` is requ to start up
`maxReservedBrokerId`	reserved.broker.max.id Used exclusively when `KafkaServer` is reque to generateBrokerId
`messageMaxBytes`	message.max.bytes

`minInSyncReplicas`	min.insync.replicas Used when `KafkaServer` utility is used to crea default log properties for LogManager and AdminManager (and set MinInSyncReplicasP
`numIoThreads`	num.io.threads
`numNetworkThreads`	num.network.threads
`numRecoveryThreadsPerDataDir`	num.recovery.threads.per.data.dir Used when `LogManager` is created
`getNumReplicaAlterLogDirsThreads`	num.replica.alter.log.dirs.threads
`numReplicaFetchers`	num.replica.fetchers Used to create a ReplicaFetcherManager
`port`	port
`PrincipalBuilderClassProp`	principal.builder.class
`replicaFetchMaxBytes`	replica.fetch.max.bytes Used for `fetchSize` of the fetcher threads, i.e ReplicaAlterLogDirsThread and ReplicaFetcherThread
`replicaFetchResponseMaxBytes`	replica.fetch.response.max.bytes Used for `maxBytes` of the fetcher threads, i.e. ReplicaAlterLogDirsThread and ReplicaFetcherThread
`replicaLagTimeMaxMs`	replica.lag.time.max.ms Used when: • `Partition` is created • `ReplicaManager` is requested to startup (starts isr-expiration) and maybeShrinkIsr
`replicaFetchBackoffMs`	replica.fetch.backoff.ms Used as fetchBackOffMs for ReplicaAlterLogDirsThread and ReplicaFetcherThread

`replicaSocketTimeoutMs`	replica.socket.timeout.ms Used when ReplicaFetcherBlockingSend is c (for the socketTimeout)
`SaslMechanismInterBrokerProtocolProp`	sasl.mechanism.inter.broker.protocol
`SaslJaasConfigProp`	sasl.jaas.config
`SaslEnabledMechanismsProp`	sasl.enabled.mechanisms
`SaslKerberosServiceNameProp`	sasl.kerberos.service.name
`SaslKerberosKinitCmdProp`	sasl.kerberos.kinit.cmd
`SaslKerberosTicketRenewWindowFactorProp`	sasl.kerberos.ticket.renew.window.factor
`SaslKerberosTicketRenewJitterProp`	sasl.kerberos.ticket.renew.jitter
`SaslKerberosMinTimeBeforeReloginProp`	sasl.kerberos.min.time.before.relogin
`SaslKerberosPrincipalToLocalRulesProp`	sasl.kerberos.principal.to.local.rules
`SaslLoginRefreshWindowFactorProp`	sasl.login.refresh.window.factor
`SaslLoginRefreshWindowJitterProp`	sasl.login.refresh.window.jitter
`SaslLoginRefreshMinPeriodSecondsProp`	sasl.login.refresh.min.period.seconds
`SaslLoginRefreshBufferSecondsProp`	sasl.login.refresh.buffer.seconds
`SslProtocolProp`	ssl.protocol
`SslProviderProp`	ssl.provider
`SslCipherSuitesProp`	ssl.cipher.suites
`SslEnabledProtocolsProp`	ssl.enabled.protocols
`SslKeystoreTypeProp`	ssl.keystore.type
`SslKeystoreLocationProp`	ssl.keystore.location
`SslKeystorePasswordProp`	ssl.keystore.password
`SslKeyPasswordProp`	ssl.key.password

`SslPrincipalMappingRulesProp`	ssl.principal.mapping.rules
`SslTruststoreTypeProp`	ssl.truststore.type
`SslTruststoreLocationProp`	ssl.truststore.location
`SslTruststorePasswordProp`	ssl.truststore.password
`SslKeyManagerAlgorithmProp`	ssl.keymanager.algorithm
`SslTrustManagerAlgorithmProp`	ssl.trustmanager.algorithm
`SslEndpointIdentificationAlgorithmProp`	ssl.endpoint.identification.algorithm
`SslSecureRandomImplementationProp`	ssl.secure.random.implementation
`SslClientAuthProp`	ssl.client.auth
`socketRequestMaxBytes`	socket.request.max.bytes Used when `SocketServer` is created and requested to create a new network processor thread
`tokenAuthEnabled`	delegation.token.master.key
`transactionMaxTimeoutMs`	transaction.max.timeout.ms Used exclusively when TransactionCoordinat created
`uncleanLeaderElectionEnable`	unclean.leader.election.enable Default: `false` Topic-level property: unclean.leader.election.e Used when: • `TopicConfigHandler` is requested to proc configuration changes (for `unclean.leader.election.enable` configu property that triggers the `KafkaControlle` enableDefaultUncleanLeaderElection)
`queuedMaxRequests`	queued.max.requests Used exclusively when `SocketServer` is creat

`zkConnect`	zookeeper.connect Used when: • `KafkaServer` is requested to initZkClient • FIXME
`zkConnectionTimeoutMs`	zookeeper.connection.timeout.ms when set o zookeeper.session.timeout.ms Used when: • `KafkaServer` is requested to initZkClient • FIXME
`zkEnableSecureAcls`	zookeeper.set.acl Used when: • `KafkaServer` is requested to initZkClient • FIXME
`zkMaxInFlightRequests`	zookeeper.max.in.flight.requests Used when: • `KafkaServer` is requested to initZkClient • FIXME
`zkSessionTimeoutMs`	zookeeper.session.timeout.ms Used when: • `KafkaServer` is requested to initZkClient • FIXME

`authorizer` Method

```
authorizer: Option[Authorizer]
```

`authorizer` takes the value of KafkaConfig.AuthorizerClassNameProp property.

If defined, `authorizer` creates an instance and returns it as an Authorizer (or a AuthorizerWrapper for the deprecated `kafka.security.auth.Authorizer` API). Otherwise, `authorizer` returns `None`.

`authorizer` throws a `ConfigException` for unsupported `Authorizer` API:

```
Authorizer does not implement org.apache.kafka.server.authorizer.Authorizer or kafka.s
ecurity.auth.Authorizer .
```

Note	`authorizer` is used when `KafkaServer` is requested to start up.

`listeners` Method

```
listeners: Seq[EndPoint]
```

`listeners` creates the `EndPoints` based on listeners configuration property if defined. Otherwise, `listeners` assumes `PLAINTEXT://[hostName]:[port]` (per hostName and port).

`listeners` uses listenerSecurityProtocolMap for protocol mapping.

Note	`listeners` is used when: • `SocketServer` is requested for the endpoints and protectedListener • `DynamicListenerConfig` is requested to validateReconfiguration and reconfigure • `KafkaConfig` is requested for the controlPlaneListener, dataPlaneListeners, advertisedListeners, and validateValues

`advertisedListeners` Method

```
advertisedListeners: Seq[EndPoint]
```

`advertisedListeners` ...FIXME

Note	`advertisedListeners` is used when...FIXME

`listenerSecurityProtocolMap` Method

```
listenerSecurityProtocolMap: Map[ListenerName, SecurityProtocol]
```

`listenerSecurityProtocolMap` ...FIXME

Note	`listenerSecurityProtocolMap` is used when...FIXME

`getNumReplicaAlterLogDirsThreads` Method

```
getNumReplicaAlterLogDirsThreads: Int
```

`getNumReplicaAlterLogDirsThreads` is num.replica.alter.log.dirs.threads if defined or the number of log directories.

Note	`getNumReplicaAlterLogDirsThreads` is used when ReplicaAlterLogDirsManager is created.

Creating KafkaConfig Instance

`KafkaConfig` takes the following to be created:

- Key-value properties
- `doLog` flag
- DynamicBrokerConfig

`KafkaConfig` initializes the internal properties.

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.

Adding Reconfigurable to DynamicBrokerConfig — `addReconfigurable` Method

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

`addReconfigurable` simply requests the DynamicBrokerConfig to add the Reconfigurable.

Note	`addReconfigurable` is used when: • `ControllerChannelManager` is requested to register a new broker (and adds a ChannelBuilder) • TransactionMarkerChannelManager is created (and adds a ChannelBuilder) • Processor is created (and creates a Selector that in turn adds a ChannelBuilder) • `ConnectionQuotas` is requested to `addListener` • ReplicaFetcherBlockingSend is created (and creates a NetworkClient that in turn adds a ChannelBuilder)

`controlPlaneListener` Method

```
controlPlaneListener: Option[EndPoint]
```

`controlPlaneListener` ...FIXME

Note	`controlPlaneListener` is used when...FIXME

`dataPlaneListeners` Method

```
dataPlaneListeners: Seq[EndPoint]
```

`dataPlaneListeners` ...FIXME

Note	`dataPlaneListeners` is used when…FIXME

`validateValues` Internal Method

```
validateValues(): Unit
```

`validateValues` …FIXME

Note	`validateValues` is used when `KafkaConfig` is created.

`getInterBrokerListenerNameAndSecurityProtocol` Internal Method

```
getInterBrokerListenerNameAndSecurityProtocol: (ListenerName, SecurityProtocol)
```

`getInterBrokerListenerNameAndSecurityProtocol` …FIXME

Note	`getInterBrokerListenerNameAndSecurityProtocol` is used when `KafkaConfig` is requested for interBrokerListenerName and interBrokerSecurityProtocol.

`logRollTimeMillis` Method

```
logRollTimeMillis: java.lang.Long
```

`logRollTimeMillis` is either the value of log.roll.ms or log.roll.hours (converted to millis).

Note	`logRollTimeMillis` is used when `KafkaServer` utility is used to copyKafkaConfigToLog (and set the topic-level segment.ms).

Security Protocol for Controller-Brokers Communication — `controlPlaneSecurityProtocol` Method

```
controlPlaneSecurityProtocol: Option[SecurityProtocol]
```

`controlPlaneSecurityProtocol` is the name of the security protocol based on control.plane.listener.name and listener.security.protocol.map configuration properties (via getControlPlaneListenerNameAndSecurityProtocol).

Note	`controlPlaneSecurityProtocol` is used when `ControllerChannelManager` is requested to add a new broker.

`getControlPlaneListenerNameAndSecurityProtocol` Internal Method

```
getControlPlaneListenerNameAndSecurityProtocol: Option[(ListenerName, SecurityProtocol
)]
```

`getControlPlaneListenerNameAndSecurityProtocol` uses the control.plane.listener.name property for the listener name and looks it up in the listener.security.protocol.map for the security protocol.

`getControlPlaneListenerNameAndSecurityProtocol` returns `None` when the control.plane.listener.name property is not defined.

`getControlPlaneListenerNameAndSecurityProtocol` throws a `ConfigException` when the listener name could not be found:

```
Listener with [listenerName] defined in control.plane.listener.name not found in liste
ner.security.protocol.map.
```

Note	`getControlPlaneListenerNameAndSecurityProtocol` is used when `KafkaConfig` is requested for the controlPlaneListenerName and controlPlaneSecurityProtocol.

Internal Properties

Name	Description
`dynamicConfig`	DynamicBrokerConfig (that could be provided or will be created from scratch) Used when: • `AdminManager` is requested to describeConfigs (for `BROKER` resources), alterConfigs and configSynonyms • `BrokerConfigHandler` is requested to processConfigChanges • `DynamicMetricsReporters` is created • `DynamicClientQuotaCallback` is reconfigure • `KafkaConfig` is requested to addReconfigurable • `KafkaServer` is requested to start up

KafkaMetricsReporter

Caution	FIXME

KafkaRequestHandlerPool — Pool of KafkaRequestHandler Daemon Threads

`KafkaRequestHandlerPool` is a pool of KafkaRequestHandler daemon threads.

`KafkaRequestHandlerPool` is created exclusively when `KafkaServer` is requested to start up (and creates the data-plane and control-plane request handler pools).

Figure 1. KafkaRequestHandlerPool and KafkaRequestHandler Threads

When created, `KafkaRequestHandlerPool` requests all KafkaRequestHandler daemon threads to start.

Note	The number of **kafka-request-handler** threads is controlled by num.network.threads configuration property (default: `3`).

`KafkaRequestHandlerPool` uses **[Kafka Request Handler on Broker [brokerId]]** as the logging prefix (aka `logIdent`).

Tip	Enable `ALL` logging level for `kafka.server.KafkaRequestHandlerPool` logger to see what happens inside. Add the following line to `config/log4j.properties` : `log4j.logger.kafka.server.KafkaRequestHandlerPool=ALL` Refer to Logging.

Creating KafkaRequestHandlerPool Instance

`KafkaRequestHandlerPool` takes the following to be created:

- Broker ID
- RequestChannel

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

`KafkaRequestHandlerPool` initializes the internal properties.

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

Performance Metrics

`KafkaRequestHandlerPool` is a KafkaMetricsGroup with the following performance metrics.

Table 1. KafkaRequestHandlerPool's Performance Metrics

Metric Name	Description
`requestHandlerAvgIdleMetricName`	**Aggregare idle meter** - average free capacity of the request handlers (*idle handlers*)

The performance metrics are registered in **kafka.server:type=KafkaRequestHandlerPool** group.

Figure 2. KafkaMetricsGroup in jconsole

Creating and Starting KafkaRequestHandler Daemon Thread — `createHandler` Method

```
createHandler(id: Int): Unit
```

`createHandler` creates a new KafkaRequestHandler for the given id (and the broker ID, the aggregare idle meter, the threadPoolSize, the RequestChannel and the KafkaApis).

`createHandler` adds the `KafkaRequestHandler` to the runnables internal registry.

In the end, `createHandler` starts the `KafkaRequestHandler` as a daemon thread with the name with the format:

```
[logAndThreadNamePrefix]-kafka-request-handler-[id]
```

Note	`createHandler` is used when `KafkaRequestHandlerPool` is created (to create the numThreads handlers) and requested to resize the thread pool (with new handlers).

Resizing Thread Pool (of KafkaRequestHandlers) — `resizeThreadPool` Method

```
resizeThreadPool(newSize: Int): Unit
```

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

```
Resizing request handler thread pool size from [currentSize] to [newSize]
```

When the given `newSize` is greater than the current threadPoolSize, `resizeThreadPool` creates and starts new KafkaRequestHandler daemon threads.

When the given `newSize` is smaller than the current threadPoolSize, `resizeThreadPool` removes `KafkaRequestHandlers` from the runnables internal registry and requests them to stop.

In the end, `resizeThreadPool` sets the threadPoolSize internal registry to the given `newSize`.

Note	`resizeThreadPool` is used exclusively when `DynamicThreadPool` is requested to reconfigure (the num.io.threads configuration property).

Shutting Down — `shutdown` Method

```
shutdown(): Unit
```

`shutdown` …FIXME

Note	`shutdown` is used when…FIXME

Internal Properties

Name	Description
`runnables`	Pool of KafkaRequestHandlers daemon threads
`threadPoolSize`	

KafkaRequestHandler

`KafkaRequestHandler` is a thread of execution (java.lang.Runnable) that is responsible for relaying client requests (from a RequestChannel) to a KafkaApis (except `ShutdownRequest` requests that are handled directly).

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

Figure 1. KafkaRequestHandler's Startup and Request Relay

`KafkaRequestHandler` uses **[Kafka Request Handler [id] on Broker [brokerId]]** as the logging prefix (aka `logIdent`).

Tip	Enable `ALL` 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=ALL` Refer to Logging.

Creating KafkaRequestHandler Instance

`KafkaRequestHandler` takes the following to be created:

- ID

- Broker ID
- Aggregate Idle Meter
- Total number of handler threads
- RequestChannel
- KafkaApis
- `Time`

`KafkaRequestHandler` initializes the internal properties.

Starting KafkaRequestHandler Thread — `run` Method

```
run(): Unit
```

Note	`run` is part of the java.lang.Runnable to start itself as a separately-executing thread.

`run` runs continuously until `KafkaRequestHandler` is requested to stop (which turns the stopped internal flag on).

`run` ...FIXME

Stopping KafkaRequestHandler Thread — `stop` Method

```
stop(): Unit
```

`stop` simply turns the stopped internal flag on (`true`).

Note	`stop` is used exclusively when `KafkaRequestHandlerPool` is requested to resize the thread pool of KafkaRequestHandlers.

Internal Properties

Name	Description
`stopped`	Flag to control whether run should stop (`true`) or not (`false`) (and hence the `KafkaRequestHandler`) Default: `false` Turned on (`true`) when `KafkaRequestHandler` is requested to stop

LogManager

`LogManager` is created (via apply factory utility) and immediately started for `KafkaServer` when requested to start up.

Figure 1. LogManager and KafkaServer

`LogManager` runs until (and is requested to shut down when) `KafkaServer` is requested to shut down.

`LogManager` uses the following configuration properties.

Table 1. LogManager's Configuration Properties

Name	Description
file.delete.delay.ms	
log.cleaner.backoff.ms	
log.cleaner.dedupe.buffer.size	
log.cleaner.enable	Used to enable a LogCleaner
log.cleaner.io.buffer.load.factor	
log.cleaner.io.buffer.size	
log.cleaner.io.max.bytes.per.second	
log.cleaner.threads	
log.dirs or log.dir	Used for the log directories that are immediately validated and loaded (on a fixed thread pool with numRecoveryThreadsPerDataDir recovery threads)
log.flush.offset.checkpoint.interval.ms	
log.flush.scheduler.interval.ms	
log.flush.start.offset.checkpoint.interval.ms	
log.retention.check.interval.ms	
message.max.bytes	
num.recovery.threads.per.data.dir	Number of threads per log data directory for log recovery at startup and flushing at shutdown
transactional.id.expiration.ms	

`LogManager` is used to create a ReplicaManager, a DynamicLogConfig, a TopicConfigHandler, and a Partition.

`LogManager` defaults to `30000ms` for the **initial task delay**.

Tip	Enable `ALL` logging level for `kafka.log.LogManager` logger to see what happens inside. Add the following line to `config/log4j.properties`: `log4j.logger.kafka.log.LogManager=ALL` Refer to Logging.

LogManager and LogCleaner

`LogManager` creates a LogCleaner only when the log.cleaner.enable flag (of the given CleanerConfig) is enabled (default: `true`).

`LogManager` uses LogCleaner for the following:

- truncateTo
- truncateFullyAndStartAt
- abortAndPauseCleaning
- replaceCurrentWithFutureLog
- asyncDelete
- cleanupLogs
- startup
- shutdown
- handleLogDirFailure

Performance Metrics

`LogManager` is a KafkaMetricsGroup with the following performance metrics.

Table 2. LogManager's Performance Metrics

Metric Name	Description
`OfflineLogDirectoryCount`	The number of offline log directories
`LogDirectoryOffline`	Registered for every log directory to indicate whether it is online or offline Possible values: • `0` when a log directory is online • `1` when a log directory is offline The path of the directory is the tag of the metric.

The performance metrics are registered in **kafka.log:type=LogManager** group.

Figure 2. LogManager in jconsole

Creating LogManager — `apply` Utility

```
apply(
  config: KafkaConfig,
  initialOfflineDirs: Seq[String],
  zkClient: KafkaZkClient,
  brokerState: BrokerState,
  kafkaScheduler: KafkaScheduler,
  time: Time,
  brokerTopicStats: BrokerTopicStats,
  logDirFailureChannel: LogDirFailureChannel): LogManager
```

`apply` ...FIXME

Note	`apply` is used exclusively when `KafkaServer` is requested to start up.

Creating LogManager Instance

`LogManager` takes the following to be created:

- Absolute paths to log directories
- Initial offline directories
- Topic configurations - LogConfigs per topic name
- Initial LogConfig
- CleanerConfig
- num.recovery.threads.per.data.dir dynamic configuration property
- log.flush.scheduler.interval.ms configuration property
- log.flush.offset.checkpoint.interval.ms configuration property
- log.flush.start.offset.checkpoint.interval.ms configuration property
- log.retention.check.interval.ms configuration property
- transactional.id.expiration.ms configuration property
- Scheduler
- `BrokerState`
- BrokerTopicStats
- `LogDirFailureChannel`
- `Time`

`LogManager` initializes the internal properties.

While being created, `LogManager` loads logs.

`abortAndPauseCleaning` Method

```
abortAndPauseCleaning(
  topicPartition: TopicPartition): Unit
```

`abortAndPauseCleaning` ...FIXME

Note	`abortAndPauseCleaning` is used when `ReplicaManager` is requested to alterReplicaLogDirs and becomeLeaderOrFollower.

Recovering And Loading Logs In Log Data Directories — `loadLogs` Internal Method

```
loadLogs(): Unit
```

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

```
Loading logs.
```

For every live log directory, `loadLogs` first creates a fixed thread pool (with numRecoveryThreadsPerDataDir threads).

`loadLogs` then checks whether .kafka_cleanshutdown file exists in the log directory. If so, `loadLogs` prints out the following DEBUG message to the logs:

```
Found clean shutdown file. Skipping recovery for all logs in data directory: [dir]
```

`loadLogs` uses the recoveryPointCheckpoints to look up the `OffsetCheckpointFile` for the log directory (**recovery-point-offset-checkpoint** file) and then loads it.

`loadLogs` uses the logStartOffsetCheckpoints to look up the `OffsetCheckpointFile` for the log directory (**recovery-point-offset-checkpoint** file) and then loads it.

For every directory in the log directory, `loadLogs` creates a new thread to load the log directory with the recovery points and log start offsets (that have just been loaded).

`loadLogs` submits the new threads to load the log directory for execution on the fixed thread pool.

`loadLogs` then…FIXME (finish me)

In the end, after having loaded the log directories successfully, `loadLogs` prints out the following INFO message to the logs:

```
Logs loading complete in [duration] ms.
```

In case .kafka_cleanshutdown file does not exist, `loadLogs` transitions the BrokerState to `RecoveringFromUncleanShutdown`.

In case of an exception while loading the `OffsetCheckpointFile` of a log directory (**recovery-point-offset-checkpoint** file), `loadLogs` simply prints out the following WARN messages to the logs:

```
Error occurred while reading recovery-point-offset-checkpoint file of directory [dir]
Resetting the recovery checkpoint to 0
```

In case of an exception while loading the `OffsetCheckpointFile` of a log directory (**log-start-offset-checkpoint** file), `loadLogs` simply prints out the following WARN messages to the logs:

```
Error occurred while reading log-start-offset-checkpoint file of directory [dir]
```

In case of an exception while load the log directory or any other task, `loadLogs` adds the log directory to a **offlineDirs** internal registry with the exception and prints out the following ERROR message to the logs:

```
Error while loading log dir [dir]
```

Note	`loadLogs` is used exclusively when `LogManager` is created.

Loading Partition Log Directory — `loadLog` Internal Method

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

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

```
Loading log '[logDir]'
```

`loadLog` then parses the topic and partition out of the directory name of the log (by the given `logDir`).

`loadLog` gets the LogConfig for the topic (from the LogConfigs per topic) or defaults to the currentDefaultConfig.

`loadLog` gets `logRecoveryPoint` for the partition (from the given `recoveryPoints`) or defaults to `0`.

`loadLog` gets `logStartOffset` for the partition (from the given `logStartOffsets`) or defaults to `0`.

`loadLog` creates a Log.

In case the name of the given `logDir` ends with -delete suffix, `loadLog` addLogToBeDeleted.

Otherwise, `loadLog` adds the `Log` to the futureLogs or currentLogs internal registry whether it is isFuture or not, respectively.

In case there was `Log` already registered (the futureLogs or currentLogs internal registry), `loadLog` throws an `IllegalStateException`:

```
FIXME
```

Note	`loadLog` is used exclusively when `LogManager` is requested to recover and load the logs in log data directories.

Starting Up — `startup` Method

```
startup(): Unit
```

`startup` starts the background threads to flush logs and do log cleanup.

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

```
Starting log cleanup with a period of [retentionCheckMs] ms.
```

`startup` requests the Scheduler to schedule a task with the name **kafka-log-retention** that cleanupLogs with the InitialTaskDelayMs delay and the retentionCheckMs execution period.

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

```
Starting log flusher with a default period of [flushCheckMs] ms.
```

`startup` requests the Scheduler to schedule a task with the name **kafka-log-flusher** that flushDirtyLogs with the InitialTaskDelayMs delay and the flushCheckMs execution period.

`startup` requests the Scheduler to schedule a task with the name **kafka-recovery-point-checkpoint** that checkpointLogRecoveryOffsets with the InitialTaskDelayMs delay and the flushRecoveryOffsetCheckpointMs execution period.

`startup` requests the Scheduler to schedule a task with the name **kafka-log-start-offset-checkpoint** that checkpointLogStartOffsets with the InitialTaskDelayMs delay and the flushStartOffsetCheckpointMs execution period.

`startup` requests the Scheduler to schedule a task with the name **kafka-delete-logs** that deleteLogs with the InitialTaskDelayMs delay.

(only when the CleanerConfig has the enableCleaner flag enabled) `startup` requests the LogCleaner to start up.

Note	`startup` is used exclusively when `KafkaServer` is requested to start up.

BrokerTopicStats

When created, `LogManager` is given a BrokerTopicStats that is used exclusively to create Logs when recovering and loading logs in log data directories and looking up or creating a Log.

`cleanupLogs` Method

```
cleanupLogs(): Unit
```

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

```
Beginning log cleanup...
```

`cleanupLogs` finds so-called **deletable (non-compacted) logs** by requesting the LogCleaner (if used) to pauseCleaningForNonCompactedPartitions or simply finds all logs in the currentLogs internal registry that are not compacted (by the compact flag of the LogConfig of the Log).

For every deletable log, `cleanupLogs` prints out the following DEBUG message to the logs:

```
Garbage collecting '[log.name]'
```

`cleanupLogs` requests the log to deleteOldSegments.

`cleanupLogs` finds the future log for the partition (of the deletable log) in the futureLogs internal registry and, if available, prints out the following DEBUG message to the logs followed by requesting it to deleteOldSegments.

```
Garbage collecting future log '[futureLog.name]'
```

In the end, `cleanupLogs` requests the LogCleaner (if used) to resumeCleaning and prints out the following DEBUG message to the logs:

```
Log cleanup completed. [total] files deleted in [duration] seconds
```

Note	`cleanupLogs` is used when `LogManager` is requested to start up (and schedules kafka-log-retention periodic task).

Getting All Partition Logs — `allLogs` Method

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

`allLogs` ...FIXME

Note	`allLogs` 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: • `Partition` is requested to removeFutureLocalReplica and delete • `ReplicaManager` is requested to stopReplica

Looking Up Or Creating New Partition Log — `getOrCreateLog` Method

```
getOrCreateLog(
  topicPartition: TopicPartition,
  config: LogConfig,
  isNew: Boolean = false,
  isFuture: Boolean = false): Log
```

`getOrCreateLog` looks up the partition log for the given `TopicPartition` (and returns it if found) or creates a new one.

When looking up the partition log was unsuccessful, `getOrCreateLog` finds the log directory for the `TopicPartition`. `getOrCreateLog` finds one in the preferredLogDirs internal registry and falls back on nextLogDirs (that simply gives all the log directories sorted by the number of partition logs).

`getOrCreateLog` creates the directory name for the `TopicPartition` (based on the given `isFuture` flag).

`getOrCreateLog` tries to createLogDirectory in the available log directories (one by one) until successful.

`getOrCreateLog` creates a new Log (using the maxPidExpirationMs and `LogManager.ProducerIdExpirationCheckIntervalMs` configuration properties).

`getOrCreateLog` registers the new `Log` with the `TopicPartition` in the futureLogs or currentLogs internal registries (based on the given isFuture flag).

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

```
Created log for partition [topicPartition] in [logDir] with properties [config].
```

`getOrCreateLog` removes the `TopicPartition` from the preferredLogDirs internal registry.

Note	`getOrCreateLog` is used exclusively when `Partition` is requested to getOrCreateReplica.

`getLog` Method

```
getLog(
  topicPartition: TopicPartition,
  isFuture: Boolean = false): Option[Log]
```

`getLog` ...FIXME

Note	`getLog` is used when...FIXME

`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

`checkpointLogRecoveryOffsets` Method

```
checkpointLogRecoveryOffsets(): Unit
```

`checkpointLogRecoveryOffsets` ...FIXME

Note	`checkpointLogRecoveryOffsets` is used when...FIXME

`checkpointLogStartOffsets` Method

```
checkpointLogStartOffsets(): Unit
```

`checkpointLogStartOffsets` ...FIXME

Note	`checkpointLogStartOffsets` is used when...FIXME

`isLogDirOnline` Method

```
isLogDirOnline(logDir: String): Boolean
```

`isLogDirOnline` ...FIXME

Note	`isLogDirOnline` is used when...FIXME

Validating Data Log Directories — `createAndValidateLogDirs` Internal Method

```
createAndValidateLogDirs(
  dirs: Seq[File],
  initialOfflineDirs: Seq[File]): ConcurrentLinkedQueue[File]
```

For every directory in the given `dirs`, `createAndValidateLogDirs` makes sure that the data directory is available (i.e. it is a readable directory) or creates it.

`createAndValidateLogDirs` prints out the following INFO message to the logs when a data directory does not exist:

```
Log directory [dir] not found, creating it.
```

Note	`createAndValidateLogDirs` is given the logDirs and the initialOfflineDirs that `LogManager` is created with.

`createAndValidateLogDirs` throws...FIXME

Note	`createAndValidateLogDirs` is used exclusively when `LogManager` is created.

`truncateTo` Method

```
truncateTo(
  partitionOffsets: Map[TopicPartition, Long],
  isFuture: Boolean): Unit
```

`truncateTo` ...FIXME

Note	`truncateTo` is used when `Partition` is requested to truncateTo.

`truncateFullyAndStartAt` Method

```
truncateFullyAndStartAt(
  topicPartition: TopicPartition,
  newOffset: Long,
  isFuture: Boolean): Unit
```

`truncateFullyAndStartAt` ...FIXME

Note	`truncateFullyAndStartAt` is used exclusively when `Partition` is requested to truncateFullyAndStartAt.

`resizeRecoveryThreadPool` Method

```
resizeRecoveryThreadPool(newSize: Int): Unit
```

`resizeRecoveryThreadPool` prints out the following INFO message to the logs and reconfigures the numRecoveryThreadsPerDataDir internal registry to be the given `newSize`.

```
Resizing recovery thread pool size for each data dir from [numRecoveryThreadsPerDataDi
r] to [newSize]
```

Note	`resizeRecoveryThreadPool` is used exclusively when `DynamicThreadPool` is requested to reconfigure (with a new value of KafkaConfig.numRecoveryThreadsPerDataDir).

Shutting Down — `shutdown` Method

```
shutdown(): Unit
```

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

```
Shutting down.
```

`shutdown` then…FIXME

Note	`shutdown` is used exclusively when `KafkaServer` is requested to shutdown.

`replaceCurrentWithFutureLog` Method

```
replaceCurrentWithFutureLog(topicPartition: TopicPartition): Unit
```

`replaceCurrentWithFutureLog` …FIXME

Note	`replaceCurrentWithFutureLog` is used exclusively when `Partition` is requested to maybeReplaceCurrentWithFutureReplica.

`handleLogDirFailure` Method

```
handleLogDirFailure(
  dir: String): Unit
```

`handleLogDirFailure` …FIXME

Note	`handleLogDirFailure` is used exclusively when `ReplicaManager` is requested to handleLogDirFailure.

Marking Start of Partition Log Initialization — `initializingLog` Method

```
initializingLog(
  topicPartition: TopicPartition): Unit
```

`initializingLog` …FIXME

Note	`initializingLog` is used when `Partition` is requested to createLog.

Marking End of Partition Log Initialization — `finishedInitializingLog` Method

```
finishedInitializingLog(
  topicPartition: TopicPartition,
  maybeLog: Option[Log],
  fetchLogConfig: () => LogConfig): Unit
```

`finishedInitializingLog` ...FIXME

Note	`finishedInitializingLog` is used when `Partition` is requested to createLog.

`flushDirtyLogs` Internal Method

```
flushDirtyLogs(): Unit
```

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

```
Checking for dirty logs to flush...
```

`flushDirtyLogs` ...FIXME

Note	`flushDirtyLogs` is used exclusively for the kafka-log-flusher task (when `LogManager` is requested to start up).

`createLogDirectory` Internal Method

```
createLogDirectory(
  logDir: File,
  logDirName: String): Try[File]
```

`createLogDirectory` ...FIXME

Note	`createLogDirectory` is used when `LogManager` is requested to look up or create a new partition log.

`nextLogDirs` Internal Method

```
nextLogDirs(): List[File]
```

`nextLogDirs` ...FIXME

Note	`nextLogDirs` is used when `LogManager` is requested to look up or create a new partition log.

Internal Properties

Name	Description
`_liveLogDirs`	Java's ConcurrentLinkedQueue of live log directories (af createAndValidateLogDirs was executed with the logDirs the initialOfflineDirs directories). Used when…FIXME
`currentDefaultConfig`	Default LogConfig Used when a custom `LogConfig` is not available in the topicConfigs
`currentLogs`	Pool of Logs per `TopicPartition` (`Pool[TopicPartition,`
`futureLogs`	Pool of Logs per `TopicPartition` (`Pool[TopicPartition,`
`logStartOffsetCheckpoints`	
`numRecoveryThreadsPerDataDir`	Number of recovery threads per log data directory Starts as the recoveryThreadsPerDataDir and can then b dynamically changed.
`preferredLogDirs`	`preferredLogDirs: ConcurrentHashMap[TopicPartition, S`
`recoveryPointCheckpoints`	

Partition Log — Collection Of Log Segments

`Log` represents a **log** of a partition of a topic.

`Log` is a collection of LogSegments that are stored on disk in a given partition log directory with the name of the form **topic-partition** or **topic-partition.uniqueId-delete** (if marked for deletion).

While being created, `Log` creates the log directory unless available already.

`Log` is created when `LogManager` is requested to:

- Load a partition log (directory) (while being created which is when `KafkaServer` is requested to start up)
- Look up or create a partition log (when `Partition` is requested to getOrCreateReplica)

`Log` uses **.kafka_cleanshutdown** file to indicate…FIXME

`Log` is **isFuture** when…FIXME

`Log` uses the file suffixes to differentiate between parts of log segments.

Table 1. File Suffixes

Suffix	Description
`-delete`	
`.index`	OffsetIndex
`.log`	FileRecords log file
`.snapshot`	
`.timeindex`	TimeIndex
`.txnindex`	TransactionIndex

The files of offset indices, time indices, and transaction indices are collectively called **index files**.

`Log` uses a Scheduler to schedule the background tasks.

Table 2. Log's Background Tasks

Name	Period	
`PeriodicProducerExpirationCheck`	producerIdExpirationCheckIntervalMs	producerIdE
`flush-log`	`-1` (once)	`0L`
`delete-file`	`-1` (once)	file.delete.d

`Log` uses **[Log partition=[topicPartition], dir=[parent]]** as the logging prefix (aka `logIdent`).

Tip	Enable `ALL` logging levels for `kafka.log.Log` logger to see what happens inside. Add the following line to `log4j.properties`: `log4j.logger.kafka.log.Log=ALL` Refer to Logging.

LogSegments — `segments` Internal Registry

```
segments: ConcurrentNavigableMap[java.lang.Long, LogSegment]
```

`segments` is a Java's ConcurrentSkipListMap of LogSegments per baseOffset.

A new `LogSegment` is added when `Log` is requested to addSegment.

- Cleared in loadSegments just before loadSegmentFiles

Performance Metrics

`Log` is a KafkaMetricsGroup with the following performance metrics.

Table 3. Log's Performance Metrics

Metric Name	Description
`NumLogSegments`	
`LogEndOffset`	
`LogStartOffset`	
`Size`	Size of the log (in bytes)

The performance metrics are registered in **kafka.log:type=Log** group.

Figure 1. Log in jconsole

Creating Log Instance

`Log` takes the following to be created:

- Partition log directory
- LogConfig
- `logStartOffset`

- `recoveryPoint`
- Scheduler
- BrokerTopicStats
- `Time`
- `maxProducerIdExpirationMs`
- `producerIdExpirationCheckIntervalMs`
- `TopicPartition`
- ProducerStateManager
- `LogDirFailureChannel`

`Log` initializes the internal properties.

While being created, `Log` creates the log directory unless already available.

`Log` initializeLeaderEpochCache.

`Log` loads segments.

`Log` updateLogEndOffset.

`Log` requests the LeaderEpochFileCache to remove epoch entries from the store with start offsets greater than or equal to the `nextOffset` (`LeaderEpochFileCache.truncateFromEnd`).

`Log` computes the new start offset to be the maximum of the logStartOffset and the `baseOffset` of the first LogSegment in the segments internal registry.

`Log` requests the LeaderEpochFileCache to remove epoch entries from the store with offsets less than or equal to the new start offset (`LeaderEpochFileCache.truncateFromStart`).

`Log` throws a `IllegalStateException` when the ProducerStateManager is not empty:

```
Producer state must be empty during log initialization
```

`Log` loadProducerState (with the `logEndOffset` and the `reloadFromCleanShutdown` based on hasCleanShutdownFile).

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

```
Completed load of log with [size] segments, log start offset
[logStartOffset] and log end offset [logEndOffset] in [time] ms
```

Creating Log Instance — `apply` Utility

```
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` parseTopicPartitionName from the log directory.

`apply` creates a new ProducerStateManager.

In the end, `apply` creates a Log.

Note	`apply` is used when `LogManager` is requested to loadLog and look up or create a new partition log.

`roll` Method

```
roll(
  expectedNextOffset: Option[Long] = None): LogSegment
```

`roll` …FIXME

Note	`roll` is used when `Log` is requested to deleteSegments and maybeRoll.

`closeHandlers` Method

```
closeHandlers(): Unit
```

`closeHandlers` …FIXME

Note	`closeHandlers` is used when…FIXME

`updateHighWatermark` Method

```
updateHighWatermark(
  hw: Long): Long
```

`updateHighWatermark` ...FIXME

Note	`updateHighWatermark` is used when...FIXME

`addAbortedTransactions` Internal Method

```
addAbortedTransactions(
  startOffset: Long,
  segmentEntry: JEntry[JLong, LogSegment],
  fetchInfo: FetchDataInfo): FetchDataInfo
```

`addAbortedTransactions` ...FIXME

Note	`addAbortedTransactions` is used when `Log` is requested to read records.

`collectAbortedTransactions` Internal Method

```
collectAbortedTransactions(
  startOffset: Long,
  upperBoundOffset: Long): List[AbortedTxn]
collectAbortedTransactions(
  startOffset: Long,
  upperBoundOffset: Long,
  startingSegmentEntry: JEntry[JLong, LogSegment],
  accumulator: List[AbortedTxn] => Unit): Unit
```

`collectAbortedTransactions` ...FIXME

Note	`collectAbortedTransactions` is used when `Cleaner` is requested to cleanSegments and buildOffsetMap.

`maybeRoll` Internal Method

```
maybeRoll(
  messagesSize: Int,
  appendInfo: LogAppendInfo): LogSegment
```

`maybeRoll` ...FIXME

Note	`maybeRoll` is used exclusively when `Log` is requested to append records.

`asyncDeleteSegment` Internal Method

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

`asyncDeleteSegment` ...FIXME

Note	`asyncDeleteSegment` is used when `Log` is requested to deleteSegment and replaceSegments.

Flushing Log Segments Out To Disk — `flush` Method

```
flush(): Unit (1)
flush(offset: Long): Unit
```

1. Uses logEndOffset for the offset (and so flushes all log segments)

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

```
Flushing log up to offset [offset], last flushed: [lastFlushTime],  current time: [tim
e], unflushed: [unflushedMessages]
```

`flush` ...FIXME

Note	`flush` is used when: • `Log` is requested to append records and for the flush-log background task • `LogManager` is requested to shut down and flushDirtyLogs

`deleteSeg` Internal Method

```
deleteSeg(): Unit
```

`deleteSeg` ...FIXME

Note	`deleteSeg` is used exclusively for the delete-file Background Task.

`appendAsLeader` Method

```
appendAsLeader(
  records: MemoryRecords,
  leaderEpoch: Int,
  isFromClient: Boolean = true): LogAppendInfo
```

`appendAsLeader` simply appends the records with the `assignOffsets` flag on.

Note	`appendAsLeader` is used exclusively when `Partition` is requested to appendRecordsToLeader.

`appendAsFollower` Method

```
appendAsFollower(records: MemoryRecords): LogAppendInfo
```

`appendAsFollower` simply append (with the `isFromClient` and `assignOffsets` flags off, and the `leaderEpoch` being `-1`).

Note	`appendAsFollower` is used exclusively when `Partition` is requested to doAppendRecordsToFollowerOrFutureReplica.

Appending Records — `append` Internal Method

```
append(
  records: MemoryRecords,
  isFromClient: Boolean,
  interBrokerProtocolVersion: ApiVersion,
  assignOffsets: Boolean,
  leaderEpoch: Int): LogAppendInfo
```

`append` ...FIXME

Note	`append` is used when `Log` is requested to appendAsLeader (with `assignOffsets` enabled) and appendAsFollower (with `assignOffsets` and `isFromClient` disabled).

`analyzeAndValidateRecords` Internal Method

```
analyzeAndValidateRecords(
  records: MemoryRecords,
  isFromClient: Boolean): LogAppendInfo
```

`analyzeAndValidateRecords` ...FIXME

Note	`analyzeAndValidateRecords` is used exclusively when `Log` is requested to append.

`deleteSegment` Internal Method

```
deleteSegment(segment: LogSegment): Unit
```

`deleteSegment` ...FIXME

Note	`deleteSegment` is used when `Log` is requested to recoverLog, deleteSegments, roll, truncateTo, and truncateFullyAndStartAt.

`replaceSegments` Internal Method

```
replaceSegments(
  newSegments: Seq[LogSegment],
  oldSegments: Seq[LogSegment],
  isRecoveredSwapFile: Boolean = false): Unit
```

`replaceSegments` ...FIXME

Note	`replaceSegments` is used when: • `Log` is requested to completeSwapOperations and splitOverflowedSegment • `Cleaner` is requested to cleanSegments

Checking Whether .kafka_cleanshutdown Is In Parent Directory of Log Directory — `hasCleanShutdownFile` Internal Method

```
hasCleanShutdownFile: Boolean
```

`hasCleanShutdownFile` is `true` when .kafka_cleanshutdown file is in the parent directory of the log directory. Otherwise, `hasCleanShutdownFile` is `false`.

Note	`hasCleanShutdownFile` is used exclusively when `Log` is created (to loadProducerState) and requested to recoverLog.

`maybeIncrementLogStartOffset` Method

```
maybeIncrementLogStartOffset(
  newLogStartOffset: Long): Unit
```

`maybeIncrementLogStartOffset` ...FIXME

Note	`maybeIncrementLogStartOffset` is used when...FIXME

`truncateTo` Internal Method

```
truncateTo(targetOffset: Long): Boolean
```

`truncateTo` ...FIXME

Note	`truncateTo` is used when `LogManager` is requested to truncateTo.

`truncateFullyAndStartAt` Internal Method

```
truncateFullyAndStartAt(newOffset: Long): Unit
```

`truncateFullyAndStartAt` ...FIXME

Note	`truncateFullyAndStartAt` is used when: • `Log` is requested to truncateTo • `LogManager` is requested to truncateFullyAndStartAt

Scheduling Deletion Of Old Segments (Log Retention) — `deleteOldSegments` Method

```
deleteOldSegments(): Long
```

`deleteOldSegments` uses the delete flag (of the given LogConfig) to determine the scope of log deletion and returns the number of segments deleted.

Note	delete flag is enabled (`true`) when delete cleanup policy is part of the cleanup.policy configuration property.

With the delete flag enabled (`true`), `deleteOldSegments` deleteRetentionMsBreachedSegments, deleteRetentionSizeBreachedSegments and deleteLogStartOffsetBreachedSegments.

With the delete flag disabled (`false`), `deleteOldSegments` merely deleteLogStartOffsetBreachedSegments.

Note	`deleteOldSegments` is used when: • `CleanerThread` (of LogCleaner) is requested to cleanFilthiestLog • `LogManager` is requested to cleanupLogs

Scheduling Deletion Of Old Log Segments (Per Predicate) — `deleteOldSegments` Internal Method

```
deleteOldSegments(
  predicate: (LogSegment, Option[LogSegment]) => Boolean,
  reason: String): Int
```

`deleteOldSegments` finds deletable segments for the given `predicate` and schedules their deletion.

If found any, `deleteOldSegments` prints out the following INFO message to the logs:

```
Found deletable segments with base offsets [[baseOffsets]] due to [reason]
```

Note	`deleteOldSegments` is used when `Log` is requested to deleteRetentionMsBreachedSegments, deleteRetentionSizeBreachedSegments, and deleteLogStartOffsetBreachedSegments.

Finding Deletable Segments (Per Predicate) — `deletableSegments` Internal Method

```
deletableSegments(
  predicate: (LogSegment, Option[LogSegment]) => Boolean): Iterable[LogSegment]
```

`deletableSegments` ...FIXME

Note	`deletableSegments` is used exclusively when `Log` is requested to schedule deletion of old segments (per predicate).

`deleteSegments` Internal Method

```
deleteSegments(
  deletable: Iterable[LogSegment]): Int
```

`deleteSegments` roll if the number of deletable LogSegments is exactly all the segments.

For every log segment, `deleteSegments` simply schedules it for deletion and maybeIncrementLogStartOffset (based on…FIXME).

Note	`deleteSegments` is used exclusively when `Log` is requested to schedule deletion of old segments (per predicate).

`deleteRetentionMsBreachedSegments` Internal Method

```
deleteRetentionMsBreachedSegments(): Int
```

`deleteRetentionMsBreachedSegments` uses the retentionMs threshold (of the given LogConfig) to determine the scope of log retention.

`deleteRetentionMsBreachedSegments` schedules deletion of segments with their largestTimestamp below the retentionMs threshold.

`deleteRetentionMsBreachedSegments` uses the following reason:

```
retention time [retentionMs]ms breach
```

`deleteRetentionMsBreachedSegments` simply returns `0` for a negative retention.ms threshold.

Note	`deleteRetentionMsBreachedSegments` is used exclusively when `Log` is requested to schedule deletion of old segments (log retention).

`deleteRetentionSizeBreachedSegments` Internal Method

```
deleteRetentionSizeBreachedSegments(): Int
```

`deleteRetentionSizeBreachedSegments` uses the retentionSize threshold (of the given LogConfig) to determine the scope of log retention.

Note	retentionSize threshold is the value of retention.bytes configuration for a topic or log.retention.bytes configuration for the cluster.

With the retentionSize threshold negative (the default is -1L) or the size of the log below it, `deleteRetentionSizeBreachedSegments` simply exits with the return value of `0`.

`deleteRetentionSizeBreachedSegments` schedules deletion of records in log segments (per their size) so the log size drops below the threshold.

`deleteRetentionSizeBreachedSegments` uses the following reason:

```
retention size in bytes [retentionSize] breach
```

Note	`deleteRetentionSizeBreachedSegments` is used exclusively when `Log` is requested to schedule deletion of old segments (log retention) (for delete cleanup policy).

`deleteLogStartOffsetBreachedSegments` Internal Method

```
deleteLogStartOffsetBreachedSegments(): Int
```

`deleteLogStartOffsetBreachedSegments`...FIXME

Note	`deleteLogStartOffsetBreachedSegments` is used when...FIXME

`splitOverflowedSegment` Internal Method

```
splitOverflowedSegment(
  segment: LogSegment): List[LogSegment]
```

`splitOverflowedSegment`...FIXME

Note	`splitOverflowedSegment` is used when: • `Log` is requested to retryOnOffsetOverflow • `LogCleaner` is requested to cleanSegments

`onHighWatermarkIncremented` Method

```
onHighWatermarkIncremented(highWatermark: Long): Unit
```

`onHighWatermarkIncremented` ...FIXME

Note	`onHighWatermarkIncremented` is used when `Replica` is created and <<kafka-cluster-Replica.adoc#highWatermark_, highWatermark_=>>.

`parseTopicPartitionName` Object Method

```
parseTopicPartitionName(dir: File): TopicPartition
```

`parseTopicPartitionName` parses the name of the given directory and creates a `TopicPartition`.

`parseTopicPartitionName` assumes that the name is of the form **topic-partition** or **topic-partition.uniqueId-delete** (if marked for deletion).

`parseTopicPartitionName` uses all characters up to the last `-` for the topic name and the rest as the partition ID.

Note	`parseTopicPartitionName` is used when: • `Log` is created • `LogManager` is requested to load a partition log directory

`offsetFromFileName` Object Method

```
offsetFromFileName(filename: String): Long
```

`offsetFromFileName` ...FIXME

Note	`offsetFromFileName` is used when `Log` is requested to removeTempFilesAndCollectSwapFiles (right when created) and offsetFromFile.

`offsetFromFile` Object Method

```
offsetFromFile(file: File): Long
```

`offsetFromFile` ...FIXME

Note	`offsetFromFile` is used when…FIXME

Reading Records — `read` Method

```
read(
  startOffset: Long,
  maxLength: Int,
  maxOffset: Option[Long],
  minOneMessage: Boolean,
  includeAbortedTxns: Boolean): FetchDataInfo
```

`read` …FIXME

Note	`read` is used when: • `Partition` is requested to read records • `GroupMetadataManager` is requested to doLoadGroupsAndOffsets • `TransactionStateManager` is requested to loadTransactionMetadata • `Log` is requested to convertToOffsetMetadata

`convertToOffsetMetadata` Method

```
convertToOffsetMetadata(
  offset: Long): Option[LogOffsetMetadata]
```

`convertToOffsetMetadata` …FIXME

Note	`convertToOffsetMetadata` is used exclusively when `Replica` is requested to convertHWToLocalOffsetMetadata

`logEndOffset` Method

```
logEndOffset: Long
```

`logEndOffset` is the offset of the next message that will be appended to the log (based on the nextOffsetMetadata internal registry).

Note	`logEndOffset` is used when…FIXME

`addSegment` Method

```
addSegment(
  segment: LogSegment): LogSegment
```

`addSegment` simply associates the given LogSegment with the baseOffset in the segments internal registry.

Note	`addSegment` is used when `Log` is requested to replaceSegments, loadSegmentFiles, loadSegments, recoverLog, roll, and truncateFullyAndStartAt.

`updateLogEndOffset` Internal Method

```
updateLogEndOffset(messageOffset: Long): Unit
```

`updateLogEndOffset` simply creates a new `LogOffsetMetadata` (with the `messageOffset`, active segment) and becomes the nextOffsetMetadata internal registry.

Note	`updateLogEndOffset` is used when `Log` is requested to append records, roll log segment, truncateTo, and truncateFullyAndStartAt.

`activeSegment` Method

```
activeSegment: LogSegment
```

`activeSegment` gives the active LogSegment that is currently taking appends (that is the greatest key in the segments internal registry).

Note	`activeSegment` is used exclusively when `Log` is created (to create a LogOffsetMetadata).

Updating Config — `updateConfig` Method

```
updateConfig(
  updatedKeys: Set[String],
  newConfig: LogConfig): Unit
```

`updateConfig` replaces the current LogConfig with the given LogConfig.

If `message.format.version` is among the updated keys, `updateConfig` ...FIXME

Note	`updateConfig` is used when: • `TopicConfigHandler` is requested to process configuration changes • `DynamicLogConfig` is requested to reconfigure

`renameDir` Method

```
renameDir(
  name: String): Unit
```

`renameDir` ...FIXME

Note	`renameDir` is used when `LogManager` is requested to replaceCurrentWithFutureLog and asyncDelete.

`initFileSize` Method

```
initFileSize: Int
```

`initFileSize` ...FIXME

Note	`initFileSize` is used when...FIXME

Creating Log File — `logFile` Utility

```
logFile(
  dir: File,
  offset: Long,
  suffix: String = ""): File
```

`logFile` creates a prefix of the file name of a log segment (for the given `offset`, the .log suffix and the optional `suffix`) in the given `dir` directory.

logFile's Example

```
import java.nio.file.{Files, Path}
import java.io.File
val tmp = "/tmp/kafka-internals"
val p = Path.of(tmp)
val dir = if (Files.exists(p)) {
  new File(tmp)
} else {
  Files.createDirectory(p).toFile()
}

import kafka.log.Log
val log_file = Log.logFile(dir, offset = 10, suffix = ".suffix")
assert(log_file.getName == "00000000000000000010.log.suffix")
```

Note	`logFile` is used when: • `LogSegment` utility is used to open a log segment and deleteIfExists • `Log` is requested to loadSegmentFiles and roll

`offsetIndexFile` Method

```
offsetIndexFile(
  dir: File,
  offset: Long,
  suffix: String = ""): File
```

`offsetIndexFile` creates a prefix of the file name of a log segment (for the given `offset`, the .index suffix and the optional `suffix`) in the given `dir` directory.

offsetIndexFile's Example

```
import java.nio.file.{Files, Path}
import java.io.File
val tmp = "/tmp/kafka-internals"
val p = Path.of(tmp)
val dir = if (Files.exists(p)) {
  new File(tmp)
} else {
  Files.createDirectory(p).toFile()
}

import kafka.log.Log
val log_file = Log.offsetIndexFile(dir, offset = 10, suffix = ".suffix")
assert(log_file.getName == "00000000000000000010.index.suffix")
```

Note	`offsetIndexFile` is used when: • `LogSegment` utility is used to open a log segment and deleteIfExists • `Log` is requested to removeTempFilesAndCollectSwapFiles and roll

`timeIndexFile` Method

```
timeIndexFile(
  dir: File,
  offset: Long,
  suffix: String = ""): File
```

`timeIndexFile` creates a prefix of the file name of a log segment (for the given `offset`, the .timeindex suffix and the optional `suffix`) in the given `dir` directory.

timeIndexFile's Example

```
import java.nio.file.{Files, Path}
import java.io.File
val tmp = "/tmp/kafka-internals"
val p = Path.of(tmp)
val dir = if (Files.exists(p)) {
  new File(tmp)
} else {
  Files.createDirectory(p).toFile()
}

import kafka.log.Log
val log_file = Log.timeIndexFile(dir, offset = 10, suffix = ".suffix")
assert(log_file.getName == "00000000000000000010.timeindex.suffix")
```

Note	`timeIndexFile` is used when: • `LogSegment` utility is used to open a log segment and deleteIfExists • `Log` is requested to removeTempFilesAndCollectSwapFiles, loadSegmentFiles and roll

`transactionIndexFile` Method

```
transactionIndexFile(
  dir: File,
  offset: Long,
  suffix: String = ""): File
```

`transactionIndexFile` creates a prefix of the file name of a log segment (for the given `offset`, the .txnindex suffix and the optional `suffix`) in the given `dir` directory.

transactionIndexFile's Example

```
import java.nio.file.{Files, Path}
import java.io.File
val tmp = "/tmp/kafka-internals"
val p = Path.of(tmp)
val dir = if (Files.exists(p)) {
  new File(tmp)
} else {
  Files.createDirectory(p).toFile()
}

import kafka.log.Log
val log_file = Log.transactionIndexFile(dir, offset = 10, suffix = ".suffix")
assert(log_file.getName == "00000000000000000010.txnindex.suffix")
```

Note	`transactionIndexFile` is used when: • `LogSegment` utility is used to open a log segment and deleteIfExists • `Log` is requested to removeTempFilesAndCollectSwapFiles and roll

`producerSnapshotFile` Method

```
producerSnapshotFile(
  dir: File,
  offset: Long): File
```

`producerSnapshotFile` creates a prefix of the file name of a log segment (for the given `offset` and the .snapshot suffix) in the given `dir` directory.

producerSnapshotFile's Example

```
import java.nio.file.{Files, Path}
import java.io.File
val tmp = "/tmp/kafka-internals"
val p = Path.of(tmp)
val dir = if (Files.exists(p)) {
  new File(tmp)
} else {
  Files.createDirectory(p).toFile()
}

import kafka.log.Log
val log_file = Log.producerSnapshotFile(dir, offset = 10)
assert(log_file.getName == "00000000000000000010.snapshot")
```

Note	`producerSnapshotFile` is used exclusively when `ProducerStateManager` is requested to takeSnapshot.

Creating File Name Of Log Segment (From Offset) — `filenamePrefixFromOffset` Utility

```
filenamePrefixFromOffset(
  offset: Long): String
```

`filenamePrefixFromOffset` uses java.text.NumberFormat to format the given `offset`:

- Minimum number of digits: 20
- Maximum number of digits in the fraction portion of a number: 0
- No grouping used

```
import kafka.log.Log
val filenamePrefix = Log.filenamePrefixFromOffset(offset = 10)
assert(filenamePrefix == "00000000000000000010")
```

Note	`filenamePrefixFromOffset` is used when `Log` utility is used to create file names for logFile, offsetIndexFile, timeIndexFile, producerSnapshotFile, and transactionIndexFile.

Closing Log — `close` Method

```
close(): Unit
```

`close` ...FIXME

Note	`close` is used when...FIXME

`maybeAssignEpochStartOffset` Method

```
maybeAssignEpochStartOffset(
  leaderEpoch: Int,
  startOffset: Long): Unit
```

`maybeAssignEpochStartOffset` ...FIXME

Note	`maybeAssignEpochStartOffset` is used when...FIXME

Log Size (in Bytes) — `size` Method

```
size: Long
```

`size` uses the `Log` utility for the size of the segments.

Note	`size` is used when...FIXME

`sizeInBytes` Utility

```
sizeInBytes(
  segments: Iterable[LogSegment]): Long
```

`sizeInBytes` sums up the sizes of the given log segments.

Note	`sizeInBytes` is used when `Log` is requested for the size.

Loading Segments — `loadSegments` Internal Method

```
loadSegments(): Long
```

`loadSegments` removeTempFilesAndCollectSwapFiles.

`loadSegments` loadSegmentFiles (with retries when there are log segments with offset overflow).

`loadSegments` completeSwapOperations.

`loadSegments` branches off per whether the log directory is scheduled to be deleted or not.

Note	`loadSegments` is used when `Log` is created (to create a LogOffsetMetadata).

`loadSegments` Internal Method and Log Directory Not Scheduled For Deletion

For the log directory that is not scheduled to be deleted, `loadSegments` recoverLog.

`loadSegments` requests the active segment to resizeIndexes (to the value of segment.index.bytes configuration property).

In the end, `loadSegments` returns the next offset after recovery.

`loadSegments` Internal Method and Log Directory Scheduled For Deletion

For the log directory that is scheduled to be deleted, `loadSegments` adds a new log segment (with base offset `0` and initFileSize).

In the end, `loadSegments` returns `0`.

`recoverLog` Internal Method

```
recoverLog(): Long
```

`recoverLog` ...FIXME

Note	`recoverLog` is used when `Log` is requested to loadSegments.

`removeTempFilesAndCollectSwapFiles` Internal Method

```
removeTempFilesAndCollectSwapFiles(): Set[File]
```

`removeTempFilesAndCollectSwapFiles` ...FIXME

Note	`removeTempFilesAndCollectSwapFiles` is used exclusively when `Log` is requested to loadSegments (right when created).

Loading Segment Files — `loadSegmentFiles` Internal Method

```
loadSegmentFiles(): Unit
```

`loadSegmentFiles` processes index and log files in the log directory (by name in ascending order).

Internally, `loadSegmentFiles` finds all the files (sorted by name) in the log directory and branches off per whether a file is an index or a log file.

Note	`loadSegmentFiles` is used when `Log` is created (and in turn loadSegments).

Loading Log Files

For a log file, `loadSegmentFiles` opens it and requests sanityCheck.

In case of `NoSuchFileException`, `loadSegmentFiles` prints out the following ERROR to the logs and recovers the segment.

```
Could not find offset index file corresponding to log file
[path], recovering segment and rebuilding index files...
```

In case of `CorruptIndexException`, `loadSegmentFiles` prints out the following ERROR to the logs and recovers the segment.

```
Found a corrupted index file corresponding to log file [path]
due to [message], recovering segment and rebuilding index
files...
```

In the end, `loadSegmentFiles` addSegment.

Loading Index Files

For index files, `loadSegmentFiles` simply makes sure that they have corresponding .log files (in the same log directory).

If an orphaned index file is found, `loadSegmentFiles` simply prints out the following WARN message and deletes the file:

```
Found an orphaned index file [path], with no corresponding log file.
```

`isIndexFile` Internal Object Method

```
isIndexFile(file: File): Boolean
```

`isIndexFile` is positive (`true`) when the given file has got one of the following file suffices:

- .index
- .timeindex
- .txnindex

Otherwise, `isIndexFile` is `false` .

Note	`isIndexFile` is used when `Log` is requested to removeTempFilesAndCollectSwapFiles and loadSegmentFiles.

`isLogFile` Internal Object Method

```
isLogFile(file: File): Boolean
```

`isLogFile` returns `true` when the given file has .log file suffix. Otherwise, `isLogFile` is `false` .

Note	`isLogFile` is used when `Log` is requested to removeTempFilesAndCollectSwapFiles, loadSegmentFiles, and splitOverflowedSegment.

Recovering Log Segment — `recoverSegment` Internal Method

```
recoverSegment(
  segment: LogSegment,
  leaderEpochCache: Option[LeaderEpochFileCache] = None): Int
```

`recoverSegment` creates a new ProducerStateManager (for the TopicPartition, log directory and maxProducerIdExpirationMs).

Note	Why does `recoverSegment` create a new ProducerStateManager rather than using the ProducerStateManager?

`recoverSegment` then rebuildProducerState (with the baseOffset of the LogSegment, the `reloadFromCleanShutdown` flag off, and the new `ProducerStateManager`).

`recoverSegment` requests the given `LogSegment` to recover (with the new `ProducerStateManager` and the optional `LeaderEpochFileCache`).

`recoverSegment` requests the `ProducerStateManager` to takeSnapshot.

`recoverSegment` returns the number of bytes truncated from the log (while doing segment recovery).

Note	`recoverSegment` is used when `Log` is requested to loadSegmentFiles, completeSwapOperations, and recoverLog.

`loadProducerState` Internal Method

```
loadProducerState(
  lastOffset: Long,
  reloadFromCleanShutdown: Boolean): Unit
```

`loadProducerState` rebuildProducerState (with the `lastOffset`, `reloadFromCleanShutdown` and the ProducerStateManager).

In the end, `loadProducerState` updateFirstUnstableOffset.

Note	`loadProducerState` is used when `Log` is created and requested to truncateTo.

`rebuildProducerState` Internal Method

```
rebuildProducerState(
  lastOffset: Long,
  reloadFromCleanShutdown: Boolean,
  producerStateManager: ProducerStateManager): Unit
```

`rebuildProducerState` ...FIXME

Note	`rebuildProducerState` is used when `Log` is requested to recoverSegment and loadProducerState.

`updateFirstUnstableOffset` Internal Method

```
updateFirstUnstableOffset(): Unit
```

`updateFirstUnstableOffset` ...FIXME

Note	`updateFirstUnstableOffset` is used when `Log` is requested to loadProducerState, append, onHighWatermarkIncremented, maybeIncrementLogStartOffset, and truncateFullyAndStartAt.

`completeSwapOperations` Internal Method

```
completeSwapOperations(
  swapFiles: Set[File]): Unit
```

`completeSwapOperations` ...FIXME

Note	`completeSwapOperations` is used when `Log` is created (and in turn loadSegments).

`retryOnOffsetOverflow` Internal Method

```
retryOnOffsetOverflow[T](fn: => T): T
```

`retryOnOffsetOverflow` executes the `fn` block and returns the result.

In case of `LogSegmentOffsetOverflowException`, `retryOnOffsetOverflow` prints out the following INFO message to the logs, splitOverflowedSegment and retries execution of the `fn` block.

```
Caught segment overflow error: [message]. Split segment and retry.
```

Note	`retryOnOffsetOverflow` is used exclusively when `Log` is requested to loadSegments.

`initializeLeaderEpochCache` Internal Method

```
initializeLeaderEpochCache(): Unit
```

`initializeLeaderEpochCache` ...FIXME

Note	`initializeLeaderEpochCache` is used when `Log` is created and later requested to updateConfig and renameDir.

Internal Properties

Name	Description
`nextOffsetMetadata`	`LogOffsetMetadata` (*log end offset*) of the next message that will be appended to the log • Initialized right when `Log` is created • Updated when updateLogEndOffset Used when: • `Log` is created and then requested to append, read, roll, and for the logEndOffset • `Replica` is requested for logEndOffsetMetadata

LogSegment — Record Segment Of Partition Log

`LogSegment` is a segment of records of a log of a partition.

Tip	Use DumpLogSegments tool to review the content of (the underlying files of) a log segment.

`LogSegment` is composed of two main file types, e.g. the log file itself (with records) and index files.

- index
- time index
- transaction index (of aborted transactions)

Note	When Log is created (and requested to loadSegmentFiles) orphaned index files are removed.

The files are all in the same directory (as specified when opening a segment).

```
$ tree /tmp/kafka-logs/t1-1/
/tmp/kafka-logs/t1-1/
├── 00000000000000000000.index
├── 00000000000000000000.log
├── 00000000000000000000.timeindex
└── leader-epoch-checkpoint
```

`LogSegment` is created (indirectly using LogSegment.open utility) when:

- `Log` is requested to loadSegmentFiles (for every log file), completeSwapOperations, loadSegments, recoverLog, roll, and truncateFullyAndStartAt
- `LogCleaner` is requested to createNewCleanedSegment

Example

```
// One of log.dirs directories
import java.nio.file.Path
val dir = Path.of("/tmp/logsegment")
import java.nio.file.Files
Files.createDirectories(dir)
import kafka.log.{LogConfig, LogSegment}
import org.apache.kafka.common.utils.Time
val config = LogConfig()
val segment = LogSegment.open(dir.toFile, baseOffset = 0, config, time = Time.SYSTEM)
scala> println(segment)
LogSegment(baseOffset=0, size=0, lastModifiedTime=1576677605692, largestTime=0)
```

`LogSegment` uses the following configuration properties (while opening a log segment):

- segment.index.bytes
- index.interval.bytes
- segment.jitter.ms or segment.ms

Creating LogSegment Instance

`LogSegment` takes the following to be created:

- FileRecords
- Lazy OffsetIndex with deferred loading (`LazyIndex[OffsetIndex]`)
- Lazy TimeIndex with deferred loading (`LazyIndex[TimeIndex]`)
- TransactionIndex
- Base offset
- index.interval.bytes
- segment.jitter.ms or segment.ms
- `Time`

`LogSegment` initializes the internal properties.

Opening Log Segment — `open` Utility

```
open(
  dir: File,
  baseOffset: Long,
  config: LogConfig,
  time: Time,
  fileAlreadyExists: Boolean = false,
  initFileSize: Int = 0,
  preallocate: Boolean = false,
  fileSuffix: String = ""): LogSegment
```

`open` uses the following configuration properties (of the given LogConfig):

- segment.index.bytes for the `maxIndexSize`
- index.interval.bytes for the indexIntervalBytes
- segment.jitter.ms or segment.ms for the rollJitterMs

`open` creates a new LogSegment with the following files in the given `dir` directory:

- Creates a log file (with the given `baseOffset`, and `fileSuffix`) and opens it (using `FileRecords.open`)
- Creates an offset index file
- Creates a time index file
- Creates a transaction index file

Note	`open` is used when: • `Log` is requested to loadSegmentFiles (for every log file), completeSwapOperations, loadSegments, recoverLog, roll, and truncateFullyAndStartAt • `LogCleaner` is requested to createNewCleanedSegment

LogSegment and TransactionIndex

```
txnIndex: TransactionIndex
```

When created, `LogSegment` is given a TransactionIndex.

`LogSegment` uses the `TransactionIndex` for the following:

- truncateTo
- updateTxnIndex

- collectAbortedTxns
- flush
- sanityCheck and recover
- updateDir and changeFileSuffixes

`txnIndex` is also used when `Cleaner` is requested to clean a log (and in turn cleanSegments)

`TransactionIndex` is closed when `LogSegment` is requested to close and closeHandlers.

`TransactionIndex` is deleted (if exists) when `LogSegment` is requested to deleteIfExists.

LogSegment and OffsetIndex

```
offsetIndex: OffsetIndex
```

When created, `LogSegment` is given an OffsetIndex with deferred loading (`LazyIndex[OffsetIndex]`).

`offsetIndex` simply gets (*unwraps*) the OffsetIndex.

`offsetIndex` is used for the following…FIXME

`collectAbortedTxns` Method

```
collectAbortedTxns(
  fetchOffset: Long,
  upperBoundOffset: Long): TxnIndexSearchResult
```

`collectAbortedTxns` …FIXME

Note	`collectAbortedTxns` is used when `Log` is requested to collectAbortedTransactions.

Closing — `close` Method

```
close(): Unit
```

`close` …FIXME

Note	`close` is used when `Log` is requested to load segments, close, and [kafka-log-Log#splitOverflowedSegment splitOverflowedSegment].

`closeHandlers` Method

```
closeHandlers(): Unit
```

`closeHandlers` ...FIXME

Note	`closeHandlers` is used when `Log` is requested to closeHandlers.

Recovering — `recover` Method

```
recover(
  producerStateManager: ProducerStateManager,
  leaderEpochCache: Option[LeaderEpochFileCache] = None): Int
```

`recover` requests the OffsetIndex, the TimeIndex, and the TransactionIndex to reset.

Note	`recover` is used when `Log` is requested to recover a log segment.

`updateProducerState` Internal Method

```
updateProducerState(
  producerStateManager: ProducerStateManager,
  batch: RecordBatch): Unit
```

`updateProducerState` ...FIXME

Note	`updateProducerState` is used when `LogSegment` is requested to recover.

`sanityCheck` Method

```
sanityCheck(
  timeIndexFileNewlyCreated: Boolean): Unit
```

`sanityCheck` ...FIXME

Note	`sanityCheck` is used exclusively when `Log` is requested to loadSegments (when created).

`updateDir` Method

```
updateDir(
  dir: File): Unit
```

`updateDir` ...FIXME

Note	`updateDir` is used exclusively when `Log` is requested to renameDir.

`changeFileSuffixes` Method

```
changeFileSuffixes(
  oldSuffix: String,
  newSuffix: String): Unit
```

`changeFileSuffixes` ...FIXME

Note	`changeFileSuffixes` is used when `Log` is requested to asyncDeleteSegment and replaceSegments.

`flush` Method

```
flush(): Unit
```

`flush` ...FIXME

Note	`flush` is used when: • `Log` is requested to flush and splitOverflowedSegment • `Cleaner` is requested to clean a log (and cleanSegments)

`deleteIfExists` Method

```
deleteIfExists(): Unit
```

`deleteIfExists` ...FIXME

Note	`deleteIfExists` is used when…FIXME

`deleteIfExists` Utility

```
deleteIfExists(
  dir: File,
  baseOffset: Long,
  fileSuffix: String = ""): Unit
```

`deleteIfExists` …FIXME

Note	`deleteIfExists` is used when…FIXME

`resizeIndexes` Method

```
resizeIndexes(
  size: Int): Unit
```

`resizeIndexes` …FIXME

Note	`resizeIndexes` is used when…FIXME

`largestTimestamp` Method

```
largestTimestamp: Long
```

`largestTimestamp` …FIXME

Note	`largestTimestamp` is used when…FIXME

`shouldRoll` Method

```
shouldRoll(
  rollParams: RollParams): Boolean
```

`shouldRoll` …FIXME

Note	`shouldRoll` is used exclusively when `Log` is requested to maybeRoll (while appending records).

`timeWaitedForRoll` Method

```
timeWaitedForRoll(
  now: Long,
  messageTimestamp: Long) : Long
```

`timeWaitedForRoll` ...FIXME

Note	`timeWaitedForRoll` is used exclusively when `LogSegment` is requested to shouldRoll.

`append` Method

```
append(
  largestOffset: Long,
  largestTimestamp: Long,
  shallowOffsetOfMaxTimestamp: Long,
  records: MemoryRecords): Unit
```

`append` ...FIXME

Note	`append` is used exclusively when: • `Log` is requested to append records • `Cleaner` is requested to clean a log (and cleanInto) • `LogSegment` is requested to appendChunkFromFile

`appendFromFile` Method

```
appendFromFile(
  records: FileRecords,
  start: Int): Int
```

`appendFromFile` ...FIXME

Note	`appendFromFile` is used exclusively when `Log` is requested to splitOverflowedSegment.

`appendChunkFromFile` Internal Method

```
appendChunkFromFile(
  records: FileRecords,
  position: Int,
  bufferSupplier: BufferSupplier): Int
```

`appendChunkFromFile` ...FIXME

Note	`appendChunkFromFile` is used when `LogSegment` is requested to appendFromFile.

Truncating To Offset — `truncateTo` Method

```
truncateTo(
  offset: Long): Int
```

`truncateTo` ...FIXME

Note	`truncateTo` is used when `Log` is created (and in turn recoverLog) and truncateTo.

`updateTxnIndex` Method

```
updateTxnIndex(
  completedTxn: CompletedTxn,
  lastStableOffset: Long): Unit
```

`updateTxnIndex` ...FIXME

Note	`updateTxnIndex` is used when: • `Log` is requested to append records • `LogSegment` is requested to updateProducerState

`size` Method

```
size: Int
```

`size` requests the FileRecords for the size (in bytes).

Note	`size` is used when...FIXME

Internal Properties

Name	Description
`created`	Time(stamp) when this `LogSegment` was created or truncated completely (to `0`) Used exclusively when `LogSegment` is requested for the time it has waited to be rolled
`bytesSinceLastIndexEntry`	Used when…FIXME
`rollingBasedTimestamp`	Used when…FIXME
`_maxTimestampSoFar`	Used when…FIXME
`_offsetOfMaxTimestampSoFar`	Used when…FIXME

LogCleanerManager

`LogCleanerManager` is created exclusively for LogCleaner.

`LogCleanerManager` is a KafkaMetricsGroup and registers performance metrics.

Table 1. LogCleanerManager's Performance Metrics

Metric Name	Description
`max-dirty-percent`	
`time-since-last-run-ms`	
`uncleanable-bytes`	(for every log directory)
`uncleanable-partitions-count`	(for every log directory)

The performance metrics are registered in **kafka.log:type=LogCleanerManager** group.

Figure 1. LogCleanerManager in jconsole

`LogCleanerManager` uses **cleaner-offset-checkpoint** for the name of the offset checkpoint files for every log directory (in checkpoints registry).

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

Name	Description
`checkpoints`	`OffsetCheckpointFiles` per log directory Used in updateCheckpoints, maybeTruncateCheckpoint, and alterCheckpointDir to look up the `OffsetCheckpointFile` per log data directory Used in allCleanerCheckpoints to access all `OffsetCheckpointFiles` Used in handleLogDirFailure to exclude a failed log directory from cleaning

`LogCleanerManager` uses kafka.log.LogCleaner logger.

Tip	Enable `INFO` logging level for `kafka.log.LogCleaner` logger to see what happens inside.

Creating LogCleanerManager Instance

`LogCleanerManager` takes the following to be created:

- Log directories
- Logs per Kafka `TopicPartition`
- `LogDirFailureChannel`

`LogCleanerManager` initializes the internal registries and counters.

`isCompactAndDelete` Utility

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

`isCompactAndDelete` flag is enabled (`true`) when compact and delete cleanup policies are both included in the cleanup.policy configuration property. Otherwise, `isCompactAndDelete` flag is disabled (`false`).

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

`grabFilthiestCompactedLog` Method

```
grabFilthiestCompactedLog(time: Time): Option[LogToClean]
```

`grabFilthiestCompactedLog` ...FIXME

Note	`grabFilthiestCompactedLog` is used when...FIXME

`deletableLogs` Method

```
deletableLogs(): Iterable[(TopicPartition, Log)]
```

`deletableLogs` ...FIXME

Note	`deletableLogs` is used when...FIXME

`doneDeleting` Method

```
doneDeleting(topicPartitions: Iterable[TopicPartition]): Unit
```

`doneDeleting` ...FIXME

Note	`doneDeleting` is used when...FIXME

`doneCleaning` Method

```
doneCleaning(
  topicPartition: TopicPartition,
  dataDir: File,
  endOffset: Long): Unit
```

`doneCleaning` ...FIXME

Note	`doneCleaning` is used exclusively when `CleanerThread` is requested to cleanLog.

`allCleanerCheckpoints` Method

```
allCleanerCheckpoints: Map[TopicPartition, Long]
```

`allCleanerCheckpoints` ...FIXME

Note	`allCleanerCheckpoints` is used when: • `LogCleanerManager` is requested to grabFilthiestCompactedLog and for the uncleanable-bytes performance metric • `LogCleaner` is requested to awaitCleaned (for tests only)

`alterCheckpointDir` Method

```
alterCheckpointDir(
  topicPartition: TopicPartition,
  sourceLogDir: File,
  destLogDir: File): Unit
```

`alterCheckpointDir` ...FIXME

Note	`alterCheckpointDir` is used exclusively when `LogCleaner` is requested to alterCheckpointDir.

`handleLogDirFailure` Method

```
handleLogDirFailure(dir: String): Unit
```

`handleLogDirFailure` ...FIXME

Note	`handleLogDirFailure` is used exclusively when `LogCleaner` is requested to handleLogDirFailure.

`updateCheckpoints` Method

```
updateCheckpoints(
  dataDir: File,
  update: Option[(TopicPartition, Long)]): Unit
```

`updateCheckpoints` ...FIXME

Note	`updateCheckpoints` is used when: • `LogCleaner` is requested to updateCheckpoints • `LogCleanerManager` is requested to alterCheckpointDir and doneCleaning

`maybeTruncateCheckpoint` Method

```
maybeTruncateCheckpoint(dataDir: File, topicPartition: TopicPartition, offset: Long):
Unit
```

`maybeTruncateCheckpoint` ...FIXME

Note	`maybeTruncateCheckpoint` is used exclusively when `LogCleaner` is requested to maybeTruncateCheckpoint.

LogCleaner

`LogCleaner` manages one or more CleanerThreads to remove obsolete records from logs with compact retention strategy.

`LogCleaner` is created exclusively for LogManager with enableCleaner flag enabled (of the CleanerConfig of the parent LogManager) (which is the default).

When created, `LogCleaner` is given a CleanerConfig that `LogManager` builds when created (when `KafkaServer` is requested to start up).

`LogCleaner` registers performance metrics (as a KafkaMetricsGroup).

`LogCleaner` supports dynamic (re)configuration (as a BrokerReconfigurable).

Tip	Enable `ALL` logging levels for `kafka.log.LogCleaner` logger to see what happens inside.
	Add the following line to `config/log4j.properties`:
	`log4j.logger.kafka.log.LogCleaner=ALL`
	Refer to Logging.
	--- 	
	Please note that Kafka comes with a preconfigured `kafka.log.LogCleaner` logger in `config/log4j.properties`:
	`log4j.appender.cleanerAppender=org.apache.log4j.DailyRollingFileAppender`
	`log4j.appender.cleanerAppender.DatePattern='.'yyyy-MM-dd-HH`
	`log4j.appender.cleanerAppender.File=${kafka.logs.dir}/log-cleaner.log`
	`log4j.appender.cleanerAppender.layout=org.apache.log4j.PatternLayout`
	`log4j.appender.cleanerAppender.layout.ConversionPattern=[%d] %p %m (%c)%n`
	`log4j.logger.kafka.log.LogCleaner=INFO, cleanerAppender`
	`log4j.additivity.kafka.log.LogCleaner=false`
	That means that the logs of `LogCleaner` go to `logs/log-cleaner.log` file at `INFO` logging level and are not added to the main logs (per `log4j.additivity` being off).

Creating LogCleaner Instance

`LogCleaner` takes the following to be created:

- CleanerConfig

- Log directories
- Logs per Kafka `TopicPartition`
- `LogDirFailureChannel`
- `Time` (default: `Time.SYSTEM`)

`LogCleaner` initializes the internal properties.

LogCleaner And CleanerThreads

`LogCleaner` manages CleanerThreads for cleaning up dirty logs.

`CleanerThreads` are created and started at startup. They are doing their join until shutdown.

The number of `CleanerThreads` in use is determined by the numThreads of the CleanerConfig (default: 1).

The statistics about the last log cleaning of `CleanerThreads` are used for the performance metrics.

Performance Metrics

`LogCleaner` is a KafkaMetricsGroup with the following performance metrics.

Table 1. LogCleaner's Performance Metrics

Metric Name	Description
`cleaner-recopy-percent`	Sum of `bytesWritten` by sum of `bytesRead` of the cleaners in the last cleaning
`max-buffer-utilization-percent`	Maximum `bufferUtilization` of the cleaners in the last cleaning
`max-clean-time-secs`	Maximum cleaning time of the cleaners in the last cleaning
`max-compaction-delay-secs`	Maximum delay between the time when a log is required to be compacted and the time of last cleaner run

The performance metrics are registered in **kafka.log:type=LogCleaner** group.

Figure 1. LogCleaner in jconsole

LogCleaner And CleanerConfig

```
config: CleanerConfig
```

`LogCleaner` uses CleanerConfig for dynamic configurations (that can be changed via reconfigure).

`LogCleaner` is first given the initial CleanerConfig when created (from LogManager based on the `KafkaConfig` from `KafkaServer`).

Dynamic (Re)Configuration

`LogCleaner` is a BrokerReconfigurable for the following dynamic configurations:

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

- message.max.bytes

`createNewCleanedSegment` Object Method

```
createNewCleanedSegment(
  log: Log,
  baseOffset: Long): LogSegment
```

`createNewCleanedSegment` ...FIXME

Note	`createNewCleanedSegment` is used when: • `Log` is requested to splitOverflowedSegment • `LogCleaner` is requested to cleanSegments

`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

Reconfiguring — `reconfigure` Method

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

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

`reconfigure` ...FIXME

Starting Up — `startup` Method

```
startup(): Unit
```

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

```
Starting the log cleaner
```

`startup` creates new CleanerThreads and starts them all immediately.

`startup` adds the cleaner threads in cleaners internal registry.

Note	The number of `CleanerThreads` is controlled by log.cleaner.threads dynamic configuration (default: `1`).

Note	`startup` is used when: • `LogManager` is requested to start up (with `enableCleaner` enabled which is the default) • `LogCleaner` is requested to reconfigure

Building CleanerConfig From KafkaConfig — `cleanerConfig` Utility

```
cleanerConfig(
  config: KafkaConfig): CleanerConfig
```

`cleanerConfig` simply creates a new CleanerConfig based on the given KafkaConfig.

Note	`cleanerConfig` is used when: • `LogCleaner` is requested to validateReconfiguration and reconfigure • `LogManager` is created

`awaitCleaned` Method

```
awaitCleaned(
  topicPartition: TopicPartition,
  offset: Long,
  maxWaitMs: Long = 60000L): Boolean
```

`awaitCleaned` ...FIXME

Note	`awaitCleaned` seems to be used exclusively in tests.

`alterCheckpointDir` Method

```
alterCheckpointDir(
  topicPartition: TopicPartition,
  sourceLogDir: File,
  destLogDir: File): Unit
```

`alterCheckpointDir` ...FIXME

Note	`alterCheckpointDir` is used exclusively when `LogManager` is requested to replaceCurrentWithFutureLog.

`handleLogDirFailure` Method

```
handleLogDirFailure(dir: String): Unit
```

`handleLogDirFailure` ...FIXME

Note	`handleLogDirFailure` is used exclusively when `LogManager` is requested to handleLogDirFailure.

`updateCheckpoints` Method

```
updateCheckpoints(dataDir: File): Unit
```

`updateCheckpoints` ...FIXME

Note	`updateCheckpoints` is used exclusively when `LogManager` is requested to asyncDelete.

`maybeTruncateCheckpoint` Method

```
maybeTruncateCheckpoint(
  dataDir: File,
  topicPartition: TopicPartition,
  offset: Long): Unit
```

`maybeTruncateCheckpoint` ...FIXME

Note	`maybeTruncateCheckpoint` is used when `LogManager` is requested to truncateTo and truncateFullyAndStartAt.

Shuting Down — `shutdown` Method

```
shutdown(): Unit
```

`shutdown` ...FIXME

Note	`shutdown` is used when...FIXME

`abortAndPauseCleaning` Method

```
abortAndPauseCleaning(
  topicPartition: TopicPartition): Unit
```

`abortAndPauseCleaning` ...FIXME

Note	`abortAndPauseCleaning` is used when...FIXME

`resumeCleaning` Method

```
resumeCleaning(
  topicPartitions: Iterable[TopicPartition]): Unit
```

`resumeCleaning` ...FIXME

Note	`resumeCleaning` is used when…FIXME

`pauseCleaningForNonCompactedPartitions` Method

```
pauseCleaningForNonCompactedPartitions(): Iterable[(TopicPartition, Log)]
```

`pauseCleaningForNonCompactedPartitions`…FIXME

Note	`pauseCleaningForNonCompactedPartitions` is used when…FIXME

Internal Properties

Name	Description
`cleanerManager`	LogCleanerManager

CleanerConfig

`CleanerConfig` represents a set of dynamic configurations of a LogCleaner:

- log.cleaner.threads (default: `1`)
- dedupeBufferSize (default: `4*1024*1024L`)
- dedupeBufferLoadFactor (default: `0.9d`)
- ioBufferSize (default: `1024*1024`)
- maxMessageSize (default: `32*1024*1024`)
- maxIoBytesPerSecond (default: `Double.MaxValue`)
- backOffMs (default: `15 * 1000`)
- log.cleaner.enable flag (default: `true`)
- `hashAlgorithm` (default: `MD5`)

`CleanerConfig` is created exclusively when `LogCleaner` is requested to build a CleanerConfig from a KafkaConfig.

CleanerThread

`CleanerThread` is a non-daemon thread of execution for cleaning up dirty logs (one at a time until no more is left).

`CleanerThread` uses the thread name of the format **kafka-log-cleaner-thread-** followed by the threadId.

`CleanerThread` is created and immediately started when `LogCleaner` is requested to start up.

Note	The number of `CleanerThreads` (that LogCleaner uses) is controlled by log.cleaner.threads dynamic configuration (default: `1`).

`CleanerThread` takes a **threadId** to be created.

`CleanerThread` uses kafka.log.LogCleaner logger.

Tip	Enable `ALL` logging level for kafka.log.LogCleaner logger to see what happens inside.

Note	`CleanerThread` is a Java private inner class of LogCleaner. It can only be used by `LogCleaner` and have a direct access to the internals of the `LogCleaner` .

CleanerThread and Cleaner

When created, `CleanerThread` creates a Cleaner (with the ID as the given threadId).

`CleanerThread` uses the `Cleaner` exclusively for cleaning up a log.

`checkDone` Internal Method

```
checkDone(topicPartition: TopicPartition): Unit
```

`checkDone` ...FIXME

Note	`checkDone` is used exclusively when `CleanerThread` is created (to create the Cleaner).

Cleaning Up Dirty Logs — `doWork` Method

```
doWork(): Unit
```

Note	`doWork` is part of the ShutdownableThread Contract to do the work.

`doWork` simply cleanFilthiestLog.

If there was no dirty log cleaned up, `doWork` pauses for the backOffMs (of the CleanerConfig of the parent LogCleaner) (default: 15 seconds).

`cleanFilthiestLog` Internal Method

```
cleanFilthiestLog(): Boolean
```

`cleanFilthiestLog` requests the LogCleanerManager to grabFilthiestCompactedLog.

If there is any filthiest log to work on, `cleanFilthiestLog` cleanLog and turns the `cleaned` flag on.

With no logs to work on, `cleanFilthiestLog` simply turns the `cleaned` flag off.

Note	The value of `cleaned` flag is what `cleanFilthiestLog` returns.

`cleanFilthiestLog` requests the LogCleanerManager for the deletable logs and then requests every `Log` to deleteOldSegments.

In the end, `cleanFilthiestLog` requests the LogCleanerManager to doneDeleting (with the `TopicPartitions` of the deletable logs).

In case of an exception, `cleanFilthiestLog` ...FIXME

Note	`cleanFilthiestLog` is used exclusively when `CleanerThread` is requested to doWork.

Cleaning Log — `cleanLog` Internal Method

```
cleanLog(cleanable: LogToClean): Unit
```

`cleanLog` requests the Cleaner to clean the given Log (as `LogToClean`) followed by recordStats.

In the end, `cleanLog` requests the LogCleanerManager to doneCleaning.

Note	`cleanLog` is used exclusively when `CleanerThread` is requested to cleanFilthiestLog.

`recordStats` Internal Method

```
recordStats(
  id: Int,
  name: String,
  from: Long,
  to: Long,
  stats: CleanerStats): Unit
```

`recordStats` ...FIXME

Note	`recordStats` is used exclusively when `CleanerThread` is requested to cleanLog.

Cleaner

`Cleaner` is created exclusively for a CleanerThread.

`Cleaner` requires that the number of slots in the given OffsetMap multiplied by the given dupBufferLoadFactor is greater than 1 or the following exception is thrown:

```
offset map is too small to fit in even a single message, so log cleaning will never ma
ke progress. You can increase log.cleaner.dedupe.buffer.size or decrease log.cleaner.t
hreads
```

`Cleaner` uses **Cleaner [id]:** prefix for logging (with the given ID).

Tip	`Cleaner` uses kafka.log.LogCleaner logger.

Creating Cleaner Instance

`Cleaner` takes the following to be created:

- ID
- `OffsetMap`
- `ioBufferSize`
- `maxIoBufferSize`
- dupBufferLoadFactor (of the CleanerConfig of the parent LogCleaner) (default: `0.9d`)
- `Throttler`
- `Time`
- "Check Done" function (`TopicPartition ⇒ Unit`)

`Cleaner` initializes the internal properties.

Cleaning Log — `clean` Method

```
clean(cleanable: LogToClean): (Long, CleanerStats)
```

`clean` ...FIXME

Note	`clean` is used exclusively when `CleanerThread` is requested to clean a log.

Cleaning Log — `doClean` Method

```
doClean(
  cleanable: LogToClean,
  deleteHorizonMs: Long): (Long, CleanerStats)
```

`doClean` ...FIXME

Note	`doClean` is used when `Cleaner` is requested to clean a log.

`cleanSegments` Internal Method

```
cleanSegments(
  log: Log,
  segments: Seq[LogSegment],
  map: OffsetMap,
  deleteHorizonMs: Long,
  stats: CleanerStats,
  transactionMetadata: CleanedTransactionMetadata): Unit
```

`cleanSegments` ...FIXME

Note	`cleanSegments` is used exclusively when `Cleaner` is requested to clean a log.

`cleanInto` Internal Method

```
cleanInto(
  topicPartition: TopicPartition,
  sourceRecords: FileRecords,
  dest: LogSegment,
  map: OffsetMap,
  retainDeletesAndTxnMarkers: Boolean,
  maxLogMessageSize: Int,
  transactionMetadata: CleanedTransactionMetadata,
  lastRecordsOfActiveProducers: Map[Long, LastRecord],
  stats: CleanerStats): Unit
```

`cleanInto` ...FIXME

Note	`cleanInto` is used exclusively when `Cleaner` is requested to cleanSegments.

`buildOffsetMap` Internal Method

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

`buildOffsetMap` ...FIXME

Note	`buildOffsetMap` is used exclusively when `Cleaner` is requested to doClean.

`buildOffsetMapForSegment` Internal Method

```
buildOffsetMapForSegment(
  topicPartition: TopicPartition,
  segment: LogSegment,
  map: OffsetMap,
  startOffset: Long,
  maxLogMessageSize: Int,
  transactionMetadata: CleanedTransactionMetadata,
  stats: CleanerStats): Boolean
```

`buildOffsetMapForSegment` ...FIXME

Note	`buildOffsetMapForSegment` is used exclusively when `Cleaner` is requested to buildOffsetMap.

Internal Properties

Name	Description
`decompressionBufferSupplier`	FIXME Used when...FIXME
`readBuffer`	FIXME Used when...FIXME
`writeBuffer`	FIXME Used when...FIXME

CleanedTransactionMetadata

`CleanedTransactionMetadata` is…FIXME

`onControlBatchRead` Method

```
onControlBatchRead(
  controlBatch: RecordBatch): Boolean
```

`onControlBatchRead` …FIXME

Note	`onControlBatchRead` is used when…FIXME

ProducerStateManager

`ProducerStateManager` is…FIXME

`removeExpiredProducers` Method

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

`removeExpiredProducers` …FIXME

Note	`removeExpiredProducers` is used when…FIXME

`takeSnapshot` Method

```
takeSnapshot(): Unit
```

`takeSnapshot` …FIXME

Note	`takeSnapshot` is used when `Log` is requested to recoverSegment, rebuildProducerState, close, and roll.

`isEmpty` Method

```
isEmpty: Boolean
```

`isEmpty` …FIXME

Note	`isEmpty` is used when…FIXME

LogConfig

`LogConfig` is an `AbstractConfig` for the topic-level configuration properties.

`LogConfig` uses the keys that are symonyms (aliases) of the keys of TopicConfig. As the TopicConfig itself says: "*Eventually this should replace LogConfig.scala.*"

Table 1. LogConfig's Topic-Level Configuration Properties

Name	Description
cleanup.policy	Cluster-wide property: log.cleanup.policy Default: `delete` Used when: • `GroupCoordinator` is requested for the offsetsTopicConfigs • `TransactionStateManager` is requested for the transactionTopicConfigs • `LogConfig` is requested for the compact and delete flags • `KafkaServer` utility is used to create a default log properties for LogManager and AdminManager
compression.type	
delete.retention.ms	
file.delete.delay.ms	The time (in millis) to wait before deleting a file from the filesystem. Must be at least `0` Property: `file.delete.delay.ms` Default: `60000`
flush.messages	Cluster-wide property: log.flush.interval.messages
flush.ms	Cluster-wide property: log.flush.interval.ms
index.interval.bytes	
max.compaction.lag.ms	

max.message.bytes	
message.downconversion.enable	
message.format.version	
message.timestamp.difference.max.ms	
message.timestamp.type	
min.cleanable.dirty.ratio	
min.compaction.lag.ms	
min.insync.replicas	Cluster-wide property: min.insync.replicas Used when: • `TransactionStateManager` is requested for transactionTopicConfigs • `KafkaServer` utility is used to create a default log properties for LogManager and AdminManager (with the value of KafkaConfig.minInSyncReplicas) • `Partition` is requested to isUnderMinIsr, isAtMinIsr, checkEnoughReplicasReachOffset, appendRecordsToLeader
preallocate	
retention.bytes	Used when: • `KafkaServer` utility is requested for default configuration properties of logs • `Log` is requested to deleteRetentionSizeBreachedSegments
retention.ms	Cluster-wide property: log.retention.ms Used exclusively when `Log` is requested to deleteRetentionMsBreachedSegments
	Cluster-wide property: log.segment.bytes Used when:

segment.bytes	• `GroupCoordinator` is requested for the offsetsTopicConfigs • `TransactionStateManager` is requested for the transactionTopicConfigs • `KafkaServer` utility is used to create a default log properties for LogManager and AdminManager The current value is available as `LogConfig.segmentSize` and used when: • `RollParams` utility is requested to create a RollParams • `Log` is requested for the initFileSize and to append records • `Cleaner` is requested to clean a log
segment.index.bytes	Cluster-wide property: log.index.size.max.bytes
segment.jitter.ms	
segment.ms	Cluster-wide property: log.roll.ms Default: `604800000` (7 days) Used when: • `Log` is requested to maybeRoll • `LogConfig` is requested for the randomSegmentJitter and maxSegmentMs • `KafkaServer` utility is used to create a default log properties for LogManager and AdminManager
unclean.leader.election.enable	Cluster-wide property: unclean.leader.election.enable Disabled (`false`) explicity for the `__transaction_state` internal topic (when `TransactionStateManager` is requested for the transactionTopicConfigs for KafkaApis) Used when: • `ZkPartitionStateMachine` is requested to collectUncleanLeaderElectionState

	• `DynamicLogConfig` is requested to reconfigure (and requests the `KafkaController` to enableDefaultUncleanLeaderElection)

TopicConfigSynonyms

`TopicConfigSynonyms` are the topic-level configuration properties (with the highest priority) with their corresponding cluster-wide (broker-level) configuration properties.

Note	`TopicConfigSynonyms` uses the same configuration properties as KafkaServer.copyKafkaConfigToLog utility.

Table 2. TopicConfigSynonyms

Topic-Level Property Name	Cluster-Wide Property Name
cleanup.policy	log.cleanup.policy
compression.type	compression.type
delete.retention.ms	log.cleaner.delete.retention.ms
file.delete.delay.ms	log.segment.delete.delay.ms
flush.messages	log.flush.interval.messages
flush.ms	log.flush.interval.ms
index.interval.bytes	log.index.interval.bytes
max.compaction.lag.ms	log.cleaner.max.compaction.lag.ms
max.message.bytes	message.max.bytes
message.downconversion.enable	log.message.downconversion.enable
message.format.version	log.message.format.version
message.timestamp.difference.max.ms	log.message.timestamp.difference.max.ms
message.timestamp.type	log.message.timestamp.type
min.cleanable.dirty.ratio	log.cleaner.min.cleanable.ratio
min.compaction.lag.ms	log.cleaner.min.compaction.lag.ms

min.insync.replicas	min.insync.replicas
preallocate	log.preallocate
retention.bytes	log.retention.bytes
retention.ms	log.retention.ms
segment.bytes	log.segment.bytes
segment.index.bytes	log.index.size.max.bytes
segment.jitter.ms	log.roll.jitter.ms
segment.ms	log.roll.ms
unclean.leader.election.enable	unclean.leader.election.enable

Note	`TopicConfigSynonyms` is used when: • `AdminManager` is requested to describe the configuration of a topic and create topics (and createTopicConfigEntry) • `DynamicLogConfig` utility is requested for the ReconfigurableConfigs and the KafkaConfigToLogConfigName

`randomSegmentJitter` Method

```
randomSegmentJitter: Long
```

`randomSegmentJitter` is one of the following values:

- `0` if segment.jitter.ms is `0`
- Random int value modulo the minimum of segment.jitter.ms and segment.ms

Note	`randomSegmentJitter` is used when `LogSegment` utility is used to open a log segment (for the rollJitterMs property).

`maxSegmentMs` Method

```
maxSegmentMs: Long
```

`maxSegmentMs` ...FIXME

Note	`maxSegmentMs` is used when `RollParams` utility is used to create a RollParams instance.

Checking Out Whether Compact Cleanup Policy Is Enabled — `compact` Flag

```
compact: Boolean
```

`compact` ...FIXME

Note	`compact` is used when...FIXME

Checking Out Whether Delete Cleanup Policy Is Enabled — `delete` Flag

```
delete: Boolean
```

`delete` flag is enabled (`true`) when delete cleanup policy is part of the cleanup.policy configuration property. Otherwise, `delete` flag is disabled (`false`).

Note	`delete` is used when: • `Partition` is requested to deleteRecordsOnLeader • `Log` is requested to deleteOldSegments • `LogCleanerManager` utility is requested for isCompactAndDelete

RollParams

`RollParams` is…FIXME

Creating RollParams Instance — `apply` Utility

```
apply(
  config: LogConfig,
  appendInfo: LogAppendInfo,
  messagesSize: Int,
  now: Long): RollParams
```

`apply` …FIXME

Note	`apply` is used when…FIXME

AbstractIndex

`AbstractIndex` is the abstraction of indices that use entrySize to index offsets and timestamps of a LogSegment.

Note	Kafka mmaps index files into memory, and all the read / write operations of the index is through OS page cache. This avoids blocked disk I/O in most cases.

Tip	Use DumpLogSegments tool to review the content of indexes of a log segment.

Table 1. AbstractIndex Contract (Abstract Methods Only)

Method	Description
`entrySize`	`entrySize: Int` Used when…FIXME

`parseEntry`	`parseEntry(`
	`  buffer: ByteBuffer,`
	`  n: Int): IndexEntry`
	
	Used when…FIXME

`sanityCheck`	`sanityCheck(): Unit` Used when…FIXME
`truncate`	`truncate(): Unit` Used when…FIXME

`truncateTo`	`truncateTo(`
	`  offset: Long): Unit`
	
	Used when…FIXME

Table 2. AbstractIndexes

AbstractIndex	Description
OffsetIndex	Index of the offsets
TimeIndex	Index of the timestamps and the offsets

Creating AbstractIndex Instance

`AbstractIndex` takes the following to be created:

- Underlying file
- Base offset
- `maxIndexSize` (default: `-1`)
- `writable` flag

`AbstractIndex` initializes the internal properties.

Note	`AbstractIndex` is a Scala abstract class and cannot be created directly. It is created indirectly for the concrete AbstractIndexes.

`reset` Method

```
reset(): Unit
```

`reset` truncate and then resize to the maxIndexSize.

Note	`reset` is used when…FIXME

`resize` Method

```
resize(newSize: Int): Boolean
```

`resize` …FIXME

Note	`resize` is used when…FIXME

`_warmEntries` Method

```
_warmEntries: Int
```

`_warmEntries` is the number of **warm entries** of the size in `8192` (bytes?)

Note	`_warmEntries` is used when `AbstractIndex` is requested to indexSlotRangeFor.

`largestLowerBoundSlotFor` Method

```
largestLowerBoundSlotFor(
  idx: ByteBuffer,
  target: Long,
  searchEntity: IndexSearchEntity): Int
```

`largestLowerBoundSlotFor` ...FIXME

Note	`largestLowerBoundSlotFor` is used when: • `OffsetIndex` is requested to lookup and truncateTo • `TimeIndex` is requested to lookup and truncateTo

`smallestUpperBoundSlotFor` Method

```
smallestUpperBoundSlotFor(
  idx: ByteBuffer,
  target: Long,
  searchEntity: IndexSearchEntity): Int
```

`smallestUpperBoundSlotFor` ...FIXME

Note	`smallestUpperBoundSlotFor` is used when `OffsetIndex` is requested to fetchUpperBoundOffset.

`indexSlotRangeFor` Internal Method

```
indexSlotRangeFor(
  idx: ByteBuffer,
  target: Long,
  searchEntity: IndexSearchEntity): (Int, Int)
```

`indexSlotRangeFor` ...FIXME

Note	`indexSlotRangeFor` is used when `AbstractIndex` is requested for the largestLowerBoundSlotFor and the smallestUpperBoundSlotFor.

`compareIndexEntry` Internal Method

```
compareIndexEntry(
  indexEntry: IndexEntry,
  target: Long,
  searchEntity: IndexSearchEntity): Int
```

`compareIndexEntry` ...FIXME

Note	`compareIndexEntry` is used when...FIXME

Internal Properties

Name	Description
`_length`	Length of the index Used when...FIXME

OffsetIndex — Index Of Offsets Of Log Segment

`OffsetIndex` is an index of the offsets of a LogSegment.

`OffsetIndex` uses entrySize of 8 bytes (*8-byte offset index entries*).

`OffsetIndex` is created when:

- Lazily for a log segment (via forOffset utility that defers loading the underlying index file until needed)
- DumpLogSegments tool is used (to dump the offset and time indices of a log segment)

Tip	Enable `ALL` logging level for `kafka.log.OffsetIndex` logger to see what happens inside. Add the following line to `config/log4j.properties`: `log4j.logger.kafka.log.OffsetIndex=ALL` Refer to Logging.

Creating OffsetIndex Instance

`OffsetIndex` takes the following to be created:

- Underlying File
- Base offset
- `maxIndexSize` (default: `-1`)
- `writable` flag (default: `true`)

`OffsetIndex` initializes the internal properties.

While being created, `OffsetIndex` prints out the following DEBUG message to the logs:

```
Loaded index file [absolutePath] with maxEntries = [maxEntries],
maxIndexSize = [maxIndexSize], entries = [_entries], lastOffset
= [_lastOffset], file position = [position]
```

`append` Method

```
append(
  offset: Long,
  position: Int): Unit
```

`append` ...FIXME

Note	`append` is used when `LogSegment` is requested to append and recover.

`lookup` Method

```
lookup(
  targetOffset: Long): OffsetPosition
```

`lookup` ...FIXME

Note	`lookup` is used when...FIXME

`fetchUpperBoundOffset` Method

```
fetchUpperBoundOffset(
  fetchOffset: OffsetPosition,
  fetchSize: Int): Option[OffsetPosition]
```

`fetchUpperBoundOffset` ...FIXME

Note	`fetchUpperBoundOffset` is used when `Log` is requested to addAbortedTransactions.

`truncateTo` Method

```
truncateTo(
  offset: Long): Unit
```

Note	`truncateTo` is part of the AbstractIndex Contract to...FIXME.

`truncateTo` ...FIXME

`lastEntry` Internal Method

```
lastEntry: OffsetPosition
```

`lastEntry` ...FIXME

Note	`lastEntry` is used when...FIXME

Internal Properties

Name	Description
`_lastOffset`	The `offset` of the last OffsetPosition Used when...FIXME

TimeIndex — Index Of Timestamp And Offsets Of Log Segment

`TimeIndex` is an index of the timestamps and the offsets of a LogSegment.

`TimeIndex` uses entrySize of 12 bytes (*12-byte time index entries*).

`TimeIndex` is created when:

- Lazily for a log segment (via forTime utility that defers loading the underlying index file until needed)
- DumpLogSegments tool is used (to dump a time index (of a log segment))

Tip	Enable `ALL` logging level for `kafka.log.TimeIndex` logger to see what happens inside. Add the following line to `config/log4j.properties`: `log4j.logger.kafka.log.TimeIndex=ALL` Refer to Logging.

Creating TimeIndex Instance

`TimeIndex` takes the following to be created:

- Underlying File
- Base offset
- `maxIndexSize` (default: `-1`)
- `writable` flag (default: `true`)

`TimeIndex` initializes the internal properties.

While being created, `TimeIndex` prints out the following DEBUG message to the logs:

```
Loaded index file [absolutePath] with maxEntries = [maxEntries],
maxIndexSize = [maxIndexSize], entries = [_entries], lastOffset
= [_lastOffset], file position = [position]
```

`lookup` Method

```
lookup(
  targetTimestamp: Long): TimestampOffset
```

`lookup` ...FIXME

Note	`lookup` is used when...FIXME

`truncateTo` Method

```
truncateTo(
  offset: Long): Unit
```

Note	`truncateTo` is part of the AbstractIndex Contract to...FIXME.

`truncateTo` ...FIXME

`lastEntryFromIndexFile` Internal Method

```
lastEntryFromIndexFile: TimestampOffset
```

`lastEntryFromIndexFile` ...FIXME

Note	`lastEntryFromIndexFile` is used when...FIXME

Internal Properties

Name	Description
`_lastEntry`	lastEntryFromIndexFile Used when...FIXME

TransactionIndex

`TransactionIndex` is a **transaction index** of a LogSegment.

`TransactionIndex` is created when:

- `LogSegment` utility is used to open a log segment
- DumpLogSegments tool is used (to dump a transaction index (of a log segment))

Creating TransactionIndex Instance

`TransactionIndex` takes the following to be created:

- Start offset
- Underlying file (Java's File)

`TransactionIndex` initializes the internal properties.

While being created, `TransactionIndex` opens a channel to the file if exists already.

`append` Method

```
append(
  abortedTxn: AbortedTxn): Unit
```

`append` ...FIXME

Note	`append` is used when: • `CleanedTransactionMetadata` is requested to onControlBatchRead • `LogSegment` is requested to updateTxnIndex

`flush` Method

```
flush(): Unit
```

`flush` ...FIXME

Note	`flush` is used when `LogSegment` is requested to flush.

`truncateTo` Method

```
truncateTo(
  offset: Long): Unit
```

`truncateTo` ...FIXME

Note	`truncateTo` is used when `LogSegment` is requested to truncate (to a given offset).

`deleteIfExists` Method

```
deleteIfExists(): Boolean
```

`deleteIfExists` ...FIXME

Note	`deleteIfExists` is used when `LogSegment` is requested to deleteIfExists.

`collectAbortedTxns` Method

```
collectAbortedTxns(
  fetchOffset: Long,
  upperBoundOffset: Long): TxnIndexSearchResult
```

`collectAbortedTxns` ...FIXME

Note	`collectAbortedTxns` is used when `LogSegment` is requested to collectAbortedTxns.

Opening FileChannel — `openChannel` Internal Method

```
openChannel(): FileChannel
```

`openChannel` uses Java's FileChannel utility to open a file channel to the underlying file for read and write access (and creating a new file unless it exists already).

`openChannel` registers the `FileChannel` with the maybeChannel internal registry.

`openChannel` sets the file position of the `FileChannel` to be the very end of the file.

Note	`openChannel` is used when `TransactionIndex` is created (and the underlying file is available) and requested for the FileChannel.

`channel` Internal Method

```
channel(): FileChannel
```

`channel` takes the `FileChannel` out from the maybeChannel internal registry (if available) or opens a new FileChannel.

Note	`channel` is used when `TransactionIndex` is requested to append and truncateTo.

Internal Properties

Name	Description
`lastOffset`	`lastOffset: Option[Long] = None` Used when…FIXME
`maybeChannel`	`maybeChannel: Option[FileChannel] = None` FileChannel Used when…FIXME

LazyIndex — Deferred Loading Of Index Files

`LazyIndex` is a thin wrapper over an AbstractIndex instance (of type `T`) that allows to defer loading (i.e. memory mapping) the underlying index file until it is accessed for the first time (via the get method).

`LazyIndex` is used as an optimization of broker start-up time with a large number of segments.

`LazyIndex` defines utilities to create lazy indexes for OffsetIndex and TimeIndex.

Creating TimeIndex Instance

`TimeIndex` takes the following to be created:

- `IndexWrapper`
- Load index function that creates an AbstractIndex for a given underlying index file (`File ⇒ T`)

`get` Method

```
get: T
```

`get` …FIXME

Note	`get` is used when…FIXME

`forOffset` Utility

```
forOffset(
  file: File,
  baseOffset: Long,
  maxIndexSize: Int = -1,
  writable: Boolean = true): LazyIndex[OffsetIndex]
```

`forOffset` simply creates a new `LazyIndex` with an `IndexFile` (for the given `file`) and a load index function that creates a new OffsetIndex.

Note	`forOffset` is used when `LogSegment` utility is used to open a log segment.

`forTime` Utility

```
forTime(
  file: File,
  baseOffset: Long,
  maxIndexSize: Int = -1,
  writable: Boolean = true): LazyIndex[TimeIndex]
```

`forTime` simply creates a new `LazyIndex` with an `IndexFile` (for the given `file`) and a load index function that creates a new TimeIndex.

Note	`forTime` is used when `LogSegment` utility is used to open a log segment.

MetadataCache

`MetadataCache` is…FIXME

`MetadataCache` is created exclusively for KafkaServer (when started).

`MetadataCache` takes a single broker ID when created.

`MetadataCache` maintains the metadataSnapshot and is used to…FIXME

`MetadataCache` tracks the `controllerId` …FIXME

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

Name	Description
`metadataSnapshot`	`MetadataSnapshot` `metadataSnapshot` is updated when `MetadataCache` is requested to updateMetadata
`partitionMetadataLock`	Java's ReentrantReadWriteLock
`stateChangeLogger`	StateChangeLogger

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

Tip	Enable `DEBUG` or `ERROR` logging level for `kafka.server.MetadataCache` logger to see what happens inside. Add the following line to `config/log4j.properties` : `log4j.logger.kafka.server.MetadataCache=DEBUG` Refer to Logging.

`getAllTopics` Method

```
getAllTopics(): Set[String]
```

`getAllTopics` …FIXME

Note	`getAllTopics` is used when…FIXME

`getTopicMetadata` Method

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

`getTopicMetadata` ...FIXME

Note	`getTopicMetadata` is used when...FIXME

`getAliveBrokers` Method

```
getAliveBrokers: Seq[Broker]
```

`getAliveBrokers` ...FIXME

Note	`getAliveBrokers` is used when...FIXME

Updating Metadata (and Returning Deleted Partitions) — `updateMetadata` Method

```
updateMetadata(
  correlationId: Int,
  updateMetadataRequest: UpdateMetadataRequest): Seq[TopicPartition]
```

`updateMetadata` ...FIXME

Note	`updateMetadata` is used exclusively when `ReplicaManager` is requested to maybeUpdateMetadataCache.

`getClusterMetadata` Method

```
getClusterMetadata(clusterId: String, listenerName: ListenerName): Cluster
```

`getClusterMetadata` ...FIXME

Note	`getClusterMetadata` is used exclusively when `KafkaApis` is requested to handle an UpdateMetadata request.

`getPartitionReplicaEndpoints` Method

```
getPartitionReplicaEndpoints(
  tp: TopicPartition,
  listenerName: ListenerName): Map[Int, Node]
```

`getPartitionReplicaEndpoints` ...FIXME

Note	`getPartitionReplicaEndpoints` is used when `ReplicaManager` is requested to findPreferredReadReplica.

OffsetConfig

`OffsetConfig` is…FIXME

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

Property	Default Value	Description
`offsetsTopicNumPartitions`	`50`	Number of partitions for the `__consumer_offsets` consumer group metadata topic

ReplicaManager

`ReplicaManager` manages log replicas using the LogManager.

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

Figure 1. ReplicaManager and KafkaServer

`ReplicaManager` is given all the required services (e.g. Metrics, KafkaZkClient, Scheduler, LogManager, QuotaManagers, MetadataCache) from the `KafkaServer`.

Note	`KafkaServer` creates a `ReplicaManager` for the only purpose of creating KafkaApis, GroupCoordinator, and TransactionCoordinator.

When started, `ReplicaManager` schedules isr-expiration and isr-change-propagation recurring tasks (every half of replica.lag.time.max.ms property and 2500 ms, respectively).

`ReplicaManager` manages a registry of Partitions by TopicPartition.

`ReplicaManager` can be the leader or a follower of partitions which happens when `KafkaApis` is requested to handle a LeaderAndIsr request (mostly, if not always, from the KafkaController).

`ReplicaManager` uses the MetadataCache for the following:

- …FIXME

Tip

Enable `ALL` logging level for `kafka.server.ReplicaManager` logger to see what happens inside.

Add the following line to `log4j.properties`:

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

Refer to Logging.

Please note that `ReplicaManager` also uses a preconfigured state.change.logger logger in `config/log4j.properties`:

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

log4j.logger.state.change.logger=TRACE, stateChangeAppender
log4j.additivity.state.change.logger=false
```

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

ReplicaManager and ReplicaFetcherManager

`ReplicaManager` creates a ReplicaFetcherManager while being created (via createReplicaFetcherManager).

Figure 2. ReplicaManager and ReplicaFetcherManager

`ReplicaManager` uses the `ReplicaFetcherManager` for the following:

- addFetcherForPartitions when makeFollowers
- removeFetcherForPartitions when stopReplicas, makeLeaders, makeFollowers and handleLogDirFailure
- shutdownIdleFetcherThreads when becomeLeaderOrFollower
- shutdown when shutdown

`ReplicaFetcherManager` is also used when:

- `DynamicThreadPool` is requested to reconfigure (resize) the thread pool
- `KafkaApis` is requested to handle a StopReplica request

Performance Metrics

`ReplicaManager` is a KafkaMetricsGroup with the following performance metrics.

Table 1. ReplicaManager's Performance Metrics

Metric Name	Description
`FailedIsrUpdatesPerSec`	
`IsrExpandsPerSec`	
`IsrShrinksPerSec`	
`LeaderCount`	
`OfflineReplicaCount`	
`PartitionCount`	Number of allPartitions
`UnderMinIsrPartitionCount`	
`UnderReplicatedPartitions`	underReplicatedPartitionCount

The performance metrics are registered in **kafka.server:type=ReplicaManager** group.

Figure 3. ReplicaManager in jconsole

Registry of Partitions by TopicPartition — `allPartitions` Internal Property

```
allPartitions: Pool[TopicPartition, Partition]
```

`allPartitions` is a registry of Partitions by `TopicPartition`

The number of partitions in `allPartitions` is available as the PartitionCount performance metric.

A new `TopicPartition` is added exclusively when looking up a Partition by a TopicPartition that is not registered yet.

A `TopicPartition` is removed when stopping a replica (and marked as the OfflinePartition).

Note	`allPartitions` is used when getPartition, nonOfflinePartitionsIterator, offlinePartitionsIterator, becomeLeaderOrFollower, maybeShrinkIsr, handleLogDirFailure

Creating ReplicaFetcherManager — `createReplicaFetcherManager` Internal Method

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

`createReplicaFetcherManager` simply creates a ReplicaFetcherManager.

Note	`createReplicaFetcherManager` is used when `ReplicaManager` is created.

Creating ReplicaAlterLogDirsManager — `createReplicaAlterLogDirsManager` Internal Method

```
createReplicaAlterLogDirsManager(
  quotaManager: ReplicationQuotaManager,
  brokerTopicStats: BrokerTopicStats): ReplicaAlterLogDirsManager
```

`createReplicaAlterLogDirsManager` simply creates a ReplicaAlterLogDirsManager.

Note	`createReplicaAlterLogDirsManager` is used when `ReplicaManager` is created.

`shutdown` Method

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

`shutdown` ...FIXME

Note	`shutdown` is used when `KafkaServer` is requested to shut down.

`alterReplicaLogDirs` Method

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

`alterReplicaLogDirs` ...FIXME

Note	`alterReplicaLogDirs` is used exclusively when `KafkaApis` is requested to handle an AlterReplicaLogDirs request.

Becoming Leader or Follower of Partitions — `becomeLeaderOrFollower` Method

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

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

```
Received LeaderAndIsr request [stateInfo] correlation id [correlationId] from controll
er [controllerId] epoch [controllerEpoch] for partition [topicPartition]
```

`becomeLeaderOrFollower` records the current controller epoch (of the LeaderAndIsrRequest) in the controllerEpoch internal registry.

For all *valid partition states*, `becomeLeaderOrFollower` finds the partition states with the leader being the (local) broker and that are the partitions for which the broker becomes the leader. All other partition states are for partitions for which the broker becomes a follower.

For all partitions for which the broker becomes the leader, `becomeLeaderOrFollower` makeLeaders.

For all partitions for which the broker becomes a follower, `becomeLeaderOrFollower` makeFollowers.

With the hwThreadInitialized internal flag disabled (`false`), `becomeLeaderOrFollower` startHighWaterMarksCheckPointThread and turns the flag on (`true`).

For every new partitions, `becomeLeaderOrFollower` ...FIXME

`becomeLeaderOrFollower` ...FIXME

`becomeLeaderOrFollower` calls the given `onLeadershipChange` callback with the partitions for the broker to be the leader and a follower.

In the end, `becomeLeaderOrFollower` creates a new `LeaderAndIsrResponse` to "announce" a successful request processing.

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 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, LogReadResu
lt)]
```

`updateFollowerLogReadResults` ...FIXME

Note	`updateFollowerLogReadResults` is used exclusively when `ReplicaManager` is requested to fetch messages from the leader replica.

`fetchMessages` Method

```
fetchMessages(
  timeout: Long,
  replicaId: Int,
  fetchMinBytes: Int,
  fetchMaxBytes: Int,
  hardMaxBytesLimit: Boolean,
  fetchInfos: Seq[(TopicPartition, PartitionData)],
  quota: ReplicaQuota,
  responseCallback: Seq[(TopicPartition, FetchPartitionData)] => Unit,
  isolationLevel: IsolationLevel,
  clientMetadata: Option[ClientMetadata]): Unit
```

`fetchMessages` determines `fetchIsolation`:

- `FetchLogEnd` for a request from a follower or when the `replicaId` is `Request.FutureLocalReplicaId`
- `FetchTxnCommitted` for `isolationLevel` being `READ_COMMITTED`
- `FetchHighWatermark` otherwise

`fetchMessages` readFromLocalLog (with the `fetchIsolation`).

`fetchMessages` ...FIXME

Note	`fetchMessages` is used when: • `KafkaApis` is requested to handle a Fetch request • `ReplicaAlterLogDirsThread` is requested to fetchFromLeader

`maybePropagateIsrChanges` Method

```
maybePropagateIsrChanges(): Unit
```

`maybePropagateIsrChanges` ...FIXME

Note	`maybePropagateIsrChanges` is used exclusively when isr-change-propagation task is executed (every 2500 milliseconds).

Creating ReplicaManager Instance

`ReplicaManager` takes the following when created:

- KafkaConfig
- Metrics
- `Time`
- KafkaZkClient
- Scheduler
- LogManager
- `isShuttingDown` flag
- QuotaManagers
- BrokerTopicStats
- MetadataCache
- `LogDirFailureChannel`
- `DelayedOperationPurgatory[DelayedProduce]`
- `DelayedOperationPurgatory[DelayedFetch]`
- `DelayedOperationPurgatory[DelayedDeleteRecords]`
- Optional thread name prefix

`ReplicaManager` initializes the internal registries and counters.

Starting ReplicaManager (and Scheduling ISR-Related Tasks) — `startup` Method

```
startup(): Unit
```

`startup` requests Scheduler to schedule the ISR-related tasks:

1. isr-expiration
2. isr-change-propagation

`startup` then creates a LogDirFailureHandler and requests it to start.

Note	`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:

```
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 replica.lag.time.max.ms property).

Note	`maybeShrinkIsr` is used exclusively to schedule isr-expiration recurring task when `ReplicaManager` starts up.

`makeLeaders` Internal Method

```
makeLeaders(
  controllerId: Int,
  epoch: Int,
  partitionState: Map[Partition, LeaderAndIsrRequest.PartitionState],
  correlationId: Int,
  responseMap: mutable.Map[TopicPartition, Errors]): Set[Partition]
```

`makeLeaders` ...FIXME

Note	`makeLeaders` is used exclusively when `ReplicaManager` is requested to becomeLeaderOrFollower.

`describeLogDirs` Method

```
describeLogDirs(partitions: Set[TopicPartition]): Map[String, LogDirInfo]
```

`describeLogDirs` ...FIXME

Note	`describeLogDirs` is used exclusively when `KafkaApis` is requested to handle a DescribeLogDirs request.

Finding Log For TopicPartition — `getLog` Method

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

`getLog` ...FIXME

Note	`getLog` is used when: • `GroupMetadataManager` is requested to doLoadGroupsAndOffsets • `TransactionStateManager` is requested to loadTransactionMetadata

`startHighWaterMarksCheckPointThread` Method

```
startHighWaterMarksCheckPointThread(): Unit
```

`startHighWaterMarksCheckPointThread` ...FIXME

Note	`startHighWaterMarksCheckPointThread` is used when...FIXME

`checkpointHighWatermarks` Method

```
checkpointHighWatermarks(): Unit
```

`checkpointHighWatermarks` ...FIXME

Note	`checkpointHighWatermarks` is used when...FIXME

`shutdownIdleReplicaAlterLogDirsThread` Method

```
shutdownIdleReplicaAlterLogDirsThread(): Unit
```

`shutdownIdleReplicaAlterLogDirsThread` ...FIXME

Note	`shutdownIdleReplicaAlterLogDirsThread` is used when...FIXME

`handleLogDirFailure` Method

```
handleLogDirFailure(
  dir: String,
  sendZkNotification: Boolean = true): Unit
```

`handleLogDirFailure` ...FIXME

Note	`handleLogDirFailure` is used when `LogDirFailureHandler` is requested to do the work.

`maybeUpdateMetadataCache` Method

```
maybeUpdateMetadataCache(
  correlationId: Int,
  updateMetadataRequest: UpdateMetadataRequest) : Seq[TopicPartition]
```

`maybeUpdateMetadataCache` ...FIXME

Note	`maybeUpdateMetadataCache` is used exclusively when `KafkaApis` is requested to handle an UpdateMetadata request.

Appending Records — `appendRecords` Method

```
appendRecords(
  timeout: Long,
  requiredAcks: Short,
  internalTopicsAllowed: Boolean,
  isFromClient: Boolean,
  entriesPerPartition: Map[TopicPartition, MemoryRecords],
  responseCallback: Map[TopicPartition, PartitionResponse] => Unit,
  delayedProduceLock: Option[Lock] = None,
  recordConversionStatsCallback: Map[TopicPartition, RecordConversionStats] => Unit =
_ => ()): Unit
```

`appendRecords` ...FIXME

Note	`appendRecords` is used when: • `GroupMetadataManager` is requested to request the ReplicaManager to append records • `TransactionStateManager` is requested to enableTransactionalIdExpiration and appendTransactionToLog • `KafkaApis` is requested to handle Produce and WriteTxnMarkers requests

Validating requiredAcks — `isValidRequiredAcks` Internal Method

```
isValidRequiredAcks(requiredAcks: Short): Boolean
```

`isValidRequiredAcks` is positive (`true`) when the given `requiredAcks` is one of the following:

- `-1`
- `1`
- `0`

Otherwise, `isValidRequiredAcks` is negative (`false`).

Note	`isValidRequiredAcks` is used exclusively when `ReplicaManager` is requested to appendRecords.

`appendToLocalLog` Internal Method

```
appendToLocalLog(
  internalTopicsAllowed: Boolean,
  isFromClient: Boolean,
  entriesPerPartition: Map[TopicPartition, MemoryRecords],
  requiredAcks: Short): Map[TopicPartition, LogAppendResult]
```

`appendToLocalLog` processes (*maps over*) the given `Map[TopicPartition, MemoryRecords]` (`entriesPerPartition`), so that the leader partition (of every `TopicPartition`) is requested to appendRecordsToLeader.

Internally, `appendToLocalLog` prints out the following TRACE message to the logs:

```
Append [[entriesPerPartition]] to local log
```

For every tuple in the given `entriesPerPartition` (`Map[TopicPartition, MemoryRecords]`), `appendToLocalLog` does the following steps:

1. Requests the BrokerTopicStats to mark the occurrence of an event for the totalProduceRequestRate for the topic (of the `TopicPartition`) in the topicStats and for all topics
2. Gets the partition (or throws an exception) (with `expectLeader` flag enabled)
3. Requests the `Partition` to appendRecordsToLeader (with the `MemoryRecords` , the `isFromClient` flag, and the `requiredAcks` bit map)
4. Requests the BrokerTopicStats to mark the `sizeInBytes` of the `MemoryRecords` for the bytesInRate for the topic (of the `TopicPartition`) in the bytesInRate and for all topics
5. Requests the BrokerTopicStats to mark the number of messages appended for the messagesInRate for the topic (of the `TopicPartition`) in the bytesInRate and for all topics
6. Prints out the following TRACE message to the logs:

   ```
   [sizeInBytes] written to log [topicPartition] beginning at offset [firstOffset] an
   d ending at offset [lastOffset]
   ```

In case `Topic.isInternal(topicPartition.topic) && !internalTopicsAllowed` , `appendToLocalLog` ...FIXME

In case of exceptions, `appendToLocalLog` ...FIXME

Note	`appendToLocalLog` is used exclusively when `ReplicaManager` is requested to append records.

Getting Partition Or Throwing Exception — `getPartitionOrException` Method

```
getPartitionOrException(
  topicPartition: TopicPartition,
  expectLeader: Boolean): Partition
```

`getPartitionOrException` gets the partition if available or throws one of the following exceptions:

- `KafkaStorageException` when the partition is offline

```
Partition [topicPartition] is in an offline log directory
```

- `NotLeaderForPartitionException`

```
Broker [localBrokerId] is not a replica of [topicPartition]
```

- `ReplicaNotAvailableException`

```
Partition [topicPartition] is not available
```

- `UnknownTopicOrPartitionException`

```
Partition [topicPartition] doesn't exist
```

Note	`getPartitionOrException` is used when…FIXME

Getting Partition by TopicPartition (If Available) — `getPartition` Method

```
getPartition(topicPartition: TopicPartition): Option[Partition]
```

`getPartition` gets the partition for the given `TopicPartition`.

Note	`getPartition` is used when: • `DelayedDeleteRecords` is requested to `tryComplete` • `DelayedProduce` is requested to `tryComplete` • `ReplicaAlterLogDirsThread` is requested to processPartitionData • `ReplicaFetcherThread` is requested to processPartitionData, truncate, and truncateFullyAndStartAt • `ReplicaManager` is requested to nonOfflinePartition, getPartitionOrException, alterReplicaLogDirs, appendToLocalLog, becomeLeaderOrFollower, and lastOffsetForLeaderEpoch

Stopping Partition Replica — `stopReplica` Method

```
stopReplica(
  topicPartition: TopicPartition,
  deletePartition: Boolean): Unit
```

`stopReplica` ...FIXME

Note	`stopReplica` is used exclusively when `ReplicaManager` is requested to stopReplicas.

`underReplicatedPartitionCount` Method

```
underReplicatedPartitionCount: Int
```

`underReplicatedPartitionCount` ...FIXME

Note	`underReplicatedPartitionCount` is used exclusively for the UnderReplicatedPartitions performance metric.

`leaderPartitionsIterator` Internal Method

```
leaderPartitionsIterator: Iterator[Partition]
```

`leaderPartitionsIterator` ...FIXME

Note	`leaderPartitionsIterator` is used exclusively for the performance metrics: LeaderCount, UnderMinIsrPartitionCount, and UnderReplicatedPartitions (indirectly using underReplicatedPartitionCount).

`nonOfflinePartitionsIterator` Internal Method

```
nonOfflinePartitionsIterator: Iterator[Partition]
```

`nonOfflinePartitionsIterator` ...FIXME

Note	`nonOfflinePartitionsIterator` is used when `ReplicaManager` is requested to leaderPartitionsIterator, checkpointHighWatermarks, and handleLogDirFailure.

Looking Up Partition or Creating New One (by TopicPartition) — `getOrCreatePartition` Method

```
getOrCreatePartition(topicPartition: TopicPartition): Partition
```

`getOrCreatePartition` simply looks up a Partition by the `TopicPartition` (in the allPartitions internal registry). If not found, `getOrCreatePartition` adds a new `Partition`.

Note	`getOrCreatePartition` is used exclusively when `ReplicaManager` is requested to becomeLeaderOrFollower.

`offlinePartitionsIterator` Internal Method

```
offlinePartitionsIterator: Iterator[Partition]
```

`offlinePartitionsIterator` ...FIXME

Note	`offlinePartitionsIterator` is used when...FIXME

`markPartitionOffline` Method

```
markPartitionOffline(tp: TopicPartition): Unit
```

`markPartitionOffline` ...FIXME

Note	`markPartitionOffline` is used when...FIXME

`lastOffsetForLeaderEpoch` Method

```
lastOffsetForLeaderEpoch(
  requestedEpochInfo: Map[TopicPartition, OffsetsForLeaderEpochRequest.PartitionData]
): Map[TopicPartition, EpochEndOffset]
```

`lastOffsetForLeaderEpoch` ...FIXME

Note	`lastOffsetForLeaderEpoch` is used when...FIXME

`nonOfflinePartition` Method

```
nonOfflinePartition(topicPartition: TopicPartition): Option[Partition]
```

`nonOfflinePartition` ...FIXME

Note	`nonOfflinePartition` is used when…FIXME

`deleteRecords` Method

```
deleteRecords(
  timeout: Long,
  offsetPerPartition: Map[TopicPartition, Long],
  responseCallback: Map[TopicPartition, DeleteRecordsResponse.PartitionResponse] => Un
it): Unit
```

`deleteRecords` …FIXME

Note	`deleteRecords` is used when…FIXME

`fetchOffsetForTimestamp` Method

```
fetchOffsetForTimestamp(
  topicPartition: TopicPartition,
  timestamp: Long,
  isolationLevel: Option[IsolationLevel],
  currentLeaderEpoch: Optional[Integer],
  fetchOnlyFromLeader: Boolean): TimestampOffset
```

`fetchOffsetForTimestamp` …FIXME

Note	`fetchOffsetForTimestamp` is used when…FIXME

`stopReplicas` Method

```
stopReplicas(
  stopReplicaRequest: StopReplicaRequest): (mutable.Map[TopicPartition, Errors], Errors
)
```

`stopReplicas` …FIXME

Note	`stopReplicas` is used exclusively when `KafkaApis` is requested to handle a StopReplica request.

`getMagic` Method

```
getMagic(topicPartition: TopicPartition): Option[Byte]
```

`getMagic` ...FIXME

Note	`getMagic` is used when...FIXME

`localReplicaOrException` Method

```
localReplicaOrException(topicPartition: TopicPartition): Replica
```

`localReplicaOrException` finds the partition (or throws an exception) for the given `TopicPartition` (and `expectLeader` flag off) and requests the `Partition` to get the local partition replica (or throw an exception).

Note	A partition replica is local when the replica ID is exactly the local broker ID.

Note	`localReplicaOrException` is used when...FIXME

`shouldLeaderThrottle` Method

```
shouldLeaderThrottle(
  quota: ReplicaQuota,
  topicPartition: TopicPartition,
  replicaId: Int): Boolean
```

`shouldLeaderThrottle` ...FIXME

Note	`shouldLeaderThrottle` is used when...FIXME

`readFromLocalLog` Method

```
readFromLocalLog(
  replicaId: Int,
  fetchOnlyFromLeader: Boolean,
  fetchIsolation: FetchIsolation,
  fetchMaxBytes: Int,
  hardMaxBytesLimit: Boolean,
  readPartitionInfo: Seq[(TopicPartition, PartitionData)],
  quota: ReplicaQuota,
  clientMetadata: Option[ClientMetadata]): Seq[(TopicPartition, LogReadResult)]
```

`readFromLocalLog` returns records (as `FetchDataInfo`) for every `TopicPartition` in the given `readPartitionInfo`.

Internally, `readFromLocalLog` reads the requested `PartitionData` (up to `fetchMaxBytes` that is decremented sequentially for every partition) for every `TopicPartition` (in the given `readPartitionInfo` collection).

Note	`readFromLocalLog` is used when: • `DelayedFetch` is requested to `onComplete` • `ReplicaManager` is requested to fetchMessages (when `KafkaApis` is requested to handle a Fetch request)

`read` Internal Method

```
read(
  tp: TopicPartition,
  fetchInfo: PartitionData,
  limitBytes: Int,
  minOneMessage: Boolean): LogReadResult
```

`read` increments the totalFetchRequestRate metric of the topic (of the `TopicPartition`) and the allTopicsStats in the BrokerTopicStats.

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

```
Fetching log segment for partition [tp], offset [offset], partition fetch size [partit
ionFetchSize], remaining response limit [limitBytes]
```

`read` finds the Partition for the `TopicPartition`.

`read` ...FIXME

Triggering Preferred Replica Leader Election — `electPreferredLeaders` Method

```
electPreferredLeaders(
  controller: KafkaController,
  partitions: Set[TopicPartition],
  responseCallback: Map[TopicPartition, ApiError] => Unit,
  requestTimeout: Long): Unit
```

`electPreferredLeaders` simply requests the `KafkaController` to electPreferredLeaders for the partitions.

Note	`electPreferredLeaders` is used exclusively when `KafkaApis` is requested to handle an ElectPreferredLeaders request.

`tryCompleteElection` Method

```
tryCompleteElection(key: DelayedOperationKey): Unit
```

`tryCompleteElection` ...FIXME

Note	`tryCompleteElection` is used exclusively when `KafkaApis` is requested to handle an UpdateMetadata request.

`localLog` Method

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

`localLog` ...FIXME

Note	`localLog` is used when...FIXME

`findPreferredReadReplica` Method

```
findPreferredReadReplica(
  tp: TopicPartition,
  clientMetadata: ClientMetadata,
  replicaId: Int,
  fetchOffset: Long,
  currentTimeMs: Long): Option[Int]
```

`findPreferredReadReplica` finds the Partition for the `TopicPartition` (with `expectLeader` flag off).

`findPreferredReadReplica` ...FIXME

Note	`findPreferredReadReplica` is used when `ReplicaManager` is requested to readFromLocalLog.

`localLogOrException` Method

```
localLogOrException(
  topicPartition: TopicPartition): Log
```

`localLogOrException` finds the Partition for the given `TopicPartition` (and `expectLeader` flag off) and then requests it for the local Log.

`localLogOrException` throws an exception if neither the `Partition` nor the `Log` of the given `TopicPartition` is available locally.

Note	`localLogOrException` is used when `ReplicaFetcherThread` is requested for the latest epoch, logEndOffset, endOffsetForEpoch, and buildFetch.

`updateFollowerFetchState` Internal Method

```
updateFollowerFetchState(
  followerId: Int,
  readResults: Seq[(TopicPartition, LogReadResult)]): Seq[(TopicPartition, LogReadResu
lt)]
```

`updateFollowerFetchState` ...FIXME

Note	`updateFollowerFetchState` is used when `ReplicaManager` is requested to fetchMessages.

Internal Properties

Name	Description
`controllerEpoch`	
`replicaAlterLogDirsManager`	ReplicaAlterLogDirsManager Created immediately with `ReplicaManager` Used when: • `ReplicaFetcherThread` is requested to truncate • `ReplicaManager` is requested to shutdownIdleReplicaAlterLogDirsThread, stopReplicas, alterReplicaLogDirs, becomeLeaderOrFollower, handleLogDirFailure, shutdown
`highWatermarkCheckpoints`	`OffsetCheckpointFiles` per live log data directory
`stateChangeLogger`	StateChangeLogger with the broker ID and `inControllerContext` flag off (that sets `Broker` log prefix)
`hwThreadInitialized`	
`isrChangeSet`	Collection of `TopicPartition` that…FIXME
`lastIsrChangeMs`	Time when isrChangeSet has a new `TopicPartition` added.
`logDirFailureHandler`	LogDirFailureHandler
`OfflinePartition`	

ReplicaFetcherManager

`ReplicaFetcherManager` is a manager of ReplicaFetcherThreads for ReplicaManager.

Figure 1. ReplicaFetcherManager

`ReplicaFetcherManager` uses num.replica.fetchers configuration property for the number of replica fetcher threads.

Tip	Enable `ALL` logging level for `kafka.server.ReplicaFetcherManager` logger to see what happens inside. Add the following line to `config/log4j.properties` : `log4j.logger.kafka.server.ReplicaFetcherManager=ALL` Refer to Logging.

Performance Metrics

As an AbstractFetcherManager, `ReplicaFetcherManager` is a KafkaMetricsGroup.

The performance metrics are registered in **kafka.server:type=ReplicaFetcherManager** group.

Figure 2. ReplicaFetcherManager in jconsole

Creating ReplicaFetcherManager Instance

`ReplicaFetcherManager` takes the following to be created:

- KafkaConfig
- ReplicaManager
- Metrics
- `Time`
- Thread name prefix (undefined by default)
- ReplicationQuotaManager

Creating ReplicaFetcherThread — `createFetcherThread` Method

```
createFetcherThread(
  fetcherId: Int,
  sourceBroker: BrokerEndPoint): ReplicaFetcherThread
```

Note	`createFetcherThread` is part of the AbstractFetcherManager contract to… FIXME.

`createFetcherThread` creates a ReplicaFetcherThread with the thread name (that uses the threadNamePrefix if defined, the given `fetcherId` and the id of the `BrokerEndPoint`) as follows:

```
[prefix]ReplicaFetcherThread-[fetcherId]-[sourceBrokerId]
```

`shutdown` Method

```
shutdown(): Unit
```

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

```
shutting down
```

`shutdown` closeAllFetchers.

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

```
shutdown completed
```

Note	`shutdown` is used when…FIXME

ReplicaFetcherThread

`ReplicaFetcherThread` is an uninterruptible AbstractFetcherThread that…FIXME

`ReplicaFetcherThread` is used (*managed*) by ReplicaFetcherManager.

`ReplicaFetcherThread` is created when `ReplicaFetcherManager` is requested to create a fetcher thread (when…FIXME).

`ReplicaFetcherThread` uses replica.fetch.backoff.ms configuration property for the fetchBackOffMs.

`ReplicaFetcherThread` uses replica.fetch.max.bytes configuration property for…FIXME

Creating ReplicaFetcherThread Instance

`ReplicaFetcherThread` takes the following to be created:

- Name
- Fetcher ID
- Source `BrokerEndPoint`
- KafkaConfig
- ReplicaManager
- Metrics
- `Time`
- `ReplicaQuota`
- Optional `BlockingSend` (default: `None`)

`ReplicaFetcherThread` initializes the internal properties.

`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, Epoch
EndOffset]
```

Note	`fetchEpochsFromLeader` is a part of AbstractFetcherThread Contract.

`fetchEpochsFromLeader` ...FIXME

`processPartitionData` Method

```
processPartitionData(
  topicPartition: TopicPartition,
  fetchOffset: Long,
  partitionData: FetchData): Option[LogAppendInfo]
```

Note	`processPartitionData` is part of the AbstractFetcherThread Contract to... FIXME.

`processPartitionData` ...FIXME

`truncate` Method

```
truncate(
  tp: TopicPartition,
  offsetTruncationState: OffsetTruncationState): Unit
```

Note	`truncate` is a part of AbstractFetcherThread Contract.

`truncate` ...FIXME

`truncateFullyAndStartAt` Method

```
truncateFullyAndStartAt(topicPartition: TopicPartition, offset: Long): Unit
```

Note	`truncateFullyAndStartAt` is a part of AbstractFetcherThread Contract.

`truncateFullyAndStartAt` ...FIXME

`latestEpoch` Method

```
latestEpoch(topicPartition: TopicPartition): Option[Int]
```

Note	`latestEpoch` is a part of AbstractFetcherThread Contract to get the latest (current) leader epoch for the given `TopicPartition`.

`latestEpoch` requests the ReplicaManager for the local partition replica (or throw an exception) for the given `TopicPartition` and requests the replica for the latest (current) leader epoch.

`isOffsetForLeaderEpochSupported` Method

```
isOffsetForLeaderEpochSupported: Boolean
```

Note	`isOffsetForLeaderEpochSupported` is part of the AbstractFetcherThread Contract to control whether a `OffsetsForLeaderEpochRequest` is supported (that was added in Kafka 0.11.0.0).

`isOffsetForLeaderEpochSupported` simply returns the brokerSupportsLeaderEpochRequest flag.

`buildFetch` Method

```
buildFetch(
  partitionMap: Map[TopicPartition, PartitionFetchState]
): ResultWithPartitions[Option[FetchRequest.Builder]]
```

Note	`buildFetch` is part of the AbstractFetcherThread Contract to…FIXME

`buildFetch` requests the FetchSessionHandler for a new `Builder`.

For every `TopicPartition` (in the given `partitionMap`), `buildFetch` finds the logStartOffset of the local log (using the ReplicaManager) and adds them as `PartitionData` to the `Builder`.

`buildFetch` requests the `Builder` to create a `FetchRequestData`.

In the end, `buildFetch` creates a `FetchRequest.Builder` for the local partition replica.

Internal Properties

Name	Description
`brokerSupportsLeaderEpochRequest`	Flag that says whether the inter.broker.protocol.version is `0.11.0-IV2` or above Used in isOffsetForLeaderEpochSupported
`fetchSessionHandler`	`FetchSessionHandler` (with the logContext and the ID of the source broker)

ReplicaAlterLogDirsManager

`ReplicaAlterLogDirsManager` is a AbstractFetcherManager of ReplicaAlterLogDirsThreads with the properties.

Table 1. ReplicaAlterLogDirsThreads' Configuration Properties

Name	Description
`name`	`ReplicaAlterLogDirsManager on broker [brokerId]`
`clientId`	`ReplicaAlterLogDirs`
`numFetchers`	KafkaConfig.getNumReplicaAlterLogDirsThreads

`ReplicaAlterLogDirsManager` is created for ReplicaManager.

Creating ReplicaAlterLogDirsManager Instance

`ReplicaAlterLogDirsManager` takes the following to be created:

- KafkaConfig
- ReplicaManager
- ReplicationQuotaManager
- BrokerTopicStats

Creating ReplicaAlterLogDirsThread — `createFetcherThread` Method

```
createFetcherThread(
  fetcherId: Int,
  sourceBroker: BrokerEndPoint): ReplicaAlterLogDirsThread
```

Note	`createFetcherThread` is part of the AbstractFetcherManager contract to create a fetcher thread.

`createFetcherThread` creates a ReplicaAlterLogDirsThread with the thread name as `ReplicaAlterLogDirsThread-[fetcherId]`.

`shutdown` Method

```
shutdown(): Unit
```

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

```
shutting down
```

`shutdown` closeAllFetchers.

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

```
shutdown completed
```

Note	`shutdown` is used when `ReplicaManager` is requested to shut down.

ReplicaAlterLogDirsThread

`ReplicaAlterLogDirsThread` is a fetcher thread of ReplicaAlterLogDirsManager.

`ReplicaAlterLogDirsThread` is created when `ReplicaAlterLogDirsManager` is requested to create a fetcher thread.

`ReplicaAlterLogDirsThread` uses the following configuration properties.

Table 1. ReplicaAlterLogDirsThread's Configuration Properties

Name	Description
replica.fetch.backoff.ms	Used for fetchBackOffMs
broker.id	Used to buildFetchForPartition
replica.fetch.response.max.bytes	Used to buildFetchForPartition
replica.fetch.max.bytes	Used to buildFetchForPartition

`ReplicaAlterLogDirsThread` has isOffsetForLeaderEpochSupported flag always enabled (`true`).

Creating ReplicaAlterLogDirsThread Instance

`ReplicaAlterLogDirsThread` takes the following to be created:

- Thread Name
- Source `BrokerEndPoint`
- KafkaConfig
- ReplicaManager
- ReplicationQuotaManager
- BrokerTopicStats

`ReplicaAlterLogDirsThread` initializes the internal registries and counters.

`fetchFromLeader` Method

```
fetchFromLeader(
  fetchRequest: FetchRequest.Builder): Seq[(TopicPartition, FetchData)]
```

Note	`fetchFromLeader` is part of the AbstractFetcherThread contract to…FIXME.

`fetchFromLeader` …FIXME

`processPartitionData` Method

```
processPartitionData(
  topicPartition: TopicPartition,
  fetchOffset: Long,
  partitionData: PartitionData[Records]): Option[LogAppendInfo]
```

Note	`processPartitionData` is part of the AbstractFetcherThread Contract to…FIXME.

`processPartitionData` …FIXME

`truncateFullyAndStartAt` Method

```
truncateFullyAndStartAt(
  topicPartition: TopicPartition,
  offset: Long): Unit
```

Note	`truncateFullyAndStartAt` is part of the AbstractFetcherThread Contract to…FIXME.

`truncateFullyAndStartAt` …FIXME

`truncate` Method

```
truncate(
  topicPartition: TopicPartition,
  truncationState: OffsetTruncationState): Unit
```

Note	`truncate` is part of the AbstractFetcherThread Contract to…FIXME.

`truncate` …FIXME

`buildFetch` Method

```
buildFetch(
  partitionMap: Map[TopicPartition, PartitionFetchState]
): ResultWithPartitions[Option[FetchRequest.Builder]]
```

Note	`buildFetch` is part of the AbstractFetcherThread Contract to…FIXME

`buildFetch` …FIXME

`buildFetchForPartition` Internal Method

```
buildFetchForPartition(
  tp: TopicPartition,
  fetchState: PartitionFetchState): ResultWithPartitions[Option[FetchRequest.Builder]]
```

`buildFetchForPartition` …FIXME

Note	`buildFetchForPartition` is used when `ReplicaAlterLogDirsThread` is requested to buildFetch.

AbstractFetcherManager

`AbstractFetcherManager` is the abstraction of fetcher thread managers that can create fetcher threads (for ReplicaManager).

Note	`AbstractFetcherManager` is an abstract class and cannot be created directly. It is created indirectly for the concrete AbstractFetcherManagers.

Table 1. AbstractFetcherManager Contract (Abstract Methods)

Method	Description

`createFetcherThread`	`createFetcherThread(`
	`  fetcherId: Int,`
	`  sourceBroker: BrokerEndPoint): T`
	
	Creates a fetcher thread
	
	Used when `AbstractFetcherManager` is requested to addFetcherForPartitions

Table 2. AbstractFetcherManagers

AbstractFetcherManager	Description
ReplicaAlterLogDirsManager	
ReplicaFetcherManager	

Performance Metrics

AbstractFetcherManagers are KafkaMetricsGroups with the following performance metrics.

Table 3. AbstractFetcherManager's Performance Metrics

Metric Name	Description
`DeadThreadCount`	
`FailedPartitionsCount`	
`MaxLag`	
`MinFetchRate`	

The performance metrics are registered in **kafka.server:type=** group with the `type` being the name of a AbstractFetcherManager.

Creating AbstractFetcherManager Instance

`AbstractFetcherManager` takes the following to be created:

- Log Prefix
- Client ID (for a tag in performance metrics)
- num.replica.fetchers

`AbstractFetcherManager` initializes the internal properties.

`addFetcherForPartitions` Method

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

`addFetcherForPartitions` ...FIXME

Note	`addFetcherForPartitions` is used when: • `AbstractFetcherManager` is requested to resizeThreadPool (when `DynamicThreadPool` is requested to reconfigure) • `ReplicaManager` is requested to alterReplicaLogDirs, becomeLeaderOrFollower and makeFollowers

`resizeThreadPool` Method

```
resizeThreadPool(newSize: Int): Unit
```

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

```
Resizing fetcher thread pool size from [currentSize] to [newSize]
```

`resizeThreadPool` ...FIXME

Note	`resizeThreadPool` is used exclusively when `DynamicThreadPool` is requested to reconfigure.

Finding Fetcher Thread ID — `getFetcherId` Method

```
getFetcherId(
  topicPartition: TopicPartition): Int
```

`getFetcherId` calculates a hash of the topic and partition (from the given `TopicPartition`) modulo (`%`) the numFetchersPerBroker.

Note	`getFetcherId` is used when `AbstractFetcherManager` is requested to markPartitionsForTruncation and addFetcherForPartitions.

`getFetcher` Method

```
getFetcher(topicPartition: TopicPartition): Option[T]
```

`getFetcher` ...FIXME

Note	`getFetcher` is used when...FIXME

`markPartitionsForTruncation` Method

```
markPartitionsForTruncation(
  brokerId: Int,
  topicPartition: TopicPartition,
  truncationOffset: Long): Unit
```

`markPartitionsForTruncation` ...FIXME

Note	`markPartitionsForTruncation` is used when...FIXME

`removeFetcherForPartitions` Method

```
removeFetcherForPartitions(
  partitions: Set[TopicPartition]): Unit
```

`removeFetcherForPartitions` ...FIXME

Note	`removeFetcherForPartitions` is used when: • `AbstractFetcherManager` is requested to resizeThreadPool • `ReplicaManager` is requested to stopReplicas, alterReplicaLogDirs, makeLeaders, makeFollowers, and handleLogDirFailure

`closeAllFetchers` Method

```
closeAllFetchers(): Unit
```

`closeAllFetchers` requests the fetcher threads to initiateShutdown.

`closeAllFetchers` requests the fetcher threads to shutdown.

In the end, `closeAllFetchers` removes (*clears*) all entries from the fetcherThreadMap internal registry.

Note	`closeAllFetchers` is used when ReplicaAlterLogDirsManager and ReplicaFetcherManager are requested to shut down.

Internal Properties

Name	Description
`fetcherThreadMap`	AbstractFetcherThreads by (a pair of) broker ID and fetcher ID
`numFetchersPerBroker`	Number of fetcher threads per broker Initialized with the given numFetchers when `AbstractFetcherManager` is created Resized in resizeThreadPool Used in getFetcherId

AbstractFetcherThread

`AbstractFetcherThread` is the extension of the ShutdownableThread contract for fetcher threads that maybeTruncate followed by maybeFetch repeatedly (while doing the work) until shut down.

Note	`AbstractFetcherThread` is an abstract class and cannot be created directly. It is created indirectly for the concrete AbstractFetcherThreads.

Table 1. AbstractFetcherThread Contract (Abstract Methods)

Method	Description

`buildFetch`	`buildFetch(`
	`  partitionMap: Map[TopicPartition, PartitionFetch`
	`): ResultWithPartitions[Option[FetchRequest.Builde`
	
	Used when `AbstractFetcherThread` is requested to maybeFetch.

`endOffsetForEpoch`	`endOffsetForEpoch(`
	`  topicPartition: TopicPartition,`
	`  epoch: Int): Option[OffsetAndEpoch]`
	
	Used when `AbstractFetcherThread` is requested to getOffsetTruncationState.

`fetchEarliestOffsetFromLeader`	`fetchEarliestOffsetFromLeader(`
	`  topicPartition: TopicPartition,`
	`  currentLeaderEpoch: Int): Long`
	
	Used when `AbstractFetcherThread` is requested to fetchOffsetAndTruncate.

`fetchEpochEndOffsets`	`fetchEpochEndOffsets(`
	`  partitions: Map[TopicPartition, EpochData]`
	`): Map[TopicPartition, EpochEndOffset]`
	
	Used when `AbstractFetcherThread` is requested to truncateToEpochEndOffsets.

`fetchFromLeader`	`fetchFromLeader(`
	`  fetchRequest: FetchRequest.Builder`
	`): Seq[(TopicPartition, FetchData)]`

	Used when `AbstractFetcherThread` is requested to processFetchRequest.

`fetchLatestOffsetFromLeader`	`fetchLatestOffsetFromLeader(`
	`topicPartition: TopicPartition,`
	`currentLeaderEpoch: Int): Long`
	
	Used when `AbstractFetcherThread` is requested to fetchOffsetAndTruncate.

`isOffsetForLeaderEpochSupported`	`isOffsetForLeaderEpochSupported: Boolean` Controls whether an `OffsetsForLeaderEpochRequest` is supported (that was added in Kafka 0.11.0.0) Used when `AbstractFetcherThread` is requested to fetchTruncatingPartitions. Tip: Consult KIP-101: Alter Replication Protocol t Leader Epoch rather than High Watermark fo Truncation.
`latestEpoch`	`latestEpoch(topicPartition: TopicPartition): Optio` Gets the latest (current) leader epoch for the given `TopicPartition` Used when `AbstractFetcherThread` is requested to fetchTruncatingPartitions.
`logEndOffset`	`logEndOffset(topicPartition: TopicPartition): Long` Used when `AbstractFetcherThread` is requested to getOffsetTruncationState and fetchOffsetAndTruncate

`processPartitionData`	`processPartitionData(`
	`topicPartition: TopicPartition,`
	`fetchOffset: Long,`
	`partitionData: FetchData): Option[LogAppendInfo]`
	
	Used when `AbstractFetcherThread` is requested to processFetchRequest.

`truncate`	`truncate(`
	`topicPartition: TopicPartition,`
	`truncationState: OffsetTruncationState): Unit`

	Used when `AbstractFetcherThread` is requested to doTruncate and fetchOffsetAndTruncate

`truncateFullyAndStartAt`	`truncateFullyAndStartAt(`
	`  topicPartition: TopicPartition,`
	`  offset: Long): Unit`
	
	Used when `AbstractFetcherThread` is requested to fetchOffsetAndTruncate.

Table 2. AbstractFetcherThreads

AbstractFetcherThread	Description
ReplicaAlterLogDirsThread	
ReplicaFetcherThread	

Performance Metrics

AbstractFetcherThreads are KafkaMetricsGroups with the following performance metrics.

Table 3. AbstractFetcherThread's Performance Metrics

Metric Name	Description
`BytesPerSec`	
`RequestsPerSec`	

The performance metrics are registered in **kafka.server:type=FetcherStats** group.

Figure 1. AbstractFetcherThread in jconsole

Creating AbstractFetcherThread Instance

`AbstractFetcherThread` takes the following to be created:

- Thread Name
- Client ID
- `BrokerEndPoint`
- `FailedPartitions`

- replica.fetch.backoff.ms (default: `0`)
- `isInterruptible` flag (default: `true`)

`AbstractFetcherThread` initializes the internal registries and counters.

Truncating And Fetching — `doWork` Method

```
doWork(): Unit
```

Note	`doWork` is part of the ShutdownableThread Contract to do the work.

`doWork` simply maybeTruncate followed by maybeFetch.

`maybeTruncate` Internal Method

```
maybeTruncate(): Unit
```

`maybeTruncate` fetchTruncatingPartitions to find the partitions with and without epochs.

For partitions with epochs, `maybeTruncate` truncateToEpochEndOffsets.

For partitions without epochs, `maybeTruncate` truncateToHighWatermark.

Note	`maybeTruncate` is used exclusively when `AbstractFetcherThread` is requested to do the work.

`maybeFetch` Internal Method

```
maybeFetch(): Unit
```

`maybeFetch` buildFetch (with the partitionStates).

Note	buildFetch is implementation-specific.

`maybeFetch` then handlePartitionsWithErrors with partitions that buildFetch could not handle.

In the end, `maybeFetch` processes the `FetchRequest` (if created).

`maybeFetch` blocks the thread (*waits*) until the fetchBackOffMs elapses when the `FetchRequest` was not created (when buildFetch). `maybeFetch` prints out the following TRACE message to the logs:

```
There are no active partitions. Back off for [fetchBackOffMs] ms before sending a fetc
h request
```

Note	`maybeFetch` is used when `AbstractFetcherThread` is requested to do the work.

`fetchTruncatingPartitions` Internal Method

```
fetchTruncatingPartitions(): (Map[TopicPartition, EpochData], Set[TopicPartition])
```

`fetchTruncatingPartitions` finds the `TopicPartitions` (in the partitionStates registry) that are in `Truncating` state (and are not delayed) and splits them into two groups: with and without epochs.

Internally, for every truncating `TopicPartition`, `fetchTruncatingPartitions` gets the latest epoch.

If the latest epoch is available and isOffsetForLeaderEpochSupported, `fetchTruncatingPartitions` registers the `TopicPartition` as "with epoch" whereas the others as "without epoch".

Note	Both actions (getting the latest epoch and isOffsetForLeaderEpochSupported) are implementation-specific.

Note	`fetchTruncatingPartitions` is used exclusively when `AbstractFetcherThread` is requested to maybeTruncate.

`onPartitionFenced` Internal Method

```
onPartitionFenced(tp: TopicPartition): Unit
```

`onPartitionFenced` ...FIXME

Note	`onPartitionFenced` is used when `AbstractFetcherThread` is requested to maybeTruncateToEpochEndOffsets, processFetchRequest, and handleOutOfRangeError.

`maybeTruncateToEpochEndOffsets` Internal Method

```
maybeTruncateToEpochEndOffsets(
  fetchedEpochs: Map[TopicPartition, EpochEndOffset]
): ResultWithPartitions[Map[TopicPartition, OffsetTruncationState]]
```

`maybeTruncateToEpochEndOffsets` ...FIXME

Note	`maybeTruncateToEpochEndOffsets` is used when `AbstractFetcherThread` is requested to truncateToEpochEndOffsets.

`processFetchRequest` Internal Method

```
processFetchRequest(
  fetchStates: Map[TopicPartition, PartitionFetchState],
  fetchRequest: FetchRequest.Builder): Unit
```

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

```
Sending fetch request [fetchRequest]
```

`processFetchRequest` then fetchFromLeader.

Note	fetchFromLeader is implementation-specific.

`processFetchRequest` requests the FetcherStats to...FIXME

Note	`processFetchRequest` is used when `AbstractFetcherThread` is requested to maybeFetch.

`handleOutOfRangeError` Internal Method

```
handleOutOfRangeError(
  topicPartition: TopicPartition,
  fetchState: PartitionFetchState): Boolean
```

`handleOutOfRangeError` ...FIXME

Note	`handleOutOfRangeError` is used when `AbstractFetcherThread` is requested to truncate and fetch continuously (and in turn processFetchRequest).

`markPartitionsForTruncation` Method

```
markPartitionsForTruncation(
  topicPartition: TopicPartition,
  truncationOffset: Long): Unit
markPartitionsForTruncation(
  brokerId: Int,
  topicPartition: TopicPartition,
  truncationOffset: Long): Unit
```

`markPartitionsForTruncation` ...FIXME

Note	`markPartitionsForTruncation` is used when...FIXME

`addPartitions` Method

```
addPartitions(
  initialFetchStates: Map[TopicPartition, OffsetAndEpoch]): Unit
```

`addPartitions` ...FIXME

Note	`addPartitions` is used when `AbstractFetcherManager` is requested to addFetcherForPartitions.

`updateFetchOffsetAndMaybeMarkTruncationComplete` Internal Method

```
updateFetchOffsetAndMaybeMarkTruncationComplete(
  fetchOffsets: Map[TopicPartition, OffsetTruncationState]): Unit
```

`updateFetchOffsetAndMaybeMarkTruncationComplete` ...FIXME

Note	`updateFetchOffsetAndMaybeMarkTruncationComplete` is used when `AbstractFetcherThread` is requested to truncateToEpochEndOffsets and truncateToHighWatermark.

`truncateToEpochEndOffsets` Internal Method

```
truncateToEpochEndOffsets(
  latestEpochsForPartitions: Map[TopicPartition, EpochData]): Unit
```

`truncateToEpochEndOffsets` ...FIXME

Note	`truncateToEpochEndOffsets` is used exclusively when `AbstractFetcherThread` is requested to maybeTruncate.

`truncateToHighWatermark` Internal Method

```
truncateToHighWatermark(
  partitions: Set[TopicPartition]): Unit
```

`truncateToHighWatermark` ...FIXME

Note	`truncateToHighWatermark` is used exclusively when `AbstractFetcherThread` is requested to maybeTruncate.

`partitionsAndOffsets` Method

```
partitionsAndOffsets: Map[TopicPartition, InitialFetchState]
```

`partitionsAndOffsets` ...FIXME

Note	`partitionsAndOffsets` is used exclusively when `AbstractFetcherManager` is requested to resizeThreadPool.

`getOffsetTruncationState` Internal Method

```
getOffsetTruncationState(
  tp: TopicPartition,
  leaderEpochOffset: EpochEndOffset): OffsetTruncationState
```

`getOffsetTruncationState` ...FIXME

Note	`getOffsetTruncationState` is used exclusively when `AbstractFetcherThread` is requested to maybeTruncateToEpochEndOffsets.

`fetchOffsetAndTruncate` Method

```
fetchOffsetAndTruncate(
  topicPartition: TopicPartition,
  currentLeaderEpoch: Int): Long
```

`fetchOffsetAndTruncate` ...FIXME

Note	`fetchOffsetAndTruncate` is used exclusively when `AbstractFetcherThread` is requested to addPartitions and handleOutOfRangeError.

`handlePartitionsWithErrors` Method

```
handlePartitionsWithErrors(
  partitions: Iterable[TopicPartition]): Unit
```

`handlePartitionsWithErrors` ...FIXME

Note	`handlePartitionsWithErrors` is used when `AbstractFetcherThread` is requested to maybeFetch, truncateToEpochEndOffsets, truncateToHighWatermark, and processFetchRequest.

`doTruncate` Internal Method

```
doTruncate(
  topicPartition: TopicPartition,
  truncationState: OffsetTruncationState): Boolean
```

`doTruncate` ...FIXME

Note	`doTruncate` is used when `AbstractFetcherThread` is requested to maybeTruncate (and in turn truncateToHighWatermark and maybeTruncateToEpochEndOffsets).

`toMemoryRecords` Method

```
toMemoryRecords(
  records: Records): MemoryRecords
```

`toMemoryRecords` converts the given records to MemoryRecords.

If the given records are of type MemoryRecords already, `toMemoryRecords` simply returns it as-is.

If the given records are of type FileRecords, `toMemoryRecords` allocates a `ByteBuffer` of the size of the records to readInto and creates a writable MemoryRecords.

Note	`toMemoryRecords` is used when: • `ReplicaAlterLogDirsThread` is requested to processPartitionData • `ReplicaFetcherThread` is requested to processPartitionData

`shutdown` Method

```
shutdown(): Unit
```

Note	`shutdown` is part of the ShutdownableThread contract to…FIXME.

`shutdown` …FIXME

Internal Properties

Name	Description
`fetcherStats`	KafkaMetricsGroup (with two performance metrics)
`partitionStates`	`TopicPartitions` with their `PartitionFetchState` (`PartitionStates[PartitionFetchState]`) • A `TopicPartition` is removed in onPartitionFenced and removePartitions • `PartitionFetchState` updated in processFetchRequest, markPartitionsForTruncation, addPartitions, updateFetchOffsetAndMaybeMarkTruncationComplete, handleOutOfRangeError, delayPartitions Used when…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 configuration property when sending a request to the partition leader broker.

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

Internal Properties

Name	Description
`networkClient`	NetworkClient Used when...FIXME
`sourceNode`	Source `Node` Used when...FIXME

ReplicationQuotaManager

`ReplicationQuotaManager` is…FIXME

LogDirFailureHandler

`LogDirFailureHandler` is...FIXME

`start` Method

Caution	FIXME

`doWork` Method

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

Note	`doWork` is a part of ShutdownableThread Contract.

`doWork` ...FIXME

ShutdownableThread

`ShutdownableThread` is the contract for non-daemon threads of execution.

`ShutdownableThread` contract expects that the objects implement doWork method.

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

`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
```

Pausing (Until Shutdown or Timeout) — `pause` Method

```
pause(
  timeout: Long,
  unit: TimeUnit): Unit
```

`pause` ...FIXME

Note	`pause` is used when...FIXME

`initiateShutdown` Method

```
initiateShutdown(): Boolean
```

`initiateShutdown` ...FIXME

Note	`initiateShutdown` is used when...FIXME

Internal Properties

Name	Description
`isRunning`	Flag that controls how long to execute run method.
`shutdownLatch`	Java's java.util.concurrentCountDownLatch with the number of passes being `1`

SocketServer

`SocketServer` is a NIO socket server for a KafkaServer.

`SocketServer` is created and then started up exclusively when `KafkaServer` is requested to start up.

Figure 1. SocketServer's Startup

`SocketServer` uses queued.max.requests configuration property for…FIXME

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

Name	Description
`NetworkProcessorAvgIdlePercent`	
`MemoryPoolAvailable`	
`MemoryPoolUsed`	

`SocketServer` uses an empty metric prefix for…FIXME

`SocketServer` uses **ControlPlane** metric prefix for…FIXME

Creating SocketServer Instance

`SocketServer` takes the following to be created:

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

`SocketServer` initializes the internal properties.

Control-Plane Request Handler for Controller-Brokers Communication — `controlPlaneRequestChannelOpt` Internal Value

`SocketServer` creates a dedicated RequestChannel for communication between a controller and brokers when control.plane.listener.name configuration property is defined.

The `RequestChannel` is created for `KafkaServer` (when requested to start up). It is the time when `KafkaServer` creates dedicated control-plane request processor and control-plane request handler pool.

The `RequestChannel` is created with the queue size of `20` and the ControlPlane metric name prefix.

`SocketServer` manages the `RequestChannel` (when requested to stopProcessingRequests and shutdown).

The `RequestChannel` is assumed to be available when `SocketServer` is requested to createControlPlaneAcceptorAndProcessor (when requested to start up) based on control.plane.listener.name configuration property.

Creating Network Processor Thread — `newProcessor` Internal Method

```
newProcessor(
  id: Int,
  connectionQuotas: ConnectionQuotas,
  listenerName: ListenerName,
  securityProtocol: SecurityProtocol,
  memoryPool: MemoryPool): Processor
```

`newProcessor` simply creates a new Processor with the RequestChannel and the following configuration properties:

- socket.request.max.bytes
- connections.max.idle.ms
- connection.failed.authentication.delay.ms

Note	`newProcessor` is used exclusively when `SocketServer` is requested to createControlPlaneAcceptorAndProcessor and addDataPlaneProcessors.

Starting Up — `startup` Method

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

Internally, `startup` creates the ConnectionQuotas (with maxConnectionsPerIp and maxConnectionsPerIpOverrides).

For every endpoint (in endpoints registry) `startup` does the following:

1. Creates up to numProcessorThreads number of Processors (for ConnectionQuotas and MemoryPool)
2. Creates a `Acceptor` for the endpoint and processors
3. Records the `Acceptor` in acceptors internal registry
4. Starts a non-daemon thread for the `Acceptor` with the name as `kafka-socket-acceptor-[listenerName]-[securityProtocol]-[port]` (e.g. `kafka-socket-acceptor-ListenerName(PLAINTEXT)-PLAINTEXT-9092`) and waits until it has started fully

`startup` then registers metrics.

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

```
Started [dataPlaneAcceptors] acceptor threads for data-plane
```

Note	`startup` is used exclusively when `KafkaServer` is requested to start up.

`addProcessors` Internal Method

```
addProcessors(
  acceptor: Acceptor,
  endpoint: EndPoint,
  newProcessorsPerListener: Int): Unit
```

`addProcessors` ...FIXME

Note	`addProcessors` is used when `SocketServer` is requested to createAcceptorAndProcessors and resizeThreadPool.

`createAcceptorAndProcessors` Internal Method

```
createAcceptorAndProcessors(
  processorsPerListener: Int,
  endpoints: Seq[EndPoint]): Unit
```

`createAcceptorAndProcessors` ...FIXME

Note	`createAcceptorAndProcessors` is used when `SocketServer` is requested to startup and addListeners.

`resizeThreadPool` Method

```
resizeThreadPool(
  oldNumNetworkThreads: Int,
  newNumNetworkThreads: Int): Unit
```

`resizeThreadPool` ...FIXME

Note	`resizeThreadPool` is used exclusively when `DynamicThreadPool` is requested to reconfigure (the number of network threads).

`addListeners` Method

```
addListeners(listenersAdded: Seq[EndPoint]): Unit
```

`addListeners` ...FIXME

Note	`addListeners` is used exclusively when `DynamicListenerConfig` is requested to reconfigure.

Stopping Request Processors — `stopProcessingRequests` Method

```
stopProcessingRequests(): Unit
```

`stopProcessingRequests` ...FIXME

Note	`stopProcessingRequests` is used when: • `SocketServer` is requested to shutdown • `KafkaServer` is requested to shutdown

Shutting Down — `shutdown` Method

```
shutdown(): Unit
```

`shutdown` ...FIXME

Note	`shutdown` is used when...FIXME

`updateMaxConnectionsPerIpOverride` Method

```
updateMaxConnectionsPerIpOverride(
  maxConnectionsPerIpOverrides: Map[String, Int]): Unit
```

`updateMaxConnectionsPerIpOverride` ...FIXME

Note	`updateMaxConnectionsPerIpOverride` is used when...FIXME

`updateMaxConnectionsPerIp` Method

```
updateMaxConnectionsPerIp(maxConnectionsPerIp: Int): Unit
```

`updateMaxConnectionsPerIp` ...FIXME

Note	`updateMaxConnectionsPerIp` is used when...FIXME

`removeListeners` Method

```
removeListeners(listenersRemoved: Seq[EndPoint]): Unit
```

`removeListeners` ...FIXME

Note	`removeListeners` is used when...FIXME

`addDataPlaneProcessors` Internal Method

```
addDataPlaneProcessors(
  acceptor: Acceptor,
  endpoint: EndPoint,
  newProcessorsPerListener: Int): Unit
```

`addDataPlaneProcessors` ...FIXME

Note	`addDataPlaneProcessors` is used when `SocketServer` is requested to createDataPlaneAcceptorsAndProcessors and resizeThreadPool.

`createDataPlaneAcceptorsAndProcessors` Internal Method

```
createDataPlaneAcceptorsAndProcessors(
  dataProcessorsPerListener: Int,
  endpoints: Seq[EndPoint]): Unit
```

`createDataPlaneAcceptorsAndProcessors` ...FIXME

Note	`createDataPlaneAcceptorsAndProcessors` is used when `SocketServer` is requested to start up and addListeners.

`createControlPlaneAcceptorAndProcessor` Internal Method

```
createControlPlaneAcceptorAndProcessor(
  endpointOpt: Option[EndPoint]): Unit
```

`createControlPlaneAcceptorAndProcessor` ...FIXME

Note	`createControlPlaneAcceptorAndProcessor` is used when `SocketServer` is requested to start up.

`startDataPlaneProcessors` Internal Method

```
startDataPlaneProcessors(
  authorizerFutures: Map[Endpoint, CompletableFuture[Void]] = Map.empty): Unit
```

`startDataPlaneProcessors` ...FIXME

Note	`startDataPlaneProcessors` is used when...FIXME

`createAcceptor` Internal Method

```
createAcceptor(
  endPoint: EndPoint,
  metricPrefix: String) : Acceptor
```

`createAcceptor` ...FIXME

Note	`createAcceptor` is used when...FIXME

Internal Properties

Name	Description
`acceptors`	`Acceptor` threads per `EndPoint`
`connectionQuotas`	`ConnectionQuotas`
`dataPlaneRequestChannel`	RequestChannel (with the queue size of maxQueuedRequests and the DataPlaneMetricPrefix metric name prefix) Initialized when `SocketServer` is requested to addDataPlaneProcessors Used to create the dataPlaneRequestProcessor and dataPlaneRequestHandlerPool for `KafkaServer`
`endpoints`	`EndPoints` (aka *listeners*) per name (as configured using listeners Kafka property)
`maxConnectionsPerIp`	
`maxConnectionsPerIpOverrides`	
`memoryPool`	`MemoryPool`
`numProcessorThreads`	The number of processors per endpoint (as configured using num.network.threads Kafka property)
`processors`	Network processor threads per ID (initially totalProcessorThreads) New processor threads are added in addProcessors Used in stopProcessingRequests (to shut down the network processor threads)
	A RequestChannel (with queued.max.requests queue size)

`requestChannel`	Used when: • `SocketServer` is requested to create a network processor thread, addProcessors, stopProcessingRequests, resizeThreadPool, and shutdown • `KafkaServer` is requested to start up (and creates the KafkaApis and the KafkaRequestHandlerPool)
`totalProcessorThreads`	Total number of processors, i.e. numProcessorThreads for every endpoint

Network Processor Thread (Socket Server Request Processor)

`Processor` is a java.lang.Runnable (as a `AbstractServerThread`) that is executed as a separately-executing thread.

`Processor` is also a KafkaMetricsGroup that…FIXME

`Processor` is created exclusively when `SocketServer` is requested to create one.

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

Name	Description
`inflightResponses`	`Map[String, RequestChannel.Response]` Used when…FIXME
`newConnections`	`ConcurrentLinkedQueue[SocketChannel]` Used when…FIXME
`responseQueue`	`LinkedBlockingDeque[RequestChannel.Response]` Used when…FIXME
`selector`	Selector Used when…FIXME

Creating Processor Instance

`Processor` takes the following to be created:

- ID
- `Time`
- `maxRequestSize`
- RequestChannel
- `ConnectionQuotas`
- `connectionsMaxIdleMs`
- `failedAuthenticationDelayMs`
- Listener Name

- `SecurityProtocol`
- KafkaConfig
- Metrics
- `CredentialProvider`
- `MemoryPool`
- `LogContext`

`Processor` initializes the internal registries and counters.

`processCompletedReceives` Internal Method

```
processCompletedReceives()
```

`processCompletedReceives` ...FIXME

Note	`processCompletedReceives` is used exclusively when `Processor` is requested to run.

`run` Method

```
run(): Unit
```

Note	`run` is part of the java.lang.Runnable to start itself as a separately-executing thread.

`run` ...FIXME

`configureNewConnections` Internal Method

```
configureNewConnections(): Unit
```

`configureNewConnections` ...FIXME

Note	`configureNewConnections` is used exclusively when `Processor` is requested to run.

`createSelector` Internal Method

```
createSelector(
  channelBuilder: ChannelBuilder): KSelector
```

`createSelector` ...FIXME

Note	`createSelector` is used when `Processor` is created (and creates the Selector).

RequestChannel

`RequestChannel` is…FIXME

`RequestChannel` is also a KafkaMetricsGroup that…FIXME

`RequestChannel` is created exclusively when `SocketServer` is created (and is then used to create a Processor, KafkaApis, KafkaRequestHandler and KafkaRequestHandlerPool).

`RequestChannel` takes the size of the queue to be created (that is controlled by queued.max.requests configuration property).

RequestChannel.Request

`RequestChannel.Request` is created exclusively when `Processor` is requested to processCompletedReceives.

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

Name	Description
`bodyAndSize`	`RequestAndSize` that is created by requesting the RequestContext to parse a request (from the given ByteBuffer) Used when…FIXME

Creating RequestChannel.Request Instance

`Request` takes the following to be created:

- Processor ID
- RequestContext
- `startTimeNanos`
- `MemoryPool`
- java.nio.ByteBuffer
- `RequestChannel.Metrics`

`Request` initializes the internal registries and counters.

TransactionCoordinator

`TransactionCoordinator` is…FIXME

`TransactionCoordinator` is created when…FIXME

`startup` Method

```
startup
```

`startup` …FIXME

Note	`startup` is used exclusively when `KafkaServer` is requested to start up.

Creating TransactionCoordinator Instance

`TransactionCoordinator` takes the following when created:

- Broker ID
- `TransactionConfig`
- Scheduler
- `ProducerIdManager`
- TransactionStateManager
- `TransactionMarkerChannelManager`
- `Time`
- `LogContext`

`TransactionCoordinator` initializes the internal registries and counters.

Creating TransactionCoordinator — `apply` Factory Method

```
apply(
  config: KafkaConfig,
  replicaManager: ReplicaManager,
  scheduler: Scheduler,
  zkClient: KafkaZkClient,
  metrics: Metrics,
  metadataCache: MetadataCache,
  time: Time): TransactionCoordinator
```

`apply` ...FIXME

Note	`apply` is used exclusively when `KafkaServer` is requested to start up (to create a TransactionCoordinator).

`shutdown` Method

```
shutdown(): Unit
```

`shutdown` ...FIXME

Note	`shutdown` is used when...FIXME

`abortTimedOutTransactions` Internal Method

```
abortTimedOutTransactions(): Unit
```

`abortTimedOutTransactions` ...FIXME

Note	`abortTimedOutTransactions` is used when...FIXME

`partitionFor` Method

```
partitionFor(transactionalId: String): Int
```

`partitionFor` ...FIXME

Note	`partitionFor` is used when...FIXME

`transactionTopicConfigs` Method

```
transactionTopicConfigs: Properties
```

`transactionTopicConfigs` ...FIXME

Note	`transactionTopicConfigs` is used when `KafkaApis` is requested to create __transaction_state internal topic.

TransactionMarkerChannelManager

`TransactionMarkerChannelManager` is a InterBrokerSendThread that…FIXME

`TransactionMarkerChannelManager` uses the name **TxnMarkerSenderThread-[brokerId]**.

`TransactionMarkerChannelManager` uses **[Transaction Marker Channel Manager [brokerId]]** as the logging prefix (aka `logIdent`).

`TransactionMarkerChannelManager` is created (using apply factory method) exclusively when `TransactionCoordinator` is created (for a KafkaServer).

Creating TransactionMarkerChannelManager Instance

`TransactionMarkerChannelManager` takes the following to be created:

- KafkaConfig
- MetadataCache
- NetworkClient
- TransactionStateManager
- `DelayedOperationPurgatory[DelayedTxnMarker]`
- `Time`

`TransactionMarkerChannelManager` initializes the internal registries and counters.

Creating TransactionMarkerChannelManager — `apply` Factory Method

```
apply(
  config: KafkaConfig,
  metrics: Metrics,
  metadataCache: MetadataCache,
  txnStateManager: TransactionStateManager,
  txnMarkerPurgatory: DelayedOperationPurgatory[DelayedTxnMarker],
  time: Time,
  logContext: LogContext): TransactionMarkerChannelManager
```

`apply`…FIXME

Note	`apply` is used exclusively when `TransactionCoordinator` is created (when `KafkaServer` is requested to start up).

InterBrokerSendThread

`InterBrokerSendThread` is the extension of the ShutdownableThread contract for inter-broker send threads with a non-blocking NetworkClient that doWork.

Note	`TransactionMarkerChannelManager` is the only available implementation of the InterBrokerSendThread Contract in Apache Kafka.

`doWork` Method

```
doWork(): Unit
```

Note	`doWork` is a part of ShutdownableThread Contract.

`doWork` ...FIXME

`sendRequests` Internal Method

```
sendRequests(now: Long): Long
```

`sendRequests` ...FIXME

Note	`sendRequests` is used exclusively when `InterBrokerSendThread` is requested to doWork.

`completeWithDisconnect` Method

```
completeWithDisconnect(
  request: ClientRequest,
  now: Long,
  authenticationException: AuthenticationException): Unit
```

`completeWithDisconnect` ...FIXME

Note	`completeWithDisconnect` is used when `InterBrokerSendThread` is requested to checkDisconnects and failExpiredRequests (when `InterBrokerSendThread` is requested to doWork).

`checkDisconnects` Internal Method

```
checkDisconnects(now: Long): Unit
```

`checkDisconnects` ...FIXME

Note	`checkDisconnects` is used exclusively when `InterBrokerSendThread` is requested to doWork.

`failExpiredRequests` Internal Method

```
failExpiredRequests(now: Long): Unit
```

`failExpiredRequests` ...FIXME

Note	`failExpiredRequests` is used exclusively when `InterBrokerSendThread` is requested to doWork.

Creating InterBrokerSendThread Instance

`InterBrokerSendThread` takes the following to be created:

- Name
- NetworkClient
- `Time`
- `isInterruptible` flag (default: `true`)

`InterBrokerSendThread` initializes the internal registries and counters.

TransactionStateManager

`TransactionStateManager` is…FIXME

`getTransactionTopicPartitionCount` Method

```
getTransactionTopicPartitionCount
```

`getTransactionTopicPartitionCount` …FIXME

Note	`getTransactionTopicPartitionCount` is used when…FIXME

`enableTransactionalIdExpiration` Method

```
enableTransactionalIdExpiration(): Unit
```

`enableTransactionalIdExpiration` …FIXME

Note	`enableTransactionalIdExpiration` is used when…FIXME

`loadTransactionsForTxnTopicPartition` Method

```
loadTransactionsForTxnTopicPartition(
  partitionId: Int,
  coordinatorEpoch: Int,
  sendTxnMarkers: SendTxnMarkersCallback): Unit
```

`loadTransactionsForTxnTopicPartition` …FIXME

Note	`loadTransactionsForTxnTopicPartition` is used when…FIXME

`removeTransactionsForTxnTopicPartition` Method

```
removeTransactionsForTxnTopicPartition(partitionId: Int, coordinatorEpoch: Int): Unit
```

`removeTransactionsForTxnTopicPartition` …FIXME

Note	`removeTransactionsForTxnTopicPartition` is used when…FIXME

`loadTransactions` Method

```
loadTransactions(): Unit
```

`loadTransactions` ...FIXME

Note	`loadTransactions` is used when...FIXME

`removeTransactions` Method

```
removeTransactions(): Unit
```

`removeTransactions` ...FIXME

Note	`removeTransactions` is used when...FIXME

`appendTransactionToLog` Method

```
appendTransactionToLog(
  transactionalId: String,
  coordinatorEpoch: Int,
  newMetadata: TxnTransitMetadata,
  responseCallback: Errors => Unit,
  retryOnError: Errors => Boolean = _ => false): Unit
```

`appendTransactionToLog` ...FIXME

Note	`appendTransactionToLog` is used when...FIXME

`transactionTopicConfigs` Method

```
transactionTopicConfigs: Properties
```

`transactionTopicConfigs` ...FIXME

Note	`transactionTopicConfigs` is used when `TransactionCoordinator` is requested for the transactionTopicConfigs.

`loadTransactionMetadata` Internal Method

```
loadTransactionMetadata(
  topicPartition: TopicPartition,
  coordinatorEpoch: Int): Pool[String, TransactionMetadata]
```

`loadTransactionMetadata` ...FIXME

Note	`loadTransactionMetadata` is used when...FIXME

QuotaManagers

`QuotaManagers` is...FIXME

ZkUtils

`ZkUtils` is…FIXME

`ZkUtils` is created when…FIXME

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
`ControllerPath`	`/controller`

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

Name	Description
`persistentZkPaths`	
`zkPath`	

`getCluster` Method

```
getCluster(): Cluster
```

`getCluster` gets the children znodes of `/brokers/ids` znode and reads their data (as a JSON blob).

`getCluster` then adds creates a `Broker` from the znode id and the JSON blob (with a host, a port and endpoints).

Note	`getCluster` is used exclusively when `ZKRebalancerListener` does syncedRebalance (that happens for the currently-deprecated `ZookeeperConsumerConnector`).

`deletePathRecursive` Method

Caution	FIXME

`deletePath` Method

Caution	FIXME

Creating ZkUtils Instance — `apply` Factory Method

```
apply(
  zkUrl: String,
  sessionTimeout: Int,
  connectionTimeout: Int,
  isZkSecurityEnabled: Boolean): ZkUtils
```

`apply` ...FIXME

Note	`apply` is used when: 1. `KafkaServer` connects to Zookeeper 2. FIXME

Registering Listener for State Changes — `subscribeStateChanges` Method

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

`subscribeStateChanges` requests ZkClient to `subscribeStateChanges` with the `listener`.

Note	`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	`controllerZkData` is used exclusively when `KafkaController` is requested for elect.

Creating ZkUtils Instance

`ZkUtils` takes the following when created:

- `ZkClient`
- `ZkConnection`
- `isSecure` flag

`ZkUtils` initializes the internal registries and counters.

Reading Data Associated with ZNode — `readDataMaybeNull` Method

```
readDataMaybeNull(path: String): (Option[String], Stat)
```

`readDataMaybeNull` requests ZkClient to `readData` from `path` znode.

`readDataMaybeNull` returns `None` (for `Option[String]`) when `path` znode is not available.

ZKRebalancerListener

`ZKRebalancerListener` is…FIXME

`syncedRebalance` Method

```
syncedRebalance(): Unit
```

`syncedRebalance` …FIXME

Note	`syncedRebalance` is used when: • `ZKSessionExpireListener` does `handleNewSession` • That happens exclusively in (deprecated) `ZookeeperConsumerConnector` • `ZKRebalancerListener` is created (and creates `watcherExecutorThread`) • That happens exclusively in (deprecated) `ZookeeperConsumerConnector` • (deprecated) `ZookeeperConsumerConnector` does `reinitializeConsumer`

KafkaMetricsGroup

`KafkaMetricsGroup` is the abstraction of metrics groups that can register **performance metrics**, e.g. gauges, histograms, meters and timers.

```
package kafka.metrics

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

Table 1. KafkaMetricsGroups (Direct Implementations)

KafkaMetricsGroup	Description
AbstractFetcherManager	
Acceptor	
AdminManager	
AppInfo	
BrokerTopicMetrics	Topic metrics
ControllerChannelManager	
ControllerEventManager	
ControllerStats	
DelayedDeleteRecordsMetrics	
DelayedFetchMetrics	
DelayedOperationPurgatory	
DelayedProduceMetrics	
DelegationTokenManager	
FetcherLagMetrics	
FetcherStats	

FetchSessionCache	
GroupMetadataManager	Consumer group metrics
KafkaController	
KafkaRequestHandlerPool	
KafkaServer	
KafkaZkClient	
Log	
LogCleaner	
LogCleanerManager	
LogFlushStats	
LogManager	
Partition	
Processor	
ReplicaManager	
RequestChannel	
RequestMetrics	
SocketServer	
Throttler	
TransactionMarkerChannelManager	
ZooKeeperClient	

`newGauge` Method

```
newGauge[T](
  name: String,
  metric: Gauge[T],
  tags: scala.collection.Map[String, String] = Map.empty): Gauge[T]
```

`newGauge` ...FIXME

Note	`newGauge` is used when...FIXME

`newMeter` Method

```
newMeter(
  name: String,
  eventType: String,
  timeUnit: TimeUnit,
  tags: scala.collection.Map[String, String] = Map.empty): Meter
```

`newMeter` ...FIXME

Note	`newMeter` is used when...FIXME

`newHistogram` Method

```
newHistogram(
  name: String,
  biased: Boolean = true,
  tags: scala.collection.Map[String, String] = Map.empty): Histogram
```

`newHistogram` ...FIXME

Note	`newHistogram` is used when...FIXME

`newTimer` Method

```
newTimer(
  name: String,
  durationUnit: TimeUnit,
  rateUnit: TimeUnit,
  tags: scala.collection.Map[String, String] = Map.empty): Timer
```

`newTimer` ...FIXME

Note	`newTimer` is used when…FIXME

`metricName` Method

```
metricName(name: String, tags: scala.collection.Map[String, String]): MetricName
```

`metricName` …FIXME

Note	`metricName` is used when…FIXME

BrokerTopicStats

`BrokerTopicStats` holds a set of topic-related metrics:

- BrokerTopicMetrics stats per topic (by topic name)
- Accumulated BrokerTopicMetrics for all topics

`BrokerTopicStats` is created exclusively when `KafkaServer` is requested to start up (and is used to create the KafkaApis, the ReplicaManager, the LogManager).

BrokerTopicStats, BrokerTopicMetrics and KafkaMetricsGroup

`BrokerTopicStats` holds one or more BrokerTopicMetrics.

BrokerTopicMetrics is a KafkaMetricsGroup that registers topic-related metrics in **kafka.server:type=BrokerTopicMetrics** group.

Every time `BrokerTopicStats` is updated so are the `BrokerTopicMetrics` of topics. In other words, every update of `BrokerTopicStats` changes **kafka.server:type=BrokerTopicMetrics** group.

Looking Up or Creating New BrokerTopicMetrics — `topicStats` Method

```
topicStats(topic: String): BrokerTopicMetrics
```

`topicStats` looks up or creates a new BrokerTopicMetrics for the given topic name.

Note	`topicStats` is used when: • `Log` is requested to append and analyzeAndValidateRecords • `KafkaApis` is requested to updateRecordConversionStats (for `Produce` and `Fetch` requests) • `BrokerTopicStats` is requested to updateBytesOut • `ReplicaManager` is requested to appendToLocalLog and readFromLocalLog

`updateBytesOut` Method

```
updateBytesOut(
  topic: String,
  isFollower: Boolean,
  value: Long): Unit
```

`updateBytesOut` ...FIXME

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

`updateReplicationBytesOut` Internal Method

```
updateReplicationBytesOut(value: Long): Unit
```

`updateReplicationBytesOut` ...FIXME

Note	`updateReplicationBytesOut` is used exclusively when `BrokerTopicStats` is requested to updateBytesOut.

`updateReplicationBytesIn` Method

```
updateReplicationBytesIn(value: Long): Unit
```

`updateReplicationBytesIn` ...FIXME

Note	`updateReplicationBytesIn` is used exclusively when `ReplicaFetcherThread` is requested to processPartitionData.

`removeMetrics` Method

```
removeMetrics(topic: String): Unit
```

`removeMetrics` ...FIXME

Note	`removeMetrics` is used when `ReplicaManager` is requested to stopReplica and handleLogDirFailure.

`close` Method

```
close(): Unit
```

`close` ...FIXME

Note	`close` is used exclusively when `KafkaServer` is requested to shut down.

BrokerTopicMetrics

`BrokerTopicMetrics` is a KafkaMetricsGroup with performance metrics of topics.

Table 1. BrokerTopicMetrics's Performance Metrics

Metric Name	Description
`BytesInPerSec`	
`BytesOutPerSec`	
`BytesRejectedPerSec`	
`FailedFetchRequestsPerSec`	
`FailedProduceRequestsPerSec`	
`FetchMessageConversionsPerSec`	
`MessagesInPerSec`	
`ProduceMessageConversionsPerSec`	
`ReplicationBytesInPerSec`	(only when topic name is undefined)
`ReplicationBytesOutPerSec`	(only when topic name is undefined)
`TotalFetchRequestsPerSec`	
`TotalProduceRequestsPerSec`	

The performance metrics are registered in **kafka.server:type=BrokerTopicMetrics** group.

Figure 1. BrokerTopicMetrics in jconsole

`BrokerTopicMetrics` is created for `BrokerTopicStats` when requested to look up or create a new BrokerTopicMetrics for a topic or accumulated metrics for all topics.

`BrokerTopicMetrics` takes an optional **topic name** to be created. When not specified, `BrokerTopicMetrics` is assumed to be used for all topics.

`BrokerTopicMetrics` uses metric tags of `topic` and the topic name (if defined).

Consumer Contract — Kafka Clients for Consuming Records

`Consumer` is the abstraction of Kafka clients that can subscribe to topics or assign partitions to consume records from.

`Consumer` is also a Java java.io.Closeable and so can be considered as a *"a source of data that can be closed"*. A `Consumer` instance should always be closed to release system resources.

Table 1. Consumer Contract

Method	Description
`assign`	`void assign(Collection<TopicPartition> partitions)`
`assignment`	`Set<TopicPartition> assignment()`

`beginningOffsets`	`Map<TopicPartition, Long> beginningOffsets(`
	`  Collection<TopicPartition> partitions)`
	`Map<TopicPartition, Long> beginningOffsets(`
	`  Collection<TopicPartition> partitions,`
	`  Duration timeout)`

`close`	`void close()`
	`void close(Duration timeout)`
	
	Closes the `Consumer` to release system resources used

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

`commitSync`	`void commitSync()`
	`void commitSync(`
	`  Duration timeout)`
	`void commitSync(`
	`  Map<TopicPartition, OffsetAndMetadata> offsets)`
	`void commitSync(`
	`  Map<TopicPartition, OffsetAndMetadata> offsets,`
	`  Duration timeout)`

`committed`	`OffsetAndMetadata committed(`
	`  TopicPartition partition)`
	`OffsetAndMetadata committed(`
	`  TopicPartition partition,`
	`  Duration timeout)`

`endOffsets`	`Map<TopicPartition, Long> endOffsets(`
	`  Collection<TopicPartition> partitions)`
	`Map<TopicPartition, Long> endOffsets(`
	`  Collection<TopicPartition> partitions,`
	`  Duration timeout)`

`listTopics`	`Map<String, List<PartitionInfo>> listTopics()`
	`Map<String, List<PartitionInfo>> listTopics(`
	`  Duration timeout)`

`metrics`	`Map<MetricName, ? extends Metric> metrics()`

`offsetsForTimes`	`Map<TopicPartition, OffsetAndTimestamp> offsetsForTimes(`
	`  Map<TopicPartition, Long> timestampsToSearch)`
	`Map<TopicPartition, OffsetAndTimestamp> offsetsForTimes(`
	`  Map<TopicPartition, Long> timestampsToSearch,`
	`  Duration timeout)`

`partitionsFor`	`List<PartitionInfo> partitionsFor(`
	`  String topic)`
	`List<PartitionInfo> partitionsFor(`
	`  String topic,`
	`  Duration timeout)`

`pause`	`void pause(Collection<TopicPartition> partitions)`
`paused`	`Set<TopicPartition> paused()`
`poll`	`ConsumerRecords<K, V> poll(Duration timeout)`

`position`	`long position(`
	`  TopicPartition partition)`
	`long position(`
	`  TopicPartition partition,`
	`  Duration timeout)`

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

`subscribe`	`void subscribe(`
	`  Collection<String> topics)`
	`void subscribe(`
	`  Collection<String> topics,`
	`  ConsumerRebalanceListener callback)`
	`void subscribe(`
	`  Pattern pattern)`
	`void subscribe(`
	`  Pattern pattern,`
	`  ConsumerRebalanceListener callback)`

`subscription`	`Set<String> subscription()`
`unsubscribe`	`void unsubscribe()`
`wakeup`	`void wakeup()`

Table 2. Consumers

Consumer	Description
KafkaConsumer	Primary Kafka client
MockConsumer	Used for unit testing

KafkaConsumer — Primary Kafka Client

`KafkaConsumer` is a concrete Kafka client to consume records from Kafka topics for Kafka developers to write Kafka consumers.

Figure 1. KafkaConsumer

`KafkaConsumer` is created with properties and (key and value) deserializers.

Note	bootstrap.servers and group.id properties are mandatory. Use ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG and ConsumerConfig.GROUP_ID_CONFIG values in your source code, respectively.

```
val bootstrapServers = ":9092,localhost:9092"
val groupId = "kafka-sandbox"
import org.apache.kafka.clients.consumer.ConsumerConfig
val requiredConfigsOnly = Map[String, Object](
  ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG -> bootstrapServers,
  ConsumerConfig.GROUP_ID_CONFIG -> groupId
)
import org.apache.kafka.common.serialization.StringDeserializer
val keyDeserializer = new StringDeserializer
val valueDeserializer = new StringDeserializer

import scala.collection.JavaConverters._
import org.apache.kafka.clients.consumer.KafkaConsumer
val consumer = new KafkaConsumer[String, String](
  requiredConfigsOnly.asJava,
  keyDeserializer,
  valueDeserializer)
```

Once created, `KafkaConsumer` is supposed to subscribe to topics or assign partitions.

```
scala> :type consumer
org.apache.kafka.clients.consumer.KafkaConsumer[String,String]

import scala.collection.JavaConverters._
val topics = Seq("input").asJava
consumer.subscribe(topics)
```

`KafkaConsumer` is then requested to poll for records (in a loop).

```
scala> :type consumer
org.apache.kafka.clients.consumer.KafkaConsumer[String,String]

import java.time.Duration.ZERO
val records = consumer.poll(ZERO)
records.asScala.foreach(println)
```

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

Important	`KafkaConsumer` is not thread-safe, i.e. you should not use the same single instance of `KafkaConsumer` from multiple threads. You should use only one thread per `KafkaConsumer` instance. `KafkaConsumer` uses light locks protecting itself from multi-threaded access and reports `ConcurrentModificationException` when it happens. `KafkaConsumer is not safe for multi-threaded access`

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

Name	Description
`assignors`	Zero or more PartitionAssignors Configured using ConsumerConfig.PARTITION_ASSIGNMENT_STRATEGY_CONFIG (aka `partition.assignment.strategy`) configuration property Used exclusively to create the ConsumerCoordinator
`client`	ConsumerNetworkClient Used mainly *(?)* to create the Fetcher and ConsumerCoordinator Used also in poll, pollOnce and wakeup (but *I* think the usage should be limited to create Fetcher and ConsumerCoordinator)
`clientId`	
`coordinator`	ConsumerCoordinator Used in subscribe, unsubscribe, assign, updateAssignmentMetadataIfNeeded, pollForFetches, commitSync, commitAsync, committed, close, and updateFetchPositions
`fetcher`	Fetcher Created right when `KafkaConsumer` is created. Used when…FIXME
`interceptors`	`ConsumerInterceptors` that holds ConsumerInterceptor instances (defined using interceptor.classes setting). Used when…FIXME
`metadata`	Metadata Created for a KafkaConsumer with the following: • retryBackoffMs configuration property • ConsumerConfig.METADATA_MAX_AGE_CONFIG configuration property • `allowAutoTopicCreation` flag on • `topicExpiryEnabled` flag off `metadata` is requested to update itself immediately Used to create a NetworkClient (for the ConsumerNetworkClient), the ConsumerNetworkClient itself, the ConsumerCoordinator and Fetcher

	Used in…FIXME
`metrics`	`Metrics`
`retryBackoffMs`	retry.backoff.ms property or a user-defined value
`requestTimeoutMs`	Corresponds to request.timeout.ms property `KafkaConsumer` reports `ConfigException` when smaller or equal than session.timeout.ms and fetch.max.wait.ms properties.
`subscriptions`	`SubscriptionState` for auto.offset.reset setting. Created when `KafkaConsumer` is created

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 `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 For ConsumerRecords (per TopicPartitions) — `poll` Method

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

Note	`poll` is part of the Consumer Contract to poll for ConsumerRecords.

Figure 3. KafkaConsumer polls topics

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

```
scala> :type consumer
org.apache.kafka.clients.consumer.KafkaConsumer[String,String]

import java.time.Duration.ZERO
while (true) {
  println(s"Polling for records for $ZERO secs")
  val records = consumer.poll(ZERO)
  // do something with the records
  // e.g. print them out to the console
  records.asScala.foreach(println)
}
```

Note	`KafkaConsumer` has to be subscribed to topics or assigned partitions before calling poll.

```
scala> :type consumer
org.apache.kafka.clients.consumer.KafkaConsumer[String,String]

import java.time.Duration
scala> val records = consumer.poll(Duration.ZERO)
java.lang.IllegalStateException: Consumer is not subscribed to any topics or assigned
any partitions
  at org.apache.kafka.clients.consumer.KafkaConsumer.poll(KafkaConsumer.java:1171)
  at org.apache.kafka.clients.consumer.KafkaConsumer.poll(KafkaConsumer.java:1164)
  ... 36 elided
```

Internally, `poll` simply calls the internal poll method with the Time that expires after the given timeout and the `includeMetadataInTimeout` flag on.

`commitSync` Method

```
void commitSync()
```

Note	`commitSync` is part of Consumer Contract to…FIXME.

`commitSync` …FIXME

`seek` Method

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

Note	`seek` is part of Consumer Contract to…FIXME.

`seek` …FIXME

Getting Partitions For Topic — `partitionsFor` Method

Caution	FIXME

`endOffsets` Method

Caution	FIXME

`offsetsForTimes` Method

Caution	FIXME

`updateFetchPositions` Internal Method

```
boolean updateFetchPositions(final Timer timer)
```

`updateFetchPositions` …FIXME

Note	`updateFetchPositions` is used when…FIXME

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

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

`pollOnce` …FIXME

Note	`pollOnce` is used exclusively when `KafkaConsumer` is requested to poll

Requesting Metadata for All Topics (From Brokers) — `listTopics` Method

```
Map<String, List<PartitionInfo>> listTopics()
```

Internally, `listTopics` simply requests Fetcher for metadata for all topics and returns it.

```
consumer.listTopics().asScala.foreach { case (name, partitions) =>
  println(s"topic: $name (partitions: ${partitions.size()})")
}
```

Note	`listTopics` uses requestTimeoutMs that corresponds to request.timeout.ms property.

`beginningOffsets` Method

```
Map<TopicPartition, Long> beginningOffsets(Collection<TopicPartition> partitions)
```

`beginningOffsets` requests Fetcher for beginningOffsets and returns it.

Creating KafkaConsumer Instance

`KafkaConsumer` takes the following when created:

- Consumer configuration (that is converted internally to ConsumerConfig)
- Deserializer for keys
- Deserializer for values

`KafkaConsumer` initializes the internal registries and counters.

Note	`KafkaConsumer` API offers other constructors that in the end use the public 3-argument constructor that in turn passes the call on to the private internal constructor.

KafkaConsumer Public Constructor

```
// Public API
KafkaConsumer(
  Map<String, Object> configs,
  Deserializer<K> keyDeserializer,
  Deserializer<V> valueDeserializer)
```

When created, `KafkaConsumer` adds the keyDeserializer and valueDeserializer to configs (as key.deserializer and value.deserializer properties respectively) and creates a ConsumerConfig.

`KafkaConsumer` passes the call on to the internal constructor.

KafkaConsumer Internal Constructor

```
KafkaConsumer(
  ConsumerConfig config,
  Deserializer<K> keyDeserializer,
  Deserializer<V> valueDeserializer)
```

When called, the internal `KafkaConsumer` constructor prints out the following DEBUG message to the logs:

```
DEBUG Starting the Kafka consumer
```

`KafkaConsumer` sets the internal requestTimeoutMs to request.timeout.ms property.

`KafkaConsumer` sets the internal clientId to client.id or generates one with prefix **consumer-** (starting from 1) if not set.

`KafkaConsumer` sets the internal Metrics (and `JmxReporter` with **kafka.consumer** prefix).

`KafkaConsumer` sets the internal retryBackoffMs to retry.backoff.ms property.

Caution	FIXME Finish me!

`KafkaConsumer` creates the internal Metadata with the following arguments:

1. retryBackoffMs
2. metadata.max.age.ms
3. `allowAutoTopicCreation` enabled
4. `topicExpiryEnabled` disabled

5. ClusterResourceListeners with user-defined list of ConsumerInterceptors in interceptor.classes property

`KafkaConsumer` updates `metadata` with bootstrap.servers.

Caution	FIXME Finish me!

`KafkaConsumer` creates a NetworkClient with…FIXME

Caution	FIXME Finish me!

`KafkaConsumer` creates Fetcher with the following properties:

- fetch.min.bytes
- fetch.max.bytes
- fetch.max.wait.ms
- max.partition.fetch.bytes
- max.poll.records
- check.crcs

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

```
DEBUG Kafka consumer created
```

Any issues while creating a `KafkaConsumer` are reported as `KafkaException`.

```
org.apache.kafka.common.KafkaException: Failed to construct kafka consumer
```

`wakeup` Method

```
void wakeup()
```

Note	`wakeup` is a part of Consumer Contract.

`wakeup` simply requests ConsumerNetworkClient to wakeup.

Figure 4. KafkaConsumer's wakeup Method

Note	Quoting `wakeup` of Java's java.nio.channels.Selector given `wakeup` simply passes through the intermediaries and in the end triggers it. > Causes the first selection operation that has not yet returned to return immediately. Read about Selection in java.nio.channels.Selector's javadoc.

Note	`wakeup` is used when…FIXME

Configuring ClusterResourceListeners — `configureClusterResourceListeners` Internal Method

```
ClusterResourceListeners configureClusterResourceListeners(
  Deserializer<K> keyDeserializer,
  Deserializer<V> valueDeserializer,
  List<?>... candidateLists)
```

`configureClusterResourceListeners` creates a ClusterResourceListeners and registers `ClusterResourceListener` instances from the input `candidateLists`, `keyDeserializer` and `valueDeserializer`.

Note	`configureClusterResourceListeners` is used exclusively when `KafkaConsumer` is created (to create the Metadata) with the following input arguments: • key and value deserializers (defined when `KafkaConsumer` is created) • ConsumerInterceptors from interceptor.classes Kafka property • MetricsReporters from metric.reporters Kafka property

`throwIfNoAssignorsConfigured` Internal Method

```
void throwIfNoAssignorsConfigured()
```

`throwIfNoAssignorsConfigured` ...FIXME

Note	`throwIfNoAssignorsConfigured` is used exclusively when `KafkaConsumer` is requested to subscribe to topics.

`updateAssignmentMetadataIfNeeded` Internal Method

```
boolean updateAssignmentMetadataIfNeeded(
  Timer timer)
```

`updateAssignmentMetadataIfNeeded` requests the ConsumerCoordinator to poll until the `Timer` expires.

`updateAssignmentMetadataIfNeeded` returns `false` if the poll was unsuccessful, i.e. FIXME

If the poll was successful, `updateAssignmentMetadataIfNeeded` updateFetchPositions.

Note	`updateAssignmentMetadataIfNeeded` is used exclusively when `KafkaConsumer` is requested to poll for records.

`commitAsync` Method

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

Note	`commitAsync` is part of the Consumer Contract to…FIXME.

`commitAsync` ...FIXME

`committed` Method

```
OffsetAndMetadata committed(
  TopicPartition partition)
OffsetAndMetadata committed(
  TopicPartition partition,
  Duration timeout)
```

Note	`committed` is part of the Consumer Contract to...FIXME.

`committed` ...FIXME

`close` Method

```
void close()
void close(
  Duration timeout)
```

Note	`close` is part of the Consumer Contract to...FIXME.

`close` ...FIXME

`position` Method

```
long position(
  TopicPartition partition)
long position(
  TopicPartition partition,
  Duration timeout)
```

Note	`position` is part of the Consumer contract.

`position` ...FIXME

`close` Internal Method

```
void close(
  long timeoutMs,
  boolean swallowException)
```

`close` ...FIXME

`poll` Internal Method

```
ConsumerRecords<K, V> poll(
  Timer timer,
  boolean includeMetadataInTimeout)
```

`poll` first acquireAndEnsureOpen.

`poll` requests the ConsumerNetworkClient to maybeTriggerWakeup.

`poll` ...FIXME

`pollForFetches` Internal Method

```
Map<TopicPartition, List<ConsumerRecord<K, V>>> pollForFetches(
  Timer timer)
```

`pollForFetches` ...FIXME

Note	`pollForFetches` is used exclusively when `KafkaConsumer` is requested to poll for records.

MockConsumer

`MockConsumer` is a concrete Consumer to be used for unit testing.

Note	`MockConsumer` is used in Kafka Streams' `TopologyTestDriver` to write tests to verify the behavior of topologies.

`MockConsumer` takes a single `OffsetResetStrategy` to be created.

```
import org.apache.kafka.clients.consumer._
val consumer = new MockConsumer[String, String](OffsetResetStrategy.EARLIEST)
```

While being created, `MockConsumer` initializes the internal registries and counters.

Beside Consumer API, `MockConsumer` defines its own testing-specific API.

Table 1. MockConsumer API

Method	Description
`addEndOffsets`	`void addEndOffsets(Map<TopicPartition, Long> newOffsets)`
`addRecord`	`void addRecord(ConsumerRecord<K, V> record)`
`closed`	`boolean closed()`
`rebalance`	`void rebalance(Collection<TopicPartition> newAssignment)`
`scheduleNopPollTask`	`void scheduleNopPollTask()`
`schedulePollTask`	`void schedulePollTask(Runnable task)`
`setException`	`void setException(KafkaException exception)`
`updateBeginningOffsets`	`void updateBeginningOffsets(` `  Map<TopicPartition, Long> newOffsets)`
`updateEndOffsets`	`void updateEndOffsets(` `  Map<TopicPartition, Long> newOffsets)`

`updatePartitions`	`void updatePartitions(`
	`  String topic,`
	`  List<PartitionInfo> partitions)`

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

Name	Description
`subscriptions`	SubscriptionState Used when…FIXME

`partitions`	`Map<String, List<PartitionInfo>> partitions` Used when…FIXME
`records`	`Map<TopicPartition, List<ConsumerRecord<K, V>>> records` Used when…FIXME
`paused`	Kafka's `TopicPartition` paused Used when…FIXME
`closed`	Flag that says whether…FIXME Defaults to `false` Used when…FIXME
`beginningOffsets`	`Map<TopicPartition, Long> beginningOffsets` Used when…FIXME
`endOffsets`	`Map<TopicPartition, List<Long>> endOffsets` Used when…FIXME
`pollTasks`	`Queue<Runnable>` Used when…FIXME
`exception`	A `KafkaException` if thrown Used when…FIXME
`wakeup`	Flag that says whether…FIXME (a Java AtomicBoolean) Used when…FIXME
`committed`	`Map<TopicPartition, OffsetAndMetadata> committed` Used when…FIXME

ConsumerRecord

`ConsumerRecord` is...FIXME

OffsetAndMetadata

`OffsetAndMetadata` is...FIXME

OffsetAndTimestamp

`OffsetAndTimestamp` is...FIXME

OffsetCommitCallback

`OffsetCommitCallback` is...FIXME

ConsumerRebalanceListener

`ConsumerRebalanceListener` is a callback interface to be notified when the set of partitions assigned to the consumer (aka *assignment*) changes.

`ConsumerRebalanceListener` is notified about the newly-assigned partitions through onPartitionsAssigned callback that happens when `ConsumerCoordinator` is requested to onJoinComplete.

`ConsumerRebalanceListener` is notified about the partitions revoked through onPartitionsRevoked callback that happens when `ConsumerCoordinator` is requested to onJoinPrepare.

Caution	FIXME Picture when the notifications are triggered

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

interface ConsumerRebalanceListener {
  void onPartitionsAssigned(Collection<TopicPartition> partitions);
  void onPartitionsRevoked(Collection<TopicPartition> partitions);
}
```

`ConsumerRebalanceListener` is used when a Kafka `Consumer` is requested to subscribe to topics.

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

Table 1. ConsumerRebalanceListener Contract

Method	Description
`onPartitionsAssigned`	Used exclusively when `ConsumerCoordinator` is requested to onJoinComplete
`onPartitionsRevoked`	Used exclusively when `ConsumerCoordinator` is requested to onJoinPrepare

ConsumerConfig — Configuration Properties for KafkaConsumer

`ConsumerConfig` is a Apache Kafka AbstractConfig for the configuration properties of a KafkaConsumer.

```
import org.apache.kafka.clients.consumer.ConsumerConfig

val conf = new java.util.Properties()
conf.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, ":9092,localhost:9192")
import org.apache.kafka.common.serialization.Serdes
conf.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, Serdes.String.deserializer.getC
lass)
conf.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, Serdes.String.deserializer.ge
tClass)

import org.apache.kafka.clients.consumer.KafkaConsumer
val consumer = new KafkaConsumer[String, String](conf)
```

Table 1. ConsumerConfig's Configuration Properties

Name, Property, Default Value	Description
`BOOTSTRAP_SERVERS_CONFIG`	
`GROUP_ID_CONFIG`	

`METADATA_MAX_AGE_CONFIG`
`metadata.max.age.ms`
Default: 5 mins
	The period of time (in milliseconds) after which we force a refresh of metadata even if we haven't seen any partition leadership changes. That is to proactively discover any new brokers or partitions.

`PARTITION_ASSIGNMENT_STRATEGY_CONFIG`
`partition.assignment.strategy`
Default: RangeAssignor
	The class name of the partition assignment strategy that the client will use to distribute partition ownership amongst consumer instances when group management is used
	
	Tip: Read up Kafka Client-side Assignment Proposal on the group management in Apache Kafka's Consumer API.
	
	Used when:
	• `KafkaConsumer` is created
	• Kafka Streams' `StreamsConfig` is requested to `getMainConsumerConfigs`
	
	Note: Required for `KafkaConsumer` to subscribe to topics

`REQUEST_TIMEOUT_MS_CONFIG`
`request.timeout.ms`
Default: `30000` ms
	How long (in millis) a Kafka client waits for a response to a request.
	
	If the response is not received before the timeout elapses the client will resend the request if necessary or fail the request if retries are exhausted.
	
	Must be at least `0`

CommitFailedException

`CommitFailedException` is...FIXME

InvalidOffsetException

`InvalidOffsetException` is…FIXME

NoOffsetForPartitionException

`NoOffsetForPartitionException` is...FIXME

OffsetOutOfRangeException

`OffsetOutOfRangeException` is...FIXME

RetriableCommitFailedException

`RetriableCommitFailedException` is...FIXME

ConsumerInterceptor

Example

```
package pl.jaceklaskowski.kafka

import java.util

import org.apache.kafka.clients.consumer.{ConsumerInterceptor, ConsumerRecords, Offset
AndMetadata}
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

PartitionAssignor Contract

`PartitionAssignor` is the abstraction of partition assignors that are identified by name and can perform partition assignment.

Table 1. PartitionAssignor Contract

Method	Description

`assign`	`Map<String, Assignment> assign(`
	`  Cluster metadata,`
	`  Map<String, Subscription> subscriptions)`
	
	Assigns partitions to the members of a consumer group
	
	Used exclusively when `ConsumerCoordinator` is requested to perform partition assignment

`name`	`String name()` Used when `ConsumerCoordinator` is requested to metadata, lookupAssignor, and performAssignment (only for logging purposes)
`onAssignment`	`void onAssignment(Assignment assignment)` Used exclusively when `ConsumerCoordinator` is requested to onJoinComplete
`subscription`	`Subscription subscription(Set<String> topics)` Used exclusively when `ConsumerCoordinator` is requested to metadata

Table 2. PartitionAssignors (Direct Implementations)

PartitionAssignor	Description
AbstractPartitionAssignor	Extension for collecting partition counts
StreamsPartitionAssignor	Default `PartitionAssignor` in Kafka Streams

RangeAssignor

`RangeAssignor` is…FIXME

RoundRobinAssignor

`RoundRobinAssignor` is...FIXME

StickyAssignor

`StickyAssignor` is...FIXME

AbstractPartitionAssignor

`AbstractPartitionAssignor` is the extension of the PartitionAssignor contract for partition assignors that can perform group assignment given partition counts and member subscriptions.

Note	`AbstractPartitionAssignor` is a Java abstract class and cannot be created directly. It is created indirectly for the concrete PartitionAssignors.

Table 1. AbstractPartitionAssignor Contract

Method	Description

`assign`	`Map<String, List<TopicPartition>> assign(`
	`  Map<String, Integer> partitionsPerTopic,`
	`  Map<String, Subscription> subscriptions)`
	
	Performs group assignment given partition counts and member subscriptions
	
	Used exclusively when requested to assign (given cluster metadata and member subscriptions)

Table 2. AbstractPartitionAssignors

AbstractPartitionAssignor	Description
RangeAssignor	
RoundRobinAssignor	
StickyAssignor	

`assign` Method

```
Map<String, Assignment> assign(
  Cluster metadata,
  Map<String, Subscription> subscriptions)
```

Note	`assign` is part of the PartitionAssignor Contract to perform group assignment given cluster metadata and member subscriptions.

`assign` ...FIXME

ConsumerCoordinator

`ConsumerCoordinator` is a concrete AbstractCoordinator that…FIXME

`ConsumerCoordinator` is created exclusively when `KafkaConsumer` is created.

Figure 1. ConsumerCoordinator and KafkaConsumer

`KafkaConsumer` uses the following Consumer configuration properties to create a `ConsumerCoordinator`.

Table 1. ConsumerCoordinator's Consumer Configuration Properties

Configuration Property	ConsumerCoordinator's Property
`max.poll.interval.ms`	rebalanceTimeoutMs
`session.timeout.ms`	sessionTimeoutMs
`heartbeat.interval.ms`	heartbeatIntervalMs
`retry.backoff.ms`	retryBackoffMs
enable.auto.commit	autoCommitEnabled
`auto.commit.interval.ms`	autoCommitIntervalMs
`exclude.internal.topics`	excludeInternalTopics
`internal.leave.group.on.close`	leaveGroupOnClose

With autoCommitEnabled enabled (i.e. enable.auto.commit is `true`), `ConsumerCoordinator` does:

1. Sending asynchronous auto-commit of offsets

2. Sending synchronous auto-commit of offsets

`ConsumerCoordinator` uses zero or more PartitionAssignors for the following:

- metadata
- Looking up the PartitionAssignor by name (when onJoinComplete and performAssignment)

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

Name	Description
`assignmentSnapshot`	`MetadataSnapshot` Used in rejoinNeededOrPending • Set to the metadataSnapshot in performAssignment • Reset (to `null`) in onJoinComplete (when isLeader flag is off)
`completedOffsetCommits`	Java's java.util.concurrent.ConcurrentLinkedQueue of `OffsetCommitCompletions` with user-defined OffsetCommitCallbacks `OffsetCommitCallbacks` are executed in invokeCompletedOffsetCommitCallbacks `OffsetCommitCompletion` are added in commitOffsetsAsync (and also directly in doCommitOffsetsAsync)
`isLeader`	Flag that says whether the `ConsumerCoordinator` is a leader (`true`) or not (`false`). Default: `false` Turned on in performAssignment and off in onJoinPrepare Used exclusively for onJoinComplete (to reset the assignmentSnapshot when off)
`metadataSnapshot`	`MetadataSnapshot metadataSnapshot` `MetadataSnapshot` (for the SubscriptionState and the cluster metadata from the Metadata) • Updated in the `Metadata.Listener` from addMetadataListener Used in rejoinNeededOrPending and performAssignment

Tip	Enable `ERROR`, `WARN`, `INFO`, `DEBUG`, or `TRACE` logging level for `org.apache.kafka.clients.consumer.internals.ConsumerCoordinator` logger to see wha happens inside. Add the following line to `log4j.properties`: `log4j.logger.org.apache.kafka.clients.consumer.internals.ConsumerCoordinator=TRA` Refer to Logging.

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

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

`maybeAutoCommitOffsetsAsync` ...FIXME

Note	`maybeAutoCommitOffsetsAsync` is used when: • `KafkaConsumer` is requested to manually assign partitions • `ConsumerCoordinator` is requested to poll

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

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

`maybeAutoCommitOffsetsSync` ...FIXME

Note	`maybeAutoCommitOffsetsSync` is used when `ConsumerCoordinator` is requested to close or onJoinPrepare

`doAutoCommitOffsetsAsync` Internal Method

```
void doAutoCommitOffsetsAsync()
```

`doAutoCommitOffsetsAsync` ...FIXME

Note	`doAutoCommitOffsetsAsync` is used exclusively when `ConsumerCoordinator` is requested to maybeAutoCommitOffsetsAsync.

`close` Method

```
void close(long timeoutMs)
```

`close` ...FIXME

Note	`close` is used when...FIXME

`commitOffsetsAsync` Method

```
void commitOffsetsAsync(
  final Map<TopicPartition, OffsetAndMetadata> offsets,
  final OffsetCommitCallback callback)
```

`commitOffsetsAsync` ...FIXME

Note	`commitOffsetsAsync` is used when: • `KafkaConsumer` is requested to commitAsync • `ConsumerCoordinator` is requested to doAutoCommitOffsetsAsync

`commitOffsetsSync` Method

```
boolean commitOffsetsSync(
  Map<TopicPartition,
  OffsetAndMetadata> offsets,
  long timeoutMs)
```

`commitOffsetsSync` ...FIXME

Note	`commitOffsetsSync` is used when...FIXME

`refreshCommittedOffsetsIfNeeded` Method

```
void refreshCommittedOffsetsIfNeeded()
```

`refreshCommittedOffsetsIfNeeded` ...FIXME

Note	`refreshCommittedOffsetsIfNeeded` is used when...FIXME

`onJoinComplete` Callback

```
void onJoinComplete(
  int generation,
  String memberId,
  String assignmentStrategy,
  ByteBuffer assignmentBuffer)
```

Note	`onJoinComplete` is part of AbstractCoordinator Contract to...FIXME.

`onJoinComplete` ...FIXME

`onJoinPrepare` Method

```
void onJoinPrepare(int generation, String memberId)
```

Note	`onJoinPrepare` is part of AbstractCoordinator Contract to...FIXME.

`onJoinPrepare` ...FIXME

Performing Partition Assignment (using PartitionAssignor) — `performAssignment` Method

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

Note	`performAssignment` is part of AbstractCoordinator Contract to perform partition assignment (i.e. assign partitions to the members of a consumer group).

`performAssignment` tries to find the PartitionAssignor by the given `assignmentStrategy`.

`performAssignment` deserializes `Subscriptions` (from the given `allSubscriptions`) for every entry (and creates a `Map<String, Subscription>`).

`performAssignment` requests the SubscriptionState to add the topics to group subscription (i.e. add all the topics that the consumer group is subscribed to to receive topic metadata updates).

`performAssignment` requests the Metadata to setTopics to be the groupSubscription of the SubscriptionState.

`performAssignment` requests the ConsumerNetworkClient to update the cluster metadata.

`performAssignment` turns the isLeader internal flag on.

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

```
Performing assignment using strategy [assignor] with subscriptions [subscriptions]
```

`performAssignment` requests the PartitionAssignor to assign partitions to the members of the consumer group (with the current cluster metadata and subscriptions) and gets the partition assignment back (as `Map<String, Assignment>`).

Note	Here `performAssignment` does some customizations that a user-customized assignor could request. It is not a very interesting code to spend your time on.

`performAssignment` sets the assignmentSnapshot to be metadataSnapshot.

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

```
Finished assignment for group: [assignment]
```

In the end, `performAssignment` takes the partition assignment, encodes the `Assignment` per group member to create a `groupAssignment` (i.e. the `Map<String, ByteBuffer>` that is returned).

`performAssignment` throws an `IllegalStateException` when no PartitionAssignor could be found:

```
Coordinator selected invalid assignment protocol: [assignmentStrategy]
```

`maybeLeaveGroup` Method

```
void maybeLeaveGroup()
```

`maybeLeaveGroup` ...FIXME

Note	`maybeLeaveGroup` is used when…FIXME

`updatePatternSubscription` Method

```
void updatePatternSubscription(Cluster cluster)
```

`updatePatternSubscription` …FIXME

Note	`updatePatternSubscription` is used when…FIXME

`needRejoin` Method

```
boolean needRejoin()
```

Note	`needRejoin` is part of the AbstractCoordinator Contract to…FIXME.

`needRejoin` …FIXME

`timeToNextPoll` Method

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

`timeToNextPoll` …FIXME

Note	`timeToNextPoll` is used when…FIXME

Polling for Group Coordinator Events — `poll` Method

```
boolean poll(Timer timer)
```

`poll` first invokeCompletedOffsetCommitCallbacks.

`poll` branches off per whether the SubscriptionState is partitionsAutoAssigned or not.

Caution	FIXME What does `partitionsAutoAssigned` mean exactly?

In partitionsAutoAssigned, `poll` pollHeartbeat.

`poll` returns `false` if coordinatorUnknown and ensureCoordinatorReady failed (`false`).

`poll` ...FIXME

Note	`poll` is used exclusively when `KafkaConsumer` is requested to updateAssignmentMetadataIfNeeded.

Registering Metadata.Listener — `addMetadataListener` Internal Method

```
void addMetadataListener()
```

`addMetadataListener` requests the Metadata to add a new Metadata Update Listener that intercepts onMetadataUpdate events and does the following:

- FIXME

`addMetadataListener` throws a `TopicAuthorizationException` for any unauthorized topics (i.e. when the given `Cluster` has at least one topic in unauthorizedTopics).

```
FIXME
```

Note	`addMetadataListener` is used exclusively when `ConsumerCoordinator` is created.

`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`
- PartitionAssignors
- Metadata
- SubscriptionState
- Metrics
- Prefix of the metric group
- `Time`
- `retryBackoffMs`
- `autoCommitEnabled` flag
- `autoCommitIntervalMs`
- ConsumerInterceptors
- `excludeInternalTopics` flag
- `leaveGroupOnClose` flag

`ConsumerCoordinator` initializes the internal registries and counters.

In the end, `ConsumerCoordinator` requests the Metadata to update and addMetadataListener.

`rejoinNeededOrPending` Method

```
boolean rejoinNeededOrPending()
```

Note	`rejoinNeededOrPending` is part of the AbstractCoordinator Contract to…FIXME.

`rejoinNeededOrPending` …FIXME

Sending OffsetCommitRequest to Group Coordinator (Kafka Broker) — `sendOffsetCommitRequest` Internal Method

```
RequestFuture<Void> sendOffsetCommitRequest(
  final Map<TopicPartition, OffsetAndMetadata> offsets)
```

`sendOffsetCommitRequest` gets the node of the consumer group.

`sendOffsetCommitRequest` creates a OffsetCommitRequest.Builder for the group ID and an offset data based on the given offsets.

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

```
Sending OffsetCommit request with [offsets] to coordinator [coordinator]
```

`sendOffsetCommitRequest` requests the ConsumerNetworkClient to send an `OffsetCommitRequest` to the group coordinator and then creates a new OffsetCommitResponseHandler to handle a response.

`sendOffsetCommitRequest` returns immediately when there is no offsets to send.

`sendOffsetCommitRequest` returns immediately with a `COORDINATOR_NOT_AVAILABLE` failure when the consumer group coordinator is not available.

Note	`sendOffsetCommitRequest` is used when `ConsumerCoordinator` is requested to doCommitOffsetsAsync and commitOffsetsSync.

`sendOffsetFetchRequest` Internal Method

```
RequestFuture<Map<TopicPartition, OffsetAndMetadata>> sendOffsetFetchRequest(
  Set<TopicPartition> partitions)
```

`sendOffsetFetchRequest` …FIXME

Note	`sendOffsetFetchRequest` is used when…FIXME

`invokeCompletedOffsetCommitCallbacks` Internal Method

```
void invokeCompletedOffsetCommitCallbacks()
```

`invokeCompletedOffsetCommitCallbacks` takes (*polls*) every `OffsetCommitCompletion` from the completedOffsetCommits internal registry and requests it to invoke OffsetCommitCallback.onComplete.

Note	`invokeCompletedOffsetCommitCallbacks` is used when `ConsumerCoordinator` is requested to poll, close, commitOffsetsAsync, and commitOffsetsSync.

Finding PartitionAssignor by Name — `lookupAssignor` Internal Method

```
PartitionAssignor lookupAssignor(String name)
```

`lookupAssignor` tries to find the PartitionAssignor by the given `name` in the assignors.

If not found, `lookupAssignor` returns `null`.

Note	`lookupAssignor` is used when `ConsumerCoordinator` is requested to onJoinComplete and performAssignment.

`doCommitOffsetsAsync` Internal Method

```
void doCommitOffsetsAsync(
  final Map<TopicPartition, OffsetAndMetadata> offsets,
  final OffsetCommitCallback callback)
```

`doCommitOffsetsAsync` ...FIXME

Note	`doCommitOffsetsAsync` is used exclusively when `ConsumerCoordinator` is requested to commitOffsetsAsync.

AbstractCoordinator Contract

`AbstractCoordinator` is the base of Coordinators that FIXME.

Note	`AbstractCoordinator` is a Java abstract class and cannot be created directly. It is created indirectly for the concrete Coordinators.

Note	ConsumerCoordinator is the one and only known implementation of the AbstractCoordinator Contract in Apache Kafka's Consumer API.

Table 1. AbstractCoordinator Contract (Abstract Methods Only)

Method	Description
`metadata`	`List<ProtocolMetadata> metadata()` Used exclusively when `AbstractCoordinator` is requested to send a JoinGroupRequest to the group coordinator
`onJoinComplete`	`void onJoinComplete(int generation, String memberId, String protocol, ByteBuffer memberAssignment)` Used exclusively when `AbstractCoordinator` is requested to joinGroupIfNeeded (and the request to initiateJoinGroup succeeded)
`onJoinPrepare`	`void onJoinPrepare(int generation, String memberId)` Used exclusively when `AbstractCoordinator` is requested to joinGroupIfNeeded (and the needsJoinPrepare flag is on)
`performAssignment`	`Map<String, ByteBuffer> performAssignment(String leaderId, String protocol, Map<String, ByteBuffer> allMemberMetadata)` Performs partition assignment (i.e. assigns partitions to the members of a consumer group) Used exclusively when `AbstractCoordinator` is requested to perform partition assignment and notify the group coordinator
`protocolType`	`String protocolType()` Used exclusively when `AbstractCoordinator` is requested to send a JoinGroupRequest to the group coordinator

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

Name	Description
	Group Coordinator (as a Kafka Node)

`coordinator`	Initialized when `FindCoordinatorResponseHandler` is requested to onSuccess
`findCoordinatorFuture`	RequestFuture<Void> for sendFindCoordinatorRequest when lookupCoordinator Reset in clearFindCoordinatorFuture Used in lookupCoordinator and when `HeartbeatThread` is requested to run
`generation`	Defaults to `NO_GENERATION`
`heartbeatThread`	HeartbeatThread Initialized and immediately started when `AbstractCoordinator` is requested to start a HeartbeatThread
`joinFuture`	RequestFuture<ByteBuffer> for sending a JoinGroupRequest to the group coordinator when initiateJoinGroup Reset in resetJoinGroupFuture Used in rejoinNeededOrPending and initiateJoinGroup
`needsJoinPrepare`	Flag to control whether to execute onJoinPrepare while performing joinGroupIfNeeded
`rejoinNeeded`	`boolean rejoinNeeded` Flag to control whether to rejoinNeededOrPending. Enabled (`true`) by default. Turned off (`false`) when the joinFuture completed successfully in initiateJoinGroup Turned on (`true`) in resetGeneration and requestRejoin
`sensors`	GroupCoordinatorMetrics
`state`	State of the member of a consumer group Possible states: • `UNJOINED` (when not joined in a consumer group when attempting to join a consumer group or resetGeneration) • `REBALANCING` (when attempting to join a consumer group)

	• `STABLE` (when successfully joined a consumer group) Default: `UNJOINED`

Tip	Enable `WARN`, `INFO`, or `DEBUG` logging level for `org.apache.kafka.clients.consumer.internals.AbstractCoordinator` logger to see wha happens inside. Add the following line to `log4j.properties`: `log4j.logger.org.apache.kafka.clients.consumer.internals.AbstractCoordinator=DEB` Refer to Logging.

Creating AbstractCoordinator Instance

`AbstractCoordinator` takes the following when created:

- `LogContext`
- ConsumerNetworkClient
- Group ID
- `rebalanceTimeoutMs`
- `sessionTimeoutMs`
- `heartbeatIntervalMs` (or simply a `Heartbeat` instance)
- Metrics
- Metric group prefix
- `Time`
- `retryBackoffMs`
- `leaveGroupOnClose` flag

Note	`AbstractCoordinator` is a Java abstract class and cannot be created directly. It is created indirectly for the concrete Coordinators.

`joinGroupIfNeeded` Method

```
boolean joinGroupIfNeeded(
  Timer timer)
```

`joinGroupIfNeeded` ...FIXME

Note	`joinGroupIfNeeded` is used exclusively when `AbstractCoordinator` is requested to ensureActiveGroup.

Attempting to Join Consumer Group — `initiateJoinGroup` Internal Method

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

`initiateJoinGroup` disables the HeartbeatThread.

`initiateJoinGroup` sets the state as `REBALANCING`.

`initiateJoinGroup` sends a JoinGroupRequest to the group coordinator (and initializes the joinFuture).

`initiateJoinGroup` registers a RequestFutureListener to handle a response.

When the response is successful, `initiateJoinGroup` does the following:

1. Prints out the following INFO message to the logs:

   ```
   Successfully joined group with generation [generationId]
   ```

2. Sets the state as `STABLE` (and the rejoinNeeded flag off)
3. Requests the HeartbeatThread to enable itself.

When the response is a failure, `initiateJoinGroup` simply sets the state as `UNJOINED`

`initiateJoinGroup` exits immediately when the joinFuture has already been initialized (is not `null`) and gives it back.

Note	`initiateJoinGroup` is used exclusively when `AbstractCoordinator` is requested to joinGroupIfNeeded.

`ensureActiveGroup` Method

```
void ensureActiveGroup() (1)
boolean ensureActiveGroup(final Timer timer)
```

1. Calls the other `ensureActiveGroup` in an infinite `while` loop with an unexpiring timer

`ensureActiveGroup` ...FIXME

Note	`ensureActiveGroup` is used exclusively when `ConsumerCoordinator` is requested to poll for coordinator events.

Discovering Current Coordinator for Consumer Group — `lookupCoordinator` Method

```
RequestFuture<Void> lookupCoordinator()
```

`lookupCoordinator` uses the RequestFuture internal registry to know whether it was requested earlier but have not finished yet. In other words, the RequestFuture is available when looking up a coordinator of a consumer group is in progress.

If the RequestFuture internal registry is available (not a `null`), `lookupCoordinator` returns it immediately.

Otherwise, when the RequestFuture internal registry is uninitialized (`null`), `lookupCoordinator` requests the ConsumerNetworkClient for the least-loaded Kafka broker. `lookupCoordinator` then sends a FindCoordinatorRequest to the Kafka broker.

`lookupCoordinator` prints out the following DEBUG message to the logs when no nodes are available and finishes with a `RequestFuture.noBrokersAvailable` failure.

```
No broker available to send FindCoordinator request
```

Note	`lookupCoordinator` is used when: • `AbstractCoordinator` is requested to ensureCoordinatorReady • `ConsumerCoordinator` is requested to commitOffsetsAsync • `HeartbeatThread` is requested to run

`ensureCoordinatorReady` Method

```
boolean ensureCoordinatorReady(
  Timer timer)
```

`ensureCoordinatorReady` ...FIXME

Note	`ensureCoordinatorReady` is used when: • `AbstractCoordinator` is requested to ensureActiveGroup and joinGroupIfNeeded • `ConsumerCoordinator` is requested to poll, fetchCommittedOffsets, close, and commitOffsetsSync

Sending JoinGroupRequest to Group Coordinator (Kafka Broker) — `sendJoinGroupRequest` Method

```
RequestFuture<ByteBuffer> sendJoinGroupRequest()
```

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

```
(Re-)joining group
```

`sendJoinGroupRequest` creates a new JoinGroupRequest.Builder (for the groupId, the sessionTimeoutMs, the member ID, **consumer** protocol type, the supported ProtocolMetadata, and the rebalanceTimeoutMs).

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

```
Sending JoinGroup ([requestBuilder]) to coordinator [coordinator]
```

In the end, `sendJoinGroupRequest` requests the ConsumerNetworkClient to send the `JoinGroupRequest` to the group coordinator node and then creates a new JoinGroupResponseHandler to handle a response.

`sendJoinGroupRequest` returns immediately with a `RequestFuture.coordinatorNotAvailable()` when the coordinator has not been discovered yet.

Note	`sendJoinGroupRequest` is used exclusively when `AbstractCoordinator` is requested to initiateJoinGroup.

Sending SyncGroupRequest (to Kafka Broker) — `sendSyncGroupRequest` Internal Method

```
RequestFuture<ByteBuffer> sendSyncGroupRequest(
  SyncGroupRequest.Builder requestBuilder)
```

`sendSyncGroupRequest` ...FIXME

Note	`sendSyncGroupRequest` is used when...FIXME

Sending FindCoordinatorRequest (to Kafka Broker) — `sendFindCoordinatorRequest` Internal Method

```
RequestFuture<Void> sendFindCoordinatorRequest(Node node)
```

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

```
Sending FindCoordinator request to broker [node]
```

`sendFindCoordinatorRequest` creates a new FindCoordinatorRequest with `GROUP` type and the groupId.

In the end, `sendFindCoordinatorRequest` requests the ConsumerNetworkClient to send the `FindCoordinatorRequest` (to the given Kafka node) and then creates a new FindCoordinatorResponseHandler to handle a response.

Note	`sendFindCoordinatorRequest` is used exclusively when `AbstractCoordinator` is requested to lookupCoordinator.

`maybeLeaveGroup` Method

```
void maybeLeaveGroup()
```

`maybeLeaveGroup` ...FIXME

Note	`maybeLeaveGroup` is used when...FIXME

Sending HeartbeatRequest to Group Coordinator (Kafka Broker) — `sendHeartbeatRequest` Method

```
RequestFuture<Void> sendHeartbeatRequest()
```

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

```
Sending Heartbeat request to coordinator [coordinator]
```

`sendHeartbeatRequest` creates a new HeartbeatRequest.Builder (for the groupId, generation and member IDs).

In the end, `sendHeartbeatRequest` requests the ConsumerNetworkClient to send the `HeartbeatRequest` to the group coordinator node and then creates a new HeartbeatResponseHandler to handle a response.

Note	`sendHeartbeatRequest` is used exclusively when `HeartbeatThread` is requested to run.

Performing Partition Assignment and Notifying Group Coordinator — `onJoinLeader` Internal Method

```
RequestFuture<ByteBuffer> onJoinLeader(JoinGroupResponse joinResponse)
```

`onJoinLeader` performs partition assignment with the leader ID, the group protocol, and the member metadata (all from the given JoinGroupResponse).

`onJoinLeader` creates a new SyncGroupRequest.Builder (with the group, generation, and member IDs as well as the group partition assignment).

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

```
Sending leader SyncGroup to coordinator [coordinator]: [requestBuilder]
```

In the end, `onJoinLeader` sends a SyncGroupRequest (to the group coordinator node).

Note	`onJoinLeader` is used exclusively when `JoinGroupResponseHandler` is requested to handle a successful response from the group coordinator (and the consumer is the leader of the consumer group).

`resetJoinGroupFuture` Internal Method

```
void resetJoinGroupFuture()
```

`resetJoinGroupFuture` simply resets the joinFuture internal registry (i.e. sets it to `null`).

Note	`resetJoinGroupFuture` is used exclusively when `AbstractCoordinator` is requested to joinGroupIfNeeded.

Clearing FindCoordinatorFuture — `clearFindCoordinatorFuture` Internal Method

```
void clearFindCoordinatorFuture()
```

`clearFindCoordinatorFuture` simply resets the findCoordinatorFuture internal registry (to be `null`).

Note	`clearFindCoordinatorFuture` is used exclusively when `FindCoordinatorResponseHandler` is requested to onSuccess and onFailure.

`rejoinNeededOrPending` Internal Method

```
boolean rejoinNeededOrPending()
```

`rejoinNeededOrPending` is positive (`true`) when rejoinNeeded and joinFuture is initialized (i.e. not `null`).

Note	`rejoinNeededOrPending` is used when: • `ConsumerCoordinator` is requested to rejoinNeededOrPending and poll • `KafkaConsumer` is requested to pollForFetches • `AbstractCoordinator` is requested to joinGroupIfNeeded

`pollHeartbeat` Method

```
void pollHeartbeat(long now)
```

`pollHeartbeat` ...FIXME

Note	`pollHeartbeat` is used exclusively when `ConsumerCoordinator` is requested to poll for Coordinator events.

Starting HeartbeatThread — `startHeartbeatThreadIfNeeded` Internal Method

```
void startHeartbeatThreadIfNeeded()
```

`startHeartbeatThreadIfNeeded` ...FIXME

Note	`startHeartbeatThreadIfNeeded` is used exclusively when `AbstractCoordinator` is requested to ensureActiveGroup.

`needRejoin` Method

```
boolean needRejoin()
```

`needRejoin` simply returns the rejoinNeeded flag.

Note	`needRejoin` is used when...FIXME

`requestRejoin` Method

```
void requestRejoin()
```

`requestRejoin` simply turns the rejoinNeeded flag on.

Note	`requestRejoin` is used when: • `HeartbeatResponseHandler` is requested to handle a response that the consumer group is rebalancing • `SyncGroupResponseHandler` is requested to handle a response with an error

`resetGeneration` Method

```
void resetGeneration()
```

`resetGeneration` simply resets the following internal registries:

- generation becomes `NO_GENERATION`
- rejoinNeeded is turned on

- state is `UNJOINED`

Note	`resetGeneration` is used when: • `AbstractCoordinator` is requested to maybeLeaveGroup • HeartbeatResponseHandler, JoinGroupResponseHandler, SyncGroupResponseHandler, and OffsetCommitResponseHandler are requested to handle a response with an error (`UNKNOWN_MEMBER_ID` , `ILLEGAL_GENERATION` , or `REBALANCE_IN_PROGRESS`)

`coordinatorUnknown` Method

```
boolean coordinatorUnknown()
```

`coordinatorUnknown` ...FIXME

Note	`coordinatorUnknown` is used when...FIXME

`disableHeartbeatThread` Internal Method

```
void disableHeartbeatThread()
```

`disableHeartbeatThread` simply requests the HeartbeatThread to disable (if available, i.e. not `null`).

Note	`disableHeartbeatThread` is used exclusively when `AbstractCoordinator` is requested to initiateJoinGroup.

`checkAndGetCoordinator` Method

```
Node checkAndGetCoordinator()
```

`checkAndGetCoordinator` ...FIXME

Note	`checkAndGetCoordinator` is used when...FIXME

HeartbeatThread

`HeartbeatThread` is a daemon Kafka thread of execution that `AbstractCoordinator` starts to send HeartbeatRequests to the group coordinator (aka *heartbeating*) regularly every request.timeout.ms (default: `30000` ms).

`HeartbeatThread` is created and immediately started exclusively for `AbstractCoordinator` when requested to start a HeartbeatThread (when `ConsumerCoordinator` is requested to poll for coordinator events after `KafkaConsumer` was requested to poll for records).

`HeartbeatThread` uses **kafka-coordinator-heartbeat-thread** prefix followed by the group ID as the thread name.

```
kafka-coordinator-heartbeat-thread | [groupId]
```

`HeartbeatThread` uses `closed` flag to indicate…FIXME

`HeartbeatThread` uses `enabled` flag to indicate whether to wait for AbstractCoordinator to notify it (*release*). The flag is off (`false`) initially and can be turned on (`true`) and off by enable or disable, respectively.

`HeartbeatThread` takes no arguments to be created.

Starting Thread — `run` Method

```
void run()
```

Note	`run` is part of the java.lang.Runnable to start itself as a separately-executing thread.

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

```
Heartbeat thread started
```

In an infinite `while` loop, `run` does the following (infinitely):

1. Checks the internal state to exit or wait:
    i. Exits when closed
    ii. Waits for `AbstractCoordinator` to wake it up when not enabled

2. Requests the ConsumerNetworkClient to pollNoWakeup
3. Sends a HeartbeatRequest to the group coordinator
4. (when the response eventually arrives) Requests the Heartbeat to record a success (`receiveHeartbeat`) or a failure (`failHeartbeat`)

In the end, when the infinite `while` loop is terminated for any reason, `run` prints out the following DEBUG message to the logs:

```
Heartbeat thread has closed
```

Warning	`run` does some other state control checks but they look uninteresting…to me…now.

Disabling HeartbeatThread — `disable` Method

```
void disable()
```

`disable` prints out the following DEBUG message to the logs and turns the enabled flag off (`false`).

```
Disabling heartbeat thread
```

Note	`disable` is used when: • `AbstractCoordinator` is requested to disableHeartbeatThread • `HeartbeatThread` is requested to run (when in `UNJOINED` or `REBALANCING` states)

Enabling HeartbeatThread — `enable` Method

```
void enable()
```

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

```
Enabling heartbeat thread
```

`enable` turns the enabled flag on (`true`) and requests the Heartbeat to `resetTimeouts` .

In the end, `enable` notifies (*wakes up*) the owning AbstractCoordinator.

Note	`enable` is used exclusively when `AbstractCoordinator` is requested to initiateJoinGroup.

GroupCoordinatorMetrics

`GroupCoordinatorMetrics` is a set of metrics to measure performance of AbstractCoordinator.

Table 1. GroupCoordinatorMetrics's Sensors and Metrics

Sensor Name	Metric Name	Description
`heartbeat-latency`	`heartbeat-rate`	The number of heartbeats per second
`heartbeat-latency`	`heartbeat-response-time-max`	The max time taken to receive a response to a heartbeat request
`heartbeat-latency`	`heartbeat-total`	The total number of heartbeats
`join-latency`	`join-rate`	The number of group joins per second
`join-latency`	`join-time-avg`	The average time taken for a group rejoin
`join-latency`	`join-time-max`	The max time taken for a group rejoin
`join-latency`	`join-total`	The total number of group joins
	`last-heartbeat-seconds-ago`	The number of seconds since the last coordinator heartbeat was sent
`sync-latency`	`sync-rate`	The number of group syncs per second
`sync-latency`	`sync-time-avg`	The average time taken for a group sync
`sync-latency`	`sync-time-max`	The max time taken for a group sync
`sync-latency`	`sync-total`	The total number of group syncs

`GroupCoordinatorMetrics` is registered in **consumer-coordinator-metrics** group.

Figure 1. GroupCoordinatorMetrics in jconsole

`GroupCoordinatorMetrics` is created exclusively for AbstractCoordinator.

`GroupCoordinatorMetrics` takes the following to be created:

- Metrics
- Prefix of the metric group name (i.e. **consumer**)

`GroupCoordinatorMetrics` uses the prefix followed by `-coordinator-metrics` for the **metric group name** (e.g. `consumer-coordinator-metrics`).

```
[prefix]-coordinator-metrics
```

ConsumerNetworkClient

`ConsumerNetworkClient` is a high-level Kafka consumer that…FIXME

`ConsumerNetworkClient` is created for KafkaConsumer and AdminClient.

Tip	Enable `ALL` logging level for `org.apache.kafka.clients.consumer.internals.ConsumerNetworkClient` logger to see w happens inside. Add the following line to `log4j.properties`: `log4j.logger.org.apache.kafka.clients.consumer.internals.ConsumerNetworkClient=A` Refer to Logging.

`checkDisconnects` Internal Method

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

`checkDisconnects` …FIXME

Note	`checkDisconnects` is used exclusively when `ConsumerNetworkClient` is requested to poll.

"Sending" Request to Broker — `send` Method

```
RequestFuture<ClientResponse> send(Node node, AbstractRequest.Builder<?> requestBuilde
r)
```

`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	`send` 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	`wakeup` is used when…FIXME

Updating Cluster Metadata (and Possibly Blocking Until It Refreshes) — `ensureFreshMetadata` Method

```
void ensureFreshMetadata()
```

`ensureFreshMetadata` waits for metadata update when Metadata was requested for update or time to the next update is now.

Note	`ensureFreshMetadata` is used when `ConsumerCoordinator` does onJoinComplete, performAssignment and poll.

`pendingRequestCount` Method

```
int pendingRequestCount()
int pendingRequestCount(Node node)
```

`pendingRequestCount` …FIXME

Note	`pendingRequestCount` is used when…FIXME

Least-Loaded Kafka Node — `leastLoadedNode` Method

```
Node leastLoadedNode()
```

`leastLoadedNode` …FIXME

Note	`leastLoadedNode` is used when: • `AbstractCoordinator` is requested to lookupCoordinator • `Fetcher` is requested to sendMetadataRequest

`poll` Method

```
void poll(
  RequestFuture<?> future) (1)
boolean poll(
  RequestFuture<?> future,
  Timer timer)
void poll(
  Timer timer) (2)
void poll(
  Timer timer,
  PollCondition pollCondition) (3)
void poll(
  Timer timer,
  PollCondition pollCondition,
  boolean disableWakeup)
```

1. Used in tests (and uses a maximum timeout)
2. Uses uninitialized `PollCondition` (`null`)
3. Turns `disableWakeup` off (`false`)

`poll` …FIXME

Note	`poll` is used when: • `AbstractCoordinator` is requested to ensureCoordinatorReady and joinGroupIfNeeded • `ConsumerNetworkClient` is requested to pollNoWakeup, awaitMetadataUpdate, and awaitPendingRequests • `Fetcher` is requested to getTopicMetadata and fetchOffsetsByTimes • `KafkaConsumer` is requested to pollForFetches and position • `ConsumerCoordinator` is requested to close, fetchCommittedOffsets and commitOffsetsSync • `BrokerApiVersionsCommand` tool is executed (and creates a AdminClient) • Kafka Connect's `WorkerCoordinator` is requested to `poll`

Waiting for Metadata Update — `awaitMetadataUpdate` Method

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

`awaitMetadataUpdate` ...FIXME

Note	`awaitMetadataUpdate` is used when...FIXME

`awaitPendingRequests` Method

```
boolean awaitPendingRequests(Node node, Timer timer)
```

`awaitPendingRequests` ...FIXME

Note	`awaitPendingRequests` is used when...FIXME

`pollNoWakeup` Method

```
void pollNoWakeup()
```

`pollNoWakeup` ...FIXME

Note	`pollNoWakeup` is used when: • `KafkaConsumer` is requested to poll for records • `AbstractCoordinator` is requested to maybeLeaveGroup • `HeartbeatThread` is requested to run • `ConsumerCoordinator` is requested to commitOffsetsAsync

Creating ConsumerNetworkClient Instance

`ConsumerNetworkClient` takes the following when created:

- `LogContext`
- KafkaClient
- Metadata

- `Time`
- `retryBackoffMs`
- `requestTimeoutMs`

`ConsumerNetworkClient` initializes the internal registries and counters.

`trySend` Internal Method

```
long trySend(long now)
```

`trySend` ...FIXME

Note	`trySend` is used exclusively when `ConsumerNetworkClient` is requested to poll.

Initiating Connector to Kafka Node — `tryConnect` Method

```
void tryConnect(Node node)
```

`tryConnect` simply requests the KafkaClient to initiate a connection to the given broker Node.

Note	`tryConnect` is used exclusively when `FindCoordinatorResponseHandler` is requested to onSuccess.

`handlePendingDisconnects` Internal Method

```
void handlePendingDisconnects()
```

`handlePendingDisconnects` ...FIXME

Note	`handlePendingDisconnects` is used exclusively when `ConsumerNetworkClient` is requested to poll.

`maybeTriggerWakeup` Method

```
void maybeTriggerWakeup()
```

`maybeTriggerWakeup` ...FIXME

Note	`maybeTriggerWakeup` is used when...FIXME

`transmitSends` Method

```
void transmitSends()
```

`transmitSends` ...FIXME

Note	`transmitSends` is used when `KafkaConsumer` is requested to poll.

Internal Properties

Name	Description
`pendingCompletion`	
`unsent`	`UnsentRequests` with pending requests per node that have not been sent yet (i.e. awaiting transmission).

ConsumerMetrics

`ConsumerMetrics` is...FIXME

Fetcher

`Fetcher` is created exclusively when `KafkaConsumer` is created.

Figure 1. Fetcher and KafkaConsumer

`KafkaConsumer` uses the following configuration properties to create a Fetcher:

- fetch.min.bytes for minimum number of bytes
- fetch.max.bytes for maximum number of bytes
- fetch.max.wait.ms for maximum wait time
- max.partition.fetch.bytes for fetch size
- max.poll.records for how many records to poll
- check.crcs for flag to check CRC or not
- retry.backoff.ms for retry backoff (in milliseconds)
- request.timeout.ms for request timeout (in milliseconds)

Table 1. Fetcher's Internal Properties (e.g. Registries and Counters) (in alphabetical order)

Name	Description
`client`	ConsumerNetworkClient that is given when `Fetcher` is created.

Creating Fetcher Instance

`Fetcher` takes the following when created:

- ConsumerNetworkClient
- Minimum number of bytes
- Maximum number of bytes
- Maximum wait time
- Fetch size
- How many records to poll

- Flag to check CRC or not
- Deserializer for keys
- Deserializer for values
- Metadata
- `SubscriptionState`
- `Metrics`
- `FetcherMetricsRegistry`
- `Time`
- Retry backoff (in milliseconds)
- Request timeout (in milliseconds)
- `IsolationLevel`

`Fetcher` initializes the internal registries and counters.

`Fetcher` registers itself with SubscriptionState as a listener to receive notifications about… FIXME

`sendFetches` Method

```
int sendFetches()
```

`sendFetches` …FIXME

Note	`sendFetches` is used when `KafkaConsumer` is requested to poll for records.

`sendMetadataRequest` Internal Method

```
RequestFuture<ClientResponse> sendMetadataRequest(MetadataRequest.Builder request)
```

Internally, `sendMetadataRequest` requests ConsumerNetworkClient for the least loaded node.

With the node, `sendMetadataRequest` requests ConsumerNetworkClient to send the request to the node.

When no node was found, `sendMetadataRequest` returns a `RequestFuture` with `NoAvailableBrokersException`.

Note	`sendMetadataRequest` is used exclusively when `Fetcher` is requested for a topic metadata.

`beginningOffsets` Method

Caution	FIXME

`retrieveOffsetsByTimes` Method

Caution	FIXME

Getting Topic Metadata — `getTopicMetadata` Method

```
Map<String, List<PartitionInfo>> getTopicMetadata(
  MetadataRequest.Builder request,
  long timeout)
```

Internally, `getTopicMetadata` sends the metadata request and requests ConsumerNetworkClient to poll until it finishes successfully or `timeout` expires.

After `poll` finishes, `getTopicMetadata` takes the cluster information from `MetadataResponse`.

When `MetadataResponse` is successful, `getTopicMetadata` takes topics (from `Cluster`) and requests `Cluster` for available partitions for every topic.

In the end, `getTopicMetadata` creates a collection of topic and partitions pairs.

Caution	FIXME Describe the failure path

Note	`getTopicMetadata` is used when: • `Fetcher` is requested to find metadata for all topics • `KafkaConsumer` is requested to find partitions for a topic

Finding Metadata for All Topics — `getAllTopicMetadata` Method

```
Map<String, List<PartitionInfo>> getAllTopicMetadata(long timeout)
```

`getAllTopicMetadata` gets topic metadata specifying no topics (which means all topics available).

Note	`getAllTopicMetadata` is used exclusively when `KafkaConsumer` requests metadata for all topics.

`sendListOffsetRequest` Internal Method

```
RequestFuture<ListOffsetResult> sendListOffsetRequest(
  final Node node,
  final Map<TopicPartition, ListOffsetRequest.PartitionData> timestampsToSearch,
  boolean requireTimestamp)
```

`sendListOffsetRequest` ...FIXME

Note	`sendListOffsetRequest` is used when...FIXME

`fetchOffsetsByTimes` Internal Method

```
ListOffsetResult fetchOffsetsByTimes(
  Map<TopicPartition, Long> timestampsToSearch,
  Timer timer,
  boolean requireTimestamps)
```

`fetchOffsetsByTimes` ...FIXME

Note	`fetchOffsetsByTimes` is used when...FIXME

RequestFutureListener

`RequestFutureListener` is...FIXME

SubscriptionState

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

`SubscriptionState` is in **partitionsAutoAssigned** if it is `AUTO_TOPICS` or `AUTO_PATTERN`.

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

Name	Description
`listener`	ConsumerRebalanceListener for notifications about partition assignment changes Used exclusively when `ConsumerCoordinator` is notified about: • onJoinComplete (that triggers ConsumerRebalanceListener.onPartitionsAssigned) • onJoinPrepare (that triggers ConsumerRebalanceListener.onPartitionsRevoked)
`groupSubscription`	

`subscribe` Method

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

`subscribe` ...FIXME

Note	`subscribe` is used exclusively when `KafkaConsumer` is requested to subscribe to the topics that match a pattern with a ConsumerRebalanceListener.

Adding Topics to Group Subscription — `groupSubscribe` Method

```
void groupSubscribe(Collection<String> topics)
```

`groupSubscribe` ...FIXME

Note	`groupSubscribe` is used when...FIXME

RequestFuture

`RequestFuture` is...FIXME

`compose` Method

```
RequestFuture<S> compose(final RequestFutureAdapter<T, S> adapter)
```

`compose` ...FIXME

Note	`compose` is used when: • `AbstractCoordinator` is requested to send a JoinGroupRequest to the group coordinator, sendSyncGroupRequest, sendFindCoordinatorRequest, maybeLeaveGroup, and sendHeartbeatRequest • `ConsumerCoordinator` is requested to send an OffsetCommitRequest to the group coordinator (a Kafka broker) and sendOffsetFetchRequest • `Fetcher` is requested to sendListOffsetRequest

RequestFutureAdapter Contract

`RequestFutureAdapter` is the base of RequestFuture type converters that can do type conversion when a RequestFuture completes with a success or a failure.

Table 1. RequestFutureAdapter Contract

Method	Description

`onFailure`	`void onFailure(`
	`  RuntimeException e,`
	`  RequestFuture<T> future)`
	
	Performs a type conversion when a `RequestFuture` completes with a failure
	
	Used when:
	
	• `FindCoordinatorResponseHandler` is requested to onFailure
	• `RequestFuture` is requested to compose

`onSuccess`	`void onSuccess(`
	`  F value,`
	`  RequestFuture<T> future)`
	
	Performs type conversion when a `RequestFuture` completes with a success
	
	Used exclusively when `RequestFuture` is requested to compose

Table 2. RequestFutureAdapters (Direct Implementations)

RequestFutureAdapter	Description
CoordinatorResponseHandler	
FindCoordinatorResponseHandler	

CoordinatorResponseHandler Contract

`CoordinatorResponseHandler` is the extension of the RequestFutureAdapter contract for type converters that handle ClientResponses.

Table 1. CoordinatorResponseHandler Contract

Method	Description
`handle`	`void handle(R response, RequestFuture<T> future)` Handles a successful response from a group coordinator Used exclusively when `CoordinatorResponseHandler` is requested to onSuccess

Table 2. CoordinatorResponseHandlers

CoordinatorResponseHandler	Description
HeartbeatResponseHandler	
JoinGroupResponseHandler	Handles JoinGroupResponses
LeaveGroupResponseHandler	
OffsetCommitResponseHandler	
OffsetFetchResponseHandler	
SyncGroupResponseHandler	

`onFailure` Method

```
void onFailure(RuntimeException e, RequestFuture<T> future)
```

Note	`onFailure` is part of the RequestFutureAdapter Contract to perform a type conversion when a `RequestFuture` completes with a failure.

`onFailure` ...FIXME

`onSuccess` Method

```
void onSuccess(ClientResponse clientResponse, RequestFuture<T> future)
```

Note	`onSuccess` is part of the RequestFutureAdapter Contract to perform a type conversion when a `RequestFuture` completes with a success.

`onSuccess` ...FIXME

FindCoordinatorResponseHandler

`FindCoordinatorResponseHandler` is a concrete RequestFutureAdapter of ClientResponse.

`FindCoordinatorResponseHandler` is used exclusively when `AbstractCoordinator` is requested to discover the coordinator for a consumer group (and when successful sets the group coordinator as a Kafka node with the hostname and the port).

Performing Response Conversion on Success — `onSuccess` Method

```
void onSuccess(ClientResponse resp, RequestFuture<Void> future)
```

Note	`onSuccess` is part of the RequestFutureAdapter Contract to perform a type conversion when a `RequestFuture` completes with a success.

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

```
Received FindCoordinator response [resp]
```

`onSuccess` clears the findCoordinatorFuture internal registry.

`onSuccess` takes the FindCoordinatorResponse (from the given `ClientResponse`) and branches off per error.

No Error

With no error, `onSuccess` sets the group coordinator (as a Kafka node with the hostname and the port).

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

```
Discovered group coordinator [coordinator]
```

`onSuccess` requests the ConsumerNetworkClient to initiate a connection to the group coordinator.

In the end, `onSuccess` requests the Heartbeat to `resetSessionTimeout` and completes the given `RequestFuture<Void>`.

GROUP_AUTHORIZATION_FAILED Error

In case of `GROUP_AUTHORIZATION_FAILED` error, `onSuccess` raises a `GroupAuthorizationException` in the given `RequestFuture<Void>`.

Other Errors

All other errors lead `onSuccess` to print out the following DEBUG message to the logs and raise the error in the given `RequestFuture<Void>`.

```
Group coordinator lookup failed: [message]
```

Performing Response Conversion on Failure — `onFailure` Method

```
void onFailure(RuntimeException e, RequestFuture<Void> future)
```

Note	`onFailure` is part of the RequestFutureAdapter Contract to perform a type conversion when a `RequestFuture` completes with a failure.

`onFailure` ...FIXME

HeartbeatResponseHandler

`HeartbeatResponseHandler` is...FIXME

JoinGroupResponseHandler

`JoinGroupResponseHandler` is a concrete CoordinatorResponseHandler to handle a successful JoinGroupResponse from a group coordinator (and convert it to a `ByteBuffer`).

Note	`JoinGroupResponseHandler` is a Java private class of AbstractCoordinator and so can only be used by AbstractCoordinators.

`JoinGroupResponseHandler` is created exclusively when `AbstractCoordinator` is requested to send a JoinGroupRequest to the group coordinator (when requested to initiateJoinGroup when joinGroupIfNeeded which is when `ConsumerCoordinator` is requested to poll for coordinator events).

`JoinGroupResponseHandler` takes no arguments to be created.

Handling Successful Response From Group Coordinator — `handle` Method

```
void handle(
  JoinGroupResponse joinResponse,
  RequestFuture<ByteBuffer> future)
```

Note	`handle` is part of the CoordinatorResponseHandler Contract to handle a successful response from a group coordinator.

`handle` branches off per the error in the given JoinGroupResponse.

No Errors

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

```
Received successful JoinGroup response: [joinResponse]
```

`handle` requests the GroupCoordinatorMetrics for the joinLatency sensor to record the request latency.

When in `REBALANCING` state, `handle` creates a new Generation (with the generation ID, the member ID, and the group protocol from the given `JoinGroupResponse`).

In the end, `handle` checks if the given `JoinGroupResponse` is for the leader of the consumer group or a follower and performs partition assignment and notifies the group coordinator or onJoinFollower, respectively.

When not in `REBALANCING` state, `handle` requests the raises the given `RequestFuture<ByteBuffer>` to raise a `UnjoinedGroupException`.

Caution	There are other errors, but of little interest currently (aka *FIXME*).

OffsetCommitResponseHandler

`OffsetCommitResponseHandler` is...FIXME

SyncGroupResponseHandler

`SyncGroupResponseHandler` is...FIXME

Broker Nodes — Kafka Servers

Note	A **Kafka server**, a **Kafka broker** and a **Kafka node** all refer to the same concept and are synonyms (see the scaladoc of KafkaServer).

A **Kafka broker** is modelled as KafkaServer that hosts topics.

Note	Given topics are always partitioned across brokers in a cluster a single broker hosts topic partitions of one or more topics actually (even when a topic is only partitioned to just a single partition).

Quoting Broker article (from Wikipedia, the free encyclopedia):

> A broker is an individual person who arranges transactions between a buyer and a seller for a commission when the deal is executed.

> A broker's prime responsibility is to bring sellers and buyers together and thus a broker is the third-person facilitator between a buyer and a seller.

A Kafka broker receives messages from producers and stores them on disk keyed by unique **offset**.

A Kafka broker allows consumers to fetch messages by topic, partition and offset.

Kafka brokers can create a Kafka cluster by sharing information between each other directly or indirectly using Zookeeper.

A Kafka cluster has exactly one broker that acts as the Controller.

You can start a single Kafka broker using `kafka-server-start.sh` script.

Starting Kafka Broker

Start Zookeeper.

```
./bin/zookeeper-server-start.sh config/zookeeper.properties
```

Only when Zookeeper is up and running you can start a Kafka server (that will connect to Zookeeper).

```
./bin/kafka-server-start.sh config/server.properties
```

Tip	Read kafka-server-start.sh script.

`kafka-server-start.sh` script

`kafka-server-start.sh` starts a Kafka broker.

```
$ ./bin/kafka-server-start.sh
USAGE: ./bin/kafka-server-start.sh [-daemon] server.properties [--override property=va
lue]*
```

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)
```

Topics

Topics are virtual groups of one or many partitions across Kafka brokers in a Kafka cluster.

A single Kafka broker stores messages in a partition in an ordered fashion, i.e. appends them one message after another and creates a log file.

Producers write messages to the tail of these logs that consumers read at their own pace.

Kafka scales topic consumption by distributing partitions among a consumer group, which is a set of consumers sharing a common group identifier.

Note	Due to limitations in metric names, topics with a period (`.`) or underscore (`_`) could collide. To avoid issues it is best to use either, but not both.

Use kafka-topics shell script to manage topics.

Partitions

Partitions with messages — topics can be partitioned to improve read/write performance and resiliency. You can lay out a topic (as partitions) across a cluster of machines to allow data streams larger than the capability of a single machine. Partitions are log files on disk with sequential write only. Kafka guarantees message ordering in a partition.

The **log end offset** is the offset of the last message written to a log.

The **high watermark offset** is the offset of the last message that was successfully copied to all of the log's replicas.

Note	A consumer can only read up to the high watermark offset to prevent reading unreplicated messages.

Messages

Messages are the data that brokers store in the partitions of a topic.

Messages are sequentially appended to the end of the partition log file and numbered by unique offsets. They are persisted on disk (aka *disk-based persistence*) and replicated within the cluster to prevent data loss. It has an in-memory page cache to improve data reads. Messages are in partitions until deleted when **TTL** occurs or after **compaction**.

Offsets

Offsets are message positions in a topic.

Kafka Clients

- Producers
- Consumers

Producers

Multiple concurrent **producers** that send (aka *push*) messages to topics which is appending the messages to the end of partitions. They can batch messages before they are sent over the wire to a topic. Producers support message compression. Producers can send messages in synchronous (with acknowledgement) or asynchronous mode.

```
import collection.JavaConversions._
import org.apache.kafka.common.serialization._
import org.apache.kafka.clients.producer.KafkaProducer
import org.apache.kafka.clients.producer.ProducerRecord

val cfg = Map(
  "bootstrap.servers" -> "localhost:9092",
  "key.serializer" -> classOf[IntegerSerializer],
  "value.serializer" -> classOf[StringSerializer])
val producer = new KafkaProducer[Int, String](cfg)
val msg = new ProducerRecord(topic = "my-topic", key = 1, value = "hello")

scala> val f = producer.send(msg)
f: java.util.concurrent.Future[org.apache.kafka.clients.producer.RecordMetadata] = org
.apache.kafka.clients.producer.internals.FutureRecordMetadata@2e9e8fe

scala> f.get
res7: org.apache.kafka.clients.producer.RecordMetadata = my-topic-0@1

producer.close
```

Kafka Consumers

Multiple concurrent **consumers** read (aka *pull*) messages from topics however they want using offsets. Unlike typical messaging systems, Kafka consumers pull messages from a topic using offsets.

A **Kafka consumer** consume records from a Kafka cluster.

Note	Kafka 0.9.0.0 was about introducing a brand new Consumer API *aka* **New Consumer**.

When a consumer is created, it requires bootstrap.servers which is the initial list of brokers to discover the full set of alive brokers in a cluster from.

A consumer has to subscribe to the topics it wants to read messages from called topic subscription.

Caution	FIXME Building a custom consumption strategy

Using Kafka Consumer API requires **kafka-clients** dependency in your project.

```
val kafka = "0.10.2.1"
libraryDependencies += "org.apache.kafka" % "kafka-clients" % kafka

// You should also define the logging binding for slf4j
// Kafka uses slf4j for logging
val logback = "1.2.3"
libraryDependencies += "ch.qos.logback" % "logback-core" % logback
libraryDependencies += "ch.qos.logback" % "logback-classic" % logback
```

Consumer Contract

```
public interface Consumer<K, V> extends Closeable {
  // FIXME more methods...
  ConsumerRecords<K, V> poll(long timeout)
}
```

Table 1. Consumer Contract

Method	Description
`poll`	Used to…

Topic Subscription

Topic Subscription is the process of announcing the topics a consumer wants to read messages from.

```
void subscribe(Collection<String> topics)
void subscribe(Collection<String> topics, ConsumerRebalanceListener callback)
void subscribe(Pattern pattern, ConsumerRebalanceListener callback)
```

Note	`subscribe` method is not incremental and you always must include the full list of topics that you want to consume from.

You can change the set of topics a consumer is 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

Clusters

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

Metrics

`Metrics` is…FIXME

Sensor

`Sensor` is…FIXME

MetricsReporter

JmxReporter

`JmxReporter` is a metrics reporter that is always included in metric.reporters setting with `kafka.consumer` metrics prefix.

ProducerMetrics

`ProducerMetrics` is...FIXME

```
// Generate HTML with the metrics
$ ./bin/kafka-run-class.sh org.apache.kafka.clients.producer.internals.ProducerMetrics
 > metrics.html
```

SenderMetrics

`SenderMetrics` is...FIXME

`updateProduceRequestMetrics` Method

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

`updateProduceRequestMetrics` ...FIXME

Note	`updateProduceRequestMetrics` is used exclusively when `Sender` is sendProducerData.

Kafka Properties

Table 1. Properties

Name	
`advertised.listeners`	Comma-separated list of URIs to publis listeners config property. Default: `null` In IaaS environments (e.g. Docker, Kub be different from the interface to which broker advertises an address that is ac the value for listeners is used. Unlike listeners it is invalid to advertise • Use KafkaConfig.AdvertisedListen • Use KafkaConfig.advertisedListene Dynamic reconfigurable configuration
`authorizer.class.name`	Fully-qualified class name of the Autho Default: (empty) Supports authorizers that implement th which was previously used for authoriz Authorizer Interface). • Use KafkaConfig.AuthorizerClassN • Use KafkaConfig.authorizerClassN
`auto.commit.interval.ms`	How often (in milliseconds) consumer c enable.auto.commit is enabled
`auto.create.topics.enable`	Enables auto creation of a topic Default: `true` • Use `KafkaConfig.AutoCreateTopicsE` • Use KafkaConfig.autoCreateTopics
`auto.leader.rebalance.enable`	Enables auto leader balancing Default: `true` A background thread checks and trigge • Use `KafkaConfig.AutoLeaderRebalar`

	• Use KafkaConfig.autoLeaderReba
`auto.offset.reset`	Reset policy — what to do when there i not exist any more on the server (e.g. b • **earliest** — automatically reset the • **latest** — automatically reset the off • **none** — throw an exception to the consumer's group • anything else: throw an exception t Default: `latest`
`background.threads`	The number of threads to use for variou Default: `10` Must be at least `1` Dynamic reconfigurable configuration
`bootstrap.servers`	A comma-separated list of `host:port` p a Kafka cluster, e.g. `localhost:9092` or Default: `(empty)`
`broker.id`	The broker id of a Kafka broker for iden If unset, a unique broker id will be gene generated broker id's and user configur reserved.broker.max.id + 1. Use KafkaConfig.brokerId to access the
`broker.id.generation.enable`	Enables **automatic broker id generati** Default: `true` When enabled (`true`) the value config • Use `KafkaConfig.BrokerIdGeneratio` • Use KafkaConfig.brokerIdGenerati
`broker.rack`	
`check.crcs`	Automatically check the CRC32 of the r disk corruption to the messages occurr disabled in cases seeking extreme perf Use `ConsumerConfig.CHECK_CRCS_CONFIG`

`client.id`	Default: `(randomly-generated)`
`connection.failed.authentication.delay.ms`	Connection close delay on failed authe connection close will be delayed on aut than connections.max.idle.ms to preve Default: `100` Has to be at least `0`
`connections.max.idle.ms`	Idle connections timeout: the server so more than this Default: `10 * 60 * 1000L`

`control.plane.listener.name`	Name of the listener for communication
	
	Default: `null` (undefined), i.e. no dedi
	
	Must be defined in listener.security.prot
	
	Note: FIXME Is this true? Must b
	
	Must be different than interBrokerLister
	
	Broker will use the name to locate the e
	controller.
	
	For example, if a broker's config is:
	
	`listeners=INTERNAL://192.1.1.8:9092,`
	`listener.security.protocol.map=INTER`
	`control.plane.listener.name=CONTROLL`
	
	On startup, the broker will start listenin
	controller side, when it discovers a brok
	the name to find the endpoint, which it
	
	For example, if the broker's published e
	
	`"endpoints": [`
	`"INTERNAL://broker1:9092",`
	`"EXTERNAL://broker1:9093",`
	`"CONTROLLER://broker1:9094"`
	`]`
	
	and the controller's config is:
	
	`listener.security.protocol.map=INTER`
	`control.plane.listener.name=CONTROLL`
	
	then a controller will use "broker1:9094
	
	• Use KafkaConfig.ControlPlaneListe

	• Use KafkaConfig.controlPlaneListe
`default.replication.factor`	The default replication factor that is use Default: `1` Increase the default value to at least `2`
`delegation.token.master.key`	
`delete.topic.enable`	Enables topic deletion Note: Deleting topic through the ac Default: `true`
`enable.auto.commit`	When enabled (i.e. `true`) consumer of (aka *consumer auto commit*) every auto Default: `true` When disabled, offsets have to be com KafkaConsumer.commitSync or asynch restore the position of a consumer usin Used when `KafkaConsumer` is created a
`fetch.max.bytes`	The maximum amount of data the serve fetched in batches by the consumer, an of the fetch is larger than this value, the consumer can make progress. As such batch size accepted by the broker is de max.message.bytes (topic config). Note parallel. Use `ConsumerConfig.FETCH_MAX_BYTES_CO`
`fetch.max.wait.ms`	The maximum amount of time the serve isn't sufficient data to immediately satis Use `ConsumerConfig.FETCH_MAX_WAIT_MS_`
`fetch.min.bytes`	The minimum amount of data the serve available the request will wait for that m The default setting of 1 byte means tha of data is available or the fetch request something greater than 1 will cause the accumulate which can improve server t Use `ConsumerConfig.FETCH_MIN_BYTES_CO`
`file.delete.delay.ms`	

`group.id`	The name of the consumer group the c
`heartbeat.interval.ms`	The expected time between heartbeats management facilities.
`host.name`	The hostname a Kafka broker listens o Default: `(empty)`
`inter.broker.listener.name`	Name of the listener for inter-broker co listener.security.protocol.map) Default: security.inter.broker.protocol Must be among advertised.listeners It is an error to use together with securi • Use KafkaConfig.InterBrokerListen • Use KafkaConfig.interBrokerListen
`inter.broker.protocol.version`	Version of the inter-broker protocol Default: the latest `ApiVersion` (e.g. `2.` Typically bumped up after all brokers w • Use KafkaConfig.InterBrokerProto • Use KafkaConfig.interBrokerProto
`interceptor.classes`	Comma-separated list of ConsumerInte Default: `(empty)`
`key.deserializer`	How to deserialize message keys.
`leader.imbalance.check.interval.seconds`	How often the active KafkaController s *AutoLeaderRebalance* or *AutoPreferre* Default: `300` • Use KafkaConfig.LeaderImbalance • Use KafkaConfig.leaderImbalance
`leader.imbalance.per.broker.percentage`	Allowed ratio of leader imbalance per b goes above this value per broker. The Default: `10` • Use KafkaConfig.LeaderImbalance • Use KafkaConfig.leaderImbalance

`listeners`	Comma-separated list of URIs and liste Default: `PLAINTEXT://host.name:port` Use `0.0.0.0` to bind to all the network the default interface. • Use KafkaConfig.ListenersProp to • Use KafkaConfig.listeners to acces Dynamic reconfigurable configuration
`listener.security.protocol.map`	Map of listener names and security pro map entries are separated by commas) map. Default: Map with `PLAINTEXT` , `SSL` , `SA` This map must be defined for the same IP. For example, internal and external tr Concretely, the user could define listen property as: `INTERNAL:SSL,EXTERNAL:SSL` Different security (SSL and SASL) setti normalised prefix (the listener name is l different keystore for the INTERNAL list `listener.name.internal.ssl.keystore.lo` name is not set, the config will fallback • Use KafkaConfig.ListenerSecurityP • Use KafkaConfig.listenerSecurityP Dynamic reconfigurable configuration
`log.cleaner.backoff.ms`	Dynamic reconfigurable configuration
`log.cleaner.dedupe.buffer.size`	Dynamic reconfigurable configuration
`log.cleaner.enable`	Enables the log cleaner process to run any topics with a cleanup.policy=compa topics will not be compacted and contin Default: `true` • Use KafkaConfig.LogCleanerEnab • Use KafkaConfig.logCleanerEnabl
`log.cleaner.io.buffer.load.factor`	Dynamic reconfigurable configuration
`log.cleaner.io.buffer.size`	Dynamic reconfigurable configuration
`log.cleaner.io.max.bytes.per.second`	Dynamic reconfigurable configuration

`log.cleaner.io.max.bytes.per.second`	Dynamic reconfigurable configuration
`log.cleaner.threads`	Dynamic reconfigurable configuration
`log.cleanup.policy`	Log Cleanup Policies (Strategies) — Lo Default: delete Included in copyKafkaConfigToLog (to s • Use KafkaConfig.LogCleanupPolic • Use KafkaConfig.logCleanupPolicy
`log.dir`	The directory in which the log data is ke Default: `/tmp/kafka-logs`
`log.dirs`	The directories in which the log data is Default: log.dir Use KafkaConfig.logDirs to access the
`log.flush.interval.messages`	Number of messages written to a log pa forcing an fsync) Default: `Long.MaxValue` (maximum pos E.g. if this was set to `1` we would fsyn after every five messages. It is recommended not setting this and operating system's background flush ca Must be at least `0` Topic-level configuration: flush.messag • Use KafkaConfig.LogFlushInterval • Use KafkaConfig.logFlushIntervalM
`log.flush.interval.ms`	How long (in millis) a message written t disk (by forcing an fsync). If not set, the Default: `null` (undefined) E.g. if this was set to `1000` we would fs Used exclusively when `LogManager` is r It is recommended not setting this and operating system's background flush ca Must be undefined or at least `0`

	• Use KafkaConfig.LogFlushInterval • Use KafkaConfig.logFlushIntervalM
`log.flush.offset.checkpoint.interval.ms`	
`log.flush.scheduler.interval.ms`	
`log.flush.start.offset.checkpoint.interval.ms`	
`log.index.size.max.bytes`	Maximum size (in bytes) of the offset in preallocated and shrinked only after log Default: `10 * 1024 * 1024` You generally should not need to chang Must be at least `0` • Use KafkaConfig.LogIndexSizeMa • Use KafkaConfig.logIndexSizeMax
`log.retention.bytes`	Maximum size of a partition (which con segments and free up space. `log.retention.bytes` is enforced at the compute the topic retention in bytes. Default: `-1L` Must be at least `-1`
`log.retention.check.interval.ms`	How often (in millis) the `LogManager` (a eligible for deletion Default: `5 * 60 * 1000L` (millis) Must be at least `1` • Use KafkaConfig.LogCleanupInter • Use KafkaConfig.logCleanupInterv
`log.retention.ms`	How long (in millis) to keep a log file be Default: `24 * 7 * 60 * 60 * 1000L` (7 d Must be at least `-1` Unless set, the value of log.retention.m
`log.retention.minutes`	How long (in mins) to keep a log file be Unless set, the value of log.retention.ho

`log.retention.hours`	How long (in hours) to keep a log file be Considered the last unless log.retentior
`log.roll.ms`	Time (in millis) after which Kafka forces ensure that retention can delete or com Default: `604800000` (7 days) Must be at least `1` • Use KafkaConfig.LogRollTimeMillis • Use KafkaConfig.logRollTimeMillis
`log.segment.bytes`	The maximum size of a segment file of at a time so a larger segment size mea Default: `1 * 1024 * 1024 * 1024` Must be at least `14` bytes (`LegacyReco` Use KafkaConfig.logSegmentBytes to a
`max.block.ms`	
`max.partition.fetch.bytes`	The maximum amount of data per-parti batches by the consumer. If the first rec larger than this limit, the batch will still b progress. The maximum record batch s message.max.bytes (broker config) or r Use `ConsumerConfig.MAX_PARTITION_FETC` Note: Use fetch.max.bytes for lir
`max.poll.records`	(KafkaConsumer) The maximum numb polling topics for records. The default setting (`-1`) sets no upper `Consumer.poll()` will return as soon as expires. `max.poll.records` was added to Kafka From kafka-clients mailing list: > `max.poll.records` only controls the r affect fetching. The consumer will try It will then buffer those records and r (either all from the same topic partitic records, or from multiple topic partitic partitions does not cover the `max.po` Use `ConsumerConfig.MAX_POLL_RECORDS_C`

	Internally, `max.poll.records` is used ex Fetcher).
`message.max.bytes`	Dynamic reconfigurable configuration
`metadata.max.age.ms`	
`metric.reporters`	The list of fully-qualified classes names Default: JmxReporter
`metrics.num.samples`	Number of samples to compute metrics
`metrics.sample.window.ms`	Time window (in milliseconds) a metrics
`min.insync.replicas`	The minimum number of replicas in ISR `required.acks=-1` (or `all`) Default: `1` Must be at least `1` When a Kafka producer sets acks to `a` number of replicas that must acknowled If this minimum cannot be met, then the `NotEnoughReplicas` or `NotEnoughReplica` Used together with acks allows you to e A typical scenario would be to create a `min.insync.replicas` to 2, and produce raises an exception if a majority of repli • Use KafkaConfig.MinInSyncReplic • Use KafkaConfig.minInSyncReplic
`num.io.threads`	The number of threads that KafkaServe disk I/O Default: `8` Must be at least `1` Dynamic reconfigurable configuration
`max.connections.per.ip`	The maximum number of connections a Default: `Int.MaxValue` Must be at least `0` (with `0` if there are max.connections.per.ip.overrides prope

	Dynamic reconfigurable configuration
`max.connections.per.ip.overrides`	A comma-separated list of per-ip or hos connections, e.g. `hostName:100,127.0.0` Default: (empty) Dynamic reconfigurable configuration
`num.network.threads`	The number of threads that SocketServ receiving requests from the network an Default: `3` Must be at least `1` Dynamic reconfigurable configuration
`num.partitions`	The number of log partitions for auto-cr Default: `1` Increase the default value (`1`) since it data balancing and aids consumer para
`num.recovery.threads.per.data.dir`	The number of threads per log data dire shutdown Default: `1` Must be at least `1` Dynamic reconfigurable configuration
`num.replica.alter.log.dirs.threads`	The number of threads that can move r disk I/O Default: `null` • Use KafkaConfig.NumReplicaAlter • Use KafkaConfig.getNumReplicaA
`num.replica.fetchers`	The number of fetcher threads that Rep from a source broker Default: `1` The higher the value the higher degree Dynamic reconfigurable configuration • Use KafkaConfig.NumReplicaFetc • Use KafkaConfig.numReplicaFetch

	• Use KafkaConfig.numReplicaFetch
`port`	The port a Kafka broker listens on Default: `9092`
`principal.builder.class`	Fully-qualified name of KafkaPrincipalB object for authorization Default: `null` (i.e. DefaultKafkaPrincip Supports the deprecated `PrincipalBuil` authentication over SSL. If no principal builder is defined, the def • For SSL authentication, the princip ssl.principal.mapping.rules applied one is provided; otherwise, if client be ANONYMOUS. • For SASL authentication, the princi sasl.kerberos.principal.to.local.rule for other mechanisms. For PLAINT Used when `ChannelBuilders` is reques Dynamic reconfigurable configuration • Use KafkaConfig.PrincipalBuilderC
`replica.fetch.backoff.ms`	How long (in millis) a fetcher thread is g (while sending a fetch request) or after Default: `1000` (millis) Must be at least `0` • Use KafkaConfig.ReplicaFetchBac • Use KafkaConfig.replicaFetchBack
`replica.fetch.max.bytes`	The number of bytes of messages to at Default: `1024 * 1024` (1 MB) Must be at least `0` This is not an absolute maximum, if the fetch is larger than this value, the recor can be made. The maximum record bat message.max.bytes (broker config) or r • Use KafkaConfig.ReplicaFetchMax • Use KafkaConfig.replicaFetchMaxI

`replica.fetch.response.max.bytes`	Maximum bytes expected for the entire Default: `10 * 1024 * 1024` Must be at least `0` Records are fetched in batches, and if t the fetch is larger than this value, the re can be made. As such, this is not an ab accepted by the broker is defined via m max.message.bytes (topic config). • Use KafkaConfig.ReplicaFetchRes • Use KafkaConfig.replicaFetchResp
`queued.max.requests`	The number of queued requests allowe Default: `500` Must be at least `1`
`rebalance.timeout.ms`	The maximum allowed time for each wo
`receive.buffer.bytes`	The hint about the size of the TCP netw when reading data. If the value is -1, th
`replica.fetch.wait.max.ms`	
`replica.lag.time.max.ms`	How long to wait for a follower to consu leader removes the follower from the IS Default: `10000L` (millis) Note: replica.fetch.wait.max.ms sh `replica.lag.time.max.ms` to • Use KafkaConfig.ReplicaLagTimeM • Use KafkaConfig.replicaLagTimeM
`replica.socket.timeout.ms`	Socket timeout of `ReplicaFetcherBlocki` leader brokers Default: `30 * 1000` (30 seconds) Should always be at least replica.fetch. • Use KafkaConfig.ReplicaSocketTir • Use KafkaConfig.replicaSocketTim
`request.timeout.ms`	The configuration controls the maximur a request. If the response is not receive request if necessary or fail the request

	Use `ConsumerConfig.REQUEST_TIMEOUT_MS`
`reserved.broker.max.id`	Maximum number that can be used for Default: `1000` * Use `KafkaConfig.MaxReservedBrokerId` * Use KafkaConfig.maxReservedBroke
`retry.backoff.ms`	Time to wait before attempting to retry a repeatedly sending requests in a tight l Use `ConsumerConfig.RETRY_BACKOFF_MS_C`
`sasl.mechanism.inter.broker.protocol`	
`sasl.jaas.config`	
`sasl.enabled.mechanisms`	
`sasl.kerberos.service.name`	
`sasl.kerberos.kinit.cmd`	
`sasl.kerberos.ticket.renew.window.factor`	
`sasl.kerberos.ticket.renew.jitter`	
`sasl.kerberos.min.time.before.relogin`	
`sasl.kerberos.principal.to.local.rules`	
`sasl.login.refresh.window.factor`	
`sasl.login.refresh.window.jitter`	
`sasl.login.refresh.min.period.seconds`	
`sasl.login.refresh.buffer.seconds`	
`security.inter.broker.protocol`	Security protocol for inter-broker comm Default: `PLAINTEXT` Possible values: • `PLAINTEXT` • `SSL` • `SASL_PLAINTEXT` • `SASL_SSL`

	It is an error when defined with inter.bro listener.security.protocol.map). It is validated when the inter-broker con `SASL_SSL`) for…FIXME • Use KafkaConfig.InterBrokerSecur • Use KafkaConfig.interBrokerSecur
`send.buffer.bytes`	The hint about the size of the TCP netw when sending data. If the value is -1, th
`session.timeout.ms`	The timeout used to detect worker failu Default: `10000`
`socket.request.max.bytes`	The maximum number of bytes in a soc Default: `100 * 1024 * 1024` Must be at least `1`
`ssl.principal.mapping.rules`	Rules for mapping from the distinguishe Default: `DEFAULT` (i.e. the distinguished The rules are evaluated in order and th map it to a short name. Any later rules i This configuration is ignored for a custo principal.builder.class configuration. Used when `SslChannelBuilder` is confi • Use KafkaConfig.SslPrincipalMapp
`ssl.protocol`	
`ssl.provider`	
`ssl.cipher.suites`	
`ssl.enabled.protocols`	
`ssl.keystore.type`	
`ssl.keystore.location`	
`ssl.keystore.password`	
`ssl.key.password`	
`ssl.truststore.type`	

`ssl.truststore.location`	
`ssl.truststore.password`	
`ssl.keymanager.algorithm`	
`ssl.trustmanager.algorithm`	
`ssl.endpoint.identification.algorithm`	
`ssl.secure.random.implementation`	
`ssl.client.auth`	Client authentication Default: `none` Supported values (case-insensitive): `r` Non-reconfigurable ReconfigurableConfigs • Use KafkaConfig.SslClientAuthPro
`transactional.id.expiration.ms`	
`transaction.max.timeout.ms`	The maximum allowed timeout for trans If a client's requested transaction time e `InitProducerIdRequest`. This prevents a consumers reading from topics include Default: `15 minutes` Must be at least `1` • Use KafkaConfig.transactionMaxTi
`unclean.leader.election.enable`	Enables Unclean Partition Leader Elect Cluster-wide property: unclean.leader.e Topic-level property: unclean.leader.ele
`value.deserializer`	How to deserialize message values
`zookeeper.connect`	Comma-separated list of Zookeeper ho `localhost:2181` , `127.0.0.1:3000,127.0` Default: `(empty)` Zookeeper URIs can have an optional c `127.0.0.1:3000,127.0.0.1:3001,127.0.0.` If the optional chroot path suffix is used

`zookeeper.connect`	Zookeeper URIs can have an optional c `127.0.0.1:3000,127.0.0.1:3001,127.0.0.` If the optional chroot path suffix is used It is recommended to include all the hos • Available as `KafkaConfig.ZkConnect` • Use KafkaConfig.zkConnect to acc
`zookeeper.connection.timeout.ms`	The max time that the client waits to es Default: zookeeper.session.timeout.ms • Available as `KafkaConfig.ZkConnect` • Use KafkaConfig.zkConnectionTim
`zookeeper.max.in.flight.requests`	The maximum number of unacknowled blocking. Has to be at least 1 Default: `10` • Available as `KafkaConfig.ZkMaxInFl` • Use KafkaConfig.zkMaxInFlightRe
`zookeeper.session.timeout.ms`	Zookeeper session timeout Default: `6000` • Available as `KafkaConfig.ZkSessior` • Use KafkaConfig.zkSessionTimeou
`zookeeper.set.acl`	Enables secure ACLs Default: `false` • Available as `KafkaConfig.ZkEnableS` • Use KafkaConfig.zkEnableSecureA

bootstrap.servers Property

`bootstrap.servers` is a comma-separated list of host and port pairs that are the addresses of the Kafka brokers in a "bootstrap" Kafka cluster that a Kafka client connects to initially to bootstrap itself.

A host and port pair uses `:` as the separator.

```
localhost:9092
localhost:9092,another.host:9092
```

`bootstrap.servers` provides the initial hosts that act as the starting point for a Kafka client to discover the full set of alive servers in the cluster.

Note	Since these servers are just used for the initial connection to discover the full cluster membership (which may change dynamically), this list does not have to contain the full set of servers (you may want more than one, though, in case a server is down).

Note	Clients (producers or consumers) make use of all servers irrespective of which servers are specified in `bootstrap.servers` for bootstrapping.

Tip	Use `org.apache.kafka.clients.CommonClientConfigs.BOOTSTRAP_SERVERS_CONFIG` public value to refer to the property.

client.id Property

An optional identifier of a Kafka consumer (in a consumer group) that is passed to a Kafka broker with every request.

The sole purpose of this is to be able to track the source of requests beyond just ip and port by allowing a logical application name to be included in Kafka logs and monitoring aggregates.

enable.auto.commit Property

`enable.auto.commit` ...FIXME

By default, as the consumer reads messages from Kafka, it will periodically commit its current offset (defined as the offset of the next message to be read) for the partitions it is reading from back to Kafka. Often you would like more control over exactly when offsets are committed. In this case you can set `enable.auto.commit` to `false` and call the `commit` method on the consumer.

group.id Property

`group.id` specifies the name of the consumer group a Kafka consumer belongs to.

When the Kafka consumer is constructed and `group.id` does not exist yet (i.e. there are no existing consumers that are part of the group), the consumer group will be created automatically.

Note	If all consumers in a group leave the group, the group is automatically destroyed.

retry.backoff.ms Property

`retry.backoff.ms` is the time to wait before attempting to retry a failed request to a given topic partition.

This avoids repeatedly sending requests in a tight loop under some failure scenarios.

Logging

Kafka Streams uses Apache Log4j 2 for logging service.

The main logger is **org.apache.kafka.streams**. Use log4j.properties to set the logging levels.

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

log4j.properties

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

log4j.logger.org.apache.kafka.streams=DEBUG
log4j.additivity.org.apache.kafka.streams=false
```

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

Simple Logging Facade for Java (SLF4J)

Kafka uses Simple Logging Facade for Java (SLF4J) for logging.

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
```

Note	With logback's configuration (as described in the above tip) you may see the follow
	
	``
	SLF4J: Class path contains multiple SLF4J bindings.
	SLF4J: Found binding in [jar:file:/Users/jacek/.ivy2/cache/org.slf4j/slf4j-log4
	SLF4J: Found binding in [jar:file:/Users/jacek/.ivy2/cache/ch.qos.logback/logba
	SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanatior
	SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
	``
	
	Commenting out `logback-classic` in `build.sbt` resolves it.
	
	build.sbt
	
	``
	val logback = "1.2.3"
	libraryDependencies += "ch.qos.logback" % "logback-core" % logback
	//libraryDependencies += "ch.qos.logback" % "logback-classic" % logback
	``
	
	FIXME: Explain why the commenting out is required?

Gradle Tips

Building Kafka Distribution

```
gradle -PscalaVersion=2.11.8 clean releaseTarGz install
```

It takes around 2 minutes (after all the dependencies were downloaded once).

After the command, you can unpack the release as follows:

```
tar -zxvf core/build/distributions/kafka_2.11-0.10.1.0-SNAPSHOT.tgz
```

Executing Single Test

```
gradle -PscalaVersion=2.11.8 :core:test --no-rebuild --tests "*PlaintextProducerSendTe
st"
```

Zookeeper Tips

The zookeeper shell shipped with Kafka works with no support for command line history because jline jar is missing (see KAFKA-2385).

A solution is to use the official distribution of Apache Zookeeper (**3.4.10** as of this writing) from Apache ZooKeeper Releases.

Once downloaded, use `./bin/zkCli.sh` to connect to Zookeeper that is used for Kafka.

```
$ ./bin/zkCli.sh -server localhost:2181
...
JLine support is enabled

[zk: localhost:2181(CONNECTED) 0] ls /
[cluster, controller_epoch, controller, brokers, zookeeper, admin, isr_change_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/ka
fka-sandbox/)
[info] sbt server started at 127.0.0.1:4408
sbt:kafka-sandbox> console
[info] Starting scala interpreter...
Welcome to Scala 2.12.3 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_144).
Type in expressions for evaluation. Or try :help.

scala> import kafka.utils._
import kafka.utils._

scala> ZkUtils.controllerZkData(1, System.currentTimeMillis())
res0: String = {"version":1,"brokerid":1,"timestamp":"1506162122097"}
```

Running Kafka Broker in Docker

A very basic installation of Apache Kafka is made up of the following components:

- An instance of Apache Zookeeper
- 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:

- wurstmeister/kafka
- spotify/kafka

Note	ches/docker-kafka is another Docker image (that I have not tried myself yet).

wurstmeister/kafka gives separate images for Apache Zookeeper and Apache Kafka while `spotify/kafka` runs both Zookeeper and Kafka in the same container.

wurstmeister/kafka

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 that allows for further customizations.

Tip	Read the official tutorial on how to use `wurstmeister/kafka` 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
// On macOS use the following script to set DOCKERHOST env var
// DOCKERHOST=$(ifconfig | grep -E "([0-9]{1,3}\.){3}[0-9]{1,3}" | grep -v 127.0.0.1 |
 awk '{ print $2 }' | cut -f2 -d: | head -n1)
// https://github.com/wurstmeister/kafka-docker/issues/17#issuecomment-370237590
```

```
// Start services based on the default docker-compose.yml
// You may want to use -d to detach from the terminal (daemon mode)
$ docker-compose up

// Connect to the container of the Kafka broker
// NOTE: Asciidoc would try to substitute curly braces
//       Remove the spaces between curly braces
$ docker ps --format "{ { .Names } }"
kafka-docker_zookeeper_1
kafka-docker_kafka_1

// NOTE: Asciidoc would try to substitute curly braces
//       Remove the spaces between curly braces
$ docker inspect --format='{ {range $k, $v := .NetworkSettings.Ports} }{ {range $v} }{
 {$k} } -> { {.HostIp} } { {.HostPort} }{ {end} }{ {end} }' kafka-docker_kafka_1
9092/tcp -> 0.0.0.0 32778

// Check the connection from the host to the single Kafka broker
$ nc -vz 0.0.0.0 32778
found 0 associations
found 1 connections:
     1: flags=82<CONNECTED,PREFERRED>
        outif lo0
        src 127.0.0.1 port 59000
        dst 127.0.0.1 port 32778
        rank info not available
        TCP aux info available

Connection to 0.0.0.0 port 32778 [tcp/*] succeeded!

// Print out the topics
// You should see no topics listed
$ docker exec -t kafka-docker_kafka_1 \
  kafka-topics.sh \
    --bootstrap-server :9092 \
    --list

// Create a topic t1
$ docker exec -t kafka-docker_kafka_1 \
  kafka-topics.sh \
    --bootstrap-server :9092 \
    --create \
    --topic t1 \
    --partitions 3 \
    --replication-factor 1

// Describe topic t1
$ docker exec -t kafka-docker_kafka_1 \
  kafka-topics.sh \
    --bootstrap-server :9092 \
    --describe \
    --topic t1
Topic:t1        PartitionCount:3        ReplicationFactor:1     Configs:segment.bytes=
```

```
1073741824
        Topic: t1       Partition: 0    Leader: 1001    Replicas: 1001  Isr: 1001
        Topic: t1       Partition: 1    Leader: 1001    Replicas: 1001  Isr: 1001
        Topic: t1       Partition: 2    Leader: 1001    Replicas: 1001  Isr: 1001

// Connect with the Kafka console consumer in another terminal
$ docker exec -t kafka-docker_kafka_1 \
  kafka-console-consumer.sh \
    --bootstrap-server :9092 \
    --group jacek-japila-pl \
    --topic t1

// Connect with the Kafka console producer in one terminal
$ docker exec -it kafka-docker_kafka_1 \
  kafka-console-producer.sh \
    --broker-list :9092 \
    --topic t1

// Type a message in producer window to see the message printed out in consumer's
// Observe the logs of the running docker-compose up (with no -d)
// Or use docker logs -f kafka-docker_kafka_1

// Ctrl-C to shut down all the processes

// You may also consider removing the containers
// so you always start afresh
// docker-compose rm
```

KafkaClient

`KafkaClient` is the abstraction of Kafka clients.

Note	NetworkClient is the one and only known implementation of the KafkaClient Contract in Apache Kafka.

Table 1. KafkaClient Contract

Method	Description
`authenticationException`	`AuthenticationException authenticationException(Node node)` Used when…FIXME
`close`	`void close(String nodeId)` Closing the Kafka client (releasing resources) Used when: • `RequestSendThread` is requested to `doWork` and `brokerReady` • `KafkaServer` is requested to controlledShutdown • `ReplicaFetcherBlockingSend` is requested to send a client request and wait for a response
`connectionDelay`	`long connectionDelay(Node node, long now)` Used when…FIXME
`connectionFailed`	`boolean connectionFailed(Node node)` Used when…FIXME
`disconnect`	`void disconnect(String nodeId)` Used when: • `AdminClientRunnable` is requested to timeoutCallsInFlight and handleResponses • `ConsumerNetworkClient` is requested to handlePendingDisconnects

`hasInFlightRequests`	`boolean hasInFlightRequests()`
	`boolean hasInFlightRequests(String nodeId);`
	
	Used when…FIXME

`hasReadyNodes`	`boolean hasReadyNodes(long now)` Used when…FIXME

`inFlightRequestCount`	`int inFlightRequestCount()`
	`int inFlightRequestCount(String nodeId)`
	
	Used when…FIXME

`isReady`	`boolean isReady(Node node, long now)` Used when: • `NetworkClient` is requested to ready and leastLoadedNode • `NetworkClientUtils` is requested to isReady and wait unt a connection to a broker node is ready
`leastLoadedNode`	`Node leastLoadedNode(long now)` Used when: • `DefaultMetadataUpdater` is requested to maybeUpdate • `ConsumerNetworkClient` is requested to leastLoadedNode • `Sender` is requested to awaitLeastLoadedNodeReady • `LeastLoadedNodeProvider` and `MetadataUpdateNodeIdProvider` are requested to provide the least-loaded node

`newClientRequest`	`ClientRequest newClientRequest(`
	`  String nodeId,`
	`  AbstractRequest.Builder<?> requestBuilder,`
	`  long createdTimeMs,`
	`  boolean expectResponse)`
	`ClientRequest newClientRequest(`
	`  String nodeId,`
	`  AbstractRequest.Builder<?> requestBuilder,`
	`  long createdTimeMs,`
	`  boolean expectResponse,`
	`  int requestTimeoutMs,`
	`  RequestCompletionHandler callback)`
	
	Creates a new ClientRequest for a given AbstractRequest.Builder and RequestCompletionHandler
	
	Used when…FIXME

`poll`	`List<ClientResponse> poll(`
	`  long timeout,`
	`  long now)`
	
	Used when:
	• `InterBrokerSendThread` thread is started
	• `NetworkClientUtils` utility is used to isReady, awaitRead and sendAndReceive
	• `AdminClientRunnable` thread is started
	• `ConsumerNetworkClient` is requested to poll and transmitSends
	• `Sender` thread is started (and runOnce and maybeSendAndPollTransactionalRequest)

`pollDelayMs`	`long pollDelayMs(Node node, long now)` Used when…FIXME
`ready`	`boolean ready(Node node, long now)` Initiates a connection to the given broker `Node` (i.e. a connection to a broker node is ready) Used when: • `NetworkClientUtils` helper object is requested to wait un a connection to a broker node is ready • `AdminClientRunnable` is requested to sendEligibleCalls

	• `ConsumerNetworkClient` is requested to trySend and tryConnect • `Sender` is requested to sendProducerData (when requested to run a single iteration of sending) • `InterBrokerSendThread` is requested to sendRequests
`send`	`void send(ClientRequest request, long now)` Queues up the ClientRequest for sending Used when: • `InterBrokerSendThread` is requested to sendRequests • `NetworkClientUtils` is requested to sendAndReceive • `AdminClientRunnable` is requested to `sendEligibleCalls` • `ConsumerNetworkClient` is requested to trySend • `Sender` is requested to maybeSendTransactionalReques and sendProduceRequest
`wakeup`	`void wakeup()` Used when…FIXME

NetworkClient — Non-Blocking Network KafkaClient

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

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

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

Figure 1. NetworkClient

`NetworkClient` uses the MetadataUpdater for the following:

- disconnect
- close
- isReady
- poll
- close
- leastLoadedNode
- processDisconnection
- handleTimedOutRequests
- handleCompletedReceives
- handleDisconnections
- initiateConnect

`NetworkClient` is created when:

- `KafkaConsumer` is created (with `ConsumerNetworkClient`)
- `KafkaProducer` is created (with `Sender`)
- `KafkaAdminClient` is created (using `createInternal`)
- `AdminClient` is created (with `ConsumerNetworkClient`)
- `ControllerChannelManager` is requested to addNewBroker (and creates a `RequestSendThread` daemon thread and a `ControllerBrokerStateInfo`)
- `TransactionMarkerChannelManager` is created
- `KafkaServer` does `doControlledShutdown`
- `ReplicaFetcherBlockingSend` is created

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

Name	Description
`connectionStates`	`ClusterConnectionStates` Used when…FIXME

Tip	Enable `DEBUG` logging level for `org.apache.kafka.clients.NetworkClient` logger to see what happens inside. Add the following line to `config/tools-log4j.properties` (for Kafka tools): `log4j.logger.org.apache.kafka.clients.NetworkClient=DEBUG` Add the following line to `config/log4j.properties`: `log4j.logger.org.apache.kafka.clients.NetworkClient=DEBUG` Refer to Logging.

Establishing Connection to Broker Node — `initiateConnect` Internal Method

```
void initiateConnect(Node node, long now)
```

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

```
Initiating connection to node [node]
```

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

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

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

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

`initiateConnect` requests the MetadataUpdater for update.

You should see the following DEBUG message in the logs:

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

Note	`initiateConnect` is used when: • `NetworkClient` is requested to attempt to connect to a broker node • `DefaultMetadataUpdater` is requested to maybeUpdate

`ready` Method

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

Note	`ready` is a part of KafkaClient Contract.

`ready` …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 Socket — `poll` Method

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

Note	`poll` is a part of KafkaClient contract.

`poll` requests the MetadataUpdater for cluster metadata update (if needed and possible).

`poll` then requests Selectable to poll.

In the end, `poll` handles completed request sends, receives, disconnected connections, records any connections to new brokers, initiates API version requests, expire in-flight requests, and finally triggers their `RequestCompletionHandlers`.

In case abortedSends is not empty, `poll` creates a collection of `ClientResponse` with abortedSends, triggers their `RequestCompletionHandlers` and returns them.

`handleCompletedReceives` Internal Method

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

`handleCompletedReceives` ...FIXME

Note	`handleCompletedReceives` is used exclusively when `NetworkClient` is requested to poll.

Creating NetworkClient Instance

`NetworkClient` takes the following when created:

- MetadataUpdater (i.e. DefaultMetadataUpdater)
- Metadata
- Selectable
- Client ID that is used to identify the client in requests to a Kafka server (when `NetworkClient` is requested to create a new ClientRequest)
- `maxInFlightRequestsPerConnection`

- `reconnectBackoffMs`
- `reconnectBackoffMax`
- Size of the TCP send buffer (SO_SNDBUF) for socket connection (in bytes)

  Use send.buffer.bytes property to configure it.

  Used when `NetworkClient` establishes connection to a broker node.
- Size of the TCP receive buffer (SO_RCVBUF) for socket connection (in bytes)

  Use receive.buffer.bytes property to configure it.

  Used when `NetworkClient` establishes connection to a broker node
- `defaultRequestTimeoutMs`
- `Time`
- `discoverBrokerVersions` flag
- `ApiVersions`
- `throttleTimeSensor` Sensor
- `LogContext`

`NetworkClient` initializes the internal registries and counters.

Informing ClientResponse about Response Being Completed — `completeResponses` Internal Method

```
void completeResponses(List<ClientResponse> responses)
```

`completeResponses` informs every `ClientResponse` (in the input `responses`) that a response has been completed.

In case of any exception, `completeResponses` prints out the following ERROR message to the logs:

```
Uncaught error in request completion: [exception]
```

Note	`completeResponses` is used when `NetworkClient` is requested to poll (for both abortedSends and completed actions).

Creating ClientRequest — `newClientRequest` Method

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

Note	`newClientRequest` is part of the KafkaClient Contract to…FIXME.

`newClientRequest` simply creates a new `ClientRequest` (with the input parameters and the correlation incremented, the clientId and the defaultRequestTimeoutMs).

`sendInternalMetadataRequest` Internal Method

```
void sendInternalMetadataRequest(
  MetadataRequest.Builder builder,
  String nodeConnectionId,
  long now)
```

`sendInternalMetadataRequest` …FIXME

Note	`sendInternalMetadataRequest` is used exclusively when `DefaultMetadataUpdater` is requested to maybeUpdate.

`doSend` Internal Method

```
void doSend(
  ClientRequest clientRequest,
  boolean isInternalRequest,
  long now)
void doSend(
  ClientRequest clientRequest,
  boolean isInternalRequest,
  long now,
  AbstractRequest request)
```

`doSend` …FIXME

Note	`doSend` is used when `NetworkClient` is requested to send, sendInternalMetadataRequest and handleInitiateApiVersionRequests.

`send` Method

```
void send(ClientRequest request, long now)
```

Note	`send` is part of the KafkaClient Contract to…FIXME.

`send` …FIXME

`handleDisconnections` Internal Method

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

`handleDisconnections` …FIXME

Note	`handleDisconnections` is used exclusively when `NetworkClient` is requested to poll.

`handleTimedOutRequests` Internal Method

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

`handleTimedOutRequests` …FIXME

Note	`handleTimedOutRequests` is used exclusively when `NetworkClient` is requested to poll.

`processDisconnection` Internal Method

```
void processDisconnection(
  List<ClientResponse> responses,
  String nodeId,
  long now,
  ChannelState disconnectState)
```

`processDisconnection` …FIXME

Note	`processDisconnection` is used when `NetworkClient` is requested to handleTimedOutRequests, handleApiVersionsResponse, and handleDisconnections.

`handleApiVersionsResponse` Internal Method

```
void handleApiVersionsResponse(
  List<ClientResponse> responses,
  InFlightRequest req,
  long now,
  ApiVersionsResponse apiVersionsResponse)
```

`handleApiVersionsResponse` ...FIXME

Note	`handleApiVersionsResponse` is used exclusively when `NetworkClient` is requested to handleCompletedReceives (when requested to poll).

`leastLoadedNode` Method

```
Node leastLoadedNode(long now)
```

Note	`leastLoadedNode` is part of the KafkaClient Contract to...FIXME.

`leastLoadedNode` ...FIXME

`close` Method

```
void close()
```

Note	`close` is part of Java's java.io.Closeable to close this stream and releases any system resources associated with it.

`close` ...FIXME

`close` Method

```
void close(String nodeId)
```

Note	`close` is part of the KafkaClient Contract to...FIXME.

`close` ...FIXME

`isReady` Method

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

Note	`isReady` is part of the KafkaClient Contract to…FIXME.

`isReady` …FIXME

`disconnect` Method

```
void disconnect(String nodeId)
```

Note	`disconnect` is part of the KafkaClient Contract to…FIXME.

`disconnect` …FIXME

RequestCompletionHandler Contract

`RequestCompletionHandler` is the contract of request completion handlers that are executed when a request is complete, i.e. the corresponding response has been received or there was a disconnection while handling the request.

Table 1. RequestCompletionHandler Contract

Method	Description
`onComplete`	`void onComplete(ClientResponse response)` Action to be executed when a request is complete Used when: • `ClientResponse` is requested to onComplete (when `NetworkClient` is requested to poll) • `ConsumerNetworkClient` is requested to checkDisconnects (when `ConsumerNetworkClient` is requested to poll) • `InterBrokerSendThread` is requested to completeWithDisconnect (when `InterBrokerSendThread` is requested to doWork)

Table 2. RequestCompletionHandlers (Direct Implementations)

RequestCompletionHandler	Description
RequestFutureCompletionHandler	
TransactionMarkerRequestCompletionHandler	
TxnRequestHandler	

MetadataUpdater Contract

`MetadataUpdater` is an abstraction of metadata updaters that are used by NetworkClient to handle completed metadata responses.

Table 1. MetadataUpdater Contract

Method	Description
`close`	`void close()` Closes the metadata updater
`fetchNodes`	`List<Node> fetchNodes()` Used when: • `NetworkClient` is requested to leastLoadedNode • `DefaultMetadataUpdater` (of `NetworkClient`) is requested to isAnyNodeConnecting

`handleCompletedMetadataResponse`	`void handleCompletedMetadataResponse(`
	`  RequestHeader requestHeader,`
	`  long now,`
	`  MetadataResponse metadataResponse)`
	
	Handles a response to a metadata request
	
	Used when `NetworkClient` is requested to handle a completed MetadataResponse

`handleDisconnection`	`void handleDisconnection(`
	`  String destination)`
	
	Used when `NetworkClient` is requested to disconnect, close, and processDisconnection

`handleFatalException`	`void handleFatalException(`
	`  KafkaException fatalException)`
	
	Used when `NetworkClient` is requested to doSend and processDisconnection

`isUpdateDue`	`boolean isUpdateDue(long now)` Used when `NetworkClient` is requested to isReady
`maybeUpdate`	`long maybeUpdate(long now)` Starts a cluster metadata update if needed and possible. Used when `NetworkClient` is requested to read and write to sockets.
`requestUpdate`	`void requestUpdate()` Schedules an update of the current cluster metadata Used when `NetworkClient` is requested to handleTimedOutRequests, handleDisconnections and establish a connection to a broker node

Table 2. MetadataUpdaters

MetadataUpdater	Description
AdminMetadataUpdater	Used by AdminMetadataManager
DefaultMetadataUpdater	Default `MetadataUpdater` of NetworkClient
ManualMetadataUpdater	Used by ControllerChannelManager, TransactionMarkerChannelManager, KafkaServer and ReplicaFetcherBlockingSend

DefaultMetadataUpdater

`DefaultMetadataUpdater` is a MetadataUpdater that `NetworkClient` uses to…FIXME

`DefaultMetadataUpdater` is created exclusively for a NetworkClient.

Tip	Enable `ALL` logging level for `org.apache.kafka.clients.NetworkClient` logger to see what happens inside. Add the following line to `log4j.properties`: `log4j.logger.org.apache.kafka.clients.NetworkClient=ALL` Refer to Logging.

Creating DefaultMetadataUpdater Instance

`DefaultMetadataUpdater` takes the following to be created:

- Metadata

`DefaultMetadataUpdater` initializes the internal properties.

`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 to handle a response to a metadata request.

`handleCompletedMetadataResponse` ...FIXME

`handleDisconnection` Callback Method

```
void handleDisconnection(
  String destination)
```

Note	`handleDisconnection` is a part of MetadataUpdater contract.

`handleDisconnection` ...FIXME

`handleFatalException` Callback Method

```
void handleFatalException(
  KafkaException fatalException)
```

Note	`handleFatalException` is a part of MetadataUpdater contract.

`handleFatalException` ...FIXME

`maybeUpdate` Method

```
long maybeUpdate(
  long now)
```

Note	`maybeUpdate` is part of the MetadataUpdater contract to...FIXME.

`maybeUpdate` ...FIXME

`maybeUpdate` Internal Method

```
long maybeUpdate(
  long now,
  Node node)
```

`maybeUpdate` ...FIXME

Note	`maybeUpdate` is used when `DefaultMetadataUpdater` is requested to maybeUpdate.

Scheduling Update Of Cluster Metadata — `requestUpdate` Method

```
void requestUpdate()
```

Note	`requestUpdate` is part of the MetadataUpdater Contract to schedule an update of the current cluster metadata.

`requestUpdate` simply requests the Metadata for a metadata update.

`hasFetchInProgress` Internal Method

```
boolean hasFetchInProgress()
```

`hasFetchInProgress` ...FIXME

Note	`hasFetchInProgress` is used when...FIXME

`isAnyNodeConnecting` Internal Method

```
boolean isAnyNodeConnecting()
```

`isAnyNodeConnecting` ...FIXME

Note	`isAnyNodeConnecting` is used when...FIXME

Internal Properties

Name	Description
`inProgressRequestVersion`	Defined if there is a request in progress, `null` otherwise. Used in hasFetchInProgress and handleCompletedMetadataResponse Initialized to be `null` when `DefaultMetadataUpdater` is created Set to the requestVersion of the Metadata when `DefaultMetadataUpdater` is requested to maybeUpdate Reset to `null` when `DefaultMetadataUpdater` is requested to: • handleDisconnection • handleFatalException • handleCompletedMetadataResponse
`metadataFetchInProgress`	Flag to control whether a cluster metadata update is in progress, i.e. FIXME • Disabled when `DefaultMetadataUpdater` is created • Turned on exclusively when `DefaultMetadataUpdater` does maybeUpdate (with a timestamp and a broker node) • Turned off when `DefaultMetadataUpdater` handles completed metadata response, disconnection or authentication failure

Metadata

`Metadata` describes a Kafka cluster (and is created) for KafkaConsumer and KafkaProducer.

Table 1. Metadata's Properties When Created by Clients

Client	refreshBackoffMs	metadataExpireMs	allowAutoTopicCreat
`KafkaConsumer`	retry.backoff.ms	metadata.max.age.ms	on
`KafkaProducer`	retry.backoff.ms	metadata.max.age.ms	on

A (*seemingly*) common usage pattern is as follows:

1. Request `Metadata` for a update (that simply turns the `needUpdate` flag on)
2. Request (indirectly) `KafkaClient` to wake up if blocked on I/O
3. Request `Metadata` to wait for metadata change (i.e. until the metadata version has changed)

`Metadata` manages metadata update listeners that want to be notified about metadata updates. Listeners can be registered and deregistered.

Note	ConsumerCoordinator is the only entity to register a metadata update listener.

Tip	Enable `INFO`, `DEBUG` or `TRACE` logging levels for `org.apache.kafka.clients.Metadata` logger to see what happens inside. Add the following line to `log4j.properties`: `log4j.logger.org.apache.kafka.clients.Metadata=TRACE` Refer to Logging.

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 authenticationExcepti
on)
```

`failedUpdate` ...FIXME

Note	`failedUpdate` is used when...FIXME

`getClusterForCurrentTopics` Internal Method

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

`getClusterForCurrentTopics` ...FIXME

Note	`getClusterForCurrentTopics` is used exclusively when `Metadata` is requested to update.

`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

Requesting Metadata Update — `requestUpdate` Method

```
int requestUpdate()
```

`requestUpdate` simply turns the needUpdate flag on and returns the current version.

Note	`requestUpdate` is used when: • `Metadata` is requested to requestUpdateForNewTopics • `DefaultMetadataUpdater` is requested to requestUpdate • `KafkaConsumer` is requested to subscribe • `ConsumerCoordinator` is created, addMetadataListener, poll • `ConsumerNetworkClient` is requested to awaitMetadataUpdate • `Fetcher` is requested to resetOffsetsAsync, groupListOffsetRequests, prepareFetchRequests, parseCompletedFetch • `KafkaProducer` is requested to wait for cluster metadata • `Sender` is requested to sendProducerData, maybeSendTransactionalRequest, maybeWaitForProducerId, and completeBatch

Waiting for Metadata Update (i.e. Metadata Version Change) — `awaitUpdate` Method

```
synchronized void awaitUpdate(
  final int lastVersion,
  final long maxWaitMs) throws InterruptedException
```

`awaitUpdate` ...FIXME

Note	`awaitUpdate` is used when...FIXME

Getting Current Cluster Metadata — `fetch` Method

```
synchronized Cluster fetch()
```

`fetch` returns current cluster information.

Note	`fetch` is used when…FIXME

Setting Topics to Maintain — `setTopics` Method

Caution	FIXME

Updating Cluster Metadata — `update` Method

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

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

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

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

`update` notifies the Metadata.Listeners that the metadata has been updated.

`update` does getClusterForCurrentTopics for the `cluster` when needMetadataForAllTopics flag is on and turns needUpdate flag off (that may have been turned on…FIXME).

`update` sets the cluster to the given `Cluster` when the needMetadataForAllTopics flag is off.

`update` prints out the following INFO message to the logs with the cluster ID and notifies clusterResourceListeners that cluster has changed (only for a non-bootstrap cluster).

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

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

```
Updated cluster metadata version [version] to [cluster]
```

Note	`update` is used when: • `DefaultMetadataUpdater` is requested to handle a MetadataResponse • KafkaConsumer and KafkaProducer are created (and request updating the cluster metadata with a "bootstrap" cluster, i.e. with the bootstrap servers only)

Creating Metadata Instance

`Metadata` takes the following when created:

- `refreshBackoffMs`
- `metadataExpireMs`
- `allowAutoTopicCreation` flag
- `topicExpiryEnabled` flag
- ClusterResourceListeners

`Metadata` initializes the internal registries and counters.

Conditionally Requesting Update For New Topics (for KafkaConsumer) — `needMetadataForAllTopics` Method

```
synchronized void needMetadataForAllTopics(boolean needMetadataForAllTopics)
```

`needMetadataForAllTopics` requestUpdateForNewTopics when the input `needMetadataForAllTopics` flag is enabled (i.e. `true`) and the current needMetadataForAllTopics is disabled (i.e. `false`).

`needMetadataForAllTopics` sets needMetadataForAllTopics to be the input `needMetadataForAllTopics` .

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

Registering Metadata.Listener — `addListener` Method

```
void addListener(Listener listener)
```

`addListener` simply adds the given Listener to the listeners internal registry.

Note	`addListener` is used exclusively when `ConsumerCoordinator` is requested to addMetadataListener (when created).

`newMetadataRequestBuilder` Method

```
MetadataRequest.Builder newMetadataRequestBuilder()
```

`newMetadataRequestBuilder` simply requests the `MetadataRequest` to create MetadataRequest.Builder for all topics.

Note	`newMetadataRequestBuilder` is used exclusively when `Metadata` is requested to newMetadataRequestAndVersion.

`newMetadataRequestAndVersion` Method

```
MetadataRequestAndVersion newMetadataRequestAndVersion()
```

`newMetadataRequestAndVersion` simply creates a new `MetadataRequestAndVersion` with MetadataRequest.Builder and the requestVersion.

Note	`newMetadataRequestAndVersion` is used exclusively when `DefaultMetadataUpdater` is requested to maybeUpdate.

Internal Properties

Name	Description

`cluster`	Cluster with a subset of the nodes and topic partitions in a Kafka cluster.
	• Empty (with no nodes and no topic partitions) when `Metadata` is created
	• Updated when:
	  ◦ `DefaultMetadataUpdater` handles MetadataResponse
	  ◦ `KafkaConsumer`, `KafkaProducer`, `KafkaAdminClient` and `AdminClient` are created and update the cluster with a "bootstrap" cluster with bootstrap brokers
	• Can be accessed using fetch

`lastRefreshMs`	The time (in millis) of the last successful update (and failed update) • Used in timeToNextUpdate • Starts `0` when `Metadata` is created • Reset (to `0`) in requestUpdateForNewTopics
`lastSuccessfulRefreshMs`	
`needMetadataForAllTopics`	Flag…FIXME • Disabled (i.e. `false`) when `Metadata` is created • Updated when `Metadata` is requested to set state to indicate that metadata for all topics in Kafka cluster is required
`needUpdate`	Flag that controls whether a metadata update has been requested (enabled) or not (disabled). • Starts turned off when `Metadata` is created • Turned on exclusively when `Metadata` is requested for an update • Turned off when `Metadata` is updated

	Use updateRequested to know the current value.
`requestVersion`	
`version`	Metadata version • `0` when `Metadata` is created • Incremented every update

Listener Contract — Intercepting Metadata Updates

`Listener` is the contract of…FIXME

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

public final class Metadata {
  public interface Listener {
    void onMetadataUpdate(Cluster cluster, Set<String> unavailableTopics);
  }
}
```

ClientRequest

`ClientRequest` is…FIXME

`ClientRequest` is created exclusively when `NetworkClient` is requested to create one.

`ClientRequest` uses the client ID when requested to makeHeader.

Creating ClientRequest Instance

`ClientRequest` takes the following when created:

- Destination
- AbstractRequest.Builder
- Correlation ID
- Client ID
- `createdTimeMs`
- `expectResponse` flag
- `requestTimeoutMs`
- RequestCompletionHandler

`ClientRequest` initializes the internal registries and counters.

Creating RequestHeader — `makeHeader` Method

```
RequestHeader makeHeader(short version)
```

`makeHeader` creates a `RequestHeader` (with the client and the correlation IDs)

Note	`makeHeader` is used when: • `InterBrokerSendThread` is requested to completeWithDisconnect • `NetworkClient` is requested to doSend • `ConsumerNetworkClient` is requested to checkDisconnects

ClientResponse

`ClientResponse` is...FIXME

`onComplete` Method

```
void onComplete()
```

`onComplete` triggers RequestCompletionHandler if defined.

Note	`onComplete` is used exclusively when `NetworkClient` is requested to completeResponses (when `NetworkClient` is requested to poll).

StaleMetadataException

`StaleMetadataException` is...FIXME

NetworkClientUtils

`NetworkClientUtils` is…FIXME

`sendAndReceive` Static Method

```
ClientResponse sendAndReceive(
  KafkaClient client
  ClientRequest request
  Time time) throws IOException
```

`sendAndReceive` …FIXME

Note	`sendAndReceive` is used when…FIXME

Waiting Until Connection to Broker Node is Ready — `awaitReady` Static Method

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

`awaitReady` …FIXME

Note	`awaitReady` is used when…FIXME

`isReady` Static Method

```
boolean isReady(
  KafkaClient client,
  Node node,
  long currentTime)
```

`isReady` …FIXME

Note	`isReady` is used when…FIXME

CommonClientConfigs — Common Configuration Properties for Kafka Clients

`CommonClientConfigs` is the configuration properties for Kafka consumers and producers.

Table 1. CommonClientConfigs' Configuration Properties

Name	Description
`security.protocol`	Protocol for communication with brokers Valid values are: `PLAINTEXT`, `SSL`, `SASL_PLAINTEXT`, `SASL_SSL`. Default: `PLAINTEXT` Used when `ClientUtils` utility is used to createChannelBuilder

ClientUtils

`ClientUtils` is...FIXME

`createChannelBuilder` Utility

```
ChannelBuilder createChannelBuilder(
  AbstractConfig config,
  Time time)
```

`createChannelBuilder` reads security.protocol (for `SecurityProtocol`) and `sasl.mechanism` configuration properties to create a ChannelBuilder (with `CLIENT` JAAS context type and `null` listener name).

Note	`createChannelBuilder` is used when: • KafkaConsumer is created • KafkaProducer is created • Kafka Connect's `WorkerGroupMember` is created • `KafkaAdminClient` is requested to createInternal (for Kafka administration utilities) • BrokerApiVersionsCommand and ReplicaVerificationTool tools are executed

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 **boostrap cluster** that is made up of the bootstrap brokers that are mandatory (and specified explicitly) when Kafka clients are created, i.e. KafkaAdminClient, AdminClient, KafkaConsumer and KafkaProducer.

Note	A bootstrap cluster does not hold all information about the cluster.

`Cluster` is created when:

- `AdminMetadataManager` is requested to clearController
- `Cluster` is requested to create an empty Cluster metadata and Cluster metadata with given partitions, and bootstrap
- `MetadataCache` is requested to getClusterMetadata
- `Metadata` is requested to getClusterForCurrentTopics
- `MetadataResponse` is requested to cluster

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

Name	Description
`isBootstrapConfigured`	Flag…FIXME Used when…FIXME
`partitionsByTopic`	

`bootstrap` Factory Method

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

`bootstrap` …FIXME

Note	`bootstrap` is used when KafkaAdminClient, KafkaConsumer and KafkaProducer are created (and update their internal metadata).

`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	`partitionsForTopic` is used when: • `Metadata` getClusterForCurrentTopics • `KafkaAdminClient` describeTopics • `KafkaConsumer` partitionsFor • `KafkaProducer` requests partitions for a topic • `DefaultPartitioner` assigns the partition for a record

`availablePartitionsForTopic` Method

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

`availablePartitionsForTopic` …FIXME

Note	`availablePartitionsForTopic` is used when…FIXME

Creating Cluster Instance

`Cluster` takes the following when created:

- **Cluster ID**
- `isBootstrapConfigured` flag

- Kafka Nodes (`Collection<Node>`)
- Kafka PartitionInfos (`Collection<PartitionInfo>`)
- Names of the **unauthorized topics**
- Names of the **invalid topics**
- Names of the **internal topics**
- **Controller** (as a Kafka Node)

`Cluster` initializes the internal registries and counters.

Creating Empty Cluster Metadata — `empty` Factory Method

```
static Cluster empty()
```

`empty` ...FIXME

Note	`empty` is used when: • `Metadata` is created • `AdminMetadataManager` is created

Creating Cluster Metadata With Given Partitions — `withPartitions` Method

```
Cluster withPartitions(Map<TopicPartition, PartitionInfo> partitions)
```

`withPartitions` ...FIXME

`withPartitions` is used exclusively when Kafka Streams' `StreamsPartitionAssignor` is requested to `assign` and `onAssignment` .

Cluster (deprecated)

Important	It *seems* that `Cluster` class is created using ZkUtils.getCluster that is used exclusively when `ZKRebalancerListener` does syncedRebalance (that in turn happens for the currently-deprecated `ZookeeperConsumerConnector`). In other words, `Cluster` class and the page are *soon* to be removed.

`kafka.cluster.Cluster` private class represents a set of active brokers in a Kafka cluster.

Note	There is also `org.apache.kafka.common.Cluster` .

`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

TopicConfig — Topic-Level Configuration Keys

`TopicConfig` are the keys for a **topic-level configuration** of a topic (and override cluster-wide defaults that are mostly different by names with `log.` prefix).

Note	As the TopicConfig itself says: "*Eventually this should replace LogConfig.scala.*"

Import `org.apache.kafka.common.config.TopicConfig` to reference the keys.

```
import org.apache.kafka.common.config.TopicConfig
assert(TopicConfig.CLEANUP_POLICY_CONFIG == "cleanup.policy")
```

Table 1. Keys

Name	Description
`CLEANUP_POLICY_CONFIG`	cleanup.policy Used when: • `LogConfig` is requested for LogConfig.CleanupPolicyProp • (Kafka Streams) `RepartitionTopicCon` requested for `REPARTITION_TOPIC_DEFA` • (Kafka Streams) `UnwindowedChangelog` requested for `UNWINDOWED_STORE_CHANGELOG_TOPIC_DEF` • (Kafka Streams) `WindowedChangelogTo` requested for `WINDOWED_STORE_CHANGELOG_TOPIC_DEFAU`
`COMPRESSION_TYPE_CONFIG`	compression.type
`DELETE_RETENTION_MS_CONFIG`	delete.retention.ms
`FILE_DELETE_DELAY_MS_CONFIG`	file.delete.delay.ms
`FLUSH_MESSAGES_INTERVAL_CONFIG`	flush.messages
`FLUSH_MS_CONFIG`	flush.ms
`INDEX_INTERVAL_BYTES_CONFIG`	index.interval.bytes
`MAX_COMPACTION_LAG_MS_CONFIG`	max.compaction.lag.ms

`MAX_MESSAGE_BYTES_CONFIG`	max.message.bytes
`MESSAGE_DOWNCONVERSION_ENABLE_CONFIG`	message.downconversion.enable
`MESSAGE_FORMAT_VERSION_CONFIG`	message.format.version
`MESSAGE_TIMESTAMP_DIFFERENCE_MAX_MS_CONFIG`	message.timestamp.difference.max.ms
`MESSAGE_TIMESTAMP_TYPE_CONFIG`	message.timestamp.type
`MIN_CLEANABLE_DIRTY_RATIO_CONFIG`	min.cleanable.dirty.ratio
`MIN_COMPACTION_LAG_MS_CONFIG`	min.compaction.lag.ms
`MIN_IN_SYNC_REPLICAS_CONFIG`	min.insync.replicas Used when: • TopicCommand is executed • Alias for LogConfig.MinInSyncReplica
`PREALLOCATE_CONFIG`	preallocate
`RETENTION_BYTES_CONFIG`	Maximum size (in bytes) of a partition to g discarding old log segments and free up s with delete retention policy. Default: `-1L` (no size limit) By default there is no size limit only a time this limit is enforced at the partition level, r the number of partitions to compute the to bytes. Alias of LogConfig.RetentionBytesProp Cluster-wide property: log.retention.bytes Used exclusively when `Log` is requested deleteRetentionSizeBreachedSegments
`RETENTION_MS_CONFIG`	retention.ms
`SEGMENT_BYTES_CONFIG`	LogConfig.SegmentBytesProp Used when: • (Kafka Streams) `StreamsConfig` is rec `getMainConsumerConfigs` • (Kafka Streams) `RepartitionTopicCon` requested for `REPARTITION_TOPIC_DEFA`

`SEGMENT_INDEX_BYTES_CONFIG`	LogConfig.SegmentIndexBytesProp
`SEGMENT_JITTER_MS_CONFIG`	segment.jitter.ms
`SEGMENT_MS_CONFIG`	segment.ms
`UNCLEAN_LEADER_ELECTION_ENABLE_CONFIG`	unclean.leader.election.enable

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
`processNotification`	Handles a change notification Used exclusively when `ZkNodeChangeNotificationListener` is requested to processNotification

Table 2. NotificationHandlers

NotificationHandler	Description
rawHandler	
ConfigChangedNotificationHandler	
TokenChangedNotificationHandler	

ZkNodeChangeNotificationListener

`ZkNodeChangeNotificationListener` is…FIXME

`processNotification` Internal Method

```
processNotification(notification: String): Unit
```

`processNotification` …FIXME

Note	`processNotification` is used when…FIXME

`init` Method

```
init(): Unit
```

`init` …FIXME

Note	`init` is used when…FIXME

`close` Method

```
close(): Unit
```

`close` …FIXME

Note	`close` is used when…FIXME

Configurable Contract

`Configurable` is the abstraction of classes that are instantiated by reflection and need to take configuration parameters.

Table 1. Configurable Contract

Method	Description

`configure`	`void configure(`
	`  Map<String, ?> configs)`
	
	Configures the class with the given key-value configurations

Reconfigurable Contract

`Reconfigurable` is the abstraction of reconfigurables that support dynamic reconfiguration.

Table 1. Reconfigurable Contract

Method	Description
`reconfigurableConfigs`	`Set<String> reconfigurableConfigs()` Used when…FIXME
`reconfigure`	`void reconfigure(Map<String, ?> configs)` Reconfigures the Reconfigurable Used when `DynamicBrokerConfig` is requested to maybeReco processReconfigurable.
`validateReconfiguration`	`void validateReconfiguration(Map<String, ?> configs) throw` Used when…FIXME

Table 2. Reconfigurables (Direct Implementations)

Reconfigurable	Description
DynamicClientQuotaCallback	
DynamicLogConfig	
DynamicMetricsReporters	
`ListenerReconfigurable`	
`SslFactory`	

MemoryRecordsBuilder

`MemoryRecordsBuilder` is...FIXME

Records

`Records` is the extension of the `BaseRecords` contract for FIXME that FIXME.

Table 1. Records Contract

Method	Description
`batches`	`Iterable<? extends RecordBatch> batches()`
`batchIterator`	`AbstractIterator<? extends RecordBatch> batchIterator()`
`downConvert`	`ConvertedRecords<? extends Records> downConvert(byte toMagic, long firstOffset, Time time)`
`hasCompatibleMagic`	`boolean hasCompatibleMagic(byte magic)`
`hasMatchingMagic`	`boolean hasMatchingMagic(byte magic)`
`records`	`Iterable<Record> records()`
`writeTo`	`long writeTo(GatheringByteChannel channel, long position, int length) throws IOException`

Note	AbstractRecords is the default and only known `Records` extension in Apache Kafka.

FileRecords

`FileRecords` is a concrete AbstractRecords…FIXME

`FileRecords` is created when…FIXME

Creating FileRecords Instance

`FileRecords` takes the following to be created:

- Underlying `File`
- Underlying `FileChannel`
- Start
- End
- `isSlice` flag

`FileRecords` initializes the internal properties.

Opening Log File — `open` Utility

```
FileRecords open(
  File file) throws IOException (1)
FileRecords open(
  File file,
  boolean mutable) throws IOException (2)
FileRecords open(
  File file,
  boolean mutable,
  boolean fileAlreadyExists,
  int initFileSize,
  boolean preallocate) throws IOException
FileRecords open(
  File file,
  boolean fileAlreadyExists,
  int initFileSize,
  boolean preallocate) throws IOException
```

1. Turns `mutable` flag on (`true`)
2. Turns `fileAlreadyExists` flag off (`false`), `initFileSize` as `0` , and `preallocate` as `false`

`open` opens a FileChannel.

`open` determines the end of a `FileRecords` to be created:

- `0` for the given `fileAlreadyExists` flag disabled (`false`) and the given `preallocate` flag enabled (`true`)
- `Integer.MAX_VALUE` otherwise

In the end, `open` creates a new `FileRecords` for the given `file` , the channel, and the isSlice flag disabled (`false`).

Note	`open` is used when: • `LogSegment` utility is used to open a log segment • DumpLogSegments tool is executed

Reading Log Batches (Into ByteBuffer) — `readInto` Method

```
void readInto(
  ByteBuffer buffer,
  int position) throws IOException
```

`readInto` ...FIXME

Note	`readInto` is used when: • `GroupMetadataManager` is requested to doLoadGroupsAndOffsets • `TransactionStateManager` is requested to loadTransactionMetadata • `Cleaner` is requested to cleanInto and buildOffsetMapForSegment • `LogSegment` is requested to appendChunkFromFile • `AbstractFetcherThread` is requested to toMemoryRecords

`sizeInBytes` Method

```
int sizeInBytes()
```

Note	`sizeInBytes` is part of the BaseRecords contract for the size of the records (in bytes).

`sizeInBytes` simply the current value of the size internal registry.

Opening FileChannel — `openChannel` Internal Method

```
FileChannel openChannel(
  File file,
  boolean mutable,
  boolean fileAlreadyExists,
  int initFileSize,
  boolean preallocate) throws IOException
```

`openChannel` …FIXME

Note	`openChannel` is used when `FileRecords` utility is requested to open a log file.

Internal Properties

Name	Description
`batches`	(`Iterable<FileChannelRecordBatch>`)
`size`	Size (java.util.concurrent.atomic.AtomicInteger) that is `end - start` for isSlice enabled. Used when…FIXME

MemoryRecords

`MemoryRecords` is…FIXME

AbstractRecords

`AbstractRecords` is the base implementation of the Records contract for records (that… FIXME)?

Table 1. AbstractRecords

AbstractRecords	Description
FileRecords	On-Disk Representation
MemoryRecords	In-Memory Representation

BaseRecords

`BaseRecords` is…FIXME

AbstractRequestResponse Contract

`AbstractRequestResponse` is the abstraction of messages exchanged between Kafka clients and brokers.

Note	`AbstractRequestResponse` is a Java abstract class and cannot be created directly. It is created indirectly for the concrete AbstractRequestResponses.

`AbstractRequestResponse` defines the single `serialize` static method.

```
ByteBuffer serialize(Struct headerStruct, Struct bodyStruct)
```

`serialize` simply allocates a new java.nio.ByteBuffer for the given header and body structs followed by writing them out to the buffer (and rewinding the buffer).

`serialize` is used when AbstractRequest and AbstractResponse are requested to serialize.

Table 1. AbstractRequestResponses (Direct Extensions and Implementations)

AbstractRequestResponse	Description
AbstractRequest	
AbstractResponse	
RequestHeader	
ResponseHeader	

AbstractRequest Contract

`AbstractRequest` is the extension of the AbstractRequestResponse contract for message requests.

`AbstractRequest` takes the following to be created:

- `ApiKeys`
- Version

Note	`AbstractRequest` is a Java abstract class and cannot be created directly. It is created indirectly for the concrete AbstractRequests.

Table 1. AbstractRequest Contract

Method	Description
`getErrorResponse`	`AbstractResponse getErrorResponse(int throttleTimeMs, Throwable e)` Used when…FIXME
`toStruct`	`Struct toStruct()` Used when…FIXME

Table 2. AbstractRequests (Direct Implementations and Extensions Only)

AbstractRequest	Description
AbstractControlRequest	Base of controller requests that KafkaController uses to propage broker and partition state changes to brokers
AddOffsetsToTxnRequest	
AddPartitionsToTxnRequest	
AlterConfigsRequest	
AlterReplicaLogDirsRequest	
ApiVersionsRequest	

ControlledShutdownRequest	
CreateAclsRequest	
CreateDelegationTokenRequest	
CreatePartitionsRequest	
CreateTopicsRequest	
DeleteAclsRequest	
DeleteGroupsRequest	
DeleteRecordsRequest	
DeleteTopicsRequest	
DescribeAclsRequest	
DescribeConfigsRequest	
DescribeDelegationTokenRequest	
DescribeGroupsRequest	
DescribeLogDirsRequest	
ElectPreferredLeadersRequest	
EndTxnRequest	
ExpireDelegationTokenRequest	
FetchRequest	
FindCoordinatorRequest	
HeartbeatRequest	
IncrementalAlterConfigsRequest	
InitProducerIdRequest	
JoinGroupRequest	

LeaderAndIsrRequest	
LeaveGroupRequest	
ListGroupsRequest	
ListOffsetRequest	
MetadataRequest	
OffsetCommitRequest	
OffsetFetchRequest	
OffsetsForLeaderEpochRequest	
ProduceRequest	
RenewDelegationTokenRequest	
SaslAuthenticateRequest	
SaslHandshakeRequest	
SyncGroupRequest	
TxnOffsetCommitRequest	
WriteTxnMarkersRequest	

Creating AbstractRequest For API Key — `parseRequest` Factory Method

```
AbstractRequest parseRequest(
  ApiKeys apiKey,
  short apiVersion,
  Struct struct)
```

`parseRequest` simply creates a concrete AbstractRequest for the given `ApiKeys`.

Note	`parseRequest` is used exclusively when `RequestContext` is requested to parse a request.

AbstractRequest.Builder Contract — Request Builders

`AbstractRequest.Builder` is the abstraction of request builders that can build a request.

Table 1. AbstractRequest.Builder Contract

Method	Description
`build`	`T build(short version)` Builds a message request (of type `T`) Used when: • `NetworkClient` is requested to doSend • `ReplicaAlterLogDirsThread` is requested to fetchFromLeader • `OffsetFetchRequest` is requested to `forAllPartitions` • `SaslClientAuthenticator` is requested to `createSaslHandshakeRequest` and `sendSaslClientToken`

AbstractResponse

`AbstractResponse` is...FIXME

DescribeLogDirsRequest

`DescribeLogDirsRequest` is…FIXME

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

Name	Description
`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

DescribeConfigsRequest

`DescribeConfigsRequest` is a message request with **DescribeConfigs** API key and the following properties:

- Config resources with names (`Map<ConfigResource, Collection<String>>`)
- `includeSynonyms` flag
- Version

`DescribeConfigsRequest` is used exclusively by Kafka administration utilities (via `KafkaAdminClient`) to describe the configuration of specified resources which is handled by KafkaApis (on the broker).

`DescribeConfigsRequest` is created when:

- `DescribeConfigsRequest` is requested to parse a byte buffer
- `AbstractRequest` is requested to parse a request (with the DescribeConfigs API key)
- `DescribeConfigsRequest.Builder` is requested to build a DescribeConfigsRequest

`DescribeConfigsRequest.Builder` Utility

`DescribeConfigsRequest.Builder` is a request builder to build a DescribeConfigsRequest.

```
DescribeConfigsRequest build(short version)
```

`DescribeConfigsRequest.Builder` is used when…FIXME

Creating DescribeConfigsRequest from Byte Buffer — `parse` Utility

```
DescribeConfigsRequest parse(ByteBuffer buffer, short version)
```

`parse` …FIXME

Note	`parse` is used when…FIXME

ElectPreferredLeadersRequest

`ElectPreferredLeadersRequest` is a message request that is sent out from KafkaAdminClient when requested to elect preferred partition leaders (e.g. using kafka-leader-election.sh shell script) to the controller broker.

`ElectPreferredLeadersRequest` uses **ElectPreferredLeaders** API key and the following properties:

- ElectPreferredLeadersRequestData
- Version

`ElectPreferredLeadersRequest` is created when:

- `AbstractRequest` is requested to parse a request (with the ElectPreferredLeaders API key)
- `ElectPreferredLeadersRequest` is requested to parse a byte buffer
- `ElectPreferredLeadersRequest.Builder` is requested to build a ElectPreferredLeadersRequest

ElectPreferredLeadersRequestData

`ElectPreferredLeadersRequestData` is an API message with the following properties:

- Topic partitions (a topic and partition IDs) to elect the preferred leader of
- Time (ms) to wait for the election to complete

`ElectPreferredLeadersRequest.Builder` Factory Object

`ElectPreferredLeadersRequest.Builder` is a request builder to build a `ElectPreferredLeadersRequest`.

```
ElectPreferredLeadersRequest build(short version)
```

`toRequestData` Object Method

```
ElectPreferredLeadersRequestData toRequestData(
  Collection<TopicPartition> partitions,
  int timeoutMs)
```

`toRequestData` ...FIXME

Note	`toRequestData` is used when...FIXME

`fromResponseData` Object Method

```
Map<TopicPartition, ApiError> fromResponseData(
  ElectPreferredLeadersResponseData data)
```

`fromResponseData` ...FIXME

Note	`fromResponseData` is used when...FIXME

Creating ElectPreferredLeadersRequest from Byte Buffer — `parse` Object Method

```
ElectPreferredLeadersRequest parse(
  ByteBuffer buffer,
  short version)
```

`parse` ...FIXME

Note	`parse` is used when...FIXME

FetchRequest

`FetchRequest` is a message request with **Fetch** API key and the following properties:

- Version
- Replica Id
- maxWait
- minBytes
- maxBytes
- Fetch Data (`Map<TopicPartition, PartitionData>`)
- `IsolationLevel`
- `TopicPartitions` to forget (`List<TopicPartition>`)
- `FetchMetadata`
- Rack Id

`FetchRequest` is created when:

- `FetchRequest.Builder` is requested to build a FetchRequest
- `AbstractRequest` is requested to parse a request (with the Fetch API key)
- `FetchRequest` is requested to parse a byte buffer

`FetchRequest.Builder` Utility

`FetchRequest.Builder` is a request builder to build a FetchRequest.

```
FetchRequest build(
  short version)
```

`FetchRequest.Builder` is used when…FIXME

Creating FetchRequest.Builder for Kafka Consumers — `FetchRequest.Builder.forConsumer` Utility

```
Builder forConsumer(
  int maxWait,
  int minBytes,
  Map<TopicPartition, PartitionData> fetchData)
```

`forConsumer` ...FIXME

Note	`forConsumer` is used when Fetcher is requested to sendFetches (for `KafkaConsumer` to poll for records).

Creating FetchRequest.Builder for Partition Replicas — `FetchRequest.Builder.forReplica` Utility

```
Builder forReplica(
  short allowedVersion,
  int replicaId,
  int maxWait,
  int minBytes,
  Map<TopicPartition, PartitionData> fetchData)
```

`forReplica` simply creates a new FetchRequest.Builder (with the min and max versions exactly the given `allowedVersion`).

Note	`forReplica` is used when: • `ReplicaAlterLogDirsThread` is requested to buildFetchForPartition • `ReplicaFetcherThread` is requested to buildFetch • ReplicaVerificationTool is executed

Creating FetchRequest from Byte Buffer — `parse` Utility

```
FetchRequest parse(
  ByteBuffer buffer,
  short version)
```

`parse` ...FIXME

Note	`parse` is used when...FIXME

FindCoordinatorRequest

`FindCoordinatorRequest` is a message request with **FindCoordinator** API key and the following properties:

- Coordinator type (i.e. `GROUP` or `TRANSACTION`)
- Coordinator key
- Version

`FindCoordinatorRequest` is created when:

- `AbstractRequest` is requested to parse a request (with the FindCoordinator API key)
- `FindCoordinatorRequest` is requested to parse a byte buffer
- `FindCoordinatorRequest.Builder` is requested to build a FindCoordinatorRequest

`FindCoordinatorRequest.Builder` Utility

`FindCoordinatorRequest.Builder` is a request builder to build a FindCoordinatorRequest.

```
FindCoordinatorRequest build(short version)
```

`FindCoordinatorRequest.Builder` is used when…FIXME

Creating FindCoordinatorRequest from Byte Buffer — `parse` Utility

```
FindCoordinatorRequest parse(ByteBuffer buffer, short version)
```

`parse` …FIXME

Note	`parse` is used when…FIXME

FindCoordinatorResponse

`FindCoordinatorResponse` is...FIXME

HeartbeatRequest

`HeartbeatRequest` is a concrete AbstractRequest with **Heartbeat** API key and the following:

- Group ID
- Group Generation ID
- Member ID

`HeartbeatRequest` is created when:

- `AbstractRequest` is requested to parse a request (with the Heartbeat API key)
- `HeartbeatRequest` is requested to parse a byte buffer
- `HeartbeatRequest.Builder` is requested to build a HeartbeatRequest

`HeartbeatRequest.Builder` Factory Object

`HeartbeatRequest.Builder` is a concrete AbstractRequest.Builder factory object that can build a HeartbeatRequest.

```
HeartbeatRequest build(short version)
```

Creating HeartbeatRequest from Byte Buffer — `parse` Factory Method

```
HeartbeatRequest parse(ByteBuffer buffer, short version)
```

`parse` …FIXME

Note	`parse` is used when…FIXME

JoinGroupRequest

`JoinGroupRequest` is a concrete AbstractRequest with **JoinGroup** API key and the following:

- Version
- Group ID
- Session timeout
- Rebalance timeout
- Member ID
- Protocol type (e.g. `consumer`)
- Supported `ProtocolMetadata`

`JoinGroupRequest` is created when:

- `AbstractRequest` is requested to parse a request (with the JoinGroup API key)
- `JoinGroupRequest` is requested to parse a byte buffer
- `JoinGroupRequest.Builder` is requested to build a JoinGroupRequest

`JoinGroupRequest.Builder` Factory Object

`JoinGroupRequest.Builder` is a concrete AbstractRequest.Builder factory object that can build a JoinGroupRequest.

```
JoinGroupRequest build(short version)
```

Creating JoinGroupRequest from Byte Buffer — `parse` Factory Method

```
JoinGroupRequest parse(ByteBuffer buffer, short version)
```

`parse` …FIXME

Note	`parse` is used when…FIXME

`getErrorResponse` Method

```
AbstractResponse getErrorResponse(int throttleTimeMs, Throwable e)
```

Note	`getErrorResponse` is part of the AbstractRequest Contract to create a AbstractResponse for a failure.

`getErrorResponse` ...FIXME

JoinGroupResponse

`JoinGroupResponse` is a concrete AbstractResponse that…FIXME

`JoinGroupResponse` is created when:

- `JoinGroupRequest` is requested to getErrorResponse
- `KafkaApis` is requested to handle a JoinGroupRequest
- `AbstractResponse` is requested to parse a JoinGroup response
- `JoinGroupResponse` is requested to parse a ByteBuffer

`JoinGroupResponse` takes the following to be created:

- Throttle time (default: `0` ms)
- `Errors`
- Generation ID
- Group protocol
- Member ID
- Leader ID
- Group members (`Map<String, ByteBuffer>`)

Creating JoinGroupResponse from ByteBuffer — `parse` Factory Method

```
JoinGroupResponse parse(ByteBuffer buffer, short version)
```

`parse` …FIXME

Note	`parse` is used when…FIXME

MetadataRequest

`MetadataRequest` is a request with **Metadata** API key and the following properties:

- `MetadataRequestData`
- Version

`MetadataRequest` is created when:

- `MetadataRequest.Builder` is requested to build a MetadataRequest
- `MetadataRequest` is requested to parse a byte buffer
- `AbstractRequest` is requested to parse a request (with the Metadata API key)

`MetadataRequest` can be created using the following factory methods:

- MetadataRequest.Builder.allTopics
- MetadataRequest.Builder

```
import org.apache.kafka.common.requests.MetadataRequest

val r = MetadataRequest.Builder.allTopics
scala> println(r)
(type=MetadataRequest, topics=<ALL>)

import collection.JavaConverters._
val topics = Seq("t1", "t2").asJava
val allowAutoTopicCreation = false
val r = new MetadataRequest.Builder(topics, allowAutoTopicCreation)
scala> println(r)
(type=MetadataRequest, topics=t1,t2)
```

Create MetadataRequest.Builder for All Topics — `allTopics` Static Factory Method

```
// MetadataRequest.Builder.allTopics
Builder allTopics()
```

`allTopics` creates a new MetadataRequest for all topics and `allowAutoTopicCreation` flag enabled.

Note	`allTopics` is used when: • `Metadata` is requested to newMetadataRequestBuilder • `KafkaAdminClient` is requested to listTopics and describeTopics • `ConsumerMetadata` is requested to `newMetadataRequestBuilder` • `Fetcher` is requested to getAllTopicMetadata

`MetadataRequest.Builder` Utility

`MetadataRequest.Builder` is a AbstractRequest.Builder to build a MetadataRequest.

```
MetadataRequest build(short version)
```

`UpdateMetadataRequest.Builder` is used when…FIXME

Creating MetadataRequest from Byte Buffer — `parse` Utility

```
MetadataRequest parse(ByteBuffer buffer, short version)
```

`parse` …FIXME

Note	`parse` is used when…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.

OffsetCommitRequest

`OffsetCommitRequest` is a concrete AbstractRequest with **OffsetCommit** API key and the following:

- Group ID
- Generation ID
- Member ID
- Retention time
- Offset data (`Map<TopicPartition, PartitionData>`)
- Version

`OffsetCommitRequest` is used exclusively when `ConsumerCoordinator` is requested to send an OffsetCommitRequest to the group coordinator (a Kafka broker).

`OffsetCommitRequest` is created when:

- `AbstractRequest` is requested to parse a request (with the OffsetCommit API key)
- `OffsetCommitRequest` is requested to parse a byte buffer
- `OffsetCommitRequest.Builder` is requested to build an OffsetCommitRequest

`OffsetCommitRequest.Builder` Factory Object

`OffsetCommitRequest.Builder` is a concrete AbstractRequest.Builder factory object that can build an OffsetCommitRequest.

```
OffsetCommitRequest build(short version)
```

Creating OffsetCommitRequest from Byte Buffer — `parse` Factory Method

```
OffsetCommitRequest parse(ByteBuffer buffer, short version)
```

`parse` …FIXME

Note	`parse` is used when…FIXME

`getErrorResponse` Method

```
AbstractResponse getErrorResponse(int throttleTimeMs, Throwable e)
```

Note	`getErrorResponse` is part of the AbstractRequest Contract to create a AbstractResponse for a failure.

`getErrorResponse` ...FIXME

ProduceRequest

`ProduceRequest` is a concrete AbstractRequest with **Produce** API key and the following properties:

- Version
- acks
- timeout
- Partition records (`Map<TopicPartition, MemoryRecords>`)
- Transactional ID

`ProduceRequest` is used by a Kafka producer (Kafka Producer I/O Thread actually) to send partition records (with a given acks).

`ProduceRequest` is created when:

- `ProduceRequest.Builder` is requested to build a ProduceRequest
- `ProduceRequest` is requested to parse a byte buffer
- `AbstractRequest` is requested to parse a request (with the Produce API key)

`ProduceRequest.Builder` Factory Object

`ProduceRequest` comes with a concrete AbstractRequest.Builder factory object that can build a ProduceRequest.

```
ProduceRequest build(short version, boolean validate)
```

`ProduceRequest.Builder` is used when…FIXME

Creating ProduceRequest.Builder Instance (For Given Magic Number) — `forMagic` Factory Method

```
Builder forMagic(
  byte magic,
  short acks,
  int timeout,
  Map<TopicPartition, MemoryRecords> partitionRecords,
  String transactionalId)
```

`forMagic` ...FIXME

Note	`forMagic` is used excusively when `Sender` is requested to send a ProduceRequest.

Creating ProduceRequest from Byte Buffer — `parse` Factory Method

```
ProduceRequest parse(ByteBuffer buffer, short version)
```

`parse` ...FIXME

Note	`parse` is used when...FIXME

SyncGroupRequest

`SyncGroupRequest` is a concrete AbstractRequest with **SyncGroup** API key and the following:

- Group ID
- Generation ID
- Member ID
- Group partition assignment (as `Map<String, ByteBuffer>`)
- Version

`SyncGroupRequest` is created when:

- `AbstractRequest` is requested to parse a request (with the SyncGroup API key)
- `SyncGroupRequest` is requested to parse a byte buffer
- `SyncGroupRequest.Builder` is requested to build a SyncGroupRequest

`SyncGroupRequest.Builder` Factory Object

`SyncGroupRequest.Builder` is a concrete AbstractRequest.Builder factory object that can build a SyncGroupRequest.

```
SyncGroupRequest build(short version)
```

Creating SyncGroupRequest from Byte Buffer — `parse` Factory Method

```
SyncGroupRequest parse(ByteBuffer buffer, short version)
```

`parse` …FIXME

Note	`parse` is used when…FIXME

RequestContext

`RequestContext` is...FIXME

`RequestContext` is created when:

- `Processor` is requested to processCompletedReceives
- `SaslServerAuthenticator` is requested to `handleSaslToken` and `handleKafkaRequest`

`parseRequest` Method

```
RequestAndSize parseRequest(ByteBuffer buffer)
```

`parseRequest` ...FIXME

Note	`parseRequest` is used when: • RequestChannel.Request is created • `SaslServerAuthenticator` is requested to `handleSaslToken` and `handleKafkaRequest`

Creating RequestContext Instance

`RequestContext` takes the following to be created:

- `RequestHeader`
- Connection ID
- Client address (as a java.net.InetAddress)
- KafkaPrincipal
- Listener name
- `SecurityProtocol`

Serializer Contract

`Serializer` is…FIXME

Deserializer Contract

Caution	FIXME

Serde Contract

`Serde` is...FIXME

Serdes Factory Object

`Serdes` is...FIXME

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
`executor`	Java's ScheduledThreadPoolExecutor that schedules tasks. Initialized when `KafkaScheduler` starts up and shut down when `KafkaScheduler` shuts down.
`schedulerThreadId`	Java's AtomicInteger 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. (ka
fka.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 triggerred, 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	`scheduleOnce` is used when…FIXME

`resizeThreadPool` Method

```
resizeThreadPool(newSize: Int): Unit
```

`resizeThreadPool` …FIXME

Note	`resizeThreadPool` is used exclusively when `DynamicThreadPool` is requested to reconfigure.

Scheduler Contract

`Scheduler` is the contract of task schedulers that allow for scheduling tasks.

Note	Kafka Server and Periodic Tasks describes the scheduled tasks.

```
package kafka.utils

trait Scheduler {
  def isStarted: Boolean
  def shutdown()
  def schedule(
    name: String,
    fun: () => Unit,
    delay: Long = 0,
    period: Long = -1,
    unit: TimeUnit = TimeUnit.MILLISECONDS)
  def startup()
}
```

Table 1. Scheduler Contract

Property	Description
`isStarted`	Flag that indicates whether the scheduler has been started or not Used when: • `KafkaController` is requested to onControllerResignation • `GroupMetadataManager` is requested to shutdown • `LogManager` is requested to deleteLogs • `KafkaScheduler` is requested to startup and ensureRunning
`shutdown`	Shuts down the scheduler. When this method is complete no more executions of background tasks will occur. Used when: • `KafkaController` is requested to onControllerResignation • `GroupMetadataManager` is requested to shutdown • `TransactionCoordinator` is requested to shutdown • `KafkaServer` is requested to shutdown

	• `ZooKeeperClient` is requested to close
`schedule`	Schedules a task Used when: • `KafkaController` is requested to onControllerFailover and scheduleAutoLeaderRebalanceTask • `GroupMetadataManager` is requested to start up, scheduleLoadGroupAndOffsets, removeGroupsForPartition and scheduleHandleTxnCompletion • `TransactionCoordinator` is requested to start up • `TransactionStateManager` is requested to enableTransactionalIdExpiration, loadTransactionsForTxnTopicPartition and removeTransactionsForTxnTopicPartition • `Log` is created and requested to roll and asyncDeleteSegment • `LogManager` is requested to start up and deleteLogs • `ReplicaManager` is requested to startHighWaterMarksCheckPointThread and startup • `KafkaScheduler` is requested to scheduleOnce
`startup`	Initializes the scheduler so it is ready to schedule tasks Used when: • `KafkaController` is requested to onControllerFailover • `GroupMetadataManager` is requested to start up • `TransactionCoordinator` is requested to start up • `KafkaServer` is requested to [kafka-server-KafkaServer#startup] • `ZooKeeperClient` is created

Note	KafkaScheduler is the one and only known implementation of the Scheduler Contract in Apache Kafka.

ZooKeeperClient

`ZooKeeperClient` is…FIXME

`ZooKeeperClient` is a KafkaMetricsGroup.

`ZooKeeperClient` is created when…FIXME

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

Name	Description
`expiryScheduler`	KafkaScheduler with 1 daemon thread with **zk-session-expiry-handler** prefix Used when…FIXME

Creating ZooKeeperClient Instance

`ZooKeeperClient` takes the following when created:

- Comma-separated host:port pairs of Zookeeper servers
- `sessionTimeoutMs`
- `connectionTimeoutMs`
- `maxInFlightRequests`
- `Time`
- Metric group
- Metric type

`ZooKeeperClient` initializes the internal registries and counters.

While being created, `ZooKeeperClient` …FIXME

`close` Method

```
close(): Unit
```

`close` …FIXME

Note	`close` is used when…FIXME

KafkaZkClient — Higher-Level Zookeeper Client

`KafkaZkClient` is a higher-level Zookeeper client.

`KafkaZkClient` is created (using the apply factory method) when:

- `KafkaServer` is requested to start up (and initZkClient)
- `AclAuthorizer` is requested to configure (and createAclPaths)
- `ZkSecurityMigrator`, TopicCommand (with the deprecated `--zookeeper` option), ReassignPartitionsCommand, `PreferredReplicaLeaderElectionCommand` (with the deprecated `--zookeeper` option), ConfigCommand tools are executed

Tip	Enable `ALL` logging level for `kafka.zk.KafkaZkClient` logger to see what happens inside. Add the following line to `config/log4j.properties`: `log4j.logger.kafka.zk.KafkaZkClient=ALL` Refer to Logging.

Performance Metrics

`KafkaZkClient` is a KafkaMetricsGroup with the following performance metrics.

Table 1. KafkaZkClient's Performance Metrics

Name	Description
`ZooKeeperRequestLatencyMs`	Histogram that is updated when `KafkaZkClient` is requested to retryRequestsUntilConnected

The performance metrics are registered in **kafka.server:type=ZooKeeperClientMetrics** group.

Figure 1. KafkaZkClient Metrics in JConsole JMX Tool

`getTopicPartitionCount` Method

```
getTopicPartitionCount(topic: String): Option[Int]
```

`getTopicPartitionCount` ...FIXME

Note	`getTopicPartitionCount` is used when...FIXME

Registering StateChangeHandler — `registerStateChangeHandler` Method

```
registerStateChangeHandler(stateChangeHandler: StateChangeHandler): Unit
```

`registerStateChangeHandler` ...FIXME

Note	`registerStateChangeHandler` is used when...FIXME

Creating KafkaZkClient Instance

`KafkaZkClient` takes the following when created:

- ZooKeeperClient
- `isSecure` flag
- `Time`

`KafkaZkClient` initializes the internal registries and counters.

Creating KafkaZkClient Instance — `apply` Factory Method

```
apply(
  connectString: String,
  isSecure: Boolean,
  sessionTimeoutMs: Int,
  connectionTimeoutMs: Int,
  maxInFlightRequests: Int,
  time: Time,
  metricGroup: String = "kafka.server",
  metricType: String = "SessionExpireListener"): KafkaZkClient
```

`apply` creates a ZooKeeperClient that is then used to create a KafkaZkClient.

Note	`apply` is used when…FIXME

`createTopLevelPaths` Method

```
createTopLevelPaths(): Unit
```

`createTopLevelPaths` …FIXME

Note	`createTopLevelPaths` is used when…FIXME

`updateBrokerInfoInZk` Method

```
updateBrokerInfoInZk(brokerInfo: BrokerInfo): Unit
```

`updateBrokerInfoInZk` …FIXME

Note	`updateBrokerInfoInZk` is used when…FIXME

`registerZNodeChildChangeHandler` Method

```
registerZNodeChildChangeHandler(zNodeChildChangeHandler: ZNodeChildChangeHandler): Unit
```

`registerZNodeChildChangeHandler` ...FIXME

Note	`registerZNodeChildChangeHandler` is used when...FIXME

`registerZNodeChangeHandlerAndCheckExistence` Method

```
registerZNodeChangeHandlerAndCheckExistence(zNodeChangeHandler: ZNodeChangeHandler): B
oolean
```

`registerZNodeChangeHandlerAndCheckExistence` ...FIXME

Note	`registerZNodeChangeHandlerAndCheckExistence` is used when...FIXME

`deleteLogDirEventNotifications` Method

```
deleteLogDirEventNotifications(): Unit
```

`deleteLogDirEventNotifications` ...FIXME

Note	`deleteLogDirEventNotifications` is used when...FIXME

`deleteIsrChangeNotifications` Method

```
deleteIsrChangeNotifications(): Unit
```

`deleteIsrChangeNotifications` ...FIXME

Note	`deleteIsrChangeNotifications` is used when...FIXME

`unregisterZNodeChildChangeHandler` Method

```
unregisterZNodeChildChangeHandler(path: String): Unit
```

`unregisterZNodeChildChangeHandler` ...FIXME

Note	`unregisterZNodeChildChangeHandler` is used when...FIXME

`unregisterZNodeChangeHandler` Method

```
unregisterZNodeChangeHandler(path: String): Unit
```

`unregisterZNodeChangeHandler` ...FIXME

Note	`unregisterZNodeChangeHandler` is used when...FIXME

`setControllerEpochRaw` Method

```
setControllerEpochRaw(epoch: Int, epochZkVersion: Int): SetDataResponse
```

`setControllerEpochRaw` ...FIXME

Note	`setControllerEpochRaw` is used when...FIXME

`createControllerEpochRaw` Method

```
createControllerEpochRaw(epoch: Int): CreateResponse
```

`createControllerEpochRaw` ...FIXME

Note	`createControllerEpochRaw` is used when...FIXME

Fetching Metadata of Brokers in Cluster — `getAllBrokersInCluster` Method

```
getAllBrokersInCluster: Seq[Broker]
```

`getAllBrokersInCluster` fetches broker IDs followed by fetching the metadata of every broker (which is the data associated with a `/brokers/ids/[brokerId]` znode).

Note	`getAllBrokersInCluster` is used when: • `ConfigCommand` is requested to alterConfig • `ReassignPartitionsCommand` is requested to removeThrottle • `TopicCommand` is requested to describeTopic • `KafkaController` is requested to initializeControllerContext and at BrokerChange controller event • `KafkaServer` is requested to createBrokerInfo • `AdminZkClient` is requested to getBrokerMetadatas

`getAllTopicsInCluster` Method

```
getAllTopicsInCluster: Seq[String]
```

`getAllTopicsInCluster` ...FIXME

Note	`getAllTopicsInCluster` is used when...FIXME

`getReplicaAssignmentForTopics` Method

```
getReplicaAssignmentForTopics(topics: Set[String]): Map[TopicPartition, Seq[Int]]
```

`getReplicaAssignmentForTopics` ...FIXME

Note	`getReplicaAssignmentForTopics` is used when...FIXME

`getPartitionReassignment` Method

```
getPartitionReassignment: collection.Map[TopicPartition, Seq[Int]]
```

`getPartitionReassignment` ...FIXME

Note	`getPartitionReassignment` is used when...FIXME

`getTopicDeletions` Method

```
getTopicDeletions: Seq[String]
```

`getTopicDeletions` ...FIXME

Note	`getTopicDeletions` is used when...FIXME

Retrieving Partition State — `getTopicPartitionStates` Method

```
getTopicPartitionStates(
  partitions: Seq[TopicPartition]): Map[TopicPartition, LeaderIsrAndControllerEpoch]
```

`getTopicPartitionStates` getTopicPartitionStatesRaw for the given `TopicPartitions`.

For every response, `getTopicPartitionStates` decodes the JSON-encoded partition state data (for the partitions that were found in ZooKeeper).

Note	`getTopicPartitionStates` is used when `KafkaController` is requested to updateLeaderAndIsrCache, areReplicasInIsr, updateLeaderEpoch and process a PartitionReassignmentIsrChange controller event.

`registerZNodeChangeHandler` Method

```
registerZNodeChangeHandler(zNodeChangeHandler: ZNodeChangeHandler): Unit
```

`registerZNodeChangeHandler` ...FIXME

Note	`registerZNodeChangeHandler` is used when...FIXME

`getControllerEpoch` Method

```
getControllerEpoch: Option[(Int, Stat)]
```

`getControllerEpoch` ...FIXME

Note	`getControllerEpoch` is used when...FIXME

`deletePartitionReassignment` Method

```
deletePartitionReassignment(): Unit
```

`deletePartitionReassignment` ...FIXME

Note	`deletePartitionReassignment` is used when...FIXME

`setOrCreatePartitionReassignment` Method

```
setOrCreatePartitionReassignment(reassignment: collection.Map[TopicPartition, Seq[Int]
]): Unit
```

`setOrCreatePartitionReassignment` ...FIXME

Note	`setOrCreatePartitionReassignment` is used when...FIXME

Getting Active Controller ID — `getControllerId` Method

```
getControllerId: Option[Int]
```

`getControllerId` sends a request to Zookeeper for the data of the `/controller` znode and returns the following:

- The `brokerid` field of the JSON data when the response is `OK`
- `None` for a `NONODE` response
- Throws a `KeeperException` with the response code and the `/controller` path

Note	`getControllerId` is used when: • `KafkaController` is requested to elect • `ControllerEventThread` is requested to process controller events (and processes ControllerChange and Reelect events) • `KafkaServer` is requested to perform a controlled shutdown

Creating Ephemeral Znode (And Throwing Exception When Unsuccessful)-- `checkedEphemeralCreate` Method

```
checkedEphemeralCreate(path: String, data: Array[Byte]): Unit
```

`checkedEphemeralCreate` ...FIXME

Note	`checkedEphemeralCreate` is used when...FIXME

`registerControllerAndIncrementControllerEpoch` Method

```
registerControllerAndIncrementControllerEpoch(controllerId: Int): (Int, Int)
```

`registerControllerAndIncrementControllerEpoch` ...FIXME

Note	`registerControllerAndIncrementControllerEpoch` is used exclusively when `KafkaController` is requested to elect.

`retryRequestsUntilConnected` Internal Method

```
retryRequestsUntilConnected[Req <: AsyncRequest](
  requests: Seq[Req]): Seq[Req#Response]
```

`retryRequestsUntilConnected` ...FIXME

`createSequentialPersistentPath` Method

```
createSequentialPersistentPath(path: String, data: Array[Byte]): String
```

`createSequentialPersistentPath` ...FIXME

Note	`createSequentialPersistentPath` is used when `KafkaZkClient` is requested to propagateLogDirEvent and propagateIsrChanges.

`propagateLogDirEvent` Method

```
propagateLogDirEvent(brokerId: Int): Unit
```

`propagateLogDirEvent` ...FIXME

Note	`propagateLogDirEvent` is used exclusively when `ReplicaManager` is requested to handleLogDirFailure.

`propagateIsrChanges` Method

```
propagateIsrChanges(isrChangeSet: collection.Set[TopicPartition]): Unit
```

`propagateIsrChanges` ...FIXME

Note	`propagateIsrChanges` is used exclusively when `ReplicaManager` is requested to maybePropagateIsrChanges.

`getTopicPartitionStatesRaw` Method

```
getTopicPartitionStatesRaw(
  partitions: Seq[TopicPartition]): Seq[GetDataResponse]
```

`getTopicPartitionStatesRaw` gets the topic partition states (from the path `/brokers/topics/[topic]/partitions/[partition]/state` in Zookeeper) for the given partitions.

Internally, `getTopicPartitionStatesRaw` creates a ZooKeeper `GetDataRequest` for the path `/brokers/topics/[topic]/partitions/[partition]/state` for every partition (in the given `partitions`).

In the end, `getTopicPartitionStatesRaw` retryRequestsUntilConnected the `GetDataRequests` .

Note	`getTopicPartitionStatesRaw` is used when: • `PartitionStateMachine` is requested to doElectLeaderForPartitions • `ZkReplicaStateMachine` is requested to getTopicPartitionStatesFromZk • `KafkaZkClient` is requested to getTopicPartitionStates (and getTopicPartitionState)

`getTopicPartitionState` Method

```
getTopicPartitionState(partition: TopicPartition): Option[LeaderIsrAndControllerEpoch]
```

`getTopicPartitionState` ...FIXME

Note	`getTopicPartitionState` is used when...FIXME

Fetching Broker IDs — `getSortedBrokerList` Method

```
getSortedBrokerList(): Seq[Int]
```

`getSortedBrokerList` gets the child znodes at `/brokers/ids` path and sorts it by broker ID (according to the natural ordering).

Note	`getSortedBrokerList` is used when: • `ReassignPartitionsCommand` is requested to parseAndValidate • `KafkaZkClient` is requested to getAllBrokersInCluster

Fetching Child ZNodes — `getChildren` Method

```
getChildren(path : String): Seq[String]
```

`getChildren` ...FIXME

Note	`getChildren` is used when...FIXME

`getClusterId` Method

```
getClusterId: Option[String]
```

`getClusterId` ...FIXME

Note	`getClusterId` is used when...FIXME

`createOrGetClusterId` Method

```
createOrGetClusterId(proposedClusterId: String): String
```

`createOrGetClusterId` ...FIXME

Note	`createOrGetClusterId` is used when...FIXME

All Brokers In Kafka Cluster — `getAllBrokerAndEpochsInCluster` Method

```
getAllBrokerAndEpochsInCluster: Map[Broker, Long]
```

`getAllBrokerAndEpochsInCluster` ...FIXME

Note	`getAllBrokerAndEpochsInCluster` is used when...FIXME

`getBroker` Method

```
getBroker(
  brokerId: Int): Option[Broker]
```

`getBroker` ...FIXME

Note	`getBroker` is used when...FIXME

LogDirEvent Notifications (from Zookeeper) — `getAllLogDirEventNotifications` Method

```
getAllLogDirEventNotifications: Seq[String]
```

`getAllLogDirEventNotifications` ...FIXME

Note	`getAllLogDirEventNotifications` is used when...FIXME

Converting LogDirEvent Notifications to Broker IDs — `getBrokerIdsFromLogDirEvents` Method

```
getBrokerIdsFromLogDirEvents(
  sequenceNumbers: Seq[String]): Seq[Int]
```

`getBrokerIdsFromLogDirEvents` ...FIXME

Note	`getBrokerIdsFromLogDirEvents` is used when...FIXME

Creating State Znodes for Selected Partitions — `createTopicPartitionStatesRaw` Method

```
createTopicPartitionStatesRaw(
  leaderIsrAndControllerEpochs: Map[TopicPartition, LeaderIsrAndControllerEpoch],
  expectedControllerEpochZkVersion: Int
): Seq[CreateResponse]
```

`createTopicPartitionStatesRaw` createTopicPartitions with the topics (of the given partitions).

`createTopicPartitionStatesRaw` createTopicPartition with the partitions.

For every partition (in the given `leaderIsrAndControllerEpochs`), `createTopicPartitionStatesRaw` creates a new `CreateRequest` to create `/brokers/topics/[topic]/partitions/[partition]/state` persistent znode with the associated `LeaderIsrAndControllerEpoch` encoded to JSON format.

```
// zkCli :2181
$ ./bin/zookeeper-shell.sh :2181 ls /brokers/topics
[t1]

$ ./bin/zookeeper-shell.sh :2181 get /brokers/topics/t1/partitions/0/state
{"controller_epoch":1,"leader":0,"version":1,"leader_epoch":0,"isr":[0]}
```

In the end, `createTopicPartitionStatesRaw` retryRequestsUntilConnected all the `CreateRequests`.

Note	`createTopicPartitionStatesRaw` is used exclusively when `ZkPartitionStateMachine` is requested to initializeLeaderAndIsrForPartitions.

`createTopicPartitions` Internal Method

```
createTopicPartitions(
  topics: Seq[String],
  expectedControllerEpochZkVersion: Int):Seq[CreateResponse]
```

`createTopicPartitions` ...FIXME

Note	`createTopicPartitions` is used when...FIXME

`createTopicPartition` Internal Method

```
createTopicPartition(
  partitions: Seq[TopicPartition],
  expectedControllerEpochZkVersion: Int): Seq[CreateResponse]
```

`createTopicPartition` ...FIXME

Note	`createTopicPartition` is used when...FIXME

`updateLeaderAndIsr` Method

```
updateLeaderAndIsr(
  leaderAndIsrs: Map[TopicPartition, LeaderAndIsr],
  controllerEpoch: Int,
  expectedControllerEpochZkVersion: Int): UpdateLeaderAndIsrResult
```

`updateLeaderAndIsr` ...FIXME

Note	`updateLeaderAndIsr` is used when...FIXME

Partition Numbers of Given Topics — `getPartitionsForTopics` Method

```
getPartitionsForTopics(
  topics: Set[String]): Map[String, Seq[Int]]
```

`getPartitionsForTopics` ...FIXME

Note	`getPartitionsForTopics` is used when...FIXME

Reading Entity Config From Zookeeper (Under /config Node) — `getEntityConfigs` Method

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

`getEntityConfigs` reads the `/config/[rootEntityType]/[sanitizedEntityName]`.

`getEntityConfigs` ...FIXME

Note	`getEntityConfigs` is used exclusively when `AdminZkClient` is requested to fetchEntityConfig.

`createAclPaths` Method

```
createAclPaths(): Unit
```

`createAclPaths` ...FIXME

Note	`createAclPaths` is used when `AclAuthorizer` is requested to configure.

`registerBroker` Method

```
registerBroker(
  brokerInfo: BrokerInfo): Long
```

`registerBroker` ...FIXME

Note	`registerBroker` is used when...FIXME

`updateBrokerInfo` Method

```
updateBrokerInfo(
  brokerInfo: BrokerInfo): Unit
```

`updateBrokerInfo` ...FIXME

Note	`updateBrokerInfo` is used when...FIXME

`close` Method

```
close(): Unit
```

Note	`close` is part of the Java's AutoCloseable contract to close a resource (and relinquishing any underlying resources).

`close` ...FIXME

AdminZkClient

`AdminZkClient` is created when:

- `ConfigCommand` is requested to processCommandWithZk
- `ReassignPartitionsCommand` is requested to verifyAssignment, generateAssignment, executeAssignment
- `TopicCommand` is requested to createTopic, alterTopic, describeTopic
- `Partition` is requested to getOrCreateReplica
- `DynamicBrokerConfig` is requested to initialize
- AdminManager, DynamicConfigManager and KafkaApis are created

Creating AdminZkClient Instance

`AdminZkClient` takes the following to be created:

- KafkaZkClient

`AdminZkClient` initializes the internal properties.

Reading Entity Configuration From Zookeeper (Under /config Node) — `fetchEntityConfig` Method

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

`fetchEntityConfig` requests the KafkaZkClient to get the configuration from Zookeeper (under /config node) for the given `rootEntityType` and `sanitizedEntityName`.

Note	`fetchEntityConfig` is used when: • `ConfigCommand` command-line tool is requested to alterConfig and describeConfig • `ReassignPartitionsCommand` command-line tool is requested to removeThrottle, maybeLimit and assignThrottledReplicas • `TopicCommand` command-line tool is requested to alterTopic and describeTopic • `Partition` is requested to look up or create a replica • `AdminManager` is requested to describe the configuration of the specified resources (topics and brokers) and incrementalAlterConfigs • `DynamicBrokerConfig` is requested to initialize • `ConfigChangedNotificationHandler` (of `DynamicConfigManager`) is requested to processEntityConfigChangeVersion1 and processEntityConfigChangeVersion2 • `AdminZkClient` is requested to getAllTopicConfigs, fetchAllEntityConfigs, and fetchAllChildEntityConfigs

`getAllTopicConfigs` Method

```
getAllTopicConfigs(): Map[String, Properties]
```

`getAllTopicConfigs` ...FIXME

Note	`getAllTopicConfigs` is used when...FIXME

`fetchAllEntityConfigs` Method

```
fetchAllEntityConfigs(entityType: String): Map[String, Properties]
```

`fetchAllEntityConfigs` ...FIXME

Note	`fetchAllEntityConfigs` is used when...FIXME

`fetchAllChildEntityConfigs` Method

```
fetchAllChildEntityConfigs(
  rootEntityType: String,
  childEntityType: String): Map[String, Properties]
```

`fetchAllChildEntityConfigs` ...FIXME

Note	`fetchAllChildEntityConfigs` is used when...FIXME

`writeTopicPartitionAssignment` Internal Method

```
writeTopicPartitionAssignment(
  topic: String,
  replicaAssignment: Map[Int, Seq[Int]],
  update: Boolean): Unit
```

`writeTopicPartitionAssignment` ...FIXME

Note	`writeTopicPartitionAssignment` is used exclusively when `AdminZkClient` is requested to createOrUpdateTopicPartitionAssignmentPathInZK.

`createOrUpdateTopicPartitionAssignmentPathInZK` Internal Method

```
createOrUpdateTopicPartitionAssignmentPathInZK(
  topic: String,
  partitionReplicaAssignment: Map[Int, Seq[Int]],
  config: Properties = new Properties,
  update: Boolean = false): Unit
```

`createOrUpdateTopicPartitionAssignmentPathInZK` ...FIXME

Note	`createOrUpdateTopicPartitionAssignmentPathInZK` is used when: • `TopicCommand` is requested to create a topic • `AdminManager` is requested to create topics • `AdminZkClient` is requested to create a topic and addPartitions

Creating Topic — `createTopic` Method

```
createTopic(
  topic: String,
  partitions: Int,
  replicationFactor: Int,
  topicConfig: Properties = new Properties,
  rackAwareMode: RackAwareMode = RackAwareMode.Enforced): Unit
```

`createTopic` fetches the metadata of the brokers in the cluster (given the `RackAwareMode`).

`createTopic` requests `AdminUtils` helper object to assignReplicasToBrokers (given the broker metadata, partitions and replicationFactor).

In the end, `createTopic` createOrUpdateTopicPartitionAssignmentPathInZK.

Note	`createTopic` is used when: • `TopicCommand` is requested to create a topic • `KafkaApis` is requested to create a topic

`addPartitions` Method

```
addPartitions(
  topic: String,
  existingAssignment: Map[Int, Seq[Int]],
  allBrokers: Seq[BrokerMetadata],
  numPartitions: Int = 1,
  replicaAssignment: Option[Map[Int, Seq[Int]]] = None,
  validateOnly: Boolean = false): Map[Int, Seq[Int]]
```

`addPartitions` ...FIXME

Note	`addPartitions` is used when...FIXME

Fetching Metadata of Brokers in Cluster (Broker ID and Rack Information) — `getBrokerMetadatas` Method

```
getBrokerMetadatas(
  rackAwareMode: RackAwareMode = RackAwareMode.Enforced,
  brokerList: Option[Seq[Int]] = None): Seq[BrokerMetadata]
```

`getBrokerMetadatas` requests the KafkaZkClient for the metadata of the brokers in the cluster.

Note	The broker metadata includes a broker ID, endpoints, and an optional rack information.

`getBrokerMetadatas` takes the input broker IDs (`brokerList`) if defined and leaves only those that are available (i.e. among the brokers in the cluster) or falls back to all brokers in the cluster.

`getBrokerMetadatas` branches off per `RackAwareMode` as follows:

- For `RackAwareMode.Disabled` or `RackAwareMode.Safe` , `getBrokerMetadatas` returns `BrokerMetadata` with broker IDs and no rack information
- For `RackAwareMode.Enforced` , `getBrokerMetadatas` returns `BrokerMetadata` with broker IDs and rack information

In the end, `getBrokerMetadatas` sorts the brokers by broker ID.

`getBrokerMetadatas` throws an `AdminOperationException` for `RackAwareMode.Enforced` mode with some brokers without rack information:

```
Not all brokers have rack information. Add --disable-rack-aware
in command line to make replica assignment without rack
information.
```

Note	`getBrokerMetadatas` is used when: • `AdminManager` is requested to createPartitions • `AdminZkClient` is requested to create a topic • `ReassignPartitionsCommand` is requested to generateAssignment • `TopicCommand` is requested to alterTopic

ZkAclChangeStore Contract

`ZkAclChangeStore` is the contract of ZkAclChangeStores that FIXME.

```
package kafka.zk

sealed trait ZkAclChangeStore {
  // only required properties (vals and methods) that have no implementation
  // the others follow
  val aclChangePath: String
  def decode(bytes: Array[Byte]): Resource
  protected def encode(resource: Resource): Array[Byte]
}
```

Note	`ZkAclChangeStore` is a Scala sealed trait and so all the available implementations are in a single compilation unit (i.e. the file with the Scala source code).

Table 1. (Subset of) ZkAclChangeStore Contract

Property	Description
`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	`createListener` is used when…FIXME

BrokerIdZNode Utility

`BrokerIdZNode` utility is used to decode or encode broker metadata for KafkaZkClient.

`decode` Utility

```
decode(
  id: Int,
  jsonBytes: Array[Byte]): BrokerInfo
```

`decode` ...FIXME

Note	`decode` is used when `KafkaZkClient` is requested to getAllBrokersInCluster, getAllBrokerAndEpochsInCluster, and getBroker.

`encode` Utility

```
encode(
  brokerInfo: BrokerInfo): Array[Byte] (1)
encode(
  version: Int,
  host: String,
  port: Int,
  advertisedEndpoints: Seq[EndPoint],
  jmxPort: Int,
  rack: Option[String]): Array[Byte]
```

1. Uses the given `BrokerInfo` for the arguments of the other `encode`

`encode` ...FIXME

Note	`encode` is used when `BrokerInfo` is requested for a JSON representation (`toJsonBytes`) for `KafkaZkClient` to add and update a broker metadata in Zookeeper.

WorkerGroupMember

Caution	FIXME WorkerCoordinator? DistributedHerder?

ConnectDistributed

`ConnectDistributed` is a command-line utility that runs Kafka Connect in distributed mode.

Caution	FIXME Doh, I'd rather not enter Kafka Connect yet. Not interested in it yet.

Demo: Kafka Controller Election

Use the following setup with one Zookeeper server and two Kafka brokers to observe the Kafka controller election.

Start Zookeeper server.

```
$ ./bin/zookeeper-server-start.sh config/zookeeper.properties
...
INFO binding to port 0.0.0.0/0.0.0.0:2181 (org.apache.zookeeper.server.NIOServerCnxnFa
ctory)
```

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.controll
er.KafkaController)
DEBUG [Controller id=100] De-registering logDirEventNotificationListener (kafka.contro
ller.KafkaController)
INFO [Controller id=100] Resigned (kafka.controller.KafkaController)
DEBUG [Controller id=100] Broker 200 has been elected as the controller, so stopping t
he 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.c
ontroller.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.controlle
r.KafkaController)
INFO [Controller id=200] Partitions being reassigned: Map() (kafka.controller.KafkaCon
troller)
INFO [Controller id=200] Partitions already reassigned: Set() (kafka.controller.KafkaC
ontroller)
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.contr
oller.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.KafkaControl
ler)
INFO [Controller id=200] List of topics ineligible for deletion:  (kafka.controller.Ka
fkaController)
INFO [Controller id=200] Ready to serve as the new controller with epoch 40 (kafka.con
troller.KafkaController)
INFO [Controller id=200] Partitions undergoing preferred replica election:  (kafka.con
troller.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 top
ic deletion:  (kafka.controller.KafkaController)
INFO [Controller id=200] Resuming preferred replica election for partitions:  (kafka.c
ontroller.KafkaController)
INFO [Controller id=200] Starting preferred replica leader election for partitions  (k
afka.controller.KafkaController)
INFO [Controller id=200] Starting the controller scheduler (kafka.controller.KafkaCont
roller)
```

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.