

useMemory: Boolean

+ _useOffHeap: Boolean

+ _deserialized: Boolean

+ def clone(): StorageLevel

+ def isValid: Boolean

+ def toString: String

+ def description: String

BlockManager

+ def toInt: Int + def hashCode(): Int

+ def equals(other: Any): Boolean

+ def writeExternal(out: ObjectOutput): Unit

+ def readExternal(in: ObjectInput): Unit

block Manager Id: Block Manager Id,

num Replicas: Int): List [Block Manager Id]

+ def getSampleIds(n: Int, m: Int, r: Random): List[Int]

peers Replicated To: mutable. Hash Set [Block Manager Id],

BlockReplicationUtils

+ def getRandomSample[T](elems: Seq[T], m: Int, r: Random): List[T]

- rpcEnv: RpcEnv

- conf: SparkConf

+ subDirsPerLocalDir

+ diskStore:DiskStore + maxOnHeapMemory

+ maxOffHeapMemory

+ blockStoreClient

+ slaveEndpoint

+ diskBlockManager

- master: BlockManagerMaster

- serializer Manager: Serializer Manager

- memoryManager: MemoryManager

- shuffleManager: ShuffleManager

+ remoteReadNioBufferConversion

+ memoryStore:MemoryStore

+ externalShuffleServicePort

+ blockManagerId: BlockManagerId

+ maxFailuresBeforeLocationRefresh

+ cachedPeers: Seq[BlockManagerId]

+ blockReplicationPolicy: BlockReplicationPolicy

+ def registerWithExternalShuffleServer(): Unit

+ hostLocalDirManager: Option[HostLocalDirManager

+ def getLocalBlockData(blockId: BlockId): ManagedBuffer

+ def getMatchingBlockIds(filter: BlockId => Boolean): Seq[BlockId]

+ def doGetLocalBytes(blockId: BlockId, info: BlockInfo): BlockData

+ def fetchRemoteManagedBuffer(blockId: BlockId,blockSize: Long,

+ def getRemoteBytes(blockId: BlockId): Option[ChunkedByteBuffer]

+ def release All Locks For Task (task Attempt Id: Long): Seq [Block Id]

writeMetrics: ShuffleWriteMetricsReporter): DiskBlockObjectWriter

keepReadLock: Boolean)(putBody: BlockInfo => Option[T]): Option[T]

keepReadLock: Boolean = false): Option[PartiallyUnrolledIterator[T]]

+ def get[T: ClassTag](blockId: BlockId): Option[BlockResult]

+ def getPeers(forceFetch: Boolean): Seq[BlockManagerId]

+ def removeBroadcast(broadcastId: Long, tellMaster: Boolean): Int

+ def removeBlock(blockId: BlockId, tellMaster: Boolean = true): Unit

+ def removeBlockInternal(blockId: BlockId, tellMaster: Boolean): Unit

+ def addUpdatedBlockStatusToTaskMetrics(blockId: BlockId, status: BlockStatus): Unit

+ def downgradeLock(blockId: BlockId): Unit

+ def registerTask(taskAttemptId: Long): Unit

+ def removeRdd(rddId: Int): Int

+ def stop(): Unit

+ def getCurrentBlockStatus(blockId: BlockId, info: BlockInfo): BlockStatus

+ def getRemoteValues[T: ClassTag](blockId: BlockId): Option[BlockResult]

+ def preferExecutors(locations: Seq[BlockManagerId]): Seq[BlockManagerId] + def sortLocations(locations: Seq[BlockManagerId]): Seq[BlockManagerId]

+ def releaseLock(blockId: BlockId, taskContext: Option[TaskContext] = None): Unit

+ def getLocationBlockIds(blockIds: Array[BlockId]): Array[Seq[BlockManagerId]]

+ def getStatus(blockId: BlockId): Option[BlockStatus]

+ def handleLocalReadFailure(blockId: BlockId): Nothing

+ def getLocalValues(blockId: BlockId): Option[BlockResult]

+ def getLocalBytes(blockId: BlockId): Option[BlockData]

+ def getHostLocalShuffleData(blockId: BlockId,dirs: Array[String]): ManagedBuffer

+ def putBlockData(blockId: BlockId,data: ManagedBuffer,level: StorageLevel,classTag: ClassTag[]): Boolean

+ def reportBlockStatus(blockId: BlockId, status: BlockStatus, droppedMemorySize: Long = 0L): Unit

+ def getRemoteBlock[T](blockId: BlockId,bufferTransformer: ManagedBuffer => T): Option[T]

locations And Status: Block Manager Messages. Block Locations And Status): Option [Managed Buffer]

def getDiskWriter(blockId: BlockId, file: File, serializerInstance: SerializerInstance, bufferSize: Int,

+ def replicateBlock(blockId: BlockId,existingReplicas: Set[BlockManagerId],maxReplicas: Int): Unit

+ def putSingle[T: ClassTag](blockId: BlockId, value: T, level: StorageLevel, tellMaster: Boolean = true): Boolean

+ def dropFromMemory[T: ClassTag](blockId: BlockId,data: () => Either[Array[T], ChunkedByteBuffer]): StorageLevel

+ def releaseLockAndDispose(blockId: BlockId,data: BlockData,taskContext: Option[TaskContext] = None): Unit

+ def doPut[T](blockId: BlockId,level: StorageLevel,classTag: ClassTag[_],tellMaster: Boolean,

+ def readDiskBlockFromSameHostExecutor(blockId: BlockId,localDirs: Array[String],blockSize: Long): Option[ManagedBuffer]

def putIterator[T: ClassTag](blockId: BlockId, values: Iterator[T], level: StorageLevel, tellMaster: Boolean = true): Boolean

+ def putBytes[T: ClassTag](blockId: BlockId,bytes: ChunkedByteBuffer,level: StorageLevel,tellMaster: Boolean = true): Boolean

+ def doPutIterator[T](blockId: BlockId, iterator: () => Iterator[T], level: StorageLevel, classTag: ClassTag[T], tellMaster: Boolean = true,

+ def maybeCacheDiskValuesInMemory[T](blockInfo: BlockInfo,blockId: BlockId,level: StorageLevel, diskIterator: Iterator[T]): Iterator[T]

+ def getOrElseUpdate[T](blockId: BlockId,level: StorageLevel,classTag: ClassTag[T],makeIterator: () => Iterator[T]): Either[BlockResult, Iterator[T]]

+ def maybeCacheDiskBytesInMemory(blockInfo: BlockInfo,blockId: BlockId,level: StorageLevel,diskData: BlockData): Option[ChunkedByteBuffer]

+ def replicate(blockId: BlockId,data: BlockData,level: StorageLevel,classTag: ClassTag[],existingReplicas: Set[BlockManagerId] = Set.empty): Unit

+ def putBlockDataAsStream(blockId: BlockId,level: StorageLevel,classTag: ClassTag[]): StreamCallbackWithID

+ def tryToReportBlockStatus(blockId: BlockId, status: BlockStatus, droppedMemorySize: Long = 0L): Boolean

+ shuffleServerId: BlockManagerId

+ asyncReregisterTask: Future[Unit]

+ asyncReregisterLock:Object

+ remoteBlockTempFileManager

+ def initialize(appId: String): Unit

+ def shuffleMetricsSource: Source

+ def waitForAsyncReregister(): Unit

+ maxRemoteBlockToMem

+ def reportAllBlocks(): Unit

+ def asyncReregister(): Unit

+ def reregister(): Unit

+ peerFetchLock:Object

+ lastPeerFetchTimeNs

securityManager: SecurityManager

+ externalShuffleServiceEnabled: Boolea

mapOutputTracker: MapOutputTracker

blockTransferService: BlockTransferService

externalBlockStoreClient: Option[ExternalBlockStoreClient]

peers: Seq[BlockManagerId],

blockId: BlockId,

+ DISK ONLY 2 + MEMORY ONLY

+ OFF HEAP

StorageLevel

+ MEMORY ONLY 2

+ MEMORY ONLY SER

+ MEMORY ONLY SER 2

+ MEMORY AND DISK

+ MEMORY AND DISK 2

+ MEMORY AND DISK SER

+ MEMORY_AND_DISK_SER_2

+ storageLevelCache:ConcurrentHashMap

+ def fromString(s: String): StorageLevel

BlockManager

BlockStoreUpdater

+ def saveDeserializedValuesToMemoryStore(inputStream:

ByteBufferBlockStoreUpdater

+ def readToByteBuffer(): ChunkedByteBuffer

+ ID_GENERATOR : IdGenerator

ShuffleMetricsSource

+ sourceName: String

+ metricSet: MetricSet

def blockIdsToLocations(

blockIds: Array[BlockId],

env: SparkEnv,

+ def saveSerializedValuesToMemoryStore(bytes:

+ blockId: BlockId

+ level: StorageLevel

+ classTag: ClassTag[T]

+ tellMaster: Boolean

+ def blockData(): BlockData

+ def saveToDiskStore(): Unit

ChunkedByteBuffer): Boolean

+ level: StorageLevel

+ classTag: ClassTag[T]

+ bytes: ChunkedByteBuffer

+ tellMaster: Boolean = true

+ def blockData(): BlockData

+ def saveToDiskStore(): Unit

EncryptedDownloadFile

EncryptedDownloadWritableChannel

+ countingOutput: CountingWritableChannel

+ def closeAndRead(): ManagedBuffer

+ def write(src: ByteBuffer): Int

+ def isOpen: Boolean

+ def close(): Unit

+ def openForWriting(): DownloadFileWritableChannel

- key: Array[Byte]

+ env = SparkEnv.get

+ def path(): String

+ def delete(): Boolean

+ keepReadLock: Boolean = false

InputStream): Boolean

+ def save(): Boolean

TempFileBasedBlockStoreUpdater

+ def readToByteBuffer(): ChunkedByteBuffe

RemoteBlockDownloadFileManager

ReferenceWithCleanup

+ referenceQueue : JReferenceQueue[DownloadFile]

+ blockManager: BlockManager

- def keepCleaning(): Unit

referenceQueue:

+ filePath:Path

+ def cleanUp(): Unit

JReferenceQueue[DownloadFile]

+ POLL_TIMEOUT = 1000

+ cleaningThread

+ def stop(): Unit

+ blockld: Blockld + level: StorageLevel

+ blockSize: Long

BlockManager (Object)

block Manager Master: Block Manager Master = null): Map [Block Id, Seq [String]]

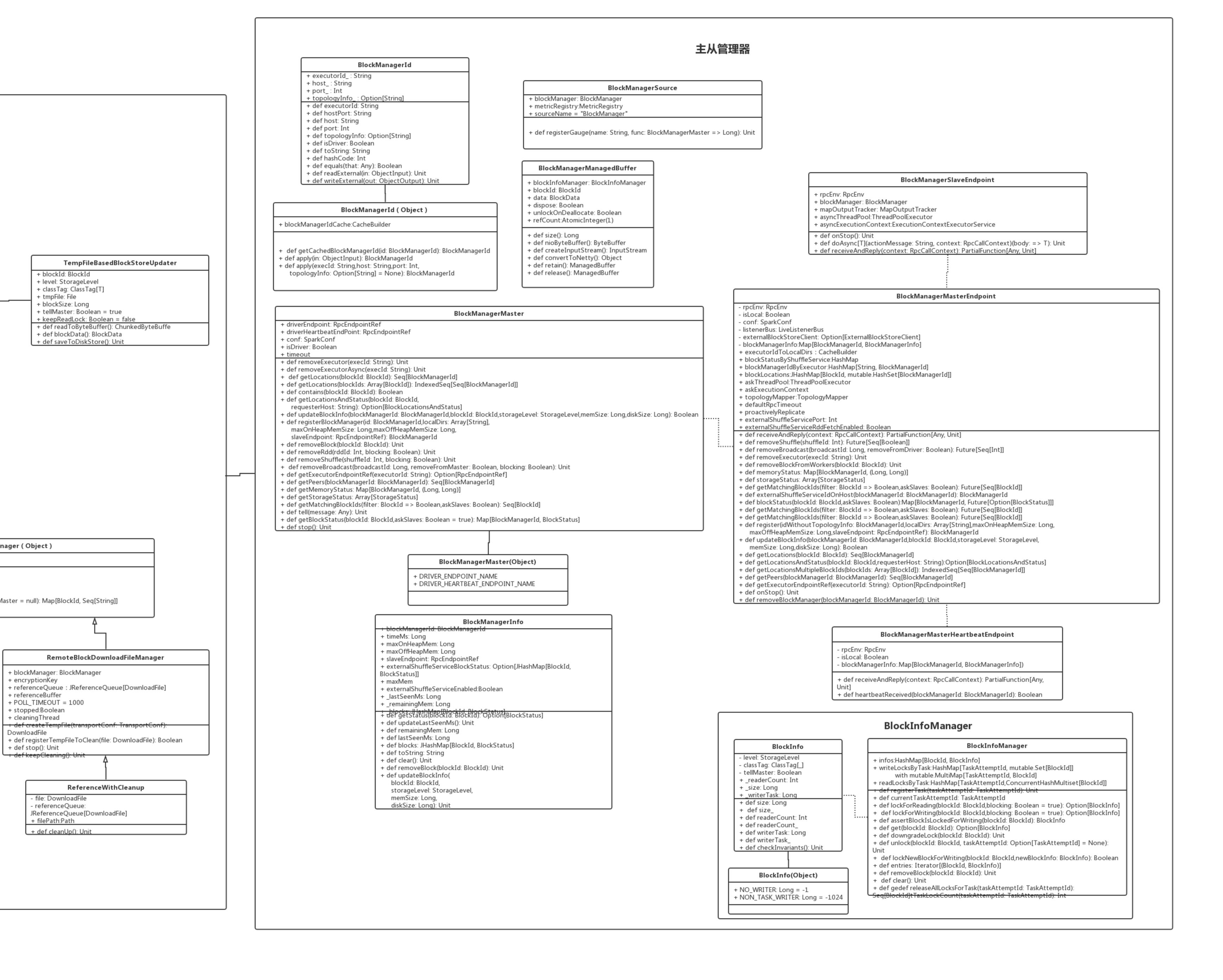
+ classTag: ClassTag[T]

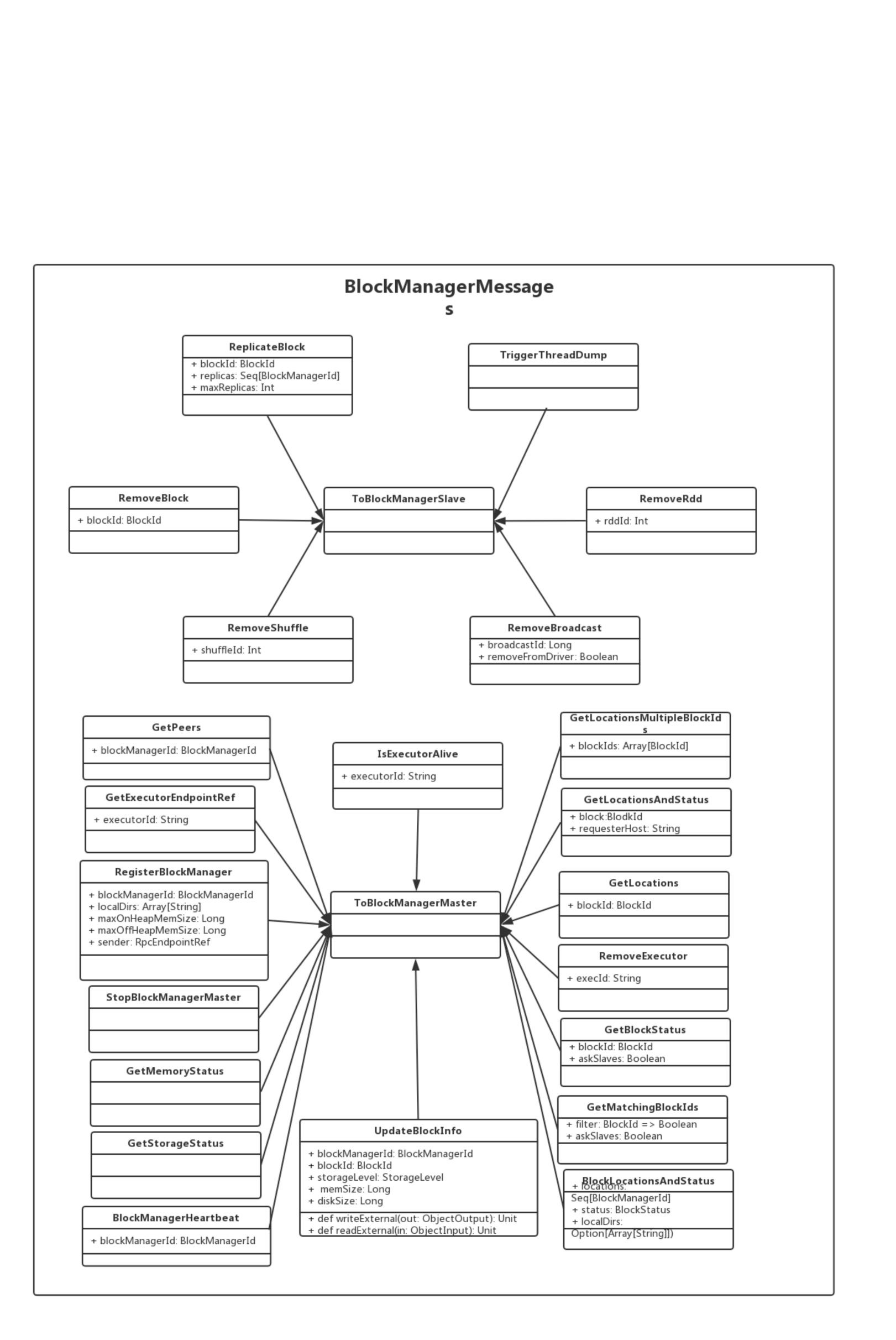
+ tellMaster: Boolean = true

+ def blockData(): BlockData

+ def saveToDiskStore(): Unit

+ def getCachedStorageLevel(level: StorageLevel):





File Based Topology Mapper

+ def getTopologyForHost(hostname: String):