**BookKeeper Getting Started Guide**

**by**

# Table of contents

1. [Programming with BookKeeper](#_bookmark0) 2
   1. [Instantiating BookKeeper.](#_bookmark1) 2
   2. [Creating a ledger.](#_bookmark2) 2
   3. [Adding entries to a ledger.](#_bookmark3) 3
   4. [Closing a ledger.](#_bookmark4) 4
   5. [Opening a ledger.](#_bookmark5) 5
   6. [Reading from ledger](#_bookmark6) 6
   7. [Deleting a ledger](#_bookmark7) 6

# Programming with BookKeeper

* [Instantiating BookKeeper.](#_bookmark1)
* [Creating a ledger.](#_bookmark2)
* [Adding entries to a ledger.](#_bookmark3)
* [Closing a ledger.](#_bookmark4)
* [Opening a ledger.](#_bookmark5)
* [Reading from ledger](#_bookmark6)
* [Deleting a ledger](#_bookmark7)
  1. **Instantiating BookKeeper.**

The first step to use BookKeeper is to instantiate a BookKeeper object:

org.apache.bookkeeper.BookKeeper

There are three BookKeeper constructors:

public BookKeeper(String servers) throws KeeperException,

IOException

where:

* servers is a comma-separated list of ZooKeeper servers.

public BookKeeper(ZooKeeper zk) throws InterruptedException,

KeeperException

where:

* zk is a ZooKeeper object. This constructor is useful when the application also using ZooKeeper and wants to have a single instance of ZooKeeper.

public BookKeeper(ZooKeeper zk, ClientSocketChannelFactory channelFactory) throws InterruptedException, KeeperException

where:

* zk is a ZooKeeper object. This constructor is useful when the application also using ZooKeeper and wants to have a single instance of ZooKeeper.
* channelFactory is a netty channel object (org.jboss.netty.channel.socket).
  1. **Creating a ledger.**

Before writing entries to BookKeeper, it is necessary to create a ledger. With the current BookKeeper API, it is possible to create a ledger both synchronously or asynchronously. The following methods belong to org.apache.bookkeeper.client.BookKeeper.

# Synchronous call:

public LedgerHandle createLedger(int ensSize, int qSize, DigestType type, byte passwd[]) throws KeeperException,

InterruptedException, IOException, BKException

where:

* ensSize is the number of bookies (ensemble size);
* qSize is the write quorum size;
* type is the type of digest used with entries: either MAC or CRC32.
* passwd is a password that authorizes the client to write to the ledger being created.

All further operations on a ledger are invoked through the LedgerHandle object returned.

As a convenience, we provide a createLedger with default parameters (3,2,VERIFIABLE), and the only two input parameters it requires are a digest type and a password.

# Asynchronous call:

public void asyncCreateLedger(int ensSize, int qSize, DigestType type, byte passwd[], CreateCallback cb, Object ctx )

The parameters are the same of the synchronous version, with the exception of cb and ctx. CreateCallback is an interface in

org.apache.bookkeeper.client.AsyncCallback, and a class implementing it has to implement a method called createComplete that has the following signature:

void createComplete(int rc, LedgerHandle lh, Object ctx);

where:

* rc is a return code (please refer to

org.apache.bookeeper.client.BKException for a list);

* lh is a LedgerHandle object to manipulate a ledger;
* ctx is a control object for accountability purposes. It can be essentially any object the application is happy with.

The ctx object passed as a parameter to the call to create a ledger is the one same returned in the callback.

* 1. **Adding entries to a ledger.**

Once we have a ledger handle lh obtained through a call to create a ledger, we can start writing entries. As with creating ledgers, we can write both synchronously and asynchronously. The following methods belong to org.apache.bookkeeper.client.LedgerHandle.

# Synchronous call:

public long addEntry(byte[] data) throws InterruptedException

where:

* data is a byte array;

A call to addEntry returns the status of the operation (please refer to

org.apache.bookeeper.client.BKDefs for a list);

# Asynchronous call:

public void asyncAddEntry(byte[] data, AddCallback cb, Object ctx)

It also takes a byte array as the sequence of bytes to be stored as an entry. Additionaly, it takes a callback object cb and a control object ctx. The callback object must implement the AddCallback interface in org.apache.bookkeeper.client.AsyncCallback, and a class implementing it has to implement a method called addComplete that has the following signature:

void addComplete(int rc, LedgerHandle lh, long entryId, Object ctx);

where:

* rc is a return code (please refer to org.apache.bookeeper.client.BKDefs for a list);
* lh is a LedgerHandle object to manipulate a ledger;
* entryId is the identifier of entry associated with this request;
* ctx is control object used for accountability purposes. It can be any object the application is happy with.
  1. **Closing a ledger.**

Once a client is done writing, it closes the ledger. The following methods belong to

org.apache.bookkeeper.client.LedgerHandle.

# Synchronous close:

public void close() throws InterruptedException

It takes no input parameters.

# Asynchronous close:

public void asyncClose(CloseCallback cb, Object ctx) throws InterruptedException

It takes a callback object cb and a control object ctx. The callback object must implement the CloseCallback interface in

org.apache.bookkeeper.client.AsyncCallback, and a class implementing it has to implement a method called closeComplete that has the following signature:

void closeComplete(int rc, LedgerHandle lh, Object ctx)

where:

* rc is a return code (please refer to org.apache.bookeeper.client.BKDefs for a list);
* lh is a LedgerHandle object to manipulate a ledger;
* ctx is control object used for accountability purposes.
  1. **Opening a ledger.**

To read from a ledger, a client must open it first. The following methods belong to

org.apache.bookkeeper.client.BookKeeper.

# Synchronous open:

public LedgerHandle openLedger(long lId, DigestType type, byte passwd[]) throws InterruptedException, BKException

* ledgerId is the ledger identifier;
* type is the type of digest used with entries: either MAC or CRC32.
* passwd is a password to access the ledger (used only in the case of VERIFIABLE

ledgers);

# Asynchronous open:

public void asyncOpenLedger(long lId, DigestType type, byte passwd[], OpenCallback cb, Object ctx)

It also takes a a ledger identifier and a password. Additionaly, it takes a callback object cb and a control object ctx. The callback object must implement the OpenCallback interface in org.apache.bookkeeper.client.AsyncCallback, and a class

implementing it has to implement a method called openComplete that has the following signature:

public void openComplete(int rc, LedgerHandle lh, Object ctx)

where:

* rc is a return code (please refer to org.apache.bookeeper.client.BKDefs for a list);
* lh is a LedgerHandle object to manipulate a ledger;
* ctx is control object used for accountability purposes.
  1. **Reading from ledger**

Read calls may request one or more consecutive entries. The following methods belong to

org.apache.bookkeeper.client.LedgerHandle.

# Synchronous read:

public Enumeration<LedgerEntry> readEntries(long firstEntry, long lastEntry) throws InterruptedException, BKException

* firstEntry is the identifier of the first entry in the sequence of entries to read;
* lastEntry is the identifier of the last entry in the sequence of entries to read.

# Asynchronous read:

public void asyncReadEntries(long firstEntry, long lastEntry, ReadCallback cb, Object ctx) throws BKException, InterruptedException

It also takes a first and a last entry identifiers. Additionaly, it takes a callback object cb and a control object ctx. The callback object must implement the ReadCallback interface in org.apache.bookkeeper.client.AsyncCallback, and a class implementing it has to implement a method called readComplete that has the following signature:

void readComplete(int rc, LedgerHandle lh, Enumeration<LedgerEntry> seq, Object ctx)

where:

* rc is a return code (please refer to org.apache.bookeeper.client.BKDefs for a list);
* lh is a LedgerHandle object to manipulate a ledger;
* seq is a Enumeration<LedgerEntry> object to containing the list of entries requested;
* ctx is control object used for accountability purposes.
  1. **Deleting a ledger**

Once a client is done with a ledger and is sure that nobody will ever need to read from it again, they can delete the ledger. The following methods belong to org.apache.bookkeeper.client.BookKeeper.

# Synchronous delete:

public void deleteLedger(long lId) throws InterruptedException, BKException

* lId is the ledger identifier;

# Asynchronous delete:

public void asyncDeleteLedger(long lId, DeleteCallback cb, Object ctx)

It takes a ledger identifier. Additionally, it takes a callback object cb and a control object ctx. The callback object must implement the DeleteCallback interface in

org.apache.bookkeeper.client.AsyncCallback, and a class implementing it has to implement a method called deleteComplete that has the following signature:

void deleteComplete(int rc, Object ctx)

where:

* rc is a return code (please refer to org.apache.bookeeper.client.BKDefs for a list);
* ctx is control object used for accountability purposes.