BookKeeper Getting Started Guide

by

Table of contents

1 Programming with BookKeeper... 2
1.1 Instantiating BookKeeper..2
1.2 Creating a ledger. 2
1.3 Adding entries to a ledger. 3
1.4 Closing a ledger. 4
1.5 Opening a ledger. 5
1.6 Reading from ledger 6

1. Programming with BookKeeper

- Instantiating BookKeeper.
- Creating a ledger.
- Adding entries to a ledger.
- Closing a ledger.
- Opening a ledger.
- Reading from ledger

1.1. Instantiating BookKeeper.

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

`org.apache.bookkeeper.BookKeeper`

There is one BookKeeper constructor:

```
public BookKeeper(String servers) throws KeeperException,
IOException
```

where `servers` is a comma-separated list of ZooKeeper servers.

1.2. 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, QMode
mode, byte passwd[]) throws KeeperException,
InterruptedException, IOException, BKException
```

where:

- `ensSize` is the number of bookies (ensemble size);
- `qSize` is the write quorum size;
- `mode` is the ledger mode (QMode.GENERIC, QMode.VERIFIABLE). If `mode` is QMode.GENERIC, then `ensSize` has to be at least *3t+1*, and `qSize` has to be *2t+1*. *t* is the maximum number of tolerated bookie failures.

- `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 input parameter it requires is a password.

Asynchronous call:

```
public void asyncCreateLedger(int ensSize, int qSize, QMode
mode, byte passwd[], CreateCallback cb, Object ctx ) throws
KeeperException, InterruptedException, IOException,
BKException
```

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.BKDefs` for a list);
- `lh` is a `LedgerHandle` object to manipulate a ledger;
- `ctx` is a control object for accountability purposes;

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

1.3. 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) throws InterruptedException
```

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.

1.4. 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 KeeperException,
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.

1.5. 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, byte passwd[]) throws
KeeperException, InterruptedException, IOException,
BKException
```

- `ledgerId` is the ledger identifier;
- `passwd` is a password to access the ledger (used only in the case of `VERIFIABLE` ledgers);

Asynchronous open:

```
public void asyncOpenLedger(long lId, byte passwd[],
OpenCallback cb, Object ctx) throws InterruptedException
```

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.

1.6. 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 LedgerSequence 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, LedgerSequence 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 `LedgerSequence` object to containing the list of entries requested;
- `ctx` is control object used for accountability purposes.