

# Proposal for a common transactional API for HBase

Version 0.3, July 7, 2014

John de Roo, Hewlett Packard

## 1 Abstract

This paper proposes a transactional API which, if adopted, would allow HBase applications and products built on top of HBase to plug into alternative transaction managers based on the customer or developers requirements. At this time there are a number of emerging transaction managers implemented on HBase with differing strengths. By adopting this API, Transaction Manager Developers would allow customers to select the transaction manager which best suits their requirements without the need to modify their code. We hope that adoption of this proposal will simplify the transaction model for HBase and accelerate the adoption and acceptance of transaction management in the HBase community.

Java Transaction API (JTA) is the transactional API associated with Java Transaction Service (JTS) from Oracle [1]. It provides a Java based transactional API which is part of Java Enterprise Edition (Java EE). JTS is the Java equivalent of the Object Transaction Service, part of CORBA. JTA includes a local transaction service interface along with heterogeneous transaction interfaces. It was originally based on the X/Open XA specification [2]. The TransactionManager interface defined as part of JTA is used as the starting point for a common transactional interface for HBase. In addition to the transaction management interface, the API must also provide HBase specific table interfaces which identify operations to be performed as part of a transaction.

## 2 Objectives

The object of this proposal is to provide a simple transaction management API that can be implemented on top of any existing or future HBase Transaction Manager with minimum work so that customers may select which transaction manager they use at deployment time and change that transaction manager as their requirements change.

The transaction identifier should be opaque to hide the implementation from applications and users.

## 3 Specification

### 3.1 API Overview

```
package org.apache.hadoop.hbase.transaction;
```

```
static enum TransactionStatus {  
    ACTIVE,  
    COMMITTED,  
    COMMITTING,  
    MARKED_ROLLBACK,  
    NO_TRANSACTION,  
}
```

## Proposal for a common transactional API for HBase

```
PREPARED,
PREPARING,
ROLLEDBACK,
ROLLING_BACK,
UNKNOWN;
}

static enum TransactionIsolationLevel {
    READ_UNCOMMITTED,
    READ_COMMITTED,
    REPEATABLE_READ,
    SERIALIZABLE;
}

static enum TransactionType {
    READ_ONLY,
    WRITE_READ;
}

public interface TransactionServiceClient {
    public Transaction begin();
    public Transaction begin(final TransactionIsolationLevel isolationLevel);
    public Transaction begin(final TransactionIsolationLevel isolationLevel,
                            final int seconds);
    public Transaction begin(final TransactionIsolationLevel isolationLevel,
                            final int seconds, final TransactionType transactionType);
    public Transaction constructFrom(Byte[] transactionString);
    public Transaction[] getAll();
    // Global methods
    public void setIsolationLevel(final TransactionIsolationLevel isolationLevel);
    public void setTransactionType(final TransactionType transactionType);
    public void setTransactionTimeout(final int seconds);
}

public interface Transaction {
```

Proposal for a common transactional API for HBase

```
public void commit();
public void rollback();
public TransactionStatus getStatus();
public void setTransactionTimeout(final int seconds);
public Byte[] streamTo();
}

public interface TransactionTable extends HTableInterface {
    public TransactionTable(final Configuration conf, final byte[] tableName,
                           final Transaction transaction);
    public TransactionTable(final Configuration conf, final String tableName,
                           final Transaction transaction);
    public TransactionTable(final Configuration conf, final byte[] tableName,
                           final ExecutorService pool, final Transaction transaction);
    public TransactionTable(final Configuration conf, final String tableName, final Transaction
                           transaction);
    public TransactionTable(final Configuration conf, TableName tableName, final Transaction
                           transaction);
    public TransactionTable(final Configuration conf, TableName tableName, ExecutorService
                           pool, final Transaction transaction);
    public TransactionTable(TableName tableName, final ClusterConnection connection,
                           ExecutorService pool, final Transaction transaction);

    public void setTransaction(final Transaction transaction);

    // All other methods including super class constructors have unchanged signatures.
}
```

## 3.2 TransactionServiceClient Interface

The TransactionServiceClient interface allows an HBase application to begin new transactions and clone transactions so that work can be performed against them in other threads and processes. In addition the default isolation level, transaction type (read-only) and timeout default can be set for all transactions instantiated through this TransactionServiceClient instance.

### 3.2.1 begin

```
public Transaction begin()
public Transaction begin(final TransactionIsolationLevel isolationLevel);
public Transaction begin(final TransactionIsolationLevel isolationLevel,
                        final int seconds);
```

Proposal for a common transactional API for HBase

```
public Transaction begin(final TransactionIsolationLevel isolationLevel,  
    final int seconds, final TransactionType transactionType);  
  
    throws SystemException
```

Create a new transaction. A Transaction object is returned to represent the transaction. If TransactionIsolationLevel, transaction timeout (seconds) or TransactionType is specified as a parameter to begin, it overrides the value set for the TransactionServiceClient and the system wide defaults.

### 3.2.2 **constructFrom**

```
public Transaction constructFrom(Byte[] transactionString)  
  
    throws InvalidTransactionException,  
           IllegalStateException, SystemException
```

constructFrom constructs and returns a Transaction object from the opaque string returned by Transaction.streamTo. The content of the string is Transaction Manager dependent. It should not be inspected or modified by the application program. The intent of constructFrom and steamTo is to allow transaction identifiers to be passed between threads or processes in an implementation independent manner.

For constructFrom to return a valid Transaction object, the transaction must be exist (have been begun) and be in an active state according to the Transaction Manager. If the transaction does not exist or has already been forgotten by the Transaction Manager, constructFrom will throw an InvalidTransactionException exception. If the transaction is in a state other than active – ie it is in the process of committing or aborting, constructFrom will throw an IllegalStateException exception.

constructFrom provides the ability to perform work on an existing transaction from within a process or thread other than the beginner. Exactly how far this capability extends will be Transaction Manager dependent. Also, exactly what operations can be performed on a Transaction object instantiated by constructFrom will be Transaction Manager dependent. For example, some Transaction Managers may only support Transaction.commit for a transaction begun in the same process and not from other processes. Also Transaction instances created by constructFrom can be used in parallel with the Transaction instance from the beginner and with any other calls to constructFrom in the same or different threads or processes.

### 3.2.3 **getAll**

```
public Transaction[] getAll
```

Returns a list of all active transactions known to the Transaction Manager.

### 3.2.4 **setIsolationLevel**

```
public static void setIsolationLevel(final TransactionIsolationLevel isolationLevel)  
  
    throws NotSupportedException
```

Sets the default isolation level for all transactions begun using this TransactionServiceClient instance from the time this call is made until it is changed by another call to setIsolationLevel. IsolationLevel can be set for individual transactions when they are begun overriding this setting. See begin for more details. The system wide default isolation level is

Transaction Manager dependent and can be overridden in the hbase-site.xml file. If the Transaction Manager does not support the specified isolation level, a `NotSupportedException` is thrown.

### 3.2.5 `setTransactionType`

`public static void setTransactionType(final TransactionType transactionType)`

throws `NotSupportedException`

Sets the default transaction type for all transactions begun using this `TransactionServiceClient` instance from the time this call is made until it is changed by another call to `setTransactionType`. `TransactionType` can be set for individual transactions when they are begun overriding this setting. See `begin` for more details. The system wide default transaction type is `WRITE_READ`.

### 3.2.6 `setTransactionTimeout`

`public void setTransactionTimeout(final int seconds)`

throws `SystemException`

Modify the default transaction timeout for transactions begun using this `TransactionSystemClient` instance from the time this call is made until it is changed by another call to `setTransactionTimeout`. The default value is Transaction Manager dependent and can be overridden in the hbase-site.xml file. A value of -1 will set the timeout off such that transactions will never timeout.

## 3.3 Transaction Interface

The Transaction interface allows an application to control transactions which were begun by a call to `TransactionServiceClient.begin` or instantiated by a call to `TransactionServiceClient.constructFrom`. It is also used to associate a transaction with a table.

### 3.3.1 `commit`

`public void commit()`

throws `RollbackException`, `IllegalStateException`,  
`SystemException`

Tells the transaction that the application decided to commit. Once commit completes, the transaction has been forgotten by the system and references to the transaction object should be cleaned up.

### 3.3.2 `getStatus`

`public TransactionStatus getStatus()`

throws `IllegalStateException`, `SystemException`

Returns the status of the Transaction object. Possible values are listed at the beginning of the API Overview above.

### 3.3.3 rollback

```
public void rollback()
```

throws `IllegalStateException`, `SystemException`

Tells the transaction to rollback. Once rollback completes, the transaction has been forgotten by the system and references to the transaction object should be cleaned up.

## 3.4 TransactionTable Interface

The `TransactionTable` interface extends `HTableInterface`, allowing a transaction object to be specified so that work performed by the operation can be associated with the transaction. `TransactionTable` provides 2 ways to associate a transaction with the `HTable`. The Transaction object can be specified either as a parameter to the constructor or using the `TransactionTable.setTransaction` method. All other methods, including the standard `HTable` constructors retain the same signature and behave in an identical manner with one exception. If a `TransactionTable` was passed a valid Transaction object reference either by the constructor or through a call to `TransactionTable.setTransaction`, the operation will be performed with the specified transaction under control of the Transaction Manager. `TransactionTable` operations performed within a transaction should also have the same semantics as their `HTable` counterparts within the context of the transaction.

The `TransactionTable` object can be reused within a thread by calling `TransactionTable.setTransaction` to change the transaction associated with it, but should not be shared between threads as this could produce unpredictable results.

## 3.5 Exception Classes

```
public class IllegalStateException extends Exception
{
    public IllegalStateException();
    public IllegalStateException(String msg);
}
```

This exception indicates that the method was performed while the transaction was in an illegal state. An example is when `Transaction.commit` is called when the transaction is already committing the transaction.

```
public class InvalidTransactionException extends Exception
{
    public InvalidTransactionException();
    public InvalidTransactionException(String msg);
}
```

This exception indicates that the request was performed with an invalid transactionString. For example, when `TransactionServiceClient.constructFrom` is called with a transactionString which is not understood by the Transaction Manager.

```
public class NotSupportedException extends Exception
{
    public NotSupportedException();
    public NotSupportedException(String msg);
}
```

This exception is thrown when specified method is not supported in the current transaction context.

```
public class RollbackException extends Exception
{
    public RollbackException();
    public RollbackException(String msg);
}
```

This exception is thrown when the transaction has been rolled back even though the commit method was called. An example of this is when a transaction has been unilaterally aborted.

```
public class SystemException extends Exception
{
    public SystemException();
    public SystemException(String msg);
}
```

The SystemException is thrown to indicate that an unexpected error condition was encountered by the method.

## 3.6 Configuration

Default values for isolation level and transaction timeout can be overridden via properties in the hbase-site.xml or through the distributions web interface (eg Horton Works) by setting the following properties. Values give here are examples only.

```
<property>
  <name>hbase.transaction.isolationlevel</name>
  <value>2</value>
</property>
<property>
  <name>hbase.transaction.timeout</name>
  <value>-1</value>
</property>
```

## 4 Considerations

### 4.1 Threading

HTable is not thread-safe in HBase and hence TransactionTables will not be. TransactionTable instances can be associated with a series of transactions, but reuse should be restricted to the thread which instantiated it. This is because the association between transactions and the TransactionTable on which work is being performed is established once during construction or via a TransactionTable.setTransaction call. TransactionTable method calls are then performed assuming the transaction association has been maintained. Reuse of TransactionTable instances across threads can produce unpredictable results as the transaction association may be changed by another thread.

Unlike TransactionTable objects, Transaction objects can be shared across threads and be cloned between processes where they can be used in parallel with multiple threads and processes participating in the same transaction. See Transaction.streamTo and TransactionServiceClient.constructFrom for more details. While the API does not restrict transaction propagation and parallel execution, the Transaction Manager implementation may place limitations on it.

## 4.2 Isolation Levels

Isolation levels are named based on the ANSI SQL standard. Transaction Manager implementations need not support all levels and should throw a `NotSupportedException` if an unsupported level is specified.

Isolation level can be defaulted through a setting in the `hbase-site.xml` file, through a call to `TransactionSystemClient.setIsolationLevel`, or set for each transaction as a parameter to the `TransactionServiceClient.begin` method.

## 4.3 Transaction Types

The only transaction types defined are write-read and read-only. This is principally to provide support for read-only transactions which is considered a common Transaction Manager feature. Any transaction type not supported by the Transaction Manager will be rejected with a `NotSupportedException`.

No provision is provided to set a default value for transaction type because write-read is considered to be the natural default behaviour. Transaction type can be set by calling `TransactionServiceClient.setTransactionType`, and for each transaction as a parameter on `TransactionServiceClient.begin`.

## 4.4 Transaction Timeout

Transaction Managers generally time transactions out after a predefined period has elapsed. However, to allow for different transaction profiles, Transaction Managers generally provide a method to modify this value. Like isolation level, this can be set by default in the `hbase-site.xml` file, through `TransactionServiceClient.setTransactionTimeout`, and for each transaction as a parameter on `TransactionServiceClient.begin`.

# 5 Examples

This section provides a few examples to illustrate use of the API.

## 5.1 Simple example

This is a very simple example where the transaction is first begun, then a `TransactionTable` object created inheriting the `Transaction` object in its constructor call. Two puts are performed simply to make the transaction useful (multi-operation).

```
private static TransactionServiceClient txSC;
private static TransactionTable txTable;

txSC = new TransactionServiceClient(conf);
Configuration config = HBaseConfiguration.create();

Transaction tx1 = txSC.begin();

TransactionTable txTable = new TransactionTable(config, "table1", tx1);

Put p1 = new Put(Bytes.toBytes("row1"));
p1.add(Bytes.toBytes("cf"), Bytes.toBytes("q"), Bytes.toBytes("value1"));
txTable.put(p1);

Put p2 = new Put(Bytes.toBytes("row2"));
p2.add(Bytes.toBytes("cf"), Bytes.toBytes("q"), Bytes.toBytes("value2"));
txTable.put(p2);

tx1.commit();
```



## 5.2 Multiple transactions in sequence

This example extends the first showing that the TransactionTable instance can be reused across subsequent transactions. Notice that here the TransactionTable.setTransaction method has been used.

```
private static TransactionServiceClient txSC;
private static TransactionTable txTable;

txSC = new TransactionServiceClient(conf);
Configuration config = HBaseConfiguration.create();

TransactionTable txTable = new TransactionTable(config, "table1");

Transaction tx1 = txSC.begin();
txTable.setTransaction(tx1);

Put p1 = new Put(Bytes.toBytes("row1"));
p1.add(Bytes.toBytes("cf"), Bytes.toBytes("q"), Bytes.toBytes("value1"));
txTable.put(p1);

Put p2 = new Put(Bytes.toBytes("row2"));
p2.add(Bytes.toBytes("cf"), Bytes.toBytes("q"), Bytes.toBytes("value2"));
txTable.put(p2);

tx1.commit();

Transaction tx2 = txSC.begin();
txTable.setTransaction(tx2);

Put p1 = new Put(Bytes.toBytes("row3"));
p1.add(Bytes.toBytes("cf"), Bytes.toBytes("q"), Bytes.toBytes("value1"));
txTable.put(p1);

Put p2 = new Put(Bytes.toBytes("row4"));
p2.add(Bytes.toBytes("cf"), Bytes.toBytes("q"), Bytes.toBytes("value2"));
txTable.put(p2);

tx2.commit();
```

## 5.3 Multithread

This example is intended to show a single transaction being used by several threads in parallel. Thread 1 begins transaction tx1 and streams it to txnString. Thread 2 then uses txnString to create a local copy of the Transaction tx1 which is tx2. Note that within a process, it should also be possible to share the Transaction object tx1 across threads without the need to stream it to a string. However this example would also work across processes.

The two threads are performing work in parallel against the same transaction. It is the responsibility of the Transaction Manager to serialize this work to ensure transactional integrity. For instance, in the example, both threads update the row "row2" with different values. Because both threads are operating under the same transaction, it is possible for "row2" to have a final value of either "value2" or "value3". The concurrency control method (locking or MVCC) and isolation level will have no effect on this because both puts are performed within the same transaction.

This API does not stipulate which thread can commit or rollback the transaction. The particular Transaction Manager implementation may place restrictions on this. For example, the thread or process which began the transaction may be the only one allowed to commit or rollback. Whatever restrictions are in place, if both threads or processes attempt to commit or rollback the transaction, only one can succeed.

```
Byte[] txnString;
```

#### Thread 1

```
private static TransactionServiceClient txSC;
private static TransactionTable txTable;

txSC = new TransactionServiceClient(conf);
Configuration config = HBaseConfiguration.create();

Transaction tx1 = txSC.begin();
txnString = tx1.streamTo();

TransactionTable txTable =
    new TransactionTable(config, "table1", tx1);

Put p1 = new Put(Bytes.toBytes("row1"));
p1.add(Bytes.toBytes("cf"), Bytes.toBytes("q"),
    Bytes.toBytes("value1"));
txTable.put(p1);

Put p2 = new Put(Bytes.toBytes("row2"));
p2.add(Bytes.toBytes("cf"), Bytes.toBytes("q"),
    Bytes.toBytes("value2"));
txTable.put(p2);

tx1.commit();
```

#### Thread 2

```
private static TransactionServiceClient txSC1;
private static TransactionTable txTable1;

txSC = new TransactionServiceClient(conf);
Configuration config =
    HBaseConfiguration.create();

TransactionTable txTable1 =
    new TransactionTable(config, "table1");

Transaction tx2 = txSC1.constructFrom(txnString);
txTable1.setTransaction(tx2);

Put p1 = new Put(Bytes.toBytes("row2"));
p1.add(Bytes.toBytes("cf"), Bytes.toBytes("q"),
    Bytes.toBytes("value3"));
txTable1.put(p1);

Put p2 = new Put(Bytes.toBytes("row3"));
p2.add(Bytes.toBytes("cf"), Bytes.toBytes("q"),
    Bytes.toBytes("value4"));
txTable1.put(p2);
```

## 5.4 Setting the Isolation level, Transaction Type or Timeout

This example is the same as example 1 but shows a transaction being set to REPEATABLE\_READ isolation level.

Transaction type and timeout can be set in the same manner for each new transaction begun overriding both the default values and those set by calls to `setIsolationLevel`, `setTransactionType` and `setTransactionTimeout` against the `TransactionServiceClient`.

```
private static TransactionServiceClient txSC;
private static TransactionTable txTable;

txSC = new TransactionServiceClient(conf);
Configuration config = HBaseConfiguration.create();

Transaction tx1 = txSC.begin(REPEATABLE_READ);

TransactionTable txTable = new TransactionTable(config, "table1", tx1);

Put p1 = new Put(Bytes.toBytes("row1"));
p1.add(Bytes.toBytes("cf"), Bytes.toBytes("q"), Bytes.toBytes("value1"));
txTable.put(p1);

Put p2 = new Put(Bytes.toBytes("row2"));
p2.add(Bytes.toBytes("cf"), Bytes.toBytes("q"), Bytes.toBytes("value2"));
txTable.put(p2);
```

## 5.5 Factory example?

## 6 Limitations and Restrictions

- DDL operations such as creating and dropping HTables could and perhaps should be covered by the API. Where applications associate metadata with tables such as SQL implementations can create database inconsistencies where metadata changes are performed within a transaction but DDL operations are omitted.
- No support for the two-phased commit protocol. This API is intended for applications using a single HBase based Transaction Manager. Two-phased commit is used to control transaction outcomes where multiple logical resources are involved with the transaction. This is a Transaction Manager implementation detail.
- Heterogeneous transaction support such as that defined by JTA's XAResource or the X/Open XA specification is not supported by this proposal. This can be added if the proposal is accepted as an extension.
- Isolation levels do not indicate whether MVCC or Lock Management is to be used. We should probably add a concurrency protocol option to allow the user to distinguish between snapshot or MVCC based concurrency control protocols and lock management. This could be an additional method against the TransactionServiceClient. Alternatively, this could be viewed as a Transaction Manager implementation specific detail.
- Need to define the values for the enums defined as part of the API.

## 7 Unresolved/Questions

- Still need to define the Transaction Manager factory class.
- Should the TransactionServiceClass contain the begin and constructFrom methods or should these be constructors for the Transaction class?

## 8 References

- [1] S. Cheung and V. Matena, "JTA Specification," [Online]. Available: [http://download.oracle.com/otn-pub/jcp/7083-jta-1.0.1B-mr-spec-oth-JSpec/jta-1\\_0\\_1B-spec.pdf?AuthParam=1403051620\\_ceaa000c96e33e4a363e6c47c2807006](http://download.oracle.com/otn-pub/jcp/7083-jta-1.0.1B-mr-spec-oth-JSpec/jta-1_0_1B-spec.pdf?AuthParam=1403051620_ceaa000c96e33e4a363e6c47c2807006).
- [2] The Open Group, "Open Group Pubs," [Online]. Available: <http://pubs.opengroup.org/onlinepubs/009680699/toc.pdf>.