

# KIP-617: Allow Kafka Streams State Stores to be iterated backwards


- [Status](#)
- [Motivation](#)
  - [Reference issues](#)
- [Proposed Changes](#)
  - [Reverse Key Ranges](#)
  - [Backward Time Ranges](#)
- [Compatibility, Deprecation, and Migration Plan](#)
- [Rejected Alternatives](#)

## Status

**Current state:** Adopted

**Discussion thread:** [here](#)

**Vote thread:** [here](#)

**JIRA:**  Unable to render Jira issues macro, execution error.

Please keep the discussion on the mailing list rather than commenting on the wiki (wiki discussions get unwieldy fast).

## Motivation

Fetching range of records from Kafka Streams state stores comes with an iterator to traverse elements *from oldest to newest*, e.g `ReadOnlyWindowStore#fetch(K key, long fromTime, long toTime)` mentions:

*For each key, the iterator guarantees ordering of windows, **starting from the oldest/earliest***

Similar guarantees are provided on other fetch and range operations. But in the case of `key` ranges, there are some nuances regarding order:

*The returned iterator must be safe from `{@link java.util.ConcurrentModificationException}`s and must not return null values. **No ordering guarantees are provided.***

Ordering is not guaranteed as backing structure is based on maps keyed by `o.a.k.common.utils.Bytes`. Though, Bytes support Lexicographic byte array comparison, which defines ordering in-memory and RocksDB stores.

These APIs constraint the usage of local state store for some use-cases:

When storing records on time windows, or records by key; and an operation wants to return the *last* `N` values inserted withing a time range containing `M` records: Currently there is no alternative other than iterating records from oldest to newest—traversing `M` records, where `M` » `N`.

If a *backward read direction* option becomes available, then we could start from the latest record within a time range and go *backwards*, returning the first `N` value more efficiently.

At [Zipkin Kafka-based storage](#), we are planning to use this feature to replace two `KeyValueStores` (one for traces indexed by id, and another with `trace_ids` indexed by timestamp) for one `WindowStore`. A backward read direction will allow to support queries like: “within this time range, find the last traces that match this criteria”, and return latest values quickly.

Internally, both implementations: persistent (RocksDB), and in-memory (`TreeMap`) support reverse/descending iteration:

```
final RocksIterator iter = db.newIterator();
iter.seekToFirst();
iter.next();
final RocksIterator reverse = db.newIterator();
reverse.seekToLast();
reverse.prev();

final TreeMap<String, String> map = new TreeMap<>();
final NavigableSet<String> nav = map.navigableKeySet();
final NavigableSet<String> rev = map.descendingKeySet();
```

## Reference issues

- <https://issues.apache.org/jira/browse/KAFKA-9929>
- <https://issues.apache.org/jira/browse/KAFKA-4212>

## Proposed Changes

There are 2 important ranges in Kafka Streams Stores:

- Key Range
- Time Range



For SessionStore/ReadOnlySessionStore: findSessions and findSession operations will be moved from SessionStore to ReadOnlySessionStore to align with how other stores are design.

## Reverse Key Ranges

Extend existing interface for reverse KeyValueStore

```
public interface ReadOnlyKeyValueStore<K, V> {
    default KeyValueIterator<K, V> reverseRange(K from, K to) {
        throw new UnsupportedOperationException();
    }
    default KeyValueIterator<K, V> reverseAll() {
        throw new UnsupportedOperationException();
    }
}
```

## Backward Time Ranges

Window and Session stores are based on a set of KeyValue Stores (Segments) organized by a time-based index. Therefore, for these stores time-range is more important than key-range to lookup for values.

Existing stores will be extended with backward methods:

```

public interface ReadOnlyWindowStore<K, V> {
    default WindowStoreIterator<V> backwardFetch(K key, Instant from, Instant to) throws
    IllegalArgumentException {
        throw new UnsupportedOperationException();
    }

    default KeyValueIterator<Windowed<K>, V> backwardFetch(K from, K to, Instant fromTime, Instant toTime)
    throws IllegalArgumentException {
        throw new UnsupportedOperationException();
    }

    default KeyValueIterator<Windowed<K>, V> backwardAll() {
        throw new UnsupportedOperationException();
    }

    default KeyValueIterator<Windowed<K>, V> backwardFetchAll(Instant from, Instant to) throws
    IllegalArgumentException {
        throw new UnsupportedOperationException();
    }
}

public interface ReadOnlySessionStore<K, AGG> {
    // Moving read functions from SessionStore to ReadOnlySessionStore
    default KeyValueIterator<Windowed<K>, AGG> findSessions(final K key, final long earliestSessionEndTime,
    final long latestSessionStartTime) {
        throw new UnsupportedOperationException("Moved from SessionStore");
    }

    default KeyValueIterator<Windowed<K>, AGG> findSessions(final K keyFrom, final K keyTo, final long
    earliestSessionEndTime, final long latestSessionStartTime) {
        throw new UnsupportedOperationException("Moved from SessionStore");
    }

    default AGG fetchSession(final K key, final long startTime, final long endTime) {
        throw new UnsupportedOperationException("Moved from SessionStore");
    }

    // New
    default KeyValueIterator<Windowed<K>, AGG> backwardFindSessions(final K key, final long
    earliestSessionEndTime, final long latestSessionStartTime) {
        throw new UnsupportedOperationException();
    }

    default KeyValueIterator<Windowed<K>, AGG> backwardFindSessions(final K keyFrom, final K keyTo, final long
    earliestSessionEndTime, final long latestSessionStartTime) {
        throw new UnsupportedOperationException();
    }

    default KeyValueIterator<Windowed<K>, AGG> backwardFetch(final K key) {
        throw new UnsupportedOperationException();
    }
    default KeyValueIterator<Windowed<K>, AGG> backwardFetch(final K from, final K to) {
        throw new UnsupportedOperationException();
    }
}

```

## Compatibility, Deprecation, and Migration Plan

New methods will have default implementations to avoid affecting current implementations.

## Rejected Alternatives

- Create a parallel hierarchy of interfaces for backward operation. Even though this option seems like the best way to extend functionality, it was proved to not work in practice in KIP-614 discussion as interfaces get wrapped in different layers (Metered, Caching, Logging) so all the current hierarchy to create stores with Kafka Streams DSL will have to be duplicated.
- Initially it was considered to have additional parameter on all `ReadOnlyStore` methods e.g. `Store#fetch(keyFrom, keyTo, timeFrom, timeTo, ReadDirection.FORWARD|BACKWARD)`, but has been declined as passing arguments in inverse is more intuitive. As this could cause unexpected effects in future versions, a flag has been added to overcome this.
- Implicit ordering by flipping `from` and `to` variables has been discouraged in favor of a more explicit approach based on new interfaces that make explicit the availability of reverse and backward fetch operations.