

KIP-244: Add Record Header support to Kafka Streams Processor API


- [Status](#)
- [Motivation](#)
- [Public Interfaces](#)
- [Proposed Changes](#)
 - [Headers Inheritance](#)
- [Compatibility, Deprecation, and Migration Plan](#)
- [Rejected Alternatives](#)
- [Future Work](#)

Status

Current state: Adopted in release 2.0

Discussion thread: [here](#)

JIRA:

key	summary	type	created	updated	due	assignee	reporter	priority	status	resolution
 JQL and issue key arguments for this macro require at least one Jira application link to be configured										

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

Motivation

Headers have been introduced in almost all Kafka components (broker, producer API, consumer API, connect API). This KIP is aimed to add Record Headers support as part of Streams Processor API first, to then discuss about how to approach its support on the DSL API level.

Headers can be used on different scenarios (e.g. propagating Tracing context between different components, operational information that can be used for filtering, etc.).

Headers must be propagated downstream to make them available on Sinks, and available on Processors to be able to manipulate them (e.g. in the cases of Distributed Tracing, it will be used to create Spans and injecting context for following Nodes), using Headers API.

Public Interfaces

Add headers as part of processing:

```
org.apache.kafka.streams.processor.ProcessorContext:
```

```
* Headers headers();
```

```
org.apache.kafka.streams.processor.MockProcessorContext:
```

```
* setRecordMetadata(final String topic, final int partition, final long offset, final Headers headers,  
final long timestamp)
```

```
* void setHeaders(final Headers headers)
```

```
* Headers headers();
```

```
org.apache.kafka.streams.test.ConsumerRecordFactory:
```

```

    * ConsumerRecord<byte[], byte[]> create(final String topicName, final K key, final V value, final Headers
headers, final long timestampMs)

    * ConsumerRecord<byte[], byte[]> create(final String topicName, final K key, final V value, final Headers
headers)

    * ConsumerRecord<byte[], byte[]> create(final String topicName, final V value, final Headers headers, final
long timestampMs)

    * ConsumerRecord<byte[], byte[]> create(final String topicName, final V value, final Headers headers)

    * ConsumerRecord<byte[], byte[]> create(final K key, final V value, final Headers headers, final long
timestampMs)

    * ConsumerRecord<byte[], byte[]> create(final K key, final V value, final Headers headers)

    * ConsumerRecord<byte[], byte[]> create(final V value, final Headers headers, final long timestampMs)

    * ConsumerRecord<byte[], byte[]> create(final V value, final Headers headers)

org.apache.kafka.streams.test.OutputVerifier

    * void compareValueHeaders(final ProducerRecord<K, V> record, final V expectedValue, final Headers
expectedHeaders)

    * void compareValueHeaders(final ProducerRecord<K, V> record, final ProducerRecord<K, V> expectedRecord)

    * void compareKeyValueHeaders(final ProducerRecord<K, V> record, final K expectedKey K, final V expectedValue,
final Headers expectedHeaders)

    * void compareKeyValueHeaders(final ProducerRecord<K, V> record, final ProducerRecord<K, V> expectedRecord)

    * void compareKeyValueHeadersTimestamp(final ProducerRecord<K, V> record, final K expectedKey K, final V
expectedValue, final Headers expectedHeaders, final long expectedTimestamp)

    * void compareKeyValueHeadersTimestamp(final ProducerRecord<K, V> record, final ProducerRecord<K, V>
expectedRecord)

```

Proposed Changes

Adding `headers()` to `ProcessorContext` will enable custom processors and future DSL processors to have Headers available.

Internally, some components need to have headers available on the ProcessorContext, like:

```

* o.a.k.s.p.i.AbstractProcessorContext
* o.a.k.s.p.i.GlobalStateUpdateTask
* o.a.k.s.p.i.ProcessorRecordContext
* o.a.k.s.p.i.RecordCollector
* o.a.k.s.p.i.RecordCollectorImpl
* o.a.k.s.p.i.RecordContext
* o.a.k.s.p.i.RecordDeserialized
* o.a.k.s.p.i.SinkNode
* o.a.k.s.p.i.StampedRecord
* o.a.k.s.p.i.LRUCacheEntry

```

More details on PR: <https://github.com/apache/kafka/pull/4955>

In the case of stateful applications, consider that headers are not stored in state-stores; therefore only the headers from the current record in process is available.

Headers Inheritance

1. To make the inheritance implementation of headers consistent with what we had with other record context fields. I.e. pass through the record context in `context.forward()`. Note that within a processor node, users can already manipulate the Headers with the given APIs, so at the time of forwarding, the library can just copy what-ever is left / updated to the next processor node.

2. In the sink node, where a record is being sent to the Kafka topic, we should consider the following:

- a. For sink topics, we will set the headers into the producer record.
- b. For repartition topics, we will the headers into the producer record.
- c. For changelog topics, we will drop the headers in the produce record since they will not be used in restoration and not stored in the state store either.

Compatibility, Deprecation, and Migration Plan

- Clients using High-Level DSL and Processor API should not be affected with changes proposed.
- As Headers are propagated downstream, Clients that have Sources with Records that have Headers, will end up with Headers on Sink Topics.

Rejected Alternatives

~~1. Adding Headers to KTable API will mean propagate Headers to Stores that are Key Value specific like RocksDB. If headers are required in stateful operations, clients will need to map headers values first to key or value and then do processing.~~

2. Adding Headers to DSL API.

Future Work

- Adding DSL Processors to use Headers to filter/map/branch. Potentially supported by [KIP-159](#).