KIP-149: Enabling key access in ValueTransformer, ValueMapper, and ValueJoiner

Status

Current state: *"accepted"*

Discussion thread: HERE

JIRA: *KAFKA-4218, KAFKA-4726, KAFKA-3745*, *KAFKA-7842*, *KAFKA-7843*

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

The PR can be found here.

Motivation

(taken from JIRA descriptions)

- **Key access to ValueTransformer**: While transforming values via `KStream.transformValues` and `ValueTransformer`, the key associated with the value may be needed, even if it is not changed. For instance, it may be used to access stores.

  As of now, the key is not available within these methods and interfaces, leading to the use of `KStream.transform` and `Transformer`, and the unnecessary creation of new `KeyValue` objects.
- **Key access to ValueMapper**: `ValueMapper` should have read-only access to the key for the value it is mapping. Sometimes the value transformation will depend on the key.

  It is possible to do this with a full blown `KeyValueMapper` but that loses the promise that you won't change the key – so you might introduce a re-keying phase that is totally unnecessary.
- **Key access to ValueJoiner interface:** In working with Kafka Stream joining, it's sometimes the case that a join key is not actually present in the values of the joins themselves (if, for example, a previous transform generated an ephemeral join key.) In such cases, the actual key of the join is not available in the ValueJoiner implementation to be used to construct the final joined value. This can be worked around by explicitly threading the join key into the value if needed, but it seems like extending the interface to pass the join key along as well would be helpful

Additionally we consider adding key access to `Initializer` and `Reducer` interfaces.

Public Changes

- **`KStream`** interface:

```
<GK, GV, RV> KStream<K, RV> leftJoin(final GlobalKTable<GK, GV> globalKTable,
                                     final KeyValueMapper<? super K, ? super V, ? extends GK> keyValueMapper,
                                     final ValueJoinerWithKey<? super K, ? super V, ? super GV, ? extends RV>
valueJoinerWithKey);

<GK, GV, RV> KStream<K, RV> join(final GlobalKTable<GK, GV> globalKTable,
                                 final KeyValueMapper<? super K, ? super V, ? extends GK> keyValueMapper,
                                 final ValueJoinerWithKey<? super K, ? super V, ? super GV, ? extends RV>
valueJoinerWithKey);

<VT, VR> KStream<K, VR> leftJoin(final KTable<K, VT> table,
                                 final ValueJoinerWithKey<? super K, ? super V, ? super VT, ? extends VR>
valueJoinerWithKey,
```

```
                                final Serde<K> keySerde,
                                final Serde<V> valSerde);

<VT, VR> KStream<K, VR> leftJoin(final KTable<K, VT> table,
                                final ValueJoinerWithKey<? super K, ? super V, ? super VT, ? extends VR>
valueJoinerWithKey);

<VT, VR> KStream<K, VR> join(final KTable<K, VT> table,
                             final ValueJoinerWithKey<? super K, ? super V, ? super VT, ? extends VR>
valueJoinerWithKey,
                             final Serde<K> keySerde,
                             final Serde<V> valSerde);

<VT, VR> KStream<K, VR> join(final KTable<K, VT> table,
                             final ValueJoinerWithKey<? super K, ? super V, ? super VT, ? extends VR>
valueJoinerWithKey);

<VO, VR> KStream<K, VR> outerJoin(final KStream<K, VO> otherStream,
                                  final ValueJoinerWithKey<? super K, ? super V, ? super VO, ? extends VR>
valueJoinerWithKey,
                                  final JoinWindows windows,
                                  final Serde<K> keySerde,
                                  final Serde<V> thisValueSerde,
                                  final Serde<VO> otherValueSerde);

<VO, VR> KStream<K, VR> outerJoin(final KStream<K, VO> otherStream,
                                  final ValueJoinerWithKey<? super K, ? super V, ? super VO, ? extends VR>
valueJoinerWithKey,
                                  final JoinWindows windows);


<VO, VR> KStream<K, VR> leftJoin(final KStream<K, VO> otherStream,
                                final ValueJoinerWithKey<? super K, ? super V, ? super VO, ? extends VR>
valueJoinerWithKey,
                                final JoinWindows windows,
                                final Serde<K> keySerde,
                                final Serde<V> thisValSerde,
                                final Serde<VO> otherValueSerde);

<VO, VR> KStream<K, VR> leftJoin(final KStream<K, VO> otherStream,
                                final ValueJoinerWithKey<? super K, ? super V, ? super VO, ? extends VR>
valueJoinerWithKey,
                                final JoinWindows windows);

<VO, VR> KStream<K, VR> join(final KStream<K, VO> otherStream,
                             final ValueJoinerWithKey<? super K, ? super V, ? super VO, ? extends VR>
valueJoinerWithKey,
                             final JoinWindows windows,
                             final Serde<K> keySerde,
                             final Serde<V> thisValueSerde,
                             final Serde<VO> otherValueSerde);

<VO, VR> KStream<K, VR> join(final KStream<K, VO> otherStream,
                             final ValueJoinerWithKey<? super K, ? super V, ? super VO, ? extends VR>
valueJoinerWithKey,
                             final JoinWindows windows);

<VR> KStream<K, VR> transformValues(final ValueTransformerWithKeySupplier<? super K, ? super V, ? extends VR>
valueTransformerWithKeySupplier,
                                    final String... stateStoreNames);

<VR> KStream<K, VR> mapValues(ValueMapperWithKey<? super K, ? super V, ? extends VR> mapperWithKey);
```

- **`KTable`** interface:

KTable interface

```
<VO, VR> KTable<K, VR> join(final KTable<K, VO> other,
                            final ValueJoinerWithKey<? super K, ? super V, ? super VO, ? extends VR> joiner);

<VO, VR> KTable<K, VR> join(final KTable<K, VO> other,
                            final ValueJoinerWithKey<? super K, ? super V, ? super VO, ? extends VR> joiner,
                            final Materialized<K, VR, KeyValueStore<Bytes, byte[]>>
materialized);

<VO, VR> KTable<K, VR> join(final KTable<K, VO> other,
                            final ValueJoinerWithKey<? super K, ? super V, ? super VO, ? extends VR> joiner,
                            final Named named);

<VO, VR> KTable<K, VR> join(final KTable<K, VO> other,
                            final ValueJoinerWithKey<? super K, ? super V, ? super VO, ? extends VR> joiner,
                            final Named named,
                            final Materialized<K, VR, KeyValueStore<Bytes, byte[]>> materialized);

<VR, KO, VO> KTable<K, VR> join(final KTable<KO, VO> other,
                                final Function<V, KO> foreignKeyExtractor,
                                final ValueJoinerWithKey<KO, V, VO, VR> joiner);

<VR, KO, VO> KTable<K, VR> join(final KTable<KO, VO> other,
                                final Function<V, KO> foreignKeyExtractor,
                                final ValueJoinerWithKey<KO, V, VO, VR> joiner,
                                final Materialized<K, VR, KeyValueStore<Bytes, byte[]>> materialized);

<VR, KO, VO> KTable<K, VR> join(final KTable<KO, VO> other,
                                final Function<V, KO> foreignKeyExtractor,
                                final ValueJoinerWithKey<KO, V, VO, VR> joiner,
                                final Named named);

<VR, KO, VO> KTable<K, VR> join(final KTable<KO, VO> other,
                                final Function<V, KO> foreignKeyExtractor,
                                final ValueJoinerWithKey<KO, V, VO, VR> joiner,
                                final Named named,
                                final Materialized<K, VR, KeyValueStore<Bytes, byte[]>> materialized);


<VO, VR> KTable<K, VR> leftJoin(final KTable<K, VO> other,
                                final ValueJoinerWithKey<? super K, ? super V, ? super VO, ? extends VR>
joiner);

<VO, VR> KTable<K, VR> leftJoin(final KTable<K, VO> other,
                                final ValueJoinerWithKey<? super K, ? super V, ? super VO, ? extends VR> joiner,
                                final Materialized<K, VR, KeyValueStore<Bytes, byte[]>>
materialized);

<VO, VR> KTable<K, VR> leftJoin(final KTable<K, VO> other,
                                final ValueJoinerWithKey<? super K, ? super V, ? super VO, ? extends VR> joiner,
                                final Named named);

<VO, VR> KTable<K, VR> leftJoin(final KTable<K, VO> other,
                                final ValueJoinerWithKey<? super K, ? super V, ? super VO, ? extends VR> joiner,
                                final Named named,
                                final Materialized<K, VR, KeyValueStore<Bytes, byte[]>> materialized);

<VR, KO, VO> KTable<K, VR> leftJoin(final KTable<KO, VO> other,
                                    final Function<V, KO> foreignKeyExtractor,
                                    final ValueJoinerWithKey<KO, V, VO, VR>
joiner);

<VR, KO, VO> KTable<K, VR> leftJoin(final KTable<KO, VO> other,
                                          final Function<V, KO> foreignKeyExtractor,
                                          final ValueJoinerWithKey<KO, V, VO, VR> joiner,
                                          final Materialized<K, VR, KeyValueStore<Bytes, byte[]>>
materialized);

<VR, KO, VO> KTable<K, VR> leftJoin(final KTable<KO, VO> other,
```

```
                                        final Function<V, KO> foreignKeyExtractor,
                                        final ValueJoinerWithKey<KO, V, VO, VR> joiner,
                                        final Named named);

<VR, KO, VO> KTable<K, VR> join(final KTable<KO, VO> other,
                                final Function<V, KO> foreignKeyExtractor,
                                final ValueJoinerWithKey<KO, V, VO, VR> joiner,
                                final Named named,
                                final Materialized<K, VR, KeyValueStore<Bytes, byte[]>> materialized);
```

- **`KGroupedStream`** interface:

```
KTable<K, V> reduce(final ReducerWithKey<K, V> reducerWithKey);

KTable<K, V> reduce(final ReducerWithKey<K, V> reducerWithKey,
                    final String queryableStoreName);

KTable<K, V> reduce(final ReducerWithKey<K, V> reducerWithKey,
                    final StateStoreSupplier<KeyValueStore> storeSupplier);

<W extends Window> KTable<Windowed<K>, V> reduce(final ReducerWithKey<K, V> reducerWithKey,
                                                 final Windows<W> windows,
                                                 final String queryableStoreName);
<W extends Window> KTable<Windowed<K>, V> reduce(final Reducer<V> reducer,
                                                 final Windows<W> windows);

<W extends Window> KTable<Windowed<K>, V> reduce(final ReducerWithKey<K, V> reducerWithKey,
                                                 final Windows<W> windows,
                                                 final StateStoreSupplier<WindowStore> storeSupplier);

KTable<Windowed<K>, V> reduce(final ReducerWithKey<K, V> reducerWithKey,
                              final SessionWindows sessionWindows,
                              final String queryableStoreName);

KTable<Windowed<K>, V> reduce(final ReducerWithKey<K, V> reducerWithKey,
                              final SessionWindows sessionWindows);

KTable<Windowed<K>, V> reduce(final ReducerWithKey<K, V> reducerWithKey,
                              final SessionWindows sessionWindows,
                              final StateStoreSupplier<SessionStore> storeSupplier);



<VR> KTable<K, VR> aggregate(final InitializerWithKey<K, VR> initializerWithKey,
                             final Aggregator<? super K, ? super V, VR> aggregator,
                             final Serde<VR> aggValueSerde,
                             final String queryableStoreName);

<VR> KTable<K, VR> aggregate(final InitializerWithKey<K, VR> initializerWithKey,
                             final Aggregator<? super K, ? super V, VR> aggregator,
                             final Serde<VR> aggValueSerde);

<VR> KTable<K, VR> aggregate(final InitializerWithKey<K, VR> initializerWithKey,
                             final Aggregator<? super K, ? super V, VR> aggregator,
                             final StateStoreSupplier<KeyValueStore> storeSupplier);

<W extends Window, VR> KTable<Windowed<K>, VR> aggregate(final InitializerWithKey<K, VR> initializerWithKey,
                                                         final Aggregator<? super K, ? super V, VR> aggregator,
                                                         final Windows<W> windows,
                                                         final Serde<VR> aggValueSerde,
                                                         final String queryableStoreName);
```

```
<W extends Window, VR> KTable<Windowed<K>, VR> aggregate(final InitializerWithKey<K, VR> initializerWithKey,
                                                         final Aggregator<? super K, ? super V, VR> aggregator,
                                                         final Windows<W> windows,
                                                         final Serde<VR> aggValueSerde);

<W extends Window, VR> KTable<Windowed<K>, VR> aggregate(final InitializerWithKey<K, VR> initializerWithKey,
                                                         final Aggregator<? super K, ? super V, VR> aggregator,
                                                         final Windows<W> windows,
                                                         final StateStoreSupplier<WindowStore> storeSupplier);



<T> KTable<Windowed<K>, T> aggregate(final InitializerWithKey<K, VR> initializerWithKey,
                                     final Aggregator<? super K, ? super V, T> aggregator,
                                     final Merger<? super K, T> sessionMerger,
                                     final SessionWindows sessionWindows,
                                     final Serde<T> aggValueSerde,
                                     final String queryableStoreName);


<T> KTable<Windowed<K>, T> aggregate(final InitializerWithKey<K, VR> initializerWithKey,
                                     final Aggregator<? super K, ? super V, T> aggregator,
                                     final Merger<? super K, T> sessionMerger,
                                     final SessionWindows sessionWindows,
                                     final Serde<T> aggValueSerde);


<T> KTable<Windowed<K>, T> aggregate(final Initializer<T> initializer,
                                     final Aggregator<? super K, ? super V, T> aggregator,
                                     final Merger<? super K, T> sessionMerger,
                                     final SessionWindows sessionWindows,
                                     final Serde<T> aggValueSerde,
                                     final StateStoreSupplier<SessionStore> storeSupplier);
```

Proposed Changes

- Handling lambdas

For `ValueMapper, ValueJoiner and their "withKey"` interfaces we support lambdas. For `ValueTransformer` interface we don't need lambdas by the core definition of the class.

To support lambdas, we separate withKey interface from original ones, meaning we don't inherit or extend from one to another.

- `ValueMapperWithKey`

```
public interface ValueMapperWithKey<K, V, VR> {
    VR apply(final K readOnlyKey, final V value);
}
```

- `ValueJoinerWithKey`

```
public interface ValueJoinerWithKey<K, V1, V2, VR> {
    VR apply(final K readOnlyKey, final V1 value1, final V2 value2);
}
```

- `ValueTransformerWithKeySupplier`

```
public interface ValueTransformerWithKeySupplier<K, V, VR> {
    ValueTransformerWithKey<K, V, VR> get();
}

public interface ValueTransformerWithKey<K, V, VR> {
        void init(final ProcessorContext context);
    VR transform(final K readOnlyKey, final V value);
    void close();
}
```

- `ReducerWithKey`

```
public interface ReducerWithKey<K, V> {
        V apply(final K readOnlyKey, final V value1, final V value2);
}
```

- `InitializerWithKey`

```
public interface InitializerWithKey<K, VA> {
        VA apply(final K readOnlyKey);
}
```

Handling withKey interfaces while building the topology

In general, we change the constructors of all related backend Processors to be withKey types as we can easily convert regular (withoutKey) interfaces to withKey interfaces.

- `ValueMapperWithKey`

```
@Override
public <V1> KStream<K, V1> mapValues(final ValueMapper< ? super V, ? extends V1> mapper) {
    Objects.requireNonNull(mapperWithKey, "mapperWithKey can't be null");
    String name = topology.newName(MAPVALUES_NAME);
    final ValueMapperWithKey<K, V, V1> valueMapperWithKey = new ValueMapperWithKey<K, V, V1>() {
            @Override
            public V1 apply(K key, V value) {
                return mapper(value);
            }
        };
    }
    topology.addProcessor(name, new KStreamMapValues<>(valueMapperWithKey), this.name);
    return new KStreamImpl<>(topology, name, sourceNodes, this.repartitionRequired);
}
```

- `ValueJoinerWithKey`

```
static <K, T1, T2, R> ValueJoinerWithKey<K, T1, T2, R> convertToValueJoinerWithKey(final ValueJoiner<T1, T2, R>
valueJoiner) {
    Objects.requireNonNull(valueJoiner, "valueJoiner can't be null");
    return new ValueJoinerWithKey<K, T1, T2, R>() {
        @Override
        public R apply(K key, T1 value1, T2 value2) {
            return valueJoiner.apply(value1, value2);
        }
    };
}


public <V1, R> KStream<K, R> leftJoin(
    final KStream<K, V1> other,
    final ValueJoiner<? super V, ? super V1, ? extends R> joiner,
    final JoinWindows windows,
    final Serde<K> keySerde,
    final Serde<V> thisValSerde,
    final Serde<V1> otherValueSerde) {

    return doJoin(other,
        convertToValueJoinerWithKey(joiner),   // doJoin, join methods, and corresponding Processors accept
ValueJoinerWithKey type.
        windows,
        keySerde,
        thisValSerde,
        otherValueSerde,
        new KStreamImplJoin(true, false));
}
```

Test Plan

The unit tests are changed accordingly to support the changes in core classes.

Rejected Alternatives

- **Lambdas are not supported**

This document is proposed with `ValueMapper` example but it can be applied to other interfaces as well. Rich functions are proposed:

```
public interface RichFunction {
    void init(final ProcessorContext context);

 void close();
}

public abstract class AbstractRichFunction implements RichFunction {
    @Override
  public void init(final ProcessorContext context) {}

    @Override
  public void close() {}
}
```

```
public abstract class RichValueJoiner<K, V1, V2, VR> extends AbstractRichFunction implements ValueJoiner<V1, V2, VR
>  {
    @Override
  public final VR apply(final V1 value1, final V2 value2) {
        return apply(null, value1, value2);
  }

    public abstract VR apply(final K key, final V1 value1, final V2 value2);
}
```

Inside processor, we check if the instance (for example `ValueMapper` instance) is rich (for example `RichValueMapper`):

```
KStreamFlatMapValues(ValueMapper<? super V, ? extends Iterable<? extends V1>> mapper) {
    this.mapper = mapper;
  isRichFunction = mapper instanceof RichValueMapper ? true : false;
}

@Override
public void process(K key, V value) {
    Iterable<? extends V1> newValues;
 if (isRichFunction) {
        newValues = ((RichValueMapper<? super K, ? super V, ? extends Iterable<? extends V1>>) mapper).apply(key,
value);
  } else {
        newValues = mapper.apply(value);
  }
    for (V1 v : newValues) {
        context().forward(key, v);
  }
}
```

- **Not backward-compatible**

We propose adding key information for `ValueJoiner`, `ValueTransformer`, and `ValueMapper` classes and their `apply(...)` methods.

As a result, we perform the following public changes (and their overloaded versions)

Class	Old	New
KStream	<VR> KStream<K, VR> mapValues(ValueMapper<? super V, ? extends VR> mapper);	<VR> KStream<K, VR> mapValues(ValueMapper<? super K, ? super V , ? extends VR> mapper);
KStream	<VR> KStream<K, VR> transformValues(final ValueTransformerSupplier<? super V, ? extends VR> valueTransformerSupplier, final String... stateStoreNames);	<VR> KStream<K, VR> transformValues(final ValueTransformerSupplier<? super K, ? super V, ? extends VR> valueTransformerSupplier,final String... stateStoreNames);
KStream	<VO, VR> KStream<K, VR> join(final KStream<K, VO> otherStream , final ValueJoiner<? super V, ? super VO, ? extends VR> joiner, final JoinWindows windows);	<VO, VR> KStream<K, VR> join(final KStream<K, VO> otherStream, final ValueJoiner<? super K, ? super V, ? super VO, ? extends VR> joiner, final JoinWindows windows);
KTable	<VR> KTable<K, VR> mapValues(final ValueMapper<? super V, ? extends VR> mapper);	<VR> KTable<K, VR> mapValues(final ValueMapper<? super K, ? super V, ? extends VR> mapper);
KTable	<VO, VR> KTable<K, VR> join(final KTable<K, VO> other, final ValueJoiner<? super V, ? super VO, ? extends VR> joiner);	<VO, VR> KTable<K, VR> join(final KTable<K, VO> other, final ValueJoiner<? super K, ? super V, ? super VO, ? extends VR> joiner);

- **Lacking performance because deep-copy and need for RichFunctions**

1. We extend the target interfaces `ValueJoiner`, `ValueTransformer`, and `ValueMapper as ValueJoinerWithKey`, `ValueTransformer WithKey`, and `ValueMapperWithKey. In extended abstract classes we have an access to keys.`
2. In Processor we check the actual instance of object:

```
this.valueTransformer = valueTransformer;
if (valueTransformer instanceof ValueTransformerWithKey) {
isTransformerWithKey = true;
} else {
isTransformerWithKey = false;
}
```

.....

.....

```
@Override
public void process(K key, V value) {
if (isTransformerWithKey) {
K keyCopy = (K) Utils.deepCopy(key);
  context.forward(key, ((ValueTransformerWithKey<K, V, R>) valueTransformer).transform(keyCopy, value));
  } else {
context.forward(key, valueTransformer.transform(value));
  }
}
```

3. As we can see from the above code snippet, we can guard the key change in Processors by deeply copying the object before calling the `apply()` method.