KIP-485: Make topic optional when using through() operations in DSL

Status

Current state: *Moved to KIP-221: Enhance DSL with Connecting Topic Creation and Repartition Hint*

Discussion thread: https://www.mail-archive.com/dev@kafka.apache.org/msg99085.html

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

When using DSL in Kafka Streams, data re-partition happens only when key-changing operation is followed by stateful operation. On the other hand, in DSL, stateful computation can happen using *transform()* operation as well. Problem with this approach is that, even if any upstream operation was key-changing before calling *transform()*, no auto-repartition is triggered. If repartitioning is required, a call to *through(String)* should be performed before *transform()*. With the current implementation, burden of managing and creating the topic falls on user and introduces extra complexity of managing Kafka Streams application.

Public Interfaces

```
public interface KStream<K, V> {
    ...

    /**
     * Materialize this stream to a topic and creates a new {@code KStream} from the topic using default
serializers,
     * deserializers, and producer's {@link DefaultPartitioner}.
     * Topic will be created and managed internally by Kafka Streams.
     * <p>
     * This is similar to calling {@link #to(String) #to(someTopicName)} and
     * {@link StreamsBuilder#stream(String) StreamsBuilder#stream(someTopicName)}.
     * Note that {@code through()} uses a hard coded {@link org.apache.kafka.streams.processor.
FailOnInvalidTimestamp
     * timestamp extractor} and does not allow to customize it, to ensure correct timestamp propagation.
     *
     * @return a {@code KStream} that contains the exact same (and potentially repartitioned) records as this
{@code KStream}
     */
    KStream<K, V> through();

    /**
     * Materialize this stream to a topic and creates a new {@code KStream} using the
     * {@link Produced} instance for configuration of the {@link Serde key serde}, {@link Serde value serde},
     * and {@link StreamPartitioner}.
     * Topic will be created and managed internally by Kafka Streams using generated processor name.
     * <p>
     * This is similar to calling {@link #to(String, Produced) to(someTopic, Produced.with(keySerde,
valueSerde)}
     * and {@link StreamsBuilder#stream(String, Consumed) StreamsBuilder#stream(someTopicName, Consumed.with
(keySerde, valueSerde))}.
     * Note that {@code through()} uses a hard coded {@link org.apache.kafka.streams.processor.
FailOnInvalidTimestamp
     * timestamp extractor} and does not allow to customize it, to ensure correct timestamp propagation.
     *
     * @param produced  the options to use when producing to the topic
     * @return a {@code KStream} that contains the exact same (and potentially repartitioned) records as this
{@code KStream}
     */
    KStream<K, V> through(final Produced<K, V> produced);

    ...
}
```

Proposed Changes

Add two method overloads for **through()**

- *KStream#through()* - with default serializers, deserializers and producer's *DefaultPartitioner*
- *KStream#through(final Produced<K, V> produced) - produced* parameter will be used for configuration, in a similar way as for existing *through (String topic, final Produced<K, V> produced)* method. Topic will be generated based *through()* processor name.

Compatibility, Deprecation, and Migration Plan

Since we're adding two new method overloads without modifying existing API contract, there won't be any compatibility issues.

Rejected Alternatives

N/A