

KIP-401: TransformerSupplier/ProcessorSupplier StateStore connecting


- [Status](#)
- [Motivation](#)
- [Public Interfaces](#)
 - [ConnectedStoreProvider](#)
 - [TransformerSupplier](#)
 - [ValueTransformerSupplier](#)
 - [ValueTransformerWithKeySupplier](#)
 - [ProcessorSupplier](#)
- [Compatibility, Deprecation, and Migration Plan](#)
- [Alternatives](#)
 - [Have the added method on the Supplier interfaces only return store names, not builders](#)
 - [Do nothing](#)

Status

Current state: *Accepted*

Discussion thread: <https://lists.apache.org/thread.html/600996d83d485f2b8daf45037de64a60cebd9b234bf3449b6b753@%3Cdev.kafka.apache.org%3E>

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 writing low-level `Processors` and `Transformers` that are stateful using kafka streams, often the processors (or transformers, I'll use "processors" to refer to both for brevity) want to "own" one or more state stores, the details of which are not important to the business logic of the application. However, when incorporating these into a topology defined by the high level DSL, using `KStream.process`, you're forced to specify the state store names so the topology is wired up correctly. This creates a clumsy pattern where the "owned" state store's name must be passed alongside the `TransformerSupplier`, when the supplier itself could just as easily supply that information on their own.

An example of the clumsiness:

```
String stateStoreName = "my-store";
StoreBuilder<KeyValueStore> storeBuilder =
    Stores.keyValueStoreBuilder(Stores.inMemoryKeyValueStore(stateStoreName), keySerde, valSerde);
topology.addStateStore(storeBuilder);
ProcessorSupplier processorSupplier = new MyStatefulProcessorSupplier(stateStoreName, val -> businessLogic(val));
builder.stream("input.topic")
    .map(...)
    .filter(...)
    .process(processorSupplier, stateStoreName);
```

Both the main topology definition (the chained, high-level DSL calls on `StreamBuilder`, `KStream`, and `KTable`) and the internal implementation of `MyStatefulProcessorSupplier` need to know the state store name, when it should really only by `MyStatefulProcessorSupplier` that cares. Additionally, `topology.addStateStore(storeBuilder)` and the creation of the `StoreBuilder` are required, all of which ought to be implicit when using `MyStatefulProcessorSupplier`. Ultimately, because `KStream.process` requires store names as a separate argument, all of this "wiring" code is necessary alongside or nearby actual business logic.

Ideally, it would be reducible to something like:

```
builder.stream("input.topic")
    .map(...)
    .filter(...)
    .process(MyStatefulProcessorSupplier.make(val -> businessLogic(val)));
```

This allows for the same "reads top to bottom" type of clarity as when using `Processors` (and `Transformers`) as when using the high-level DSL.

Public Interfaces

Add an interface `ConnectedStoreProvider` that allows the implementor to specify state stores that should be connected to this processor/transformer (defaulting to no stores).

ConnectedStoreProvider

```
public interface ConnectedStoreProvider {
    default Set<StoreBuilder> stores() {
        return null;
    }
}
```

Change all `Processor/TransformerSupplier` interfaces to extend from it:

TransformerSupplier

```
public interface TransformerSupplier<K, V, R> extends ConnectedStoreProvider {
    ...
}
```

ValueTransformerSupplier

```
public interface ValueTransformerSupplier<V, VR> extends ConnectedStoreProvider {
    ...
}
```

ValueTransformerWithKeySupplier

```
public interface ValueTransformerWithKeySupplier<K, V, VR> extends ConnectedStoreProvider {
    ...
}
```

ProcessorSupplier

```
public interface ProcessorSupplier<K, V> extends ConnectedStoreProvider {
    ...
}
```

Proposed Changes

The proposal is to enhance the `ProcessorSupplier` and `TransformerSupplier` interfaces by allowing them to provide information about what state stores they "own" when constructing a topology using `StreamsBuilder`, `KStream::process`, `KStream::transform`, `KStream::transformValues`, and `Topology::addProcessor`.

The public interface changes above directly imply what needs to be changed in `KStream`: The `process` etc methods would get state store names from the list of `StoreBuilders` that the supplier (which implements `ConnectedStoreProvider`) provides, rather than the var args `stateStoreNames`.

The `process` method would add the `StoreBuilders` to the topology using `builder.addStateStore()` and connect the store to that processor, rather than requiring the user to do it themselves. In order to solve the problem of `addStateStore` potentially being called twice for the same store (because more than one `Supplier` specifies it), the check for duplicate stores in `addStateStores` will be relaxed to allow for duplicates if the same `StoreBuilder` instance for the same store name (compared by referenced, not `equals()`).

Compatibility, Deprecation, and Migration Plan

Because the added interface methods are default with a reasonable default, those additions are backwards compatible.

A user may continue to "connect" stores to a processor by passing `stateStoreNames` when calling `stream.process/transform(...)`. This may be used in combination with a `Supplier` that provides its own state stores by implementing `ConnectedStoreProvider::stores()`.

If a `StoreBuilder` that was manually added is also returned by a `ConnectedStoreProvider`, there is no issue since adding a state store will now be idempotent.

No migration tools are required since it's a relatively minor library change.

Alternatives

Have the added method on the Supplier interfaces only return store names, not builders

This solves the original issue only partially, but with perhaps less "API risk." The `String... stateStoreNames` argument would no longer be needed on the `KStream` methods, but the user would still need to manually add the `StoreBuilders` to the `Topology`. The downside is we don't achieve the full reduction of "wiring up" code required when building the topology (the user still needs to know to call `topology.addStateStore()`), but the upside is that the `StoreBuilder` is less coupled to the `*Supplier`. I don't consider this upside significant, but perhaps there are other use cases I'm not considering.

Do nothing

This is a "quality of life" API improvement, and nothing more, so maybe it's unneeded churn. I favor doing something (obviously) because I think that while small, this change can be a major usability improvement for the low-level API.