# DSL Optimizer



> **Warning**
>
> We try to keep this doc up to date, however, as it describes internals that might change at any point in time, there is no guarantee that this doc reflects the latest state of the code base.

Kafka Streams includes an optimizer that rewrite a DSL specified program before it's translated into a `Topology` . We describe the internals of the optimizer in this wiki page.

## Logical Representation of a DSL Program

When a user specifies a Kafka Streams program via the `StreamsBuilder`, all operators are collected into an internal logical representation of the specified data flow program.

Each operator is represented as a `StreamsGraphNode` (or a subclass) that contain all required meta data of the operator. For example it tracks it's name, parents and children, as well as information if it is a key and/or value changing operation. Because a DSL program may have multiple sources, a "dummy root" node is used as top level parent node to connect the whole program into a single graph structure.

To represent individual DSL operator, there are multiple subclasses of `StreamGraphNode`, for example (this is not an exhaustive list), `StreamSourceNod` (tracking topic name or pattern and `Consumed` configuration), `StreamSinkNode`, specific join nodes (stream-stream, stream-table, table-table, etc), and generic stateful or stateless processor nodes. Additionally, if the DSL triggers auto-repartitioning, special repartitioning nodes are inserted.

To keep track of dependencies between `StreamGraphNodes` and `buildPriority` parameter is maintains.

Lastly, each `StreamGraphNode` implements a method `writeToTopology()` that adds the required `Processors` and `StateStores` to the `Topology`. This method is called in the final translation step, *after* the optimizer rewrote the graph of operators.

## Optimization Rules

Given the logical representation of the DSL specified data flow program, the optimizer applies certain rules to rewrite this logical representation before it is translated into a physical `Topology` for execution. In the following we describe the applied optimization rules.

### Remove Source KTable Changelogs

TODO

### Serde push down

TODO

### Merging Repartition Topics

TODO

1. single upstream KStream with key-changing operation plus downstream fan-out with multiple operator triggering repartitioning
2. merging repartition topics for `merger()` operators