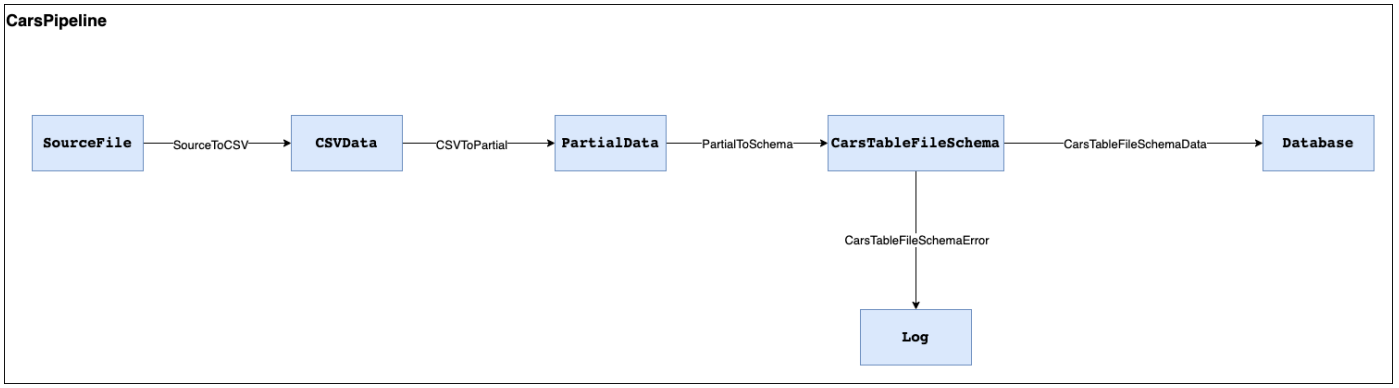
PML Draft

Pipeline Overview (Example)



Verbose syntax

- Concept
  - as “pipes and filters” here named “blocks” and “flow” (from data”flow” through a more generic word than “filter”)
  - PML only has to be able to express M1 level of Kernel (M2/M3 is provided as language features, M0 is the execution/runtime level)
- Graph
  - DAG makes sense
    - cycles might make sense in edge cases but bring large costs in a more complicated graph + special blocks
    - easy to reason over DAGs, can support users with semantic hints
- Scoping
  - file level: Does not matter, files are merged before compilation
    - every file can contain one or more namespaces
  - namespaces have to be top level element
  - namespaces can export their content explicitly
  - importing a namespace makes it’s exported content available inside the importing namespace
    - namespace A contains X, Y, Z exports Y, Z
    - namespace B imports A
      - inside B the following is available: A.Y, A.Z
  - Namespaces can contain ValueTypes, Pipelines, and Blocks
    - Pipelines can contain Blocks and Flows
    - Visibility: only within namespace unless exported (from namespace) and imported elsewhere

```
// Single line comments
// Statements are delimited by a curly brace or a semicolon


// namespaces are top-level scopes
// scopes are explicit by curly braces {}
// a file might contain multiple namespaces
```

```
namespace CarsLibrary {

      valuetype CarName extends String {
          maxLength: 5;
          regex: “$\w+$”;
      }
      export CarName; // export from namespace
}

namespace CarsNamespace {

      // a “.” is the dereferencing character so this means
      // “the element CarName in CarsLibrary”
      import CarsLibrary.CarName; // imports are per namespace

      // pipelines define the data flow graph, aka blocks and flows
      pipeline CarsPipeline {
            // Keyword “requires” indicates a required parameter used
            // for execution
            requires DatabaseLocation;
            requires LogLocation;

            // “SourceBlock” is a language feature provided by us
            // extending SourceBlock means this block needs to/can
            // provide attributes for inputs/outputs/url that our runtime
            // will use
            block SourceFile extends SourceBlock {
                inputs: []; // SourceBlocks are the only ones without an input
                outputs: [Data]; // Arrays represented as square brackets
                url: "http://";
            }

            block Database extends DatabaseSink {
                inputs: [Data];
                outputs: [];
                type: postgres;
                location: DatabaseLocation;
```

```
}

block Log extends LogSink {
    inputs: [Data];
    outputs: [];
    type: file;
    location: LogLocation;
}

block CSVData extends TableBlock {
    inputs: [Data];
    outputs: [Data, Header];
    hasHeader: true;
    rowSeparator: "\n";
    colSeparator: ",";
}

block PartialData extends SelectionBlock {
    inputs: [Data];
    outputs: [Data, Other];
    xStart: 0;
    yStart: 0;
    width: 5;
    height: 3;
}

block CarsTableFileSchema extends GuardBlock {
    inputs: [Data];
    outputs: [Data, Error];
    schema: {
        1: {
           valuetype: Integer;
           name: “Index”;
         };
        2: {
           valuetype: String;
           name: “ModelName”;
```

```
        };
        3: {
            valuetype: CarsLibrary.CarName;
            name: “CarName”;
        };
        4: {
            valuetype: Integer;
            name: “NumberOfWheels”;
        }
        5: {
            valuetype: Integer;
            name: "NumberOfDoors"
        };
    }
}

flow SourceToCSV {
      from: SourceFile.Data;
      to: CSVData.Data;
}

flow CSVToPartial {
      from: CSVData.Data;
      to: PartialData.Data;
}

flow PartialToSchema {
      from: PartialData.Data;
      to: CarsTableFileSchema.Data;
}

flow CarsTableFileSchemaData {
      from: CarsTableFileSchema.Data;
      to: Database.Data;
}

flow CarsTableFileSchemaErrors {
```

```
                from: CarsTableFileSchema.Error;
                to: Log.Data;
            }
        }
}
```