



INTRODUCTION TO MODELICA

Data management

Modelon

OVERVIEW

- Organizing Data
- Data records
- Modifiers in Models
- Deeper Hierarchies
- Data based components



DATA MANAGEMENT

ORGANIZING DATA

ORGANIZING DATA

- Data can be:
 - Organized in data packages using records
 - Constant data, never to be changed
 - Parameter data, allows for changes
 - Added to models in variants
 - Have a base model (preferably partial)
 - Create variants that extends the base model
 - Configure and set parameters in variants
 - Taken from databases outside Modelica models (or even Modelon Impact)
 - Requires customization
 - Modelon.DataAccess



DATA MANAGEMENT

DATA RECORDS

DATA RECORDS

```
record MyDataRecord
  extends Modelica.Icons.Record;
  parameter .Modelica.SIunits.Length L = 1 "Length";
  parameter .Modelica.SIunits.Diameter d = 0.1 "Diameter";
  parameter .Modelica.SIunits.Density rho = 1200 "Density";
end MyDataRecord;
```

- Create new datasets by modifying the first set:

```
record DataSet1
  extends DataPackage.MyDataRecord(L=10);
end DataSet1;
```

- In this case we modify the values of L , d and ρ is kept.

DATA RECORDS

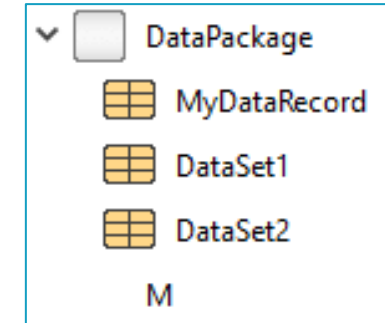
- Create new data record libraries that can be used by models.

```
package DataPackage
  extends Modelica.Icons.Package;
  record MyDataRecord
    extends Modelica.Icons.Record;
    parameter .Modelica.SIunits.Length L = 1 "Length";
    parameter .Modelica.SIunits.Diameter d = 0.1 "Diameter";
    parameter .Modelica.SIunits.Density rho = 1200 "Density";
  end MyDataRecord;

  record DataSet1
    extends DataPackage.MyDataRecord(L=10);
  end DataSet1;

  record DataSet2
    extends DataPackage.MyDataRecord(L=20);
  end DataSet2;

  model M
    parameter DataPackage.DataSet1 data;
    parameter Modelica.SIunits.Mass m=data.L*data.d*data.rho "Mass";
    annotation(...);
  end M;
end DataPackage;
```



REPLACEABLE PARAMETER SETS

- Engine model

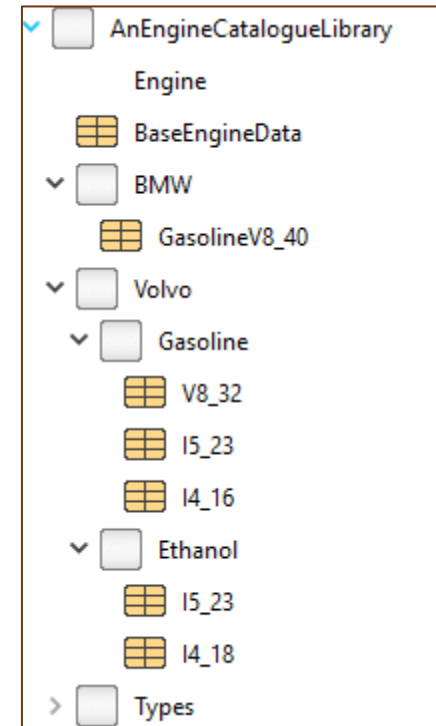
```
model Engine
  parameter BaseEngineData data;
  ...
end Engine;
```

- Base record

```
record BaseEngineData
  Types.CylinderArrangement cylinder_arrangement;
  Integer number_of_cylinders;
  Modelica.SIunits.Volume displacement;
end BaseEngineData;
```

- Specific configuration extending from engine record

```
record V8_32
  extends BaseEngineData(
    cylinder_arrangement = Types.CylinderArrangement.V,
    number_of_cylinders = 8, displacement = 3.2);
end V8_32;
```



DATA RECORDS

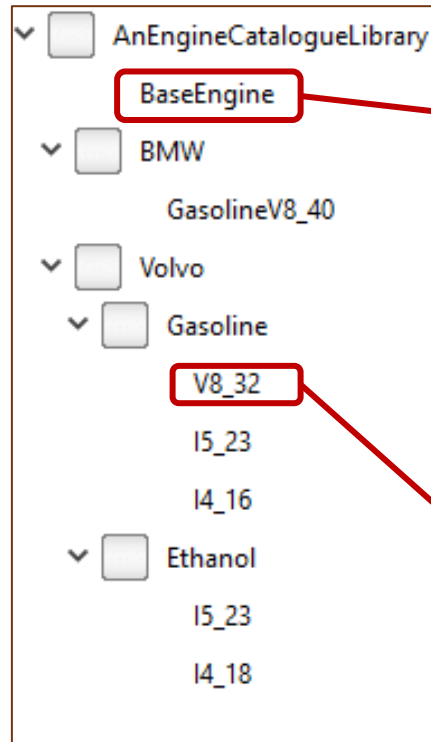
- Pros:
 - Keep data separate from models
 - Easy model parameterization with drop-down list
 - Easy to see what data is used in a particular model
- Cons:
 - Must manually make sure that data records and models match if you modify components in a hierarchical model
 - SimpleShaft and ElasticShaft might not need same parameters
 - Have to maintain two different library structures so they match
 - Models
 - Data records



DATA MANAGEMENT IN MODELS

MODIFIERS IN MODELS

VARIANT LIBRARIES



Example of a catalogue library with variants:

- **Base engine model**

```
partial model BaseEngine
  parameter Types.CylinderArrangement
    cylinder_arrangement;
  parameter Integer number_of_cylinders;
  parameter Modelica.SIunits.Volume displacement;
  ...
end BaseEngine;
```

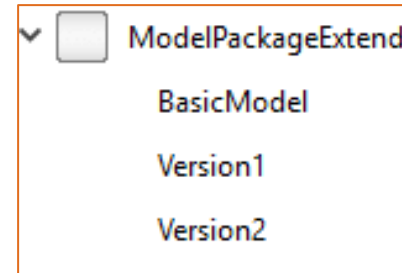
- **Specific configuration extending from base engine model**

```
model V8_32
  extends BaseEngine(
    cylinder_arrangement = Types.CylinderArrangement.V,
    number_of_cylinders = 8,
    displacement = 3.2);
end V8_32;
```

MODIFIERS IN MODELS

- Create a base class of your model
- Extend this base class and modify it

```
package ModelicaPackageExtends
  extends Modelica.Icons.Package;
  partial model BaseModel
    parameter .Modelica.SIunits.Length L = 1 "Length";
    parameter .Modelica.SIunits.Diameter d = 0.1 "Diameter";
    parameter .Modelica.SIunits.Density rho = 1200 "Density";
    annotation(...);
  end BaseModel;
  model Variant1
    extends .ModelicaClasses.ModelicaPackageExtends.BaseModel(L=10);
  end Variant1;
  model Variant2
    extends .ModelicaClasses.ModelicaPackageExtends.BaseModel(L = 20);
  end Variant2;
```



MODIFIERS IN MODELS

When using modifiers in this way, you store the model data inside the models.

Create parts catalog libraries

Pros:

- Quick and easy to create modifications from a nominal model.
- Data and model is always consistent.

Cons:

- Can be hard to get an overview of the data used in the model if it's a large data set.
- Modifiers in a deep hierarchy can be confusing



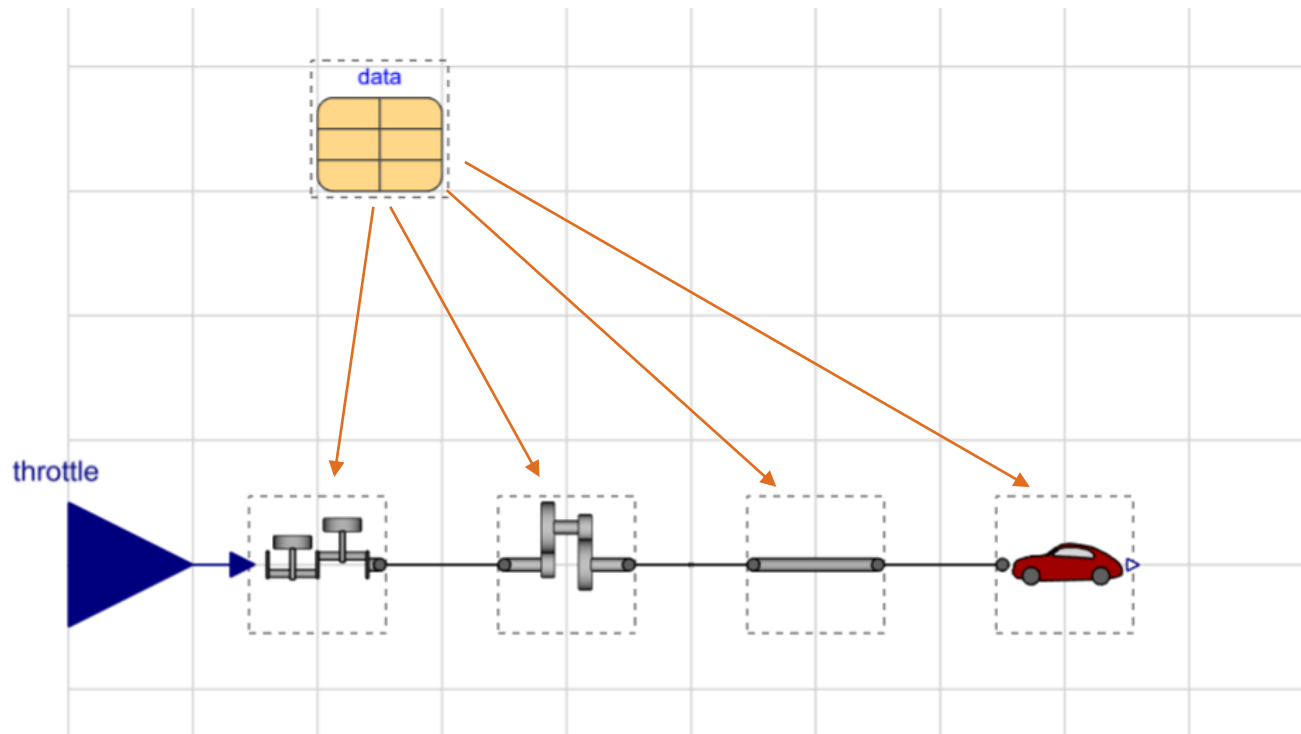
DATA MANAGEMENT

WORKING WITH DEEPER HIERARCHIES

HIERARCHICAL DATA MANAGEMENT

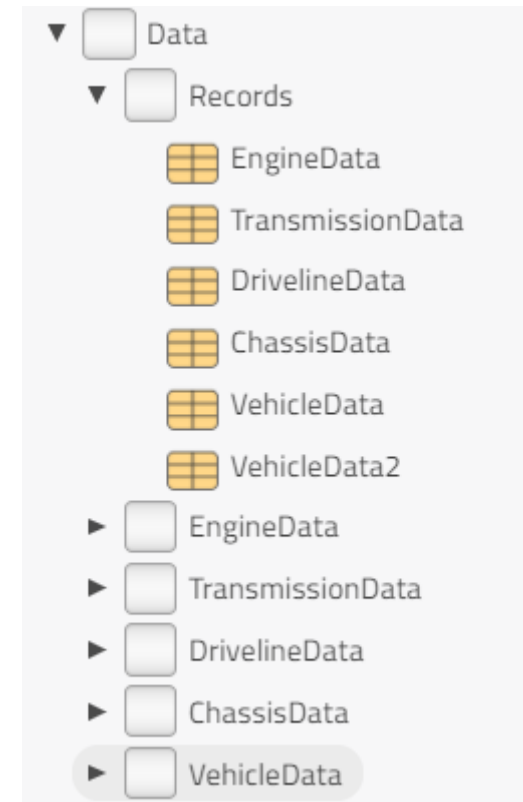
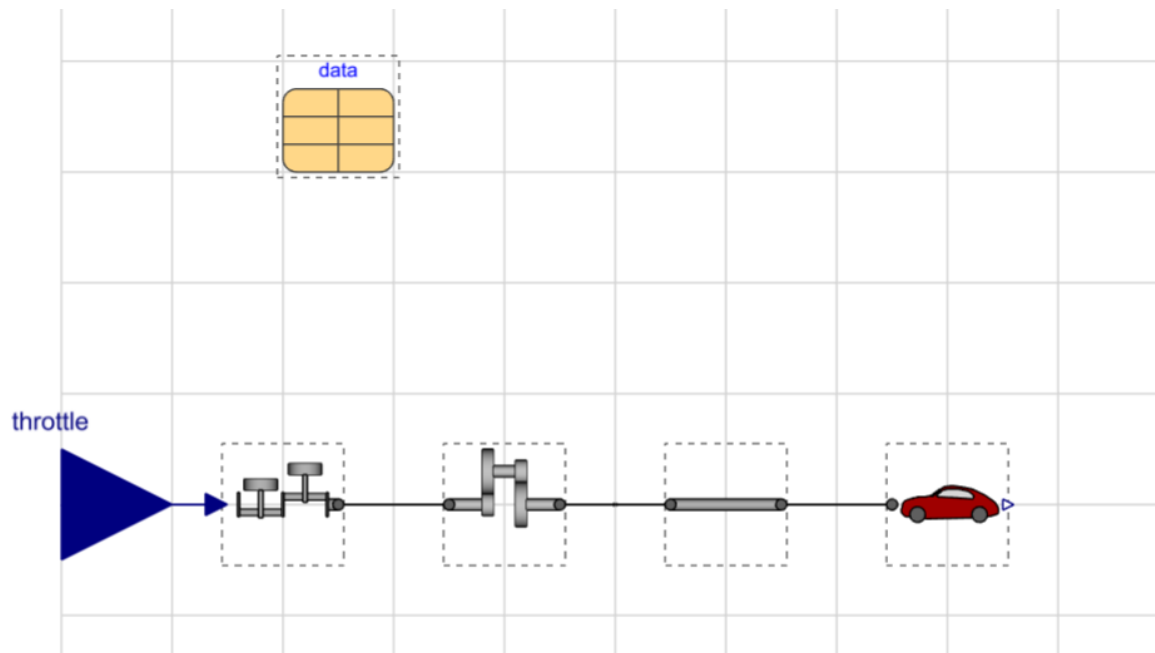
Using Data Records:

- Create a top-level Data record



HIERARCHICAL DATA MANAGEMENT

- The record can contain one replaceable record for each submodel



HIERARCHICAL DATA MANAGEMENT

- Data referencing can be done either element wise:

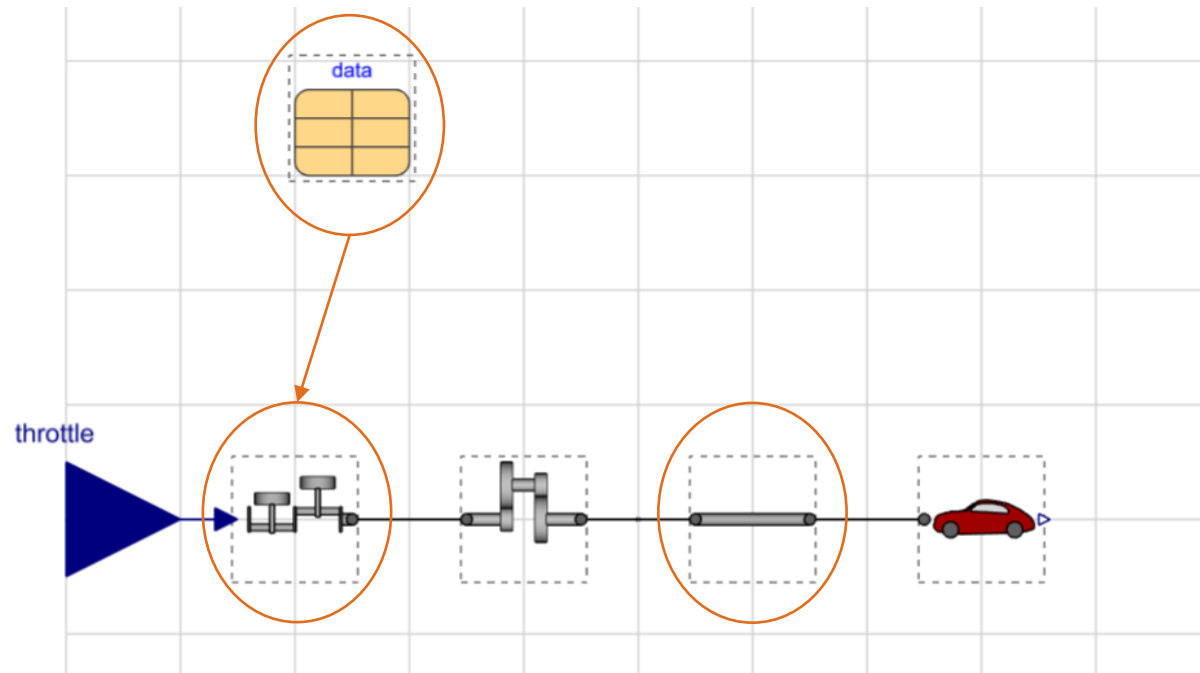
The image shows a simulation software interface. On the left, a grid-based workspace contains a 'data' component (a yellow grid) and a 'chassis' component (a red car icon). An orange arrow points from the 'data' component to the 'chassis' component. On the right, a 'COMPONENTS' panel lists 'SimpleCar', 'StandardCar', 'engine', 'transmission', 'driveline', 'chassis', 'data', and 'throttle'. The 'chassis' component is selected, and its properties are displayed in a 'PROPERTY' panel. The 'PROPERTY' panel has tabs for 'General' and 'Variables'. The 'Variables' tab is active, showing a list of variables with their values and units:

Variable	Value	Unit
J_wheel	data.chassis.J	kg·m ²
R_wheel	data.chassis.R_w	m
m_chassis	data.chassis.m_c	kg
v_start	25	m/s

- Or the chassis component could have a record of its own and the whole sub-record `data.chassis` is referenced.

SUMMARY – HIERARCHICAL DATA MANAGEMENT

- Change component model
- Change data sheet



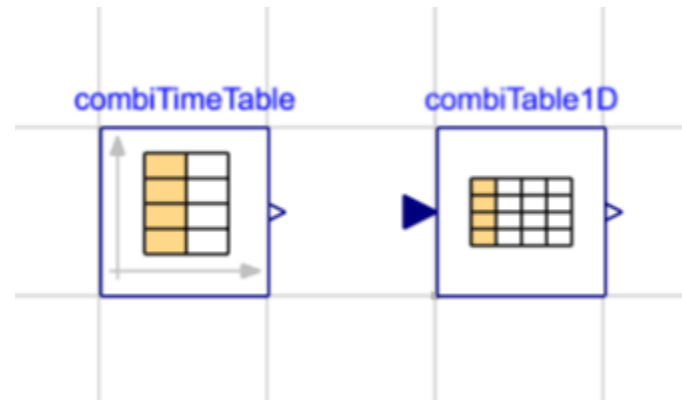


DATA MANAGEMENT IN MODELS

DATA BASED COMPONENTS

DATA INPUT BLOCKS

- MSL contains table-based input blocks.



- These can either depend on time directly or an arbitrary input signal

DATA INPUT BLOCKS

- Tables can be defined as:
 - Matrix directly in Modelica
 - A matrix stored in a file

	Table data definition
tableOnFile	<input type="checkbox"/> »
table	<input type="text" value="fill(0.0, 0, 2)"/>
tableName	<input type="text" value="NoName"/>
fileName	<input type="text" value="NoName"/>
verboseRead	<input checked="" type="checkbox"/> »

- Files can be .mat or .txt format

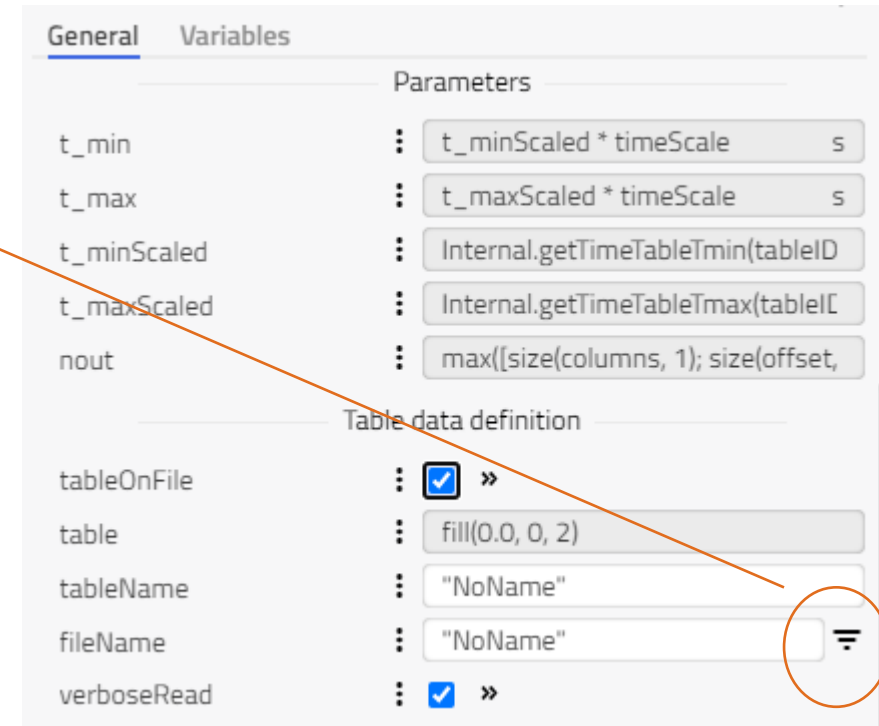
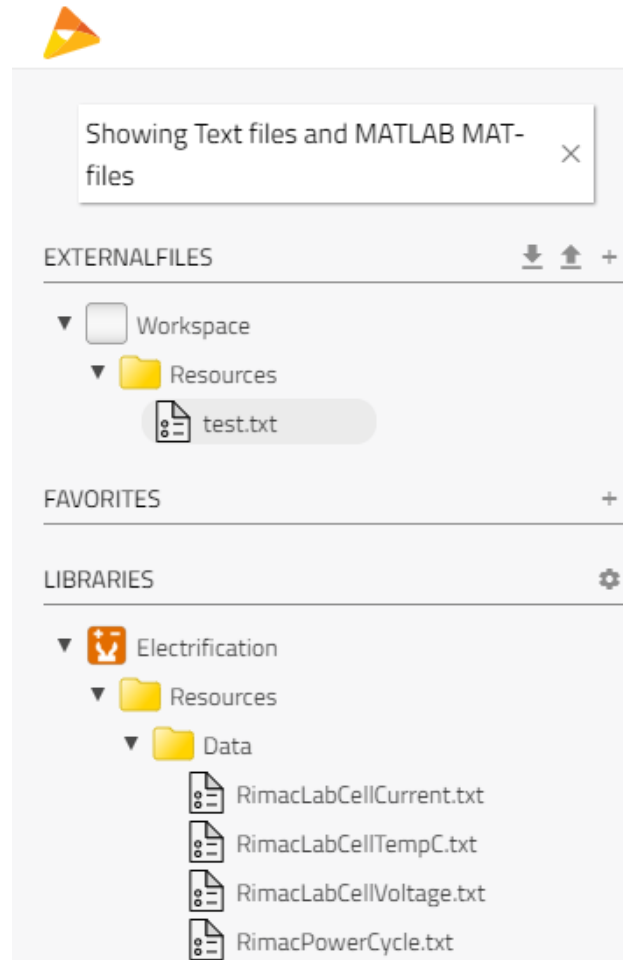
DATA INPUT BLOCKS

- The data is interpolated using piecewise polynomials
 - Constant
 - Linear
 - Continuous first derivative (splines)

The image shows a software interface with two panels. The left panel displays three input fields: 'columns' with the value '2:size(table, 2)', 'smoothness' with the value 'Table points are linearly interpolated', and 'extrapolation' with the value 'Extrapolate by using the derivative at the first point'. The right panel shows a dropdown menu titled 'Table data interpretation' with several options: 'LinearSegments', 'ContinuousDerivative', 'ConstantSegments', 'MonotoneContinuousDerivative1', and 'MonotoneContinuousDerivative2'. The 'LinearSegments' option is currently selected.

DATA INPUT BLOCKS

- Input files:



WORKSHOP 4.2

In this workshop you will:

- Create Data record structure using templates and interfaces
- Add structure to existing model
- Propagate the record data into the model