Parameter Arrays in Dymola

Managing external parameter sets

3DEXPERIENCE®

Version 1.2 - 10/3/2024

Written by: Dag Brück (QBK)
Validated by: Hans Olsson (HOS)

DASSAULT SYSTEMES | The 3DEXPERIENCE® Company

Table of contents

1. Introduction

Modelica has a powerful mechanism for parameterization within a model, for example propagating parameters down to components. Parameters can be combined into records, which makes it possible to push an entire parameter set into the model (where each component picks out the needed parts).

Traditionally this has been a static, compile-time view of parameters – especially with arrays of data used for table lookup. This has forced Modelica users to use C-code libraries that store data in external objects instead of native Modelica arrays, which defeats the notion of parameter propagation. For earlier work, see [1] and [2]. However, it should be noted that arrays or arbitrary size have always been supported as function arguments.

Starting in Dymola 2025x, there is support for dynamic parameter arrays that can be initialized at runtime. This means that a vector or matrix of arbitrary size can be read at simulation initialization into a Modelica parameter.

2. Parameter sets in Dymola

The new feature is presented with several examples of increasing complexity.

2.1 Simple parameter record

Recommended practice is to collect all parameter sets in parameter records which are initialized together and then propagating the parameters into components. This yields a model with better structure compared to component models that read parameter data directly.

A simple parameter record would be

```
record TestParameters "Global parameter record"
  parameter Modelica.Units.SI.Frequency f=0.2
      "Frequency of sine function to invoke clutch1";
  parameter Modelica.Units.SI.Time T2=0.4
      "Time when clutch2 is invoked";
end TestParameters;
```

This parameter record can then be used in the model, for example by extends plus modifiers.

```
model Test "Test case applying parameters"
  replaceable TestParameters par;
  extends Modelica.Mechanics.Rotational.Examples.CoupledClutches(
    f=par.f,
    T2=par.T2);
end Test;
```

The aspect that is missing from this example is reading parameters from file, see below.

2.2 Declaring a variable-size array

A variable-size parameter vector or matrix uses the `:` symbol to indicate an unknown dimension.

```
parameter Real p[:] “Vector of unknown size”
    annotation(__Dymola_UnknownArray=true);
```

This is the same syntax as has been used for function arguments, but note the annotation `__Dymola_UnknownArray=true` that identifies this as an array which size is determined at runtime.

2.3 Parameter record with variable-size array

For this example, we start by declaring parameter record containing a variable-size vector.

```
record ParSet "Simple parameter set with array data"
  // Old fixed-sized array: Real J[4] = { 1.0, 1.0, 1.0, 1.0};
  parameter Real J[:]  "Component inertia"
            annotation(__Dymola_UnknownArray=true);
end ParSet;
```

For this case we will read the actual parameter values from a file. For portability the function is written using Modelica Standard Library (MSL) functions supporting text files, but obviously other data formats could be supported with appropriate libraries.

```
function InitializeParameters "Reads initial values for parameter set"
  input String filename "File to read parameters from";
  output ParSet p "Parameter set from file";

protected
  Integer n=Modelica.Utilities.Streams.countLines(filename);
  String data[:]=Modelica.Utilities.Streams.readFile(filename);

algorithm
  p.J := fill(0.0, n);  // Initialization needed to get the right size

  for i in 1:n loop
    p.J[i] := Modelica.Utilities.Strings.scanReal(data[i]);
  end for;
end InitializeParameters;
```

The main program has a similar structure to the example above, but note the need for the annotation `Evaluate=false` to make sure parameters can be modified when a simulation starts, instead of being evaluated and fixed at translation.

```
model Test "Use package parameter"
  parameter ParSet par=InitializeParameters("parameterfile.txt")
      annotation (Evaluate=false);

  extends Modelica.Mechanics.Rotational.Examples.CoupledClutches(
    J1(J=par.J[1]),
    J2(J=par.J[2]),
    J3(J=par.J[3]),
    J4(J=par.J[4]));
end Test;
```

2.4 Multi-level reading of parameter files

We can generalize the above example to use nested files; one master data file to designate an actual parameter file. To enable better diagnostics, we store the parameter filename.

```
record ParSet "Simple parameter set with array data"
  String datafile "Name of actual parameter file";
  Real J[:]  "Component inertia" annotation(__Dymola_UnknownArray=true);
end ParSet;
```

The reading of parameter data is then done in two stages. First a master data file is read to obtain the name of the actual parameter file. Note the added print-outs to make certain what happens during execution.

```
function InitializeParameters "Reads initial values for parameter set"
  input String filename "File to read parameters from";
  output ParSet p "Parameter set from file";

algorithm
  Modelica.Utilities.Streams.print("Reading master data from: " +
filename);
  p.datafile :=Modelica.Utilities.Streams.readLine(filename, 1);
  p.J := InitializeArray(p.datafile);
end InitializeParameters;

function InitializeArray "Reads initial values for vector parameter"
  input String filename "File to read parameters from";
  output Real v[:] "Parameter vector from file";

protected
  Integer n=Modelica.Utilities.Streams.countLines(filename);
  String data[:]=Modelica.Utilities.Streams.readFile(filename);

algorithm
  Modelica.Utilities.Streams.print("Reading array from: " + filename);
  v := fill(0.0, n);  // Initialization needed to get the right size

  for i in 1:n loop
    v[i] := Modelica.Utilities.Strings.scanReal(data[i]);
  end for;
end InitializeArray;
```

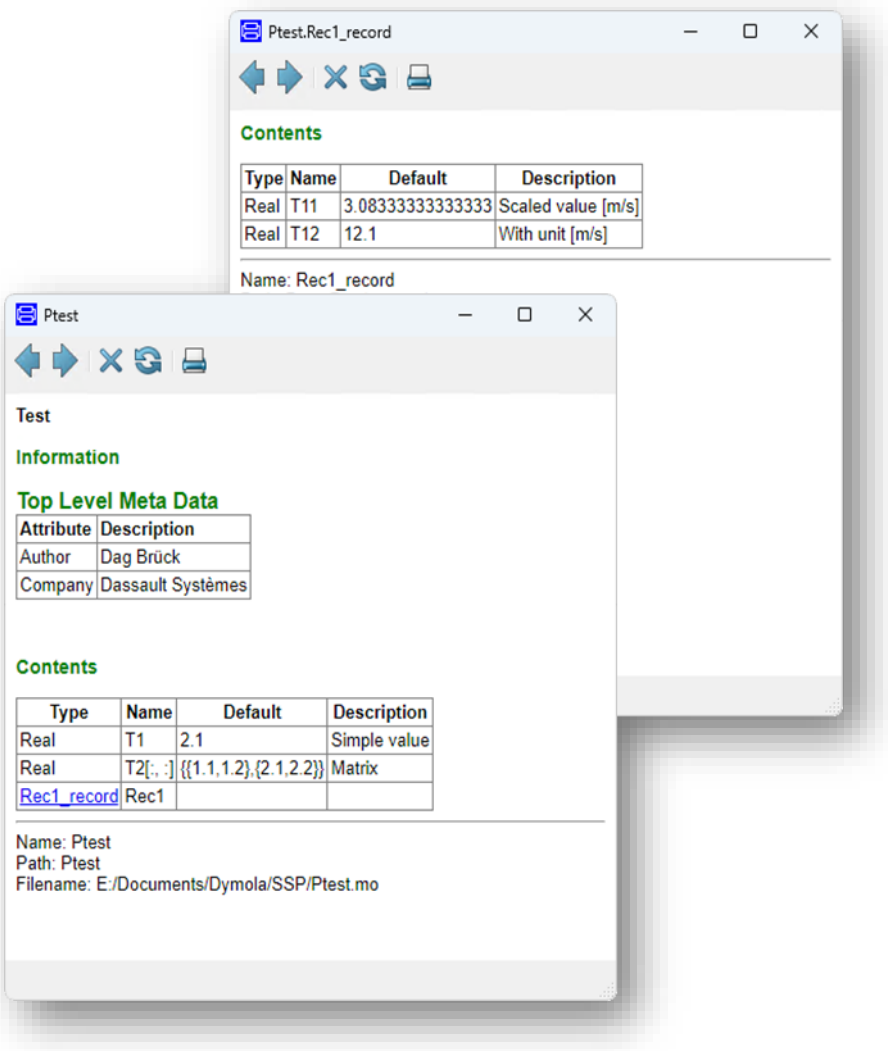
Finally, our main program is virtually unchanged.

```
model Test "Use package parameter"
  parameter String master="masterfile.txt" "Name of master data file"
     annotation (Evaluate=false);
  parameter ParSet par=InitializeParameters(master)
     annotation (Evaluate=false);

  extends Modelica.Mechanics.Rotational.Examples.CoupledClutches(
    J1(J=par.J[1]),
    J2(J=par.J[2]),
    J3(J=par.J[3]),
    J4(J=par.J[4]));
end Test;
```

2.5 Documentation of parameter records

Parameter record can also be viewed in the documentation layer of Dymola or using the Info-button in the package browser.



3. Making parameter records from data

Assuming that parameter records are the recommended best practice, an unfortunate drawback is that somebody has to create such a parameter record in the first place, before it can be filled with real data. To aid in this tedious effort, Dymola provides support both to create such a parameter record in the first place, and to apply modifications to an existing parameter record.

3.1 Creating a parameter record

The `importSSV` command in its simplest form reads a data file and creates a Modelica record with elements corresponding to each data item.

For example, the command

```
importSSV("Ptest.ssv");
```

Several properties in the data file are processed.

- The **name** of the parameter. Any dots in the name are interpreted as hierarchical delimiters, creating a nested parameter record and an instance of that record. Several parameters with the same initial path are collected in the same initial record even if they are not sequentially stored in the data file.
- A **description** of the parameter.
- Any **unit** attribute is used to define the unit of the parameter. If the data file uses derived units recognized by Dymola (km, mA, MPa), the base unit is derived and the given unit is used a display unit. Note that the value is scaled accordingly.
- The **value** attribute is used both to heuristically determine the dimensions of the parameter and to give it a default value. The Modelica syntax for vectors, matrices and arrays is recognized.

A small SSV data set is

```
<?xml version="1.0" encoding="UTF-8"?>
<ssv:ParameterSet version="1.0" name="Ptest" description="Test">
  <ssv:Parameters>
    <ssv:Parameter name="T1" description="Simple value" >
       <ssv:Real value="2.1" />
    </ssv:Parameter>
    <ssv:Parameter name="T2" description="Matrix" >
       <ssv:Real value=" {{1.1, 1.2}, {2.1, 2.2}}" />
    </ssv:Parameter>
    <ssv:Parameter name="Rec1.T11" description="Scaled value" >
       <ssv:Real value="11.1" unit="km/h"/>
    </ssv:Parameter>
    <ssv:Parameter name="Rec1.T12" description="With unit" >
       <ssv:Real value="12.1" unit="m/s"/>
    </ssv:Parameter>
  </ssv:Parameters>
</ssv:ParameterSet>
```

Reading this data file produces this parameter record.

```
record Ptest “Test”
  parameter Real T1=2.1 "Simple value";
  parameter Real T2[:,:]={{1.1,1.2},{2.1,2.2}} "Matrix";
  record Rec1_record
    parameter Real T11(
      unit="m/s",
      displayUnit="km/h") = 3.08333333333333 "Scaled value";
    parameter Real T12(unit="m/s") = 12.1 "With unit";
  end Rec1_record;
  parameter Rec1_record Rec1;
end Ptest;
```

3.2 Importing CSV data

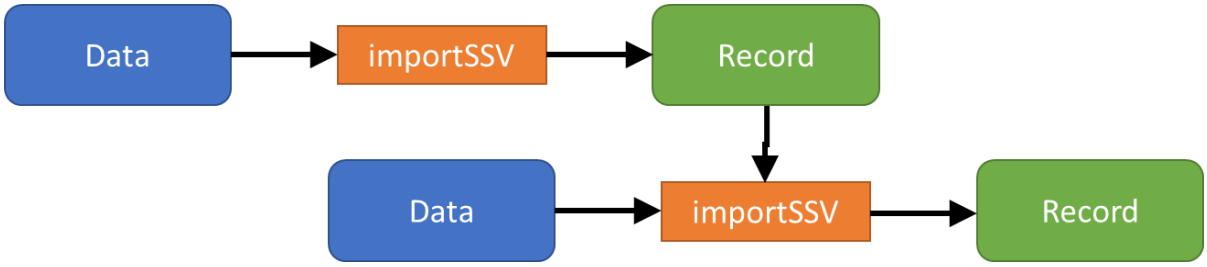
The function `importSSV` will, in spite of its name, also process simple CSV files without headings. The file shall contain data in four columns: name, value, unit and description. Columns may be empty, for example the unit property. The value is used to determine the type; recognized are: integer and real numbers, “false” and “true” yielding a Boolean parameter, and strings enclosed in double-quotes.

```
T1, 2.1, ,Simple value
T2, "Text par", ,
Rec1.T11, 11.1, km/h, Scaled value
Rec1.T12, 12.1, m/s, With unit
```

It should be noted that the array syntax is not (yet) supported in CSV files.

3.3 Modifying a parameter record

Given that we have a parameter record in Modelica, there is a need to derive new parameters sets. This is typically done by starting out with the complete parameter set and then modifying a subset of the parameters.



The `importSSV` function supports this, making applying changes with the common extend-and-modify idiom in Modelica. Assuming the parameter record above, we can read a second file (e.g. PtestM.ssv) which contains

```
<?xml version="1.0" encoding="UTF-8"?>
<ssv:ParameterSet version="1.0" name="PtestM" description="Modified set">
  <ssv:Parameters>
    <ssv:Parameter name="T1" description="Simple value" >
       <ssv:Real value="42" />
    </ssv:Parameter>
    <ssv:Parameter name="Rec1.T12" description="With unit" >
       <ssv:Real value="72.6" unit="m/s"/>
    </ssv:Parameter>
  </ssv:Parameters>
</ssv:ParameterSet>
```

with this command

```
importSSV("PtestM.ssv", "Ptest");
```

That will create a derived parameter record.

```
record PtestM "Modified set"
  extends .Ptest(
    T1=42 "Simple value",
    Rec1(T12(displayUnit="m/s") = 72.6 "With unit"));
end PtestM;
```

4. Variable-size arrays in FMUs

4.1 Reading from file

Models with variable-size arrays as described above can be exported as FMUs. The data files are read during FMU initialization. The filename can be specified as a string parameter of the FMU.

4.2 Setting from the simulation master

Assuming the name of our parameter in the model is **`par.J`**, Dymola will create two variables in the FMU.

Name	Type	Causality
`J.par.size`	Int32	Structural parameter
`J.par.data`	Float64	Parameter

`J.par.size` is the structural parameter which must be set in FMI configuration mode, before entering initialization mode. This step sets the size of `J.par.data` so it can be initialized.

Support for setting variable-size arrays from the simulation master at FMU initialization is by default enabled, but can be disabled by setting

```
Advanced.FMI3.ExposeDynamicArrays=false;
```

Current restrictions: only FMI 3.0; only vectors of Real, Integer and Boolean supported, not matrices.

5. Constraints

This new feature has been implemented with certain constraints, in order not to compromise code size and simulation speed.

- Only parameter arrays can have variable size at runtime.
- Such parameter arrays must be initialized at model initialization.
- Variable size arrays are not stored in the result file.
- Only arrays of simple types, not array of records.
- The size of variable arrays is by default limited to `70000` elements, but can be increased if needed by setting `Advanced.Translation.RealBufferSize=10*N`, where *N* is the number of parameters.
- Supported in Dymola 2025x and later.

6. References

[1] Beutlich, Thomas and Dietmar Winkler (2021): “Efficient Parameterization of Modelica Models”, Proceedings of the 14thInternational Modelica Conference, Linköping, Sweden. DOI 10.3384/ecp21181141.

[2] Tiller, Michael (2005): “Implementation of a Generic Data Retrieval API for Modelica”, Proceedings of the 4th International Modelica Conference, Hamburg, Germany.