

Dymola

Dynamic Modeling Laboratory

What is Dymola

Contents: Chapter 1 “What is Dymola”
extracted from the manual “Dymola User
Manual 1A: Introduction, Getting Started, and
Installation”.

March 2024 (Dymola 2024x Refresh 1)

The information in this document is subject to change without notice.

© Copyright 1992-2024 by Dassault Systèmes AB. All rights reserved.
Dymola® is a registered trademark of Dassault Systèmes AB.
Modelica® is a registered trademark of the Modelica Association.

Other product or brand names are trademarks or registered trademarks of their respective holders.

Dassault Systèmes AB
Ideon Gateway
Scheelevägen 27 – Floor 9
SE-223 63 Lund
Sweden

E-mail: <https://www.3ds.com/support>
URL: <https://www.dymola.com/>
Phone: +46 46 270 67 00

Contents

1	What is Dymola?	5
1.1	Features of Dymola	5
1.2	Simulating an existing model	6
1.3	Building a model	10
1.4	Architecture of Dymola	12
1.5	Features of Modelica	13
1.6	Functional Mockup Interface	15
1.7	FMI for embedded systems (eFMI®)	16
1.8	System Structure and Parameterization	17

1 What is Dymola?

1.1 Features of Dymola

**Modelica is the
modeling language;
Dymola is the tool.**

Dymola – Dynamic Modeling Laboratory – is suitable for modeling of various kinds of physical systems. It supports hierarchical model composition, libraries of truly reusable components, connectors and composite acausal connections. Model libraries are available in many engineering domains.

Employing the Modelica language, Dymola uses a modeling methodology based on object orientation and equations. The usual need for manual conversion of equations to a block diagram is removed by the use of automatic formula manipulation, leading to greater robustness and easier model building. Other highlights of Dymola are:

- Handles large, complex multi-engineering models. Uses symbolic methods to generate faster simulation code.
- Faster modeling by graphical model composition (drag-and-drop).
- Model libraries for several application domains, but also open for user defined model components.
- Open interface to other programs, including Python, Java and JavaScript.
- Code export using Functional Mockup Interface (FMI) or for real-time simulation.

Dymola supports the entire model development process from model creation to simulation and result analysis.

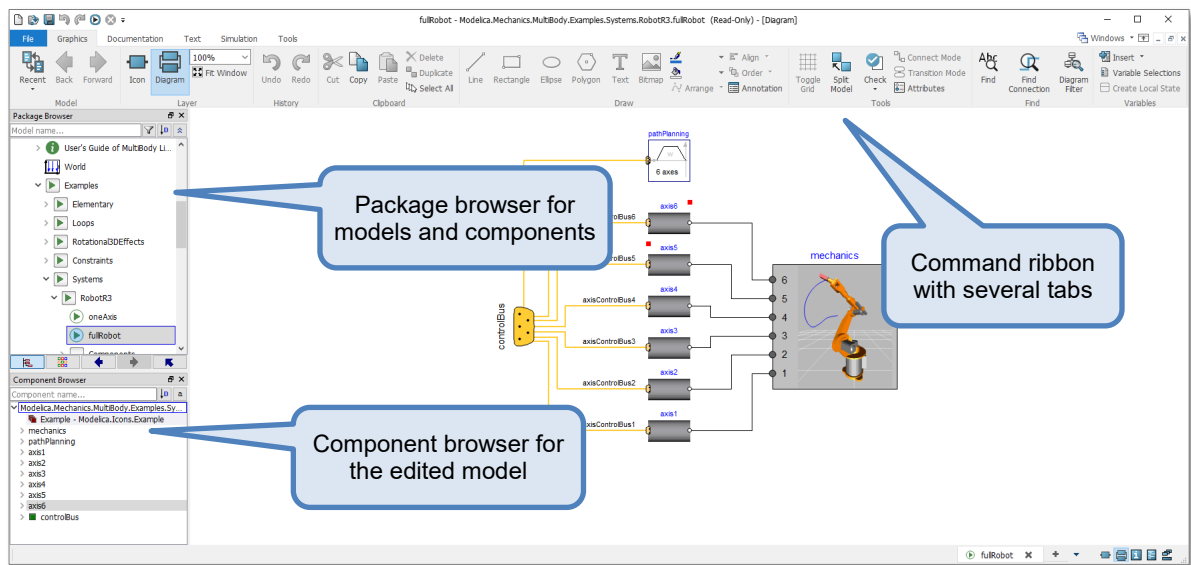
- Graphical and textual editors are used to create models and model components, typically by using component models from existing model libraries.
- The Simulation tab is used to make experiments on the model, plot results and animate the behavior. It also has a scripting window for automation of experimentation and performing calculations.

The sections below will give a short hands-on introduction to the basic features of Dymola.

1.2 Simulating an existing model

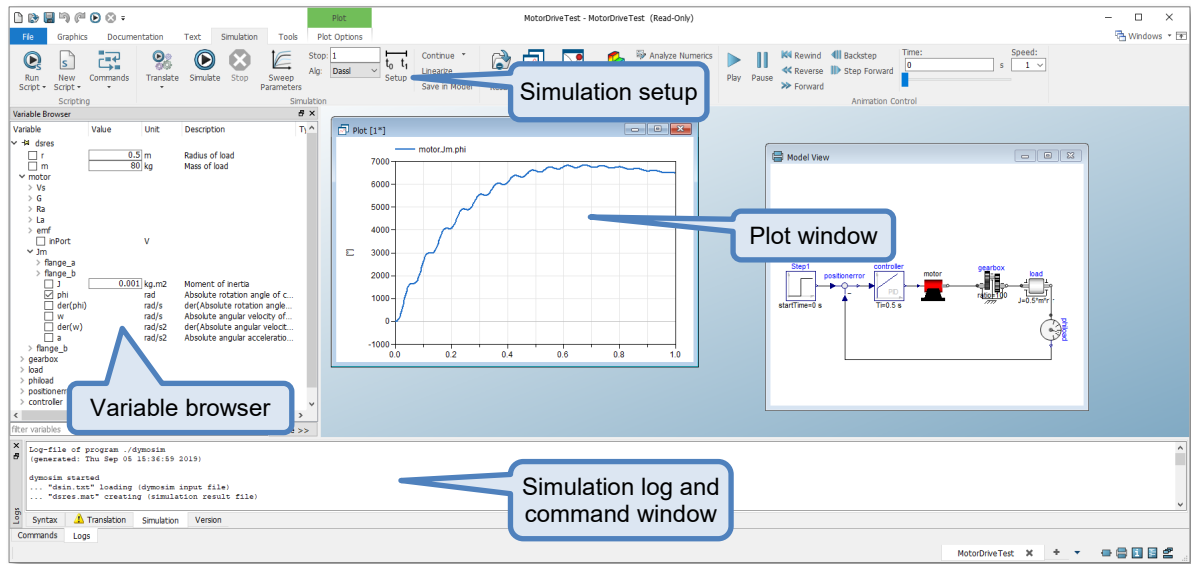
Finding the model

Dymola starts in the graphical editor. The model to simulate is found by using the **File > Open...** or **File > Demos** commands. Example models are also found in the Modelica Standard Library or in other libraries opened by the **File > Libraries** command.



Simulation

The Simulation tab is used for experimentation. It has a simulation setup to define duration of simulation, etc., plot windows, animation windows and a variable browser. The model view can also be used for navigating the hierarchy of the model.

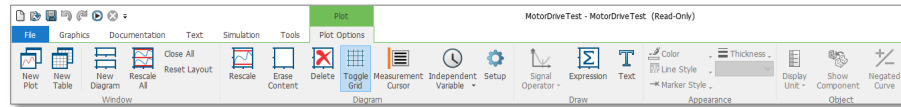


The command ribbon has controls to run scripts and pre-defined commands, as well as to setup and to run the simulation. The most common settings, the simulation stop time and the numerical integration algorithm are available in the ribbon; additional setting by clicking **Setup**.

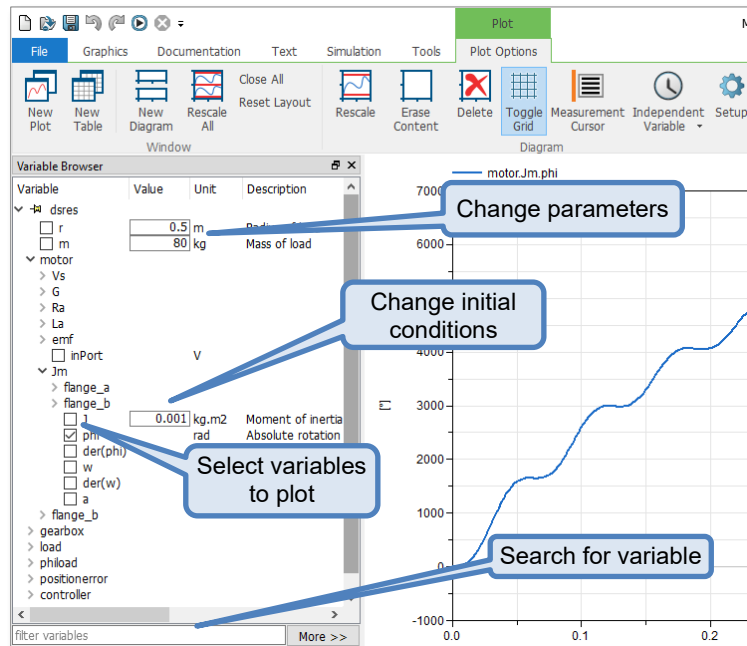


Experimentation and plotting

Additional controls are shown when the plot window is active. They are used to setup new plot windows, or the layout of what is plotted in the window.



The variable `browser` allows selecting plot variables, changing parameters and initial conditions.

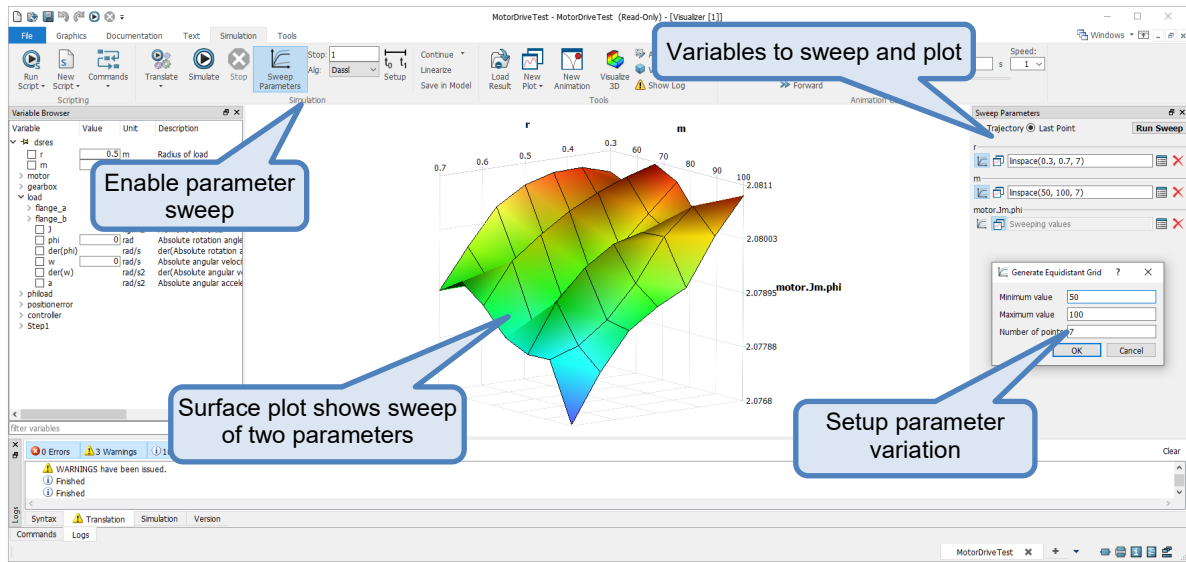


An animation window shows a 3D view of the simulated model. The animation can be run at different speeds, halted, single stepped and run backwards.

Sweeping parameters

Model experimentation involves running simulations for various combinations of parameters to determine the modeled system's properties. To increase performance, parameter sweeps are run on multiple CPU cores when possible.

A convenient user interface facilitates such experimentation. Sweeps over one or two parameters are visualized by plotting variables used as key performance indicators, or the surface formed by varying two parameters.

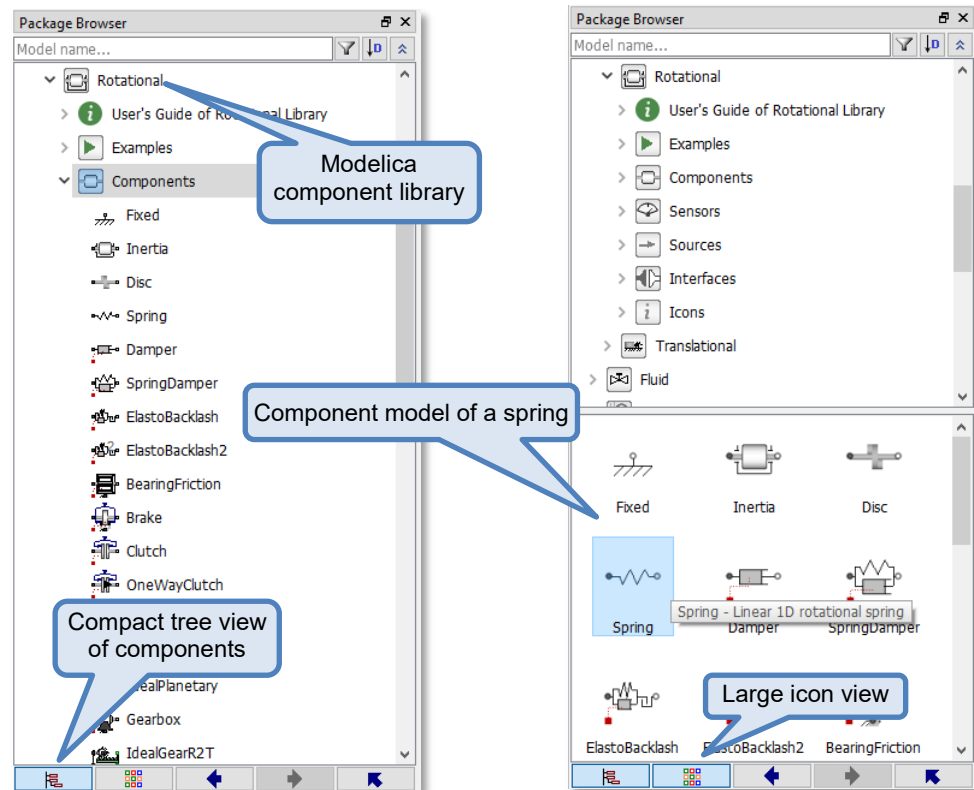


1.3 Building a model

The graphical model editor is used for creating and editing models in Dymola. Structural properties, such as components, connectors and connections are edited graphically, while equations and declarations are edited with a built-in text editor.

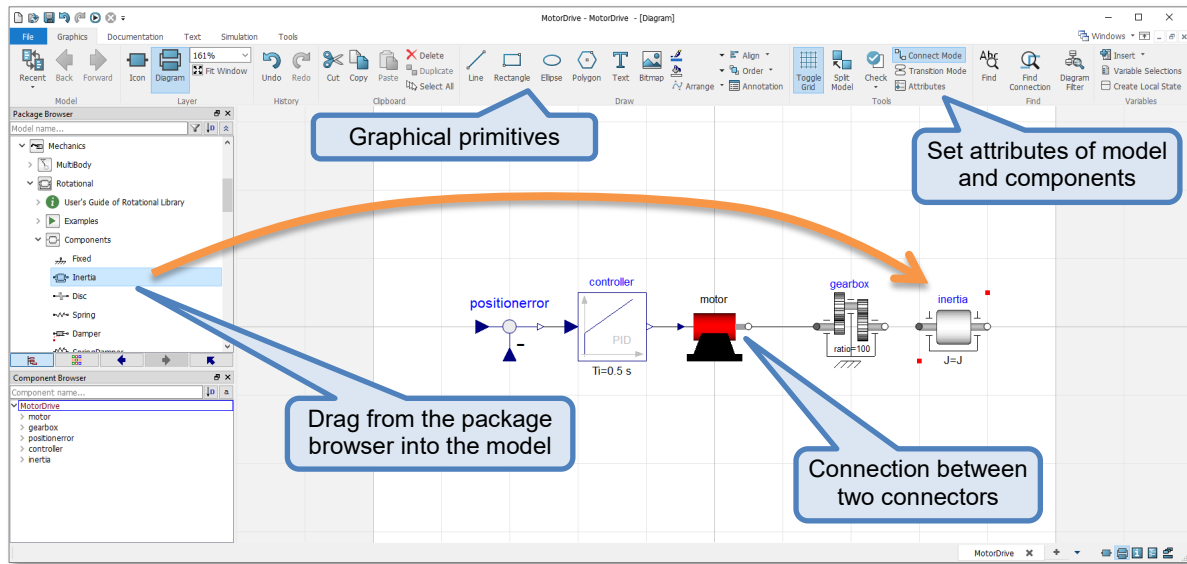
Finding a component model

The package browser allows viewing and selecting component models from a list of models with small icons. It is also possible to get a large icon view.



Composing a new model

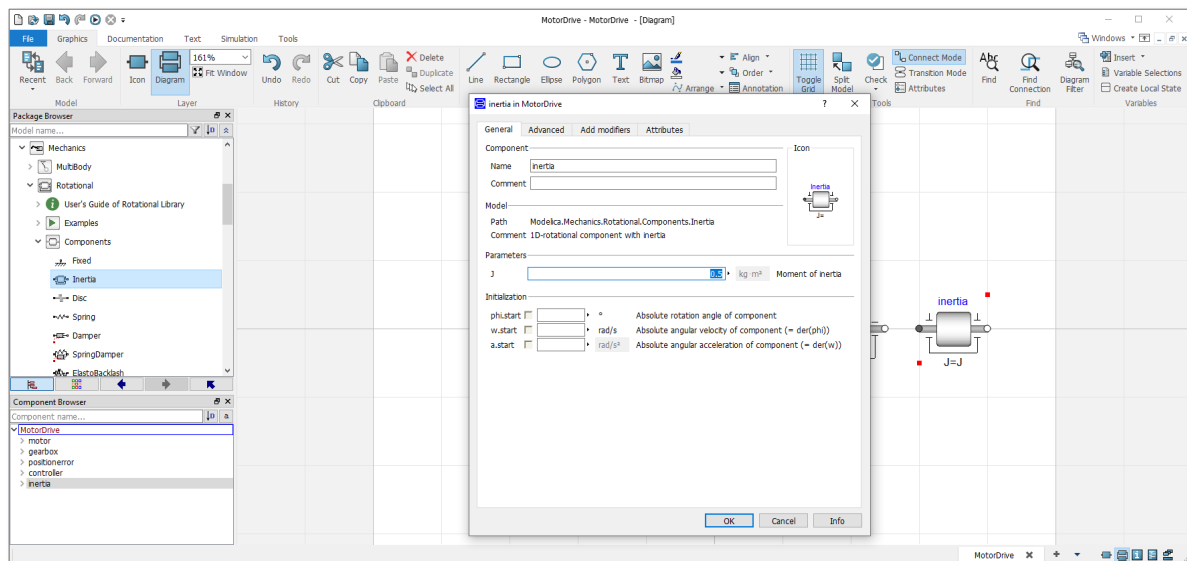
Components are dragged from the package browser to the diagram layer and connected. The component hierarchy is shown in the component browser.



A connection is made by clicking on one connector and drawing to the other connector.

Setting component parameters

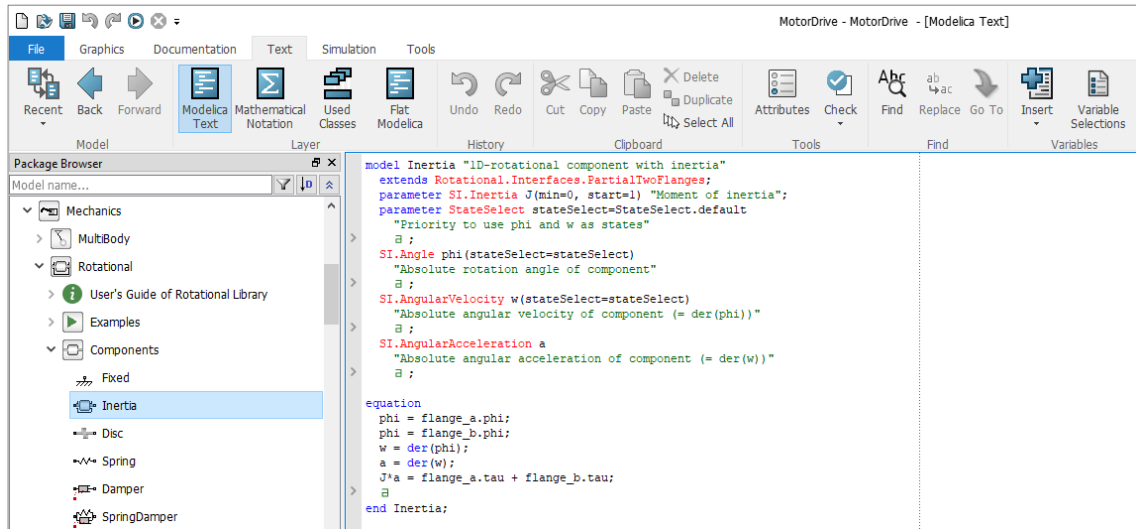
By double-clicking on a component, or right-clicking the component and selecting **Parameters**, a dialog for giving the name of the component and the values of its parameters is shown.



Editing Modelica text

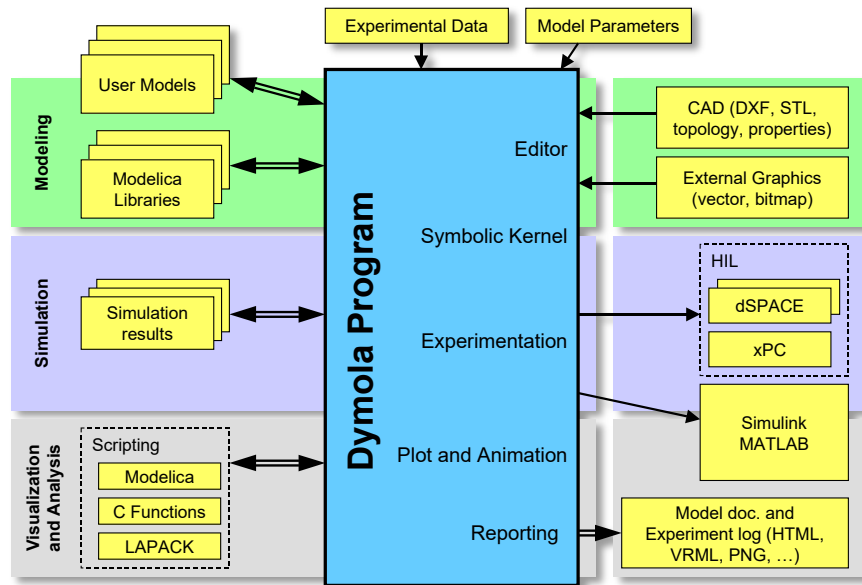
Clicking on the **Documentation** or **Text** tabs allows for inspection of the documentation or the model itself or the underlying Modelica code. This is also the editor for entering Modelica code, i.e. declarations and equations for low-level models.

In each tab there are different views, for example **Text** is used both to edit the Modelica text, and to view the equations in mathematical notation, etc.



1.4 Architecture of Dymola

The overall architecture of the Dymola program is shown below. Dymola has a powerful graphic editor for composing models, using Modelica models stored on files. Dymola can also import other data and graphics files. Dymola contains a translator for Modelica equations generating C code for simulation. The code can also be exported using the Functional Mockup Interface (FMI), or to Matlab/Simulink and hardware-in-the-loop platforms.



1.5 Features of Modelica



Modelica is an object-oriented language for modeling of large, complex and heterogeneous physical systems. It is suited for multi-domain modeling, for example for modeling of mechatronic systems within automotive, aerospace and robotics applications. Such systems are composed of mechanical, electrical and hydraulic subsystems, as well as control systems.

General equations are used for modeling of the physical phenomena. The language has been designed to allow tools to generate efficient code automatically. The modeling effort is thus reduced considerably since model components can be reused, and tedious and error-prone manual manipulations are not needed.

Background

Modeling and simulation are becoming more important since engineers need to analyze increasingly complex systems composed of components from different domains. Current tools are generally weak in treating multi-domain models because the general tools are block-oriented and thus demand a huge amount of manual rewriting to get the equations into explicit form. The domain-specific tools, such as circuit simulators or multibody programs, cannot handle components of other domains in a reasonable way.

There is traditionally too large a gap between the user's problem and the model description that the simulation program understands. Modeling should be much closer to the way an engineer builds a real system, first trying to find standard components like motors, pumps and valves from manufacturers' catalogues with appropriate specifications and interfaces.

Equations facilitate true model reuse.

Equations and reuse

Equations are used in Modelica for modeling of the physical phenomena. No particular variable needs to be solved for manually because Dymola has enough information to decide that automatically. This is an important property of Dymola to enable handling of large models having more than hundred thousand equations. Modelica supports several formalisms: ordinary differential equations (ODE), differential-algebraic equations (DAE), bond graphs, finite state automata, Petri nets etc.

The language has been designed to allow tools to generate very efficient code. Modelica models are used, for example, in Hardware-in-the-Loop (HIL) simulation of automatic gearboxes, which have variable structure models. Such models have so far usually been treated by hand, modeling each mode of operation separately. In Modelica, component models are used for shafts, clutches, brakes, gear wheels etc. and Dymola can find the different modes of operation automatically. The modeling effort is considerably reduced since model components can be reused and tedious and error-prone manual manipulations are not needed.

Interfaces and inheritance

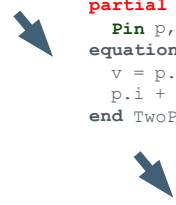
Modelica has been designed to provide very strong support for expressing common properties of models and interfaces that can be shared throughout an entire engineering domain. The Modelica Standard Library defines many such interfaces that are shared by commercial and third party libraries.

Inheritance is a composition mechanism inspired by object-oriented programming languages. It gives an efficient and safe construct to share common properties, as shown in this definition of an electrical capacitor.

```
connector Pin
  Voltage v;
  flow Current i; // Sums to zero
end Pin;

partial model TwoPin
  Pin p, n; Voltage v;
  equation
    v = p.v - n.v;
    p.i + n.i = 0;
  end TwoPin;

model Capacitor
  extends TwoPin;
  parameter Capacitance C;
  equation
    i = C*der(v);
  end Capacitor;
```



Modelica history

Reuse is a key issue for handling complexity. There had been several attempts to define object-oriented languages for physical modeling. However, the ability to reuse and exchange models relies on a standardized format. It was thus important to bring this expertise together to unify concepts and notations.

A design group was formed in September 1996 and one year later, the first version of the Modelica language was available (<http://www.Modelica.org>). It has been designed by a group

of more than 50 experts with previous knowledge of modeling languages and differential-algebraic equation models.

A paper by Hilding Elmqvist: “Modelica Evolution – From My Perspective” is available by the command **Help > Documentation**. The paper describes the history of Modelica and Dymola from the author’s perspective.

At the time of writing (2022) we have reached version 3.5 of the Modelica language specification and version 4.0.0 of the Modelica Standard Library (MSL).

1.6 Functional Mockup Interface



The Functional Mockup Interface (FMI) is an industry standard for combining simulation code modules (FMUs) from multiple tools and vendors. Developed under the auspices of the Modelica Association, the specification provides a well-defined and vendor-independent exchange format for code (binary or source) as well as associated data and documentation.

FMI is supported by a large number of authoring tools, including tools which are not Modelica based, making it the ideal foundation for a vendor independent simulation infrastructure.

The FMI 2.0 specification standardizes important extensions that improve simulation speed and accuracy.

- Classification of interface variables
- Save and restore FMI module state
- Variable dependency information
- Partial derivatives
- Precise time event handling
- Improved unit definitions

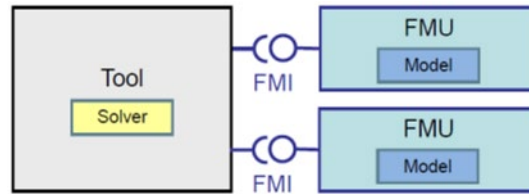
The FMI 3.0 specification contains further useful extensions. The most important extensions are:

- Array variables
- Terminals and icons
- Early return from co-simulation at event

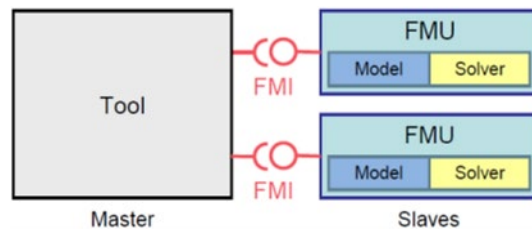
For more information, see the paper [The Functional Mock-up Interface 3.0 – New Features Enabling New Applications](#).

Model exchange and Co-simulation

The FMI specification defines two exchange formats. FMI for model exchange defines the interface for simulation code modules that must be combined with a common, central, solver. This ensures a uniform numeric solution and centralized strict simulation error control.



FMI for co-simulation defines the interface for code modules with embedded numeric solvers, as used by the generating tool. This approach gives the opportunity to embed dedicated solvers for the modeled application, and facilitates compatibility with simulation in the authoring tool.



FMI tools for Simulink – FMI Kit

Free download available.

Dassault Systèmes provides tools with full support for FMU export and import with Simulink. The toolkit can be used free without any license key. Support is available through the download site on GitHub.

<https://github.com/CATIA-Systems/FMKit-Simulink/releases>

1.7 FMI for embedded systems (eFMI®)



The [eFMI® Standard](#) is an open standard for the stepwise, model transformation-based development of advanced control functions suited for safety critical and real time targets. Its container architecture defines the common ground for collaboration among the stakeholders and tooling along the various abstraction levels from high-level – e.g. acausal, equation based physics – modelling and simulation down to actual embedded code.

eFMI defines a high-level intermediate representation language (GALEC) that provides a number of important guarantees for embedded systems that are missing in (basic) FMI. For example,

- Target independence.
- No hidden side effects.
- Vector and matrix operations optimized for the target platform.
- Static memory allocation and indexing that leads to generation of safe code with a guaranteed upper bound on execution time.

The eFMI toolchain consists minimally of a tool to translate from a high-level modeling language to GALEC, and (in most cases) a different tool to translate GALEC into certified executable code for the target environment.

The interoperability of eFMI tools is guaranteed by the well-defined model representations specified in the *eFMI Standard*. The following containers are supported:

1. Algorithm Code: Algorithmic, causal, high-level solution in GALEC.
2. Production Code: Target platform tailored and optimized C code for (1).
3. Behavioral Model: Test scenarios with well-defined, unique interpretation.
4. Binary Code: Target platform binaries for (2).

Dymola supports the generation of Algorithm Code, Behavioral Model, and Production Code containers – the latter by means of a seamless integration with the Software Product Engineering app on **3DEXPERIENCE** (separate Dassault Systèmes product) – eventually providing a model- and software-in-the-loop (MiL, SiL) development environment for advanced embedded control.

Further information about Dymola's eFMI facilities can be found in the *User's guide* of the *DymolaEmbedded* library distributed with Dymola (cf. the *eFMI* button in the *Tools* ribbon, menu entry *Load Libraries...*).

1.8 System Structure and Parameterization



[System Structure and Parameterization \(SSP\)](#) is a new system specification that allows tool independent specification of interconnected FMUs. Dymola has support for import and export of basic SSP files comprising:

- System Structure Descriptions (SSDs).
- Functional Mockup Units (FMUs).
- The connection structure between SSP components.
- Parameter bindings as defined in SSP.
- Parameter mappings and linear transformations (SSM).
- Metadata, including embedded SRMD metadata.

