[Referencing Package Contents](http://book.xogeny.com/components/packages/fqn/)

[Package Definitions](http://book.xogeny.com/components/packages/package_def/)



[http://book.xogeny.com/_static/TitleHeading.png](http://book.xogeny.com/)

Release: v0.3.0-12-g226adba-Early Access

[Packages](http://book.xogeny.com/components/packages/)

**Importing Physical Types**

*Table Of Contents*

* [Packages](http://book.xogeny.com/components/packages/)
  + [Importing Physical Types](http://book.xogeny.com/components/packages/nimport/)

*Importing Physical Types*

In the previous section, we learned how to reference types defined in other packages. This spares the developer from having to constantly define things in their local model. Instead, they can place definitions in packages and then reference those packages.

However, references with long fully qualified names can be tedious to type over and over again. For that reason, Modelica includes an import statement that allows us to use a definition as if it were defined locally.

Recall again, this example from a previous discussion on [*Physical Types*](http://book.xogeny.com/behavior/equations/physical/#physical-types):

**within** **ModelicaByExample.BasicEquations.CoolingExample**;

**model** **NewtonCoolingWithTypes** "Cooling example with physical types"

*// Types*

**type** **Temperature**=Real(unit="K", min=0);

**type** **ConvectionCoefficient**=Real(unit="W/(m2.K)", min=0);

**type** **Area**=Real(unit="m2", min=0);

**type** **Mass**=Real(unit="kg", min=0);

**type** **SpecificHeat**=Real(unit="J/(K.kg)", min=0);

*// Parameters*

**parameter** Temperature T\_inf=298.15 "Ambient temperature";

**parameter** Temperature T0=363.15 "Initial temperature";

**parameter** ConvectionCoefficient h=0.7 "Convective cooling coefficient";

**parameter** Area A=1.0 "Surface area";

**parameter** Mass m=0.1 "Mass of thermal capacitance";

**parameter** SpecificHeat c\_p=1.2 "Specific heat";

*// Variables*

Temperature T "Temperature";

**initial equation**

T = T0 "Specify initial value for T";

**equation**

m\*c\_p\*der(T) = h\*A\*(T\_inf-T) "Newton's Law of Cooling";

**end** NewtonCoolingWithTypes;

The previous section described how we could avoid defining these types locally by using types from the [*Modelica Standard Library*](http://book.xogeny.com/components/packages/msl/#msl). But we can also use the importcommand to import those types from the Modelica Standard Library once and then use them without having to specify their fully qualified names. The resulting code would look something like:

**within** **ModelicaByExample.PackageExamples**;

**model** **NewtonCooling**

"Cooling example importing physical types from the Modelica Standard Library"

**import** **Modelica.SIunits.Temperature**;

**import** **Modelica.SIunits.Mass**;

**import** **Modelica.SIunits.Area**;

**import** ConvectionCoefficient = **Modelica.SIunits.CoefficientOfHeatTransfer**;

**import** SpecificHeat = **Modelica.SIunits.SpecificHeatCapacity**;

*// Parameters*

**parameter** Temperature T\_inf=300.0 "Ambient temperature";

**parameter** Temperature T0=280.0 "Initial temperature";

**parameter** ConvectionCoefficient h=0.7 "Convective cooling coefficient";

**parameter** Area A=1.0 "Surface area";

**parameter** Mass m=0.1 "Mass of thermal capacitance";

**parameter** SpecificHeat c\_p=1.2 "Specific heat";

*// Variables*

Temperature T "Temperature";

**initial equation**

T = T0 "Specify initial value for T";

**equation**

m\*c\_p\*der(T) = h\*A\*(T\_inf-T) "Newton's Law of Cooling";

**end** NewtonCooling;

Here we have replaced the type definitions with import statements. Note how the highlighted lines are identical to the previous code. Let’s look at two of these import statements more closely to understand what effect they have on the model. Let’s start with the following import statement:

**import** **Modelica.SIunits.Temperature**;

This imports the type Modelica.SIunits.Temperature into the current model. By default, the name of this imported type will be the last name in the fully qualified name, *i.e.,* Temperature. This means that with this import statement present, we can simply use the type name Temperature and that will automatically refer back to Modelica.SIunits.Temperature.

Now let’s look at another import statement:

**import** ConvectionCoefficient = **Modelica.SIunits.CoefficientOfHeatTransfer**;

The syntax here is a little bit different. In this case, the type that we are importing is Modelica.SIunits.CoefficientOfHeatTransfer. But instead of creating a local type based on the last name in the fully qualified name, *i.e.,* CoefficientOfHeatTransfer we are specifying that the local type should beConvectionCoefficient. In this case, this allows us to use the name we originally used in our earliest examples. In this way, we can avoid refactoring any code that used the previous name. Another reason for specifying an alternative name (other than the default one that the Modelica compiler would normally assign) would be to avoid name collision. Imagine we wished to import two types from two different packages, *e.g.,*

**import** **Modelica.SIunits.Temperature**; *// Celsius*

**import** **ImperialUnits.Temperature**; *// Fahrenheit*

This would leave us two types both named Temperature. By defining an alternative name for the local alias, we could do something like this:

**import** DegK = **Modelica.SIunits.Temperature**; *// Kelvin*

**import** DegR = **ImperialUnits.Temperature**; *// Rankine*

**SI Units**

Note that this example imports imperial units just to demonstrate how a potential name clash might occur. But it is very bad practice to do this in practice. When using Modelica you should always use SI units and never use any other system of units. If you want to enter data or display results in other units, please use thedisplayUnit attribute discussed previously in the section on [*Attributes*](http://book.xogeny.com/behavior/equations/variables/#attributes).

There is one last form of the import statement worth discussing which is the wildcard import statement. Importing units one unit at a time can be tedious. The wildcard import allows us to import **all** types from a given package at once. Recall the following earlier example:

**within** **ModelicaByExample.BasicEquations.RotationalSMD**;

**model** **SecondOrderSystem** "A second order rotational system"

**type** **Angle**=Real(unit="rad");

**type** **AngularVelocity**=Real(unit="rad/s");

**type** **Inertia**=Real(unit="kg.m2");

**type** **Stiffness**=Real(unit="N.m/rad");

**type** **Damping**=Real(unit="N.m.s/rad");

**parameter** Inertia J1=0.4 "Moment of inertia for inertia 1";

**parameter** Inertia J2=1.0 "Moment of inertia for inertia 2";

**parameter** Stiffness k1=11 "Spring constant for spring 1";

**parameter** Stiffness k2=5 "Spring constant for spring 2";

**parameter** Damping d1=0.2 "Damping for damper 1";

**parameter** Damping d2=1.0 "Damping for damper 2";

Angle phi1 "Angle for inertia 1";

Angle phi2 "Angle for inertia 2";

AngularVelocity omega1 "Velocity of inertia 1";

AngularVelocity omega2 "Velocity of inertia 2";

**initial equation**

phi1 = 0;

phi2 = 1;

omega1 = 0;

omega2 = 0;

**equation**

*// Equations for inertia 1*

omega1 = der(phi1);

J1\*der(omega1) = k1\*(phi2-phi1)+d1\*der(phi2-phi1);

*// Equations for inertia 2*

omega2 = der(phi2);

J2\*der(omega2) = k1\*(phi1-phi2)+d1\*der(phi1-phi2)-k2\*phi2-d2\*der(phi2);

**end** SecondOrderSystem;

We could replace these type definitions with import statements, *e.g.*,

**import** **Modelica.SIunits.Angle**;

**import** **Modelica.SIunits.AngularVelocity**;

**import** **Modelica.SIunits.Inertia**;

**import** Stiffness = **Modelica.SIunits.RotationalSpringConstant**;

**import** Damping = **Modelica.SIunits.RotationalDampingConstant**;

However, the more types we bring in, the more import statements we need to add. Instead, we could write our model as follows:

**within** **ModelicaByExample.PackageExamples**;

**model** **SecondOrderSystem**

"A second order rotational system importing types from Modelica Standard Library"

**import** **Modelica.SIunits**.\*;

**parameter** Angle phi1\_init = 0;

**parameter** Angle phi2\_init = 1;

**parameter** AngularVelocity omega1\_init = 0;

**parameter** AngularVelocity omega2\_init = 0;

**parameter** Inertia J1=0.4;

**parameter** Inertia J2=1.0;

**parameter** RotationalSpringConstant k1=11;

**parameter** RotationalSpringConstant k2=5;

**parameter** RotationalDampingConstant d1=0.2;

**parameter** RotationalDampingConstant d2=1.0;

Angle phi1;

Angle phi2;

AngularVelocity omega1;

AngularVelocity omega2;

**initial equation**

phi1 = phi1\_init;

phi2 = phi2\_init;

omega1 = omega1\_init;

omega2 = omega2\_init;

**equation**

omega1 = der(phi1);

omega2 = der(phi2);

J1\*der(omega1) = k1\*(phi2-phi1)+d1\*der(phi2-phi1);

J2\*der(omega2) = k1\*(phi1-phi2)+d1\*der(phi1-phi2)-k2\*phi2-d2\*der(phi2);

**end** SecondOrderSystem;

Note the highlighted import statement. This single (wildcard) import statement imports all definitions from Modelica.SIunits into the current model. With wildcard imports, there is no option to “rename” the types. They will have exactly the name locally as they have in the named package.

Before using wildcard imports, be sure to read [*this caveat*](http://book.xogeny.com/components/packages/importing/#wildcards-harmful).

In this chapter, we’ve seen how import statements can be used to import types from other packages. As it turns out, import statements are not always that useful. When models are being developed within a graphical modeling environment, tools generally use the least ambiguous and most explicit method for reference types: using fully qualified names. After all, when using a graphical tool the length of the name is not an issue because it doesn’t need to be typed. This also avoids issues with name lookup, naming conflicts, etc.