[Organizing Content](http://book.xogeny.com/components/packages/organizing/)

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



[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/)

**Referencing Package Contents**

*Table Of Contents*

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

*Referencing Package Contents*

Now that we’ve covered [*Organizing Content*](http://book.xogeny.com/components/packages/organizing/#organizing-content), we’ll discuss how to access that content across different packages. Let’s consider the following example:

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

**model** **RLC** "An RLC circuit referencing types from the Modelica Standard Library"

**parameter** **Modelica.SIunits.Voltage** Vb=24 "Battery voltage";

**parameter** **Modelica.SIunits.Inductance** L = 1;

**parameter** **Modelica.SIunits.Resistance** R = 100;

**parameter** **Modelica.SIunits.Capacitance** C = 1e-3;

**Modelica.SIunits.Voltage** V;

**Modelica.SIunits.Current** i\_L;

**Modelica.SIunits.Current** i\_R;

**Modelica.SIunits.Current** i\_C;

**equation**

i\_R = V/R;

i\_C = C\*der(V);

i\_L=i\_R+i\_C;

L\*der(i\_L) = (Vb-V);

**end** RLC;

As we learned in the previous section, the very first line,

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

tells us that the RLC model is contained within the ModelicaByExample.PackageExamples package. As with the previous example, we are going to make use of the Modelica package system to allow us to avoid defining types directly in our model. In this way, we define the types once in one package and then we can reuse them in many places simply by referencing them.

Unlike the previous example in this chapter, we don’t define any types in this example. Instead, we rely on types that are defined in the [*Modelica Standard Library*](http://book.xogeny.com/components/packages/msl/#msl). The[*Modelica Standard Library*](http://book.xogeny.com/components/packages/msl/#msl) contains many useful types, models, constants, *etc*. For this example, we’ll just utilize a few of them. These types can be easily recognized because they start with Modelica. in the name of the type.

We look more closely at the [*Lookup Rules*](http://book.xogeny.com/components/packages/lookup/#lookup-rules) later in this chapter. For now, it is sufficient to say that all the types starting with Modelica. exist within the Modelicapackage. In this case, all types start with Modelica.SIunits. SIunits is a package within the Modelica package. The purpose of the SIunits package is to store type definitions that conform to ISO standard quantities and units.

As can be seen in the example code, these types are referenced by their “fully qualified name”. That means that type name starts with the name of a top-level package (a package that is not contained within another package). Each . in the name represents a new child package. The last name in the sequence identifies that actual type being referenced.

In this case, we are using 5 different types from within the Modelica.SIunits package: Voltage, Inductance, Resistance, Capacitance and Current. These types provide information about the units for each of these types, limitations on the values of these types (*e.g.*, a capacitance cannot be less than zero), *etc*. They are defined in the [*Modelica Standard Library*](http://book.xogeny.com/components/packages/msl/#msl) as follows:

*// Base Definitions*

**type** **ElectricPotential** = Real(**final** quantity="ElectricPotential",

**final** unit="V");

**type** **ElectricCurrent** = Real(**final** quantity="ElectricCurrent",

**final** unit="A");

*// The types referenced in our example*

**type** **Voltage** = ElectricPotential;

**type** **Inductance** = Real(**final** quantity="Inductance",

**final** unit="H");

**type** **Resistance** = Real(**final** quantity="Resistance",

**final** unit="Ohm");

**type** **Capacitance** = Real(**final** quantity="Capacitance",

**final** unit="F", min=0);

**type** **Current** = ElectricCurrent;

Apart from providing better documentation, there is an immediate benefit to associating such types with variables and that is because it enables unit consistency checking of the equations. For example, note the following equation from this example:

i\_R = V/R;

Clearly, this is a statement of Ohm’s law. But what if we made a mistake and accidentally wrote:

i\_R = V\*R;

Syntactically speaking, this equation is perfectly legal. Furthermore, if the variable i\_R, V and R were all declared to have the type Real, there would be no issue with this equation. However, because we know (from the type definitions) that these variables represent a current, a voltage and a resistance, respectively, a Modelica compiler is able to determine (in a completely automatic way using the definitions shown previously) that the left and right hand sides of this equation are inconsistent with respect to physical units. In other words, by associating a physical type with variables it is possible to detect modeling errors, automatically.