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

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



[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/)

**Organizing Content**

*Table Of Contents*

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

*Organizing Content*

Let’s start by simply demonstrating how content can be organized into packages. To do this, we will revisit the [*Classic Lotka-Volterra*](http://book.xogeny.com/behavior/equations/population/#classic-lotka-volterra) model. In our previous models, all variables had the type Real. Let’s enhance that model to include types for the various quantities in the model.

We can organize those types into a package like this:

**package** **Types**

**type** **Rabbits** = Real(quantity="Rabbits", min=0);

**type** **Wolves** = Real(quantity="Wolves", min=0);

**type** **RabbitReproduction** = Real(quantity="Rabbit Reproduction", min=0);

**type** **RabbitFatalities** = Real(quantity="Rabbit Fatalities", min=0);

**type** **WolfReproduction** = Real(quantity="Wolf Reproduction", min=0);

**type** **WolfFatalities** = Real(quantity="Wolf Fatalities", min=0);

**end** Types;

The first thing to note about this Modelica code is that it uses the package keyword. The syntax of a package definition is very similar to the definition of a model orfunction. The main difference is that a package contains only definitions or constants. It cannot contain any variable declarations except those that are constant. In this case, we see that this package contains only type definitions.

Now let’s turn our attention to the Lotka-Volterra model itself. Assuming it doesn’t need to define the types itself, but can rely on the types we’ve just defined, it can be refactored to look as follows:

**model** **LotkaVolterra** "Lotka-Volterra with types"

**parameter** **Types.RabbitReproduction** alpha=0.1;

**parameter** **Types.RabbitFatalities** beta=0.02;

**parameter** **Types.WolfReproduction** gamma=0.4;

**parameter** **Types.WolfFatalities** delta=0.02;

**parameter** **Types.Rabbits** x0=10;

**parameter** **Types.Wolves** y0=10;

**Types.Rabbits** x(start=x0);

**Types.Wolves** y(start=y0);

**equation**

der(x) = x\*(alpha-beta\*y);

der(y) = -y\*(gamma-delta\*x);

**end** LotkaVolterra;

Notice how all the parameters and variables now have a specific type (and not just the ordinary Real) type. Instead, we are able to associate additional information above and beyond the fact that these are continuous variables. For example, we can specify that these values should not be negative by adding the min=0 modifier to their type definitions.

Looking at the Lotka-Volterra model by itself, it isn’t obvious **where** it finds these type definitions. The Modelica compiler will use a collection of [*Lookup Rules*](http://book.xogeny.com/components/packages/lookup/#lookup-rules) to lookup these definitions. We’ll come to the lookup rules eventually. For now, the important point is that we have the ability to refer to things that are not in our immediate model.

Let’s “zoom out” a little bit to see some additional details related to organizing models. The Types package we showed earlier and the LotkaVolterra model that references it are contained within a package called NestedPackages which is defined as follows:

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

**package** **NestedPackages**

"An example of how packages can be used to organize things"

**package** **Types**

**type** **Rabbits** = Real(quantity="Rabbits", min=0);

**type** **Wolves** = Real(quantity="Wolves", min=0);

**type** **RabbitReproduction** = Real(quantity="Rabbit Reproduction", min=0);

**type** **RabbitFatalities** = Real(quantity="Rabbit Fatalities", min=0);

**type** **WolfReproduction** = Real(quantity="Wolf Reproduction", min=0);

**type** **WolfFatalities** = Real(quantity="Wolf Fatalities", min=0);

**end** Types;

**model** **LotkaVolterra** "Lotka-Volterra with types"

**parameter** **Types.RabbitReproduction** alpha=0.1;

**parameter** **Types.RabbitFatalities** beta=0.02;

**parameter** **Types.WolfReproduction** gamma=0.4;

**parameter** **Types.WolfFatalities** delta=0.02;

**parameter** **Types.Rabbits** x0=10;

**parameter** **Types.Wolves** y0=10;

**Types.Rabbits** x(start=x0);

**Types.Wolves** y(start=y0);

**equation**

der(x) = x\*(alpha-beta\*y);

der(y) = -y\*(gamma-delta\*x);

**end** LotkaVolterra;

**end** NestedPackages;

A really important thing to note about the NestedPackages package is that it is contained inside another package called PackageExamples which is, in turn, contained within a package called ModelicaByExample. We know this from the within clause at the top:

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

**Every single model** that we’ve simulated so far in this book is contained within a package. When we showed the source code to those examples, we clipped the top line because we were not yet ready to discuss what the within clause was used for. But it was there in all cases.

Note that the Types package and the LotkaVolterra model don’t include any kind of within clause. That’s because we **know** what package they are in because they are defined directly inside the NestedPackages package. So why does it appear immediately before the definition of NestedPackages? Because theNestedPackages package is a stand-alone file. In other words, when Modelica definitions are mapped into files and directories, we need to explicitly specify how they are related. We’ll discuss the relationship between files, directories and [*Package Definitions*](http://book.xogeny.com/components/packages/package_def/#package-definitions) later. For now, the important thing to understand is that the within clause is simply used to specify the parent package.