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

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



[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/)

**Lookup Rules**

*Table Of Contents*

* [Packages](http://book.xogeny.com/components/packages/)
  + [Lookup Rules](http://book.xogeny.com/components/packages/lookup/)

*Lookup Rules*

Recall the following example from our discussion on [*Organizing Content*](http://book.xogeny.com/components/packages/organizing/#organizing-content):

**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;

When we discussed [*Referencing Package Contents*](http://book.xogeny.com/components/packages/fqn/#ref-pkg-contents), the example that was presented used fully qualified names for all the types it referenced. But the example above doesn’t. We see, from the LotkaVolterra model that the Wolves type is referenced as:

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

And not as:

**parameter** **ModelicaByExample.PackageExamples.NestedPackages.Types.Wolves** y0=10;

In other words, we didn’t use the fully qualified name. But the LotkaVolterra model compiles just fine. So how is it that the Modelica compiler knows which definition of Wolves to use?

The answer involves “name lookup” in Modelica. Name lookup in Modelica involves searching for the named definition. Type names in Modelica are generally qualified (although not necessarily **fully** qualified) names. This means they may contain a . in them, *e.g.,* Modelica.SIunits.Voltage. In order to locate the matching definition associated with a name, the Modelica compiler starts by looking for the **first** name in the qualified name, *e.g.,* Modelica. It searches for a matching definition in the following order:

1. Look for a matching name among builtin types
2. Look in the current definition for a nested definition with a matching name (include inherited definitions)
3. Look in the current definition for an imported definition with a matching name (do not include inherited imports)
4. Look in the parent package of the current definition for a nested definition with a matching name (including inherited definitions)
5. Look in the parent package for an imported definition with a matching name (not including inherited imports)
6. Look in each successive parent (using the same approach) until either:
   * The parent package has the encapsulated qualifier, in which case the search terminates.
   * There are no more parent packages, in which case you search for a match among root level packages.

If the given name cannot be found after searching all these locations, then the search fails and the type cannot be resolved. If the search succeeds, then we’ve located the definition of the **first name** in the qualified name. If the name is not qualified (*i.e.,* it does not have a . in the name), then we are done. However, if it does have other components in the name, these must be nested definitions contained within the definition returned by the search. If nested definitions cannot be found for all remaining components in a qualified name, then the search fails and the type cannot be resolved.

At first, this might sound very complicated. However, most of the time these rules are not very important. The reason is that, as we discussed previously, most graphical Modelica environments will use fully qualified names. Most type names in Modelica code will either reference local definitions or will be specified with fully qualified names.

**Duplicate Names**

You should always avoid having nested packages with the same name as a top-level package. The reason this is a problem is that the lookup rules search up through the package hierarchy. As a result, they will find the nested definition before the root level one. This means that lookup of fully qualified names (ones that are referenced relative to the root of the package tree) will fail because they will find the nested definition first.