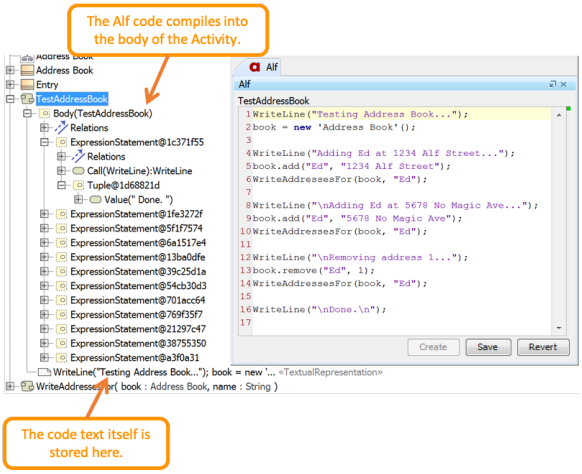
The Alf compiler

The *Alf compiler* is the part of the Alf plugin that carries out the compilation process. The Alf compiler is automatically invoked when you save Alf code into your model (unless you turn off automatic building; see Environment options). The activity model resulting from an Alf compilation is also stored in your model. Exactly where it is stored depends on which kind of element the Alf code is the body.

Table of Contents

Activity compilation

When the Alf compiler processes the Alf body of an Activity, the resulting Activity model is saved as the Activity Nodes and Edges owned within that Activity. The Alf text itself is stored in a Comment element owned by the Activity, which has the *TextualRepresentation* stereotype applied. (See also Using Alf to define Activities and Using Alf for Operation methods.)

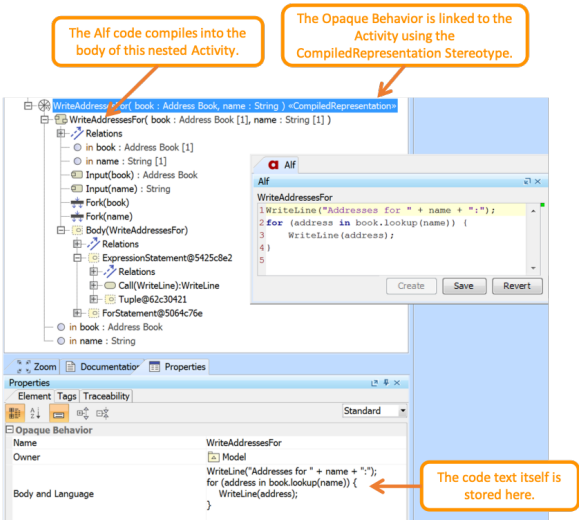


Activity compilation

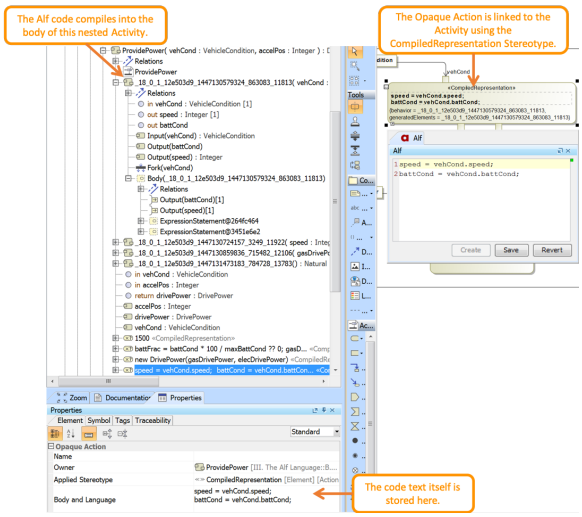
The compilation of an activity may also result in certain auxiliary elements, such as Instance Specifications, that cannot be not stored within the Activity. These are generally inserted into the nearest Package containing the Activity (which may be the top-level Model). Also, in certain cases, template instantiations are saved in a special package called *$$Template Bindings* (described below).

Opaque Behavior and Opaque Action compilation

When the Alf compiler processes the Alf body of an Opaque Behavior or Opaque Action, it creates a corresponding Activity in which to store the result of the compilation. For an Opaque Behavior, the generated Activity is saved as a child of the Opaque Behavior. For an Opaque Action, the generated Activity is saved as a child of the Activity containing the Opaque Action. In either case, the Activity is also linked to the original Opaque Behavior or Action by applying the *CompiledRepresentation* stereotype. When the Opaque Behavior or Action is later executed using Magic Model Analyst, it is the linked Activity that actually provides the executable behavior.



Opaque Behavior compilation

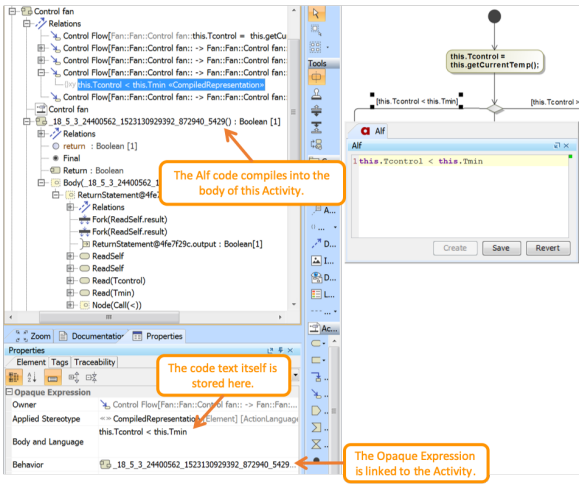


Opaque Action compilation

Opaque Expression compilation

When the Alf compiler processes the Alf body of an Opaque Expression, it creates a corresponding Activity in which to store the results of the compilation. This Activity is generally saved in the nearest Classifier containing the Opaque Expression. If the Opaque Expression specifies the guard on a Transition, then the generated Activity will be saved as a child of the State Machine containing the Transition. And, if the Opaque Expression specifies the guard on an Activity Edge, then the generated Activity will be saved as a child of the Activity containing the Activity Edge.

In all cases, the Activity is referenced in the **Behavior** property of the original Opaque Expression. When the Opaque Expression is later evaluated using Magic Model Analyst, it is the **Behavior** Activity that is actually executed in order to provide the value of the Expression.

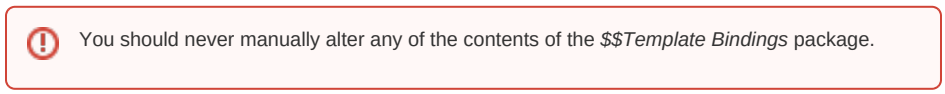


Opaque Expression compilation (example shown is for an Activity Edge guard)

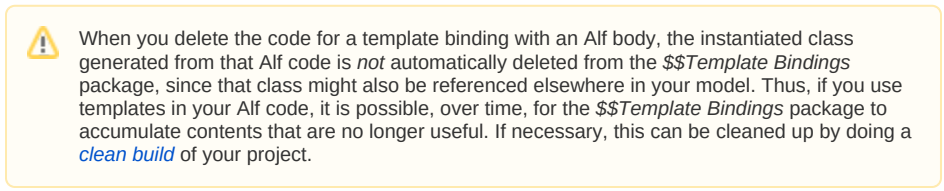
The $$Template Bindings package

The *$$Template Bindings* package is a special package used by the Alf compiler to save instantiations of template Classifiers that result of explicit template bindings in Alf code (e.g., *Set<Integer>*). These instantiations are stored in a common place, so that similar bindings in Alf code across your model can be compiled to references to the same instantiation. For example, a binding of a standard Alf collection class, such as *Set<Integer>,* results in the generation of an instantiated class in the *$$Template Bindings* package with a long encoded name similar to *$$Model$$Alf$$Library$$CollectionClasses$$Set__$$Model$$'UML Standard Profile'$$'UML2 Metamodel'PrimitiveTypes$$Integer__*. If you use *Set<Integer>* in more than once in Alf code in your model, it will still result in only the one instantiated class, and the Activity models compiled from your Alf code will all refer to this class.

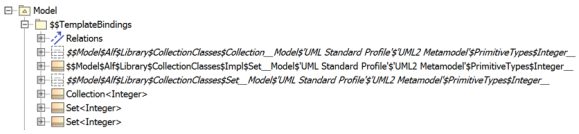
The *$$Template Bindings* package should only appear if you use template bindings in your Alf code. It is automatically created the first time such a binding is compiled in your model.



You should never manually alter any of the contents of the *$$Template Bindings* package.



When you delete the code for a template binding with an Alf body, the instantiated class generated from that Alf code is *not* automatically deleted from the *$$Template Bindings* package, since that class might also be referenced elsewhere in your model. Thus, if you use templates in your Alf code, it is possible, over time, for the *$$Template Bindings* package to accumulate contents that are no longer useful. If necessary, this can be cleaned up by doing a *clean build* of your project.



The *$$Template Bindings* package, showing the instantiation of *Set<Integer>*