19.0 Version News

Alf Plugin

Released on: July 2, 2018

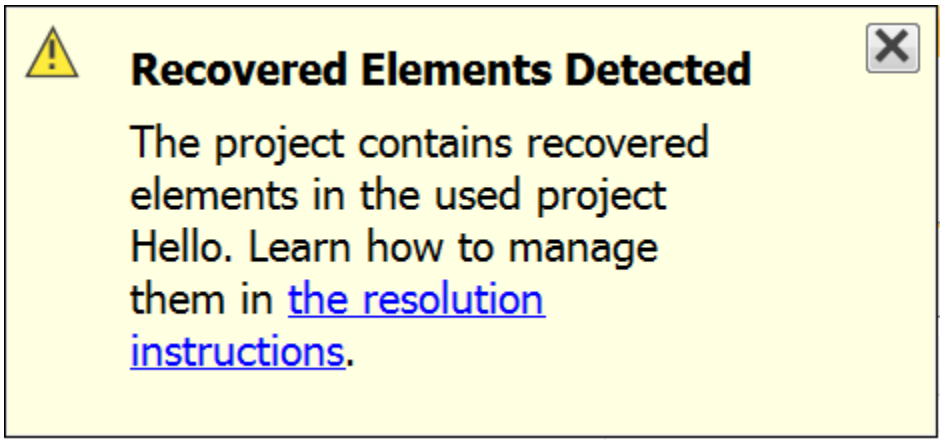
No Magic, Inc., is pleased to release the Alf Plugin version 19.0 for MagicDraw. Alf (the Action Language for Foundational UML) is an Object Management Group (OMG) standard textual language for representing behaviors within a UML model. When used in conjunction with Cameo Simulation Toolkit, behavioral specifications entered using the syntax-aware Alf editor are compiled into fully executable fUML activity models. The version 19.0 release provides better user interface integration, performance improvements and new features, such as support for using decision values in activity edge guards and event data in state machine behaviors.

What you get:

- Migrating from Version 18.5
- User Interface Integration
- Performance Improvement
- New Features
- Open API

Migrating from Version 18.5

If you have been using Alf Plugin 18.5, then you can still open Alf projects created using 18.5 in MagicDraw 19.0 with the new version of the plugin. However, when you do this, you will initially get a **Recovered Elements Detected** warning, as shown in the image below. This is because the v18.5 Alf Library contained its own version of the fUML Library, while the v19.0 Alf Library uses the fUML Library provided by Cameo Simulation Toolkit. This can be fixed by updating the project from v18.5 to v19.0 using the procedure below.



Recovered Elements Detected warning message

To update an Alf project from v18.5 to v19.0:

---

1. Open the project.
2. If the project is a server project, right click on the top level Model element in the model browser and select **Lock > Lock Elements for Edit Recursively**.
3. Select **Tools > Alf > Clean Project** to do a clean build of the project.
4. Save the project.
5. Close the project and re-open it.
6. If there is still a warning for recovered elements, but they are all marked with (yellow) warning annotations, *not* (red) error annotations, then do the following:
    a. Right click on *fUML_Library [Alf-Library.mdzip]*. (Make sure you select the *fUML_Library* from *Alf-Library.mdzip,* not the one from *fUML-Library.mdzip.*)
    b. Select **Validation > Unused Recovered Element > Remove Unused Recovered Element**.
7. If, on the other hand, there are any error annotations, then see below.

In most cases, the above procedure should be sufficient to resolve all recovered elements. However, if the original model explicitly referenced elements from the fUML Library (not just as a result of compiled Alf code), then these references will need to be corrected manually.

To correct usages of fUML Library elements:

---

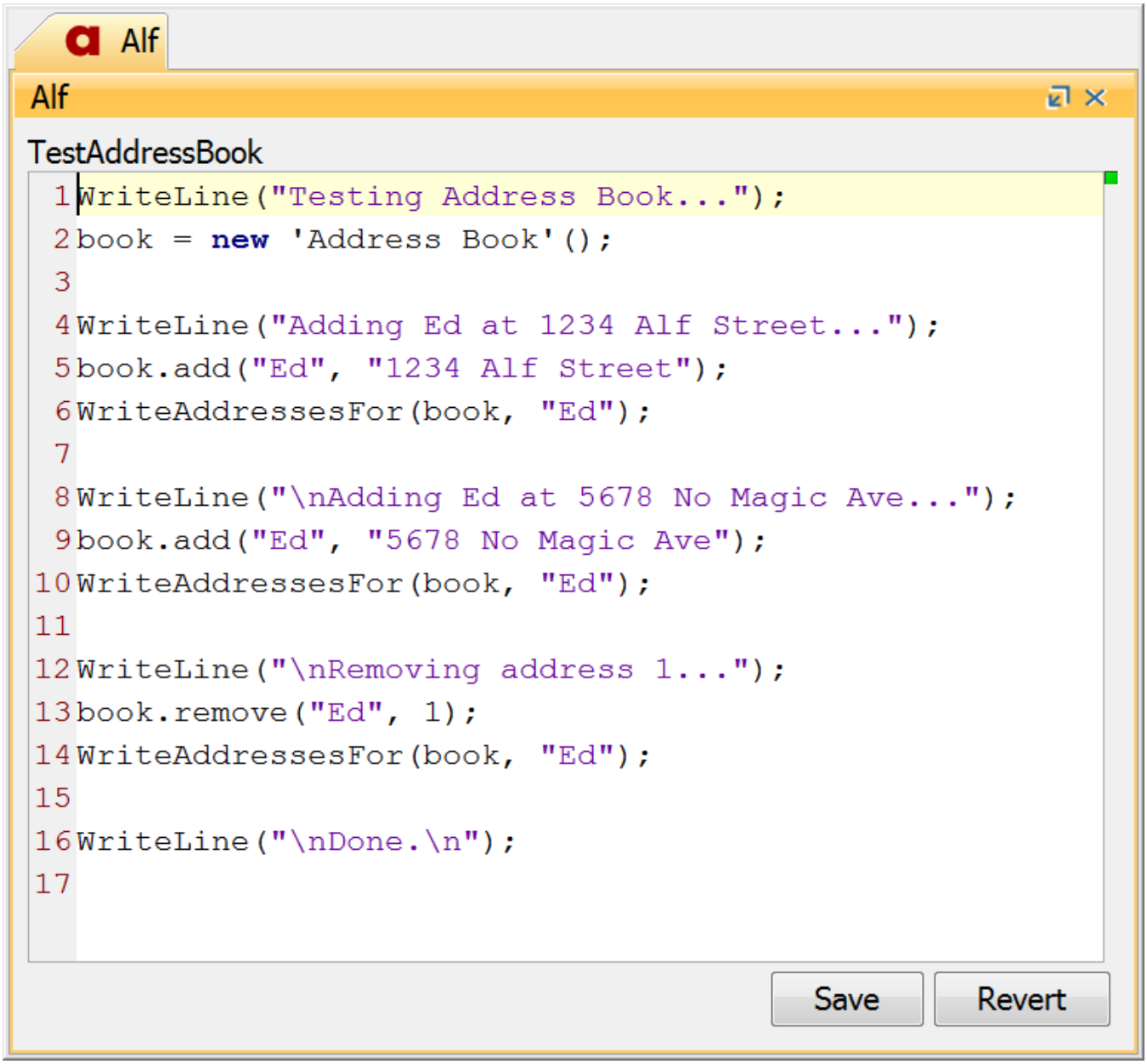
1. If used projects are not being shown in the Model Browser, then click on the gear icon in the upper right corner of the Containment pane and select **Show Auxiliary Resources**.
2. In the Model Browser, find *fUML_Library [Alf-Library.mdzip]*. (Make sure you select the *fUML_Library* from *Alf-Library.mdzip,* not the one from *fUML-Library.mdzip.*)
3. Expand the subtree under *fUML_Library* until you find the one or more recovered elements that are explicitly referenced in your model.
4. For each of these elements, do the following:
    a. Right click and select **Validation > Recovered Element > Change Usages To...**
    b. In the lower left corner of the **Select Element** window, make sure that **Apply Filter** is *not* checked.
    c. Select the element under *fUML_Library [fUML-Library.mdzip]* that corresponds to the recovered element being resolved.
    d. Click **OK**.
5. Select **Tools > Alf > Clean Project** to do a clean build of the project.
6. Save the project.
7. Close the project and re-open it.

For more information on managing recovered elements, click on the "resolution instructions" link in the **Recovered Elements Detected** message.

Back to top

User Interface Integration

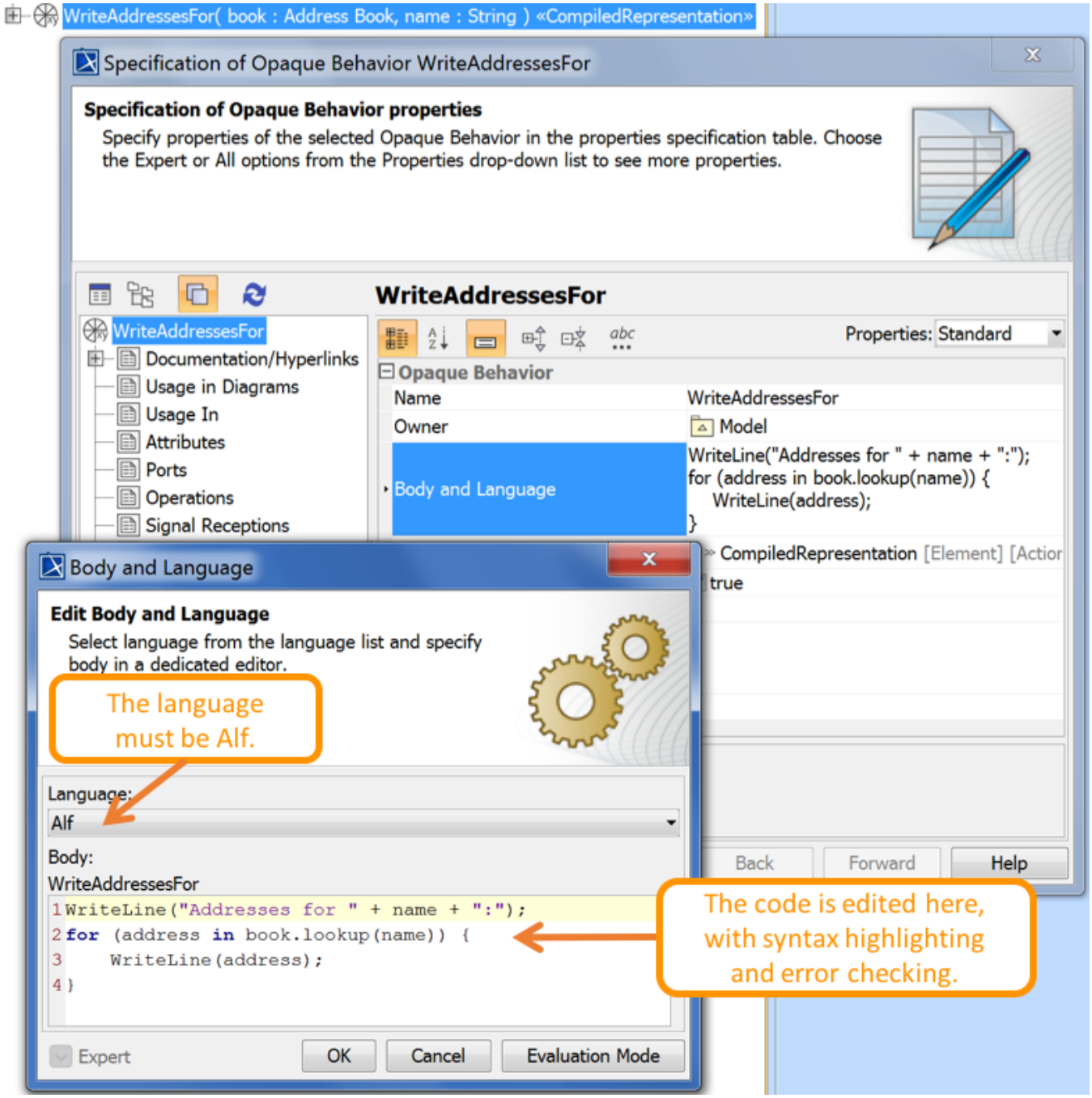
The **Alf** editor window is a dockable project window. Open it using **Windows** > **Alf** and select any model element with an Alf body to edit the Alf text in the window.



The Alf Editor Window

Learn more about the Alf editor >>

The Alf Editor capabilities for syntax highlighting and constraint checking are now also available when Alf text is edited directly in the bodies of Opaque Behaviors, Opaque Actions and Opaque Expressions.



Editing Alf text in an Opaque Behavior

Learn about editing Alf for

- Opaque Behavior bodies >>
- State Behaviors >>
- Transition Behaviors >>
- Transition guards >>
- Opaque Action bodies >>
- Activity Edge guards >>

Back to top

Performance Improvement

The Alf Compiler is now based on v1.1.0 of the Alf Reference Implementation, which allows for significant improvement in the performance of the parsing of Alf text. Simply parsing and checking Alf text in the Alf editor also no longer results in *Check Alf* sessions being inserted into the undo buffer (compiling and saving Alf text is still undoable).

In the original 18.5 beta version of the Alf plugin, the Activities and other Elements generated from Alf compilation were saved together in a common *$$Auxiliary Elements,* which caused difficulties when using Teamwork Server and Teamwork Cloud. Starting with the second 18.5 beta version of the plugin, generated Elements are stored in the project containment hierarchy as close as possible to the Element with the Alf body from which they were generated. If you still have an Alf project created using the original 18.5 beta version, then, when you do a clean compile of a project in 19.0, the *$$Auxiliary Element* package will be removed from the project. A common *$$Template Bindings* package is added to the project only if there are explicit template instantiations in the Alf code.

Learn more about the Alf compiler >>

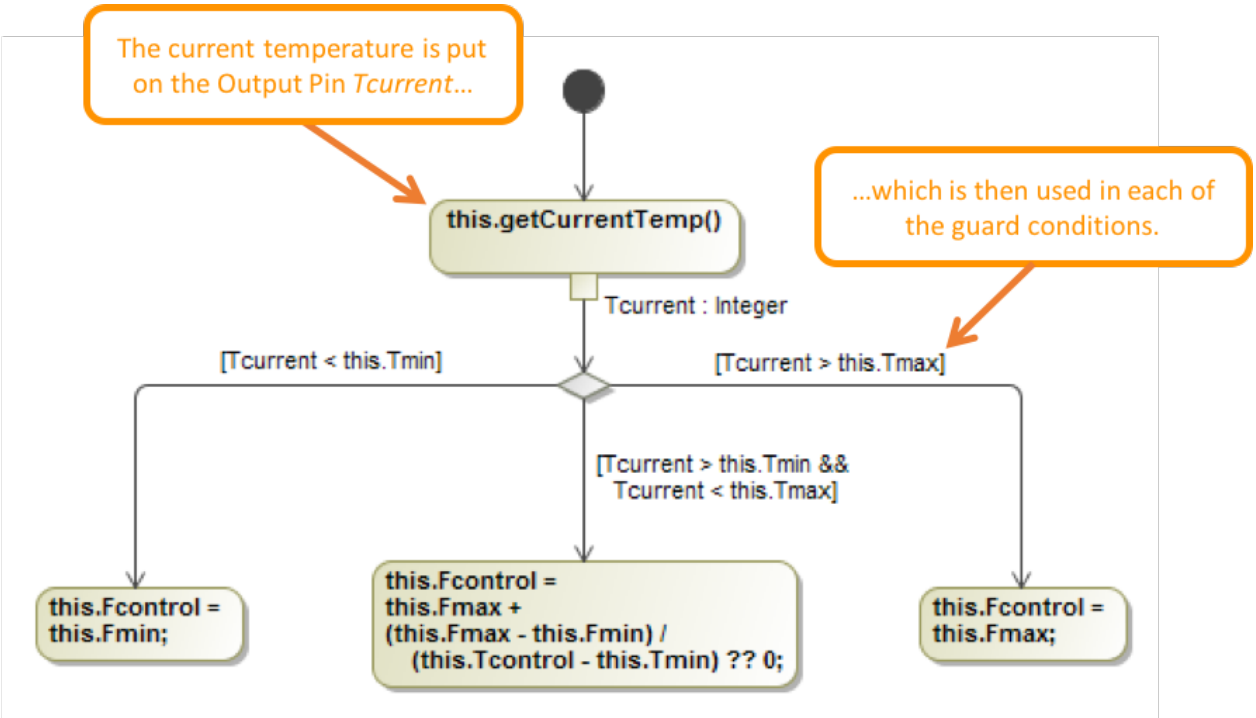
Back to top

New Features

If a project was not created as an Alf project, the Alf Library is not loaded automatically when Alf code is first entered. Instead, convert a non-Alf project into an Alf project by using the **Tools > Alf > Load Library** command.

Learn more about Alf projects >>

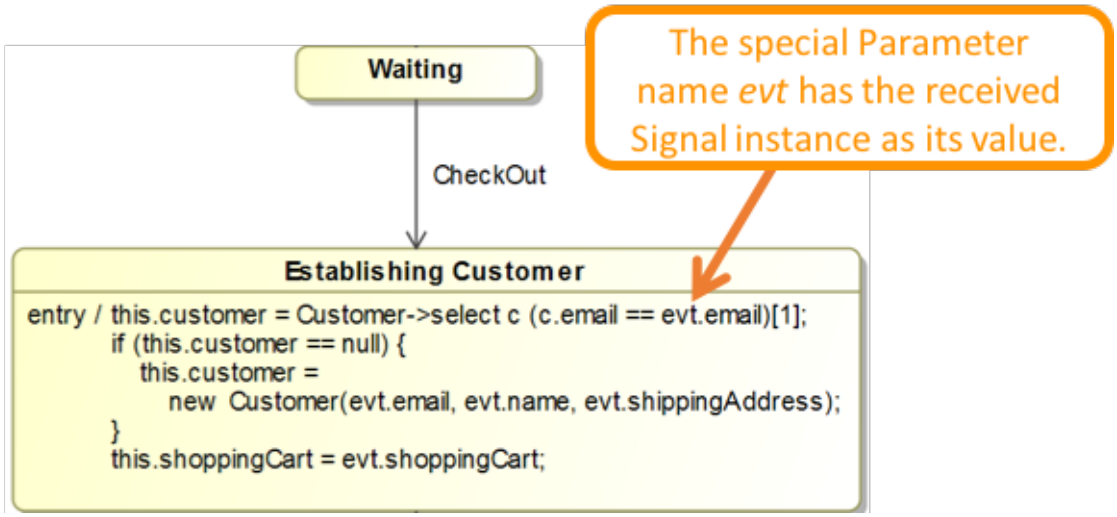
You can use the decision input value for a Decision Node in Alf expressions used to specify the guards on Activity Edges outgoing from the Decision Node.



Accessing the decision input value in guards

Learn more about accessing data in Activity Edge guards >>

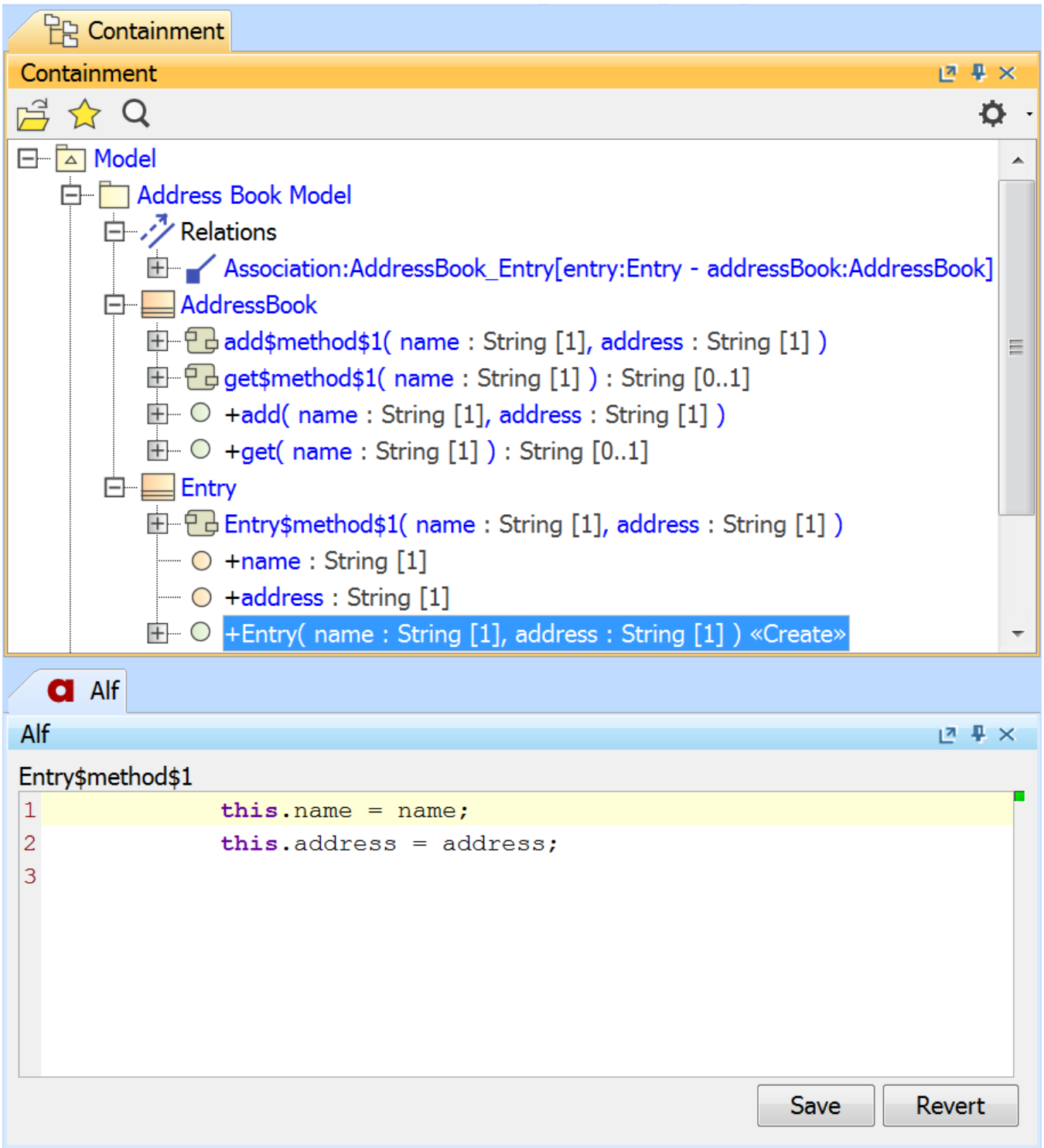
You can access the data in the guards and effect Behaviors of a Transition from the event that triggered the Transition. For Signal events, the event data is the values of the attributes of the Signals. For Call events, the event data is the values of the Parameters of the called Operation. You can also access event data from the exit Behavior of the State being exited and from the entry and do-activity Behaviors of the State being entered.



Accessing event data in an entry Behavior

Learn more about accessing event data in State Machines >>

You can import Alf units from external files into an existing or new project. (Note that exporting Alf is *not* possible at this time.)



Imported Alf Code

Learn more about importing Alf >>

Back to top

Open API

The Alf Plugin now includes an Open API that you can use to compile and import Alf code when developing a new MagicDraw plugin.

Learn mode about the Alf Open API >>

Back to top