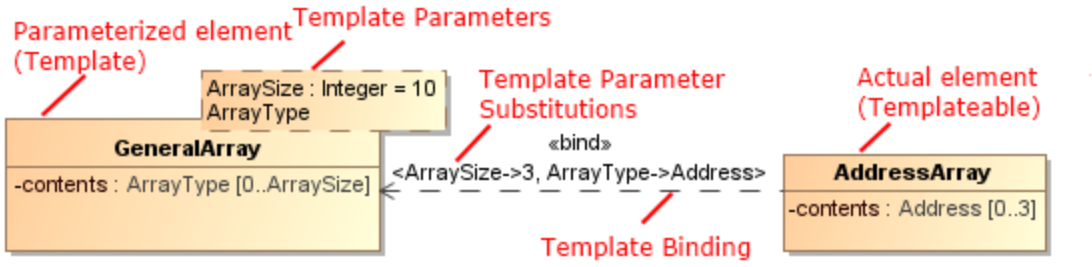
Template

Use a Template to define concrete types for elements without concrete types. You can create a template and then use it as many times as you need. Moreover, when you have elements of the same structure (or with the same parameters) but with different values, the template you use can easily define parameters for each element. You can apply template parametrization to classifiers, packages, and operations. Most often. template parametrization is used for classes and interfaces. Templates are often used in code engineering (Java, C#, and other programming languages).

The template consists of:

- A parameterized element (an element with defined template parameters). The parameterized element can also be called a template element.
- An actual element (an element for which a template is applied, an element with actual values). The actual element can also be called a templateable element.
- A template binding relationship (a relationship from the actual element to the parameterized element).
- A template parameter substitution (used to define actual parameters of the actual element that substitutes formal template parameters). Template parameter substitutions are created on the template binding relationship.



Example of Template

In this example, you can see the *GeneralArray* template class, which has the *ArraySize* and *ArrayType* template parameters, represented by the dashed rectangle in the upper-right corner of the class. These parameters represent the size and type of the general array.The attribute *GeneralArray::contents : ArrayType [0..ArraySize]* represents the *ArrayType* and the *ArraySize* template parameters. The *ArraySize* template parameter is of the Integer type, and its default value is 10. The *ArrayType* template parameter is of the Class type (it is not represented in the shape), and its default value is not specified. The default value of the template parameter is used if no actual value is supplied for the parameter in a binding. Typically, the parameter types are classifiers, but they can also be integers or other types.

The *GeneralArray* class and its template parameters create a template of the general array of undefined types. You can use this template in other concrete arrays as many times as you need.

How to use the template and how to create an actual element

To create an array of addresses, you would create the *AddressArray* class. Alone (without a template), the *AddressArray* class would have undefined types. A template binding is created from the *AddressArray* class to the *GeneralArray* class. Now the template binding connection points to the *GeneralArray* template class.

The template binding specifies template parameter substitutions, the actual values of parameters for the address array. In the example, we created the following actual values of the AdressArray class: the array size having the value 3 and the array of the *Address* type.

The output of these definitions results in the *AddressArray* class having the attribute *AddressArray::contents : Address [0..3]*.

Attributes for the actual element classes should be created manually. In the example, the *AddressArray::contents : Address [0..3]* attribute is not created automatically - you must create this attribute manually. Use these optional solutions to create these attributes in your model or not.

Working With Templates
Step by step instructions for modelling a template in the MagicDraw are described in Working With Templates.

You can format template parameter symbol properties in the **Symbol Properties** dialog.

You can specify template parameter properties in the Template Parameter Specification window and template binding properties in the Template Binding Specification window. In the specification window, you can find the description of each property. Descriptions are presented in the description area of the Specification window.

Related pages

- Model elements
- Auxiliary Diagram Symbols
- Diagramming