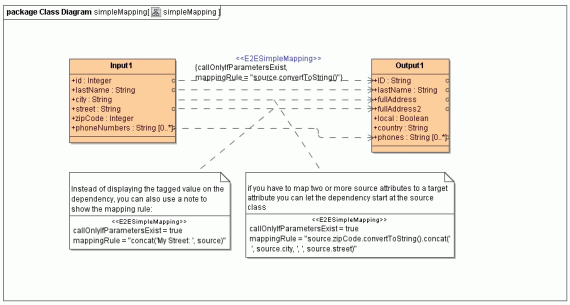
Simple Mapping of Attributes

Example File (Builder projectAdvanced Modeling/Mapping):

<your example path>\Advanced Modeling\Mapping\uml\mappingSimple.xml

The following figure shows a very simple case of mapping a class to another.



Defining the Mapping

The most trivial mapping is done by just drawing a dependency between two attributes. For example, **last Name** is identical in class **Input1** and **Output1**.

Although this situation is common, quite frequently we have the requirement that an attribute must be converted while being mapped. For instance, when **id** is being mapped to **ID** with a different data type. Or when **street** is being mapped to **fullAddress2** we prefix the string with `'My Street: '`.
This can be achieved by using the tagged value **mappingRule** on the <<E2ESimpleMapping>> stereotype. This tagged value may contain any valid action script statement. To refer the source or target attribute within the mapping rule, use the keyword source and target.

In many cases it makes no sense to apply an operation to optional input values if they are NULL. Therefore, if an operation has input parameters only that are optional at the source class, the operation is applied only, if at least one of the input values is not NULL. This is technically implemented by generating an implicit guard. This behavior can be overridden by setting the tagged value **callOnlyIfParametersExist** on the <<E2ESimpleMapping>> stereotype to false (default is true).

Caveat

Per default, **callOnlyIfParametersExist** is true, so defined **mapping rules** are only applied to existing parameters. The compiler will generate guard statements to **all** variables used in the action script in this case.

You need to be aware of this behavior when writing your mapping script. If you use nested if clauses that check for the existence of source parameters like e.g.

```
set target.field =
        if          source.fieldA.exists()         and
                 source.fieldA.normalizeSpaces().stringLength() > 0
        then source.fieldA.normalizeSpaces()
        else if   source.fieldB.exists()         and
                     source.fieldB.normalizeSpaces().stringLength() > 0
             then source.fieldB.normalizeSpaces()
[...]
```

this will result in the mapping only being performed if **all** source fields are set. Also, guard statements will be applied to local variables used the mapping script.

You can override this behavior setting **callOnlyIfParametersExist** to **false** on the mapping relation. Then, no guard statements will be generated but you need to take care for non-existing values by yourself.

On this Page:

Related Pages:

- Simple Mapping of Attributes
- Simple Mapping of Classes
- More Complex Mappings
- Mapping with a Mapping Handler
- Constraints

Related Documentation:

- The Interactive xUML Debugger

Invoking the Mapping

After defining the mapping in class diagrams, the mappings must be invoked in an activity diagram using a <<Mapping>> action.

> This action and the used mapping definitions must be kept in the same model, otherwise the compiler will complain.

This action then will internally translate the declarations defined in class diagrams into activity diagrams. In our simple example it generates the following set of action script statements that actually execute the mappings:

```
create output1;
set output1.ID =convertToString(input1.id);
set output1.lastName = input1.lastName;
set output1.fullAddress =concat( convertToString( input1.zipCode), ' ',
input1.city, ', ', input1.street);
set output1.fullAddress2 =concat('My Street: ', input1.street);
set output1.phones = input1.phoneNumbers;
```

The Interactive xUML Debugger allows you to debug all the generated activities.