Mapping with a Mapping Handler

The usage of the mapping handler is deprecated. Please use mapping dependencies and the context mapper instead:

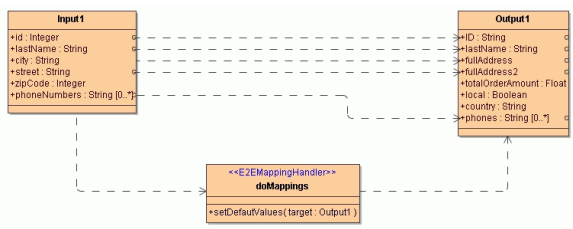
Example File (Builder projectAdvanced Modeling/Mapping):

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

Example File (Builder project Advanced Modeling/Mapping):

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

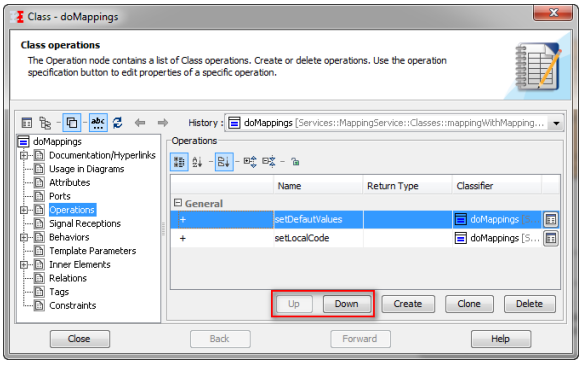
Up to now, we just mapped simple attributes using built-in operations. If it is necessary to execute self-defined operations, for example because of more complex mapping rules or to do a look-up for a code-value translation, you can use operations defined on a <<E2EMappingHandler>> class. This class intermediates the mapping between a source and a target class:



The operation **setDefaultValues** in the <<E2EMappingHandler>> example above is used to set some default values in class **Output1**. To reference the target attribute you have to use the keyword target here.
The allowed keywords are:

- self: refers to the current handler. **Note:** valid only if the operation is not static.
- source: refers to the source object
- target: refers to the target object

If <<Mapping>> action is invoked in an activity diagram for the above defined mapping handler, all operations on **doMappings** are applied to the input objects whereas all input and output parameter names must be found on the source respectively the target class. The order in which the mapping operations are executed depends on the order in which they are specified on the class. In the specification dialog of the mapping handler class this order can be rearranged if necessary.



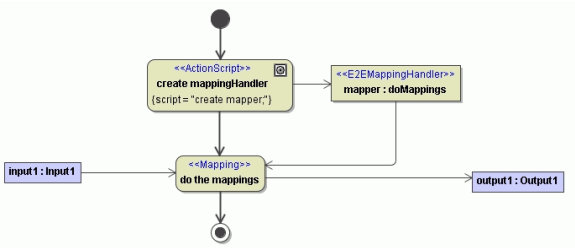
On this Page:

- Mapping Guards
- Iteration and Filtering

Related Pages:

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

If all mapping operations are static, no instance of **doMappings** must be given to the <<Mapping>> action. However, if the mapping operations are non-static, a object of type **doMappings** is required as input of the <<Mapping>> action, i.e.:

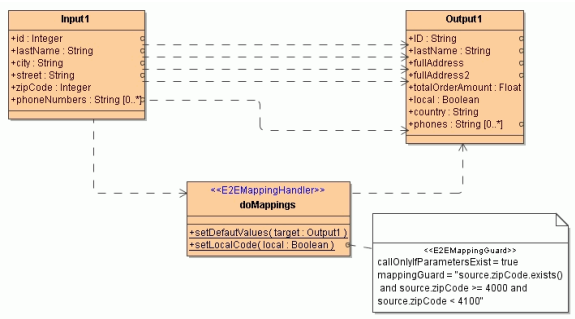


Since the mapping handler is handled like any other UML object, you can define member variables to hold contextual information when executing the mapping. Alternatively, if convenient, you can also define abstract or overridable operations that are resolved at runtime.

Mapping Guards

Up to now each operation has been applied unconditionally. If an operation shall be applied only if certain conditions apply use the tagged value **mappingGuard** on <<E2EMappingGuard>> to specify the condition.

Figure: Conditional Mapping Using Guards



The operation **setLocalCode** in the example above is only invoked if the condition zipCode between 4000 and 4099 is true.

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 <<E2EMappingGuard>> to false (default is true). Beware, the implicit guards are applied only if the operation has the stereotype <<E2EMappingGuard>>.

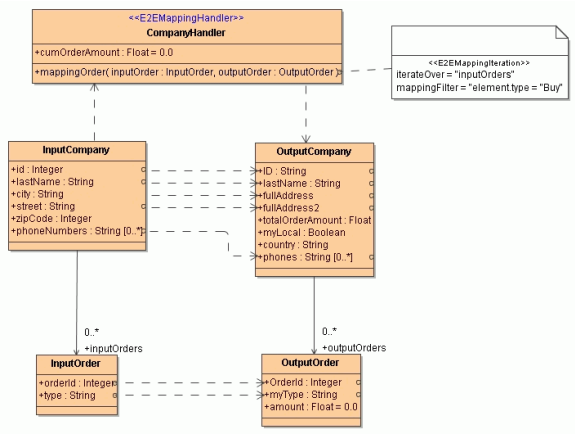
Iteration and Filtering

Example File (Builder projectAdvanced Modeling/Mapping):

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

In many situations we have to map not only single attributes but arrays. The following figure shows an <<E2EMappingHandler>> **CompanyHandler** containing the operation **mappingOrder**. This operation is applied to all elements of the **inputOrders** array because its stereotype is <<E2EMappingIteration>> having the tagged value **iterateOver** = "inputOrders".

Figure: Iterative Mapping As Defined in Class Diagrams



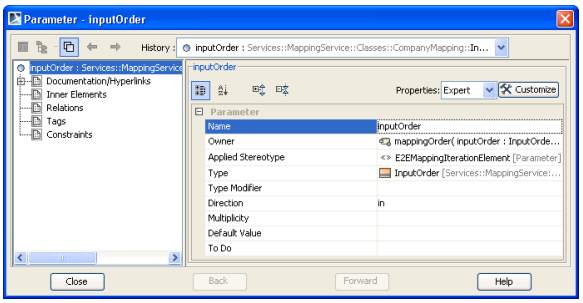
When iterating over arrays it is frequently helpful to filter the array before starting the iteration. Thus, you can use the tagged value **mappingFilter** on <<E2EMappingIteration>> to select a subset of the array. Within the filter expression you can use the following keywords to access values:

- self: refers to the current handler.

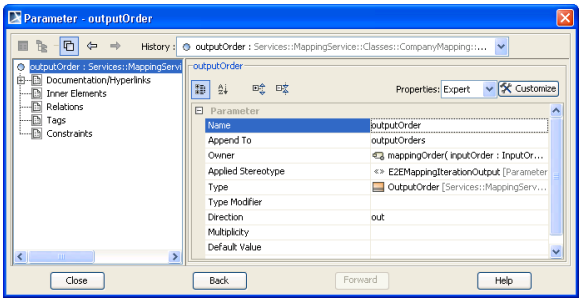
Note: Valid only if the operation is not static.

- source: refers to the source object
- element: refers the current iteration element

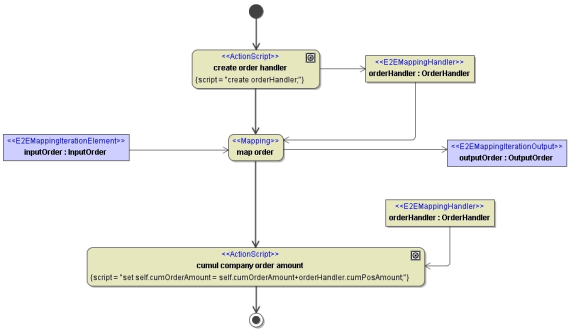
For instance, the example above selects only array elements where the order type is equal to 'Buy'. Each iterated element is put into the input parameter **inputOrder** that has the stereotype <<E2EMappingIterationElement>>.



All output elements generated while iterating are appended to the array **outputOrders**. This is defined by setting the stereotype <<E2EMappingIterationOutput>> on parameter **outputOrder**.



In the implementation of the mapping iteration **mappingOrder** the mapping specified between **InputOrder** and **OutputOrder** is invoked.



Additionally, in the example above we cumulate for every iteration the amount into the attribute **cumOrderAmount** of the <<E2EMappingHandler>> object. The attributes of the current handler can be referenced by **self**.

The aggregated amount can received from the <<E2EMappingHandler>> after the <<E2EMappingIteration>> was invoked.