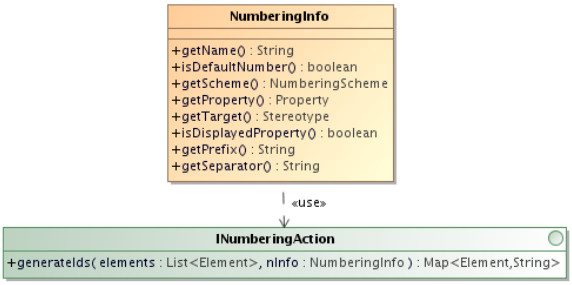
INumberingAction interface

What if we want to customize the numbering of elements even further? Let's say we need to number uml: Activities depending on their *getStructuredNode()* value. By implementing the *com.nomagic.magicdraw.autoid.INumberingAction* interface, it is possible to create the numbering that follow arbitrary conditions, this comes of course at the cost of increased complexity. The implementation must calculate the values for the numbers all by itself.

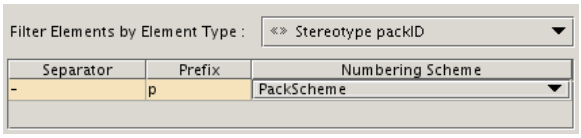
There are two parts to be explained that pertain to this interface. The actual interface that provides access to customized numbering and an immutable support class which encapsulates the values as defined in the *AutoNumber* and the associated *NumberingScheme* objects.



The *INumberingAction* Interface. The interface has one method, which is invoked whenever a new *Element* is created or when **Create** or **Renumber** buttons are clicked in the **Element Numbering** dialog. This applies of course only if the Element type equals the type for which the *AutoNumber* has been customized. So at the time when a new element is created, this method receives a *java.util.List* with a single entry. On the other hand, when the user navigates the **Element Numbering** dialog, the method receives all elements that are currently listed in the right partition of the dialog. This implies that all elements have the same owner. Since this will give access to the complete Model Tree, it is possible to collect necessary information from other parts of the model and incorporate it. This method must return a mapping of each element together with the value that was generated for it. There is no need to number every element that the method receives. The Numbering Framework will register the generated values internally and set the property value for each element.

The *com.nomagic.magicdraw.autoid.NumberingInfo* support object. This is an immutable Java Bean object which provides the implementer of the *INumberingAction* interface with all the information as defined in the *AutoNumber* and *NumberingScheme* objects:

- *getName()* returns the name of the *AutoNumber*.
- *isDefaultNumber()* returns the *AutoNumber.defaultNumber* value.
- *getScheme()* returns the *AutoNumber.numberingScheme* value, which is the *NumberingScheme* used to number this type of element.
- *getProperty()* returns the *AutoNumber.numberedProperty* value, which is the *Property* that will hold the value of the calculated number.
- *getTarget()* returns the *AutoNumber.getTarget()* value, which is the type of the elements to be numbered.
- *isDisplayedProperty()* returns true, if the value of the above *getProperty()* is displayed in the GUI. An element type can have more than one numbered *Property*, but only one can be displayed on symbols or in the Element Tree.
- *getPrefix()* returns the prefix as modified by the user in the **Element Numbering** dialog, which can be different from the *AutoNumber.getPrefix()* value (see the following figure).
- *getSeparator()* returns the separator as modified by the user in the **Element Numbering** dialog. In the implemented Numbering Framework, this value will override all the *NumberParts* of a type separator (see the following figure).



For more information on how to create Numbering customizations with «AutoNumber», «NumberingScheme», and «NumberPart» elements, see Working with Generic Numbering Mechanism.

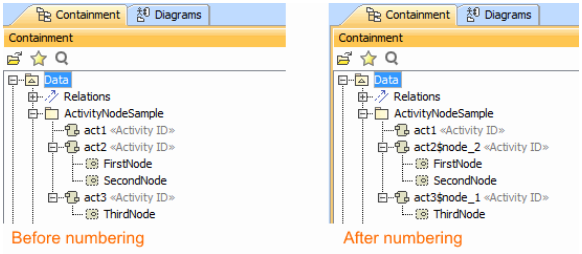
Constraints

A *NumberingScheme* that references an expression which implements this interface, can have one and only one *NumberPart*. Namely a *NumberPart* that references an *INumberingAction*.

Example #1

We shall now proceed to show a Java class that generates numbers for UML activities depending on their *getStructuredNode()* value. Since this class implements the *INumberingAction* interface, it should be set as the expression in the only *NumberPart* of this *NumberingScheme*.

We have created a few activities inside a sample *Package*. With the creation of each new activity, an ID is generated and assigned to it. We then add some *StructuredActivityNodes* into two of the given activities. Now, in order to assign *Numbers* to these *Activities* that includes the number of *StructuredActivityNodes* children, we have to open the **Element Numbering** dialog and renumber the Activities. This will generate new numbers that reflect the ownership relation.



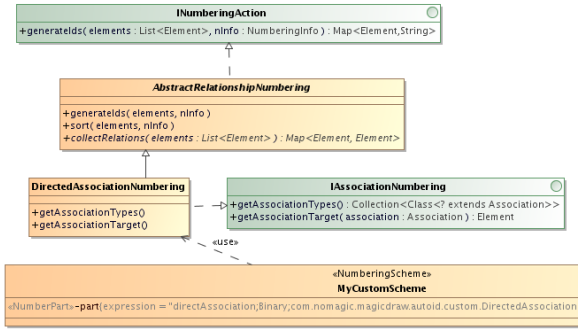
Let's have a look at the source code. This is a very simple example, which differentiates between receiving one or many elements, and calculates the values accordingly. This class can be found in *com.nomagic.magicdraw.autoid.sample*.

```
    public class ActivityNodeNumbering implements INumberingAction {
        @Override
        public Map<Element, String> generateIds(List<Element> elements,
NumberingInfo nInfo) {
            Map<Element, String> idMap = new HashMap<Element, String>();
            if (! elements.isEmpty()) {

                // the NumberingInfo object contains the values set in the
customization
                String baseId = nInfo.getPrefix() + nInfo.getSeparator();

                // when creating an Activity, this will be called with a
single element
                if (elements.size() == 1) {
                    Element e = elements.iterator().next();
                    Class<?>[] types = new Class<?>[]{Activity.class};
                    Collection<? extends Element> acts = ModelHelper.
getElementsOfType(e.getOwner(), types, false, true);
                    String id = baseId + (acts.size());
                    idMap.put(e, id);


                    // when renumbering in the dialog this will be called
with multiple elements
                } else {
                    int counter = 1;

                    // we sort so that renumber will get the same order
                    // normally this would ask for a custom comparator
class
                    // but it is left out for brevity's sake
                    Collections.sort(elements);
                    for (Element e : elements) {
                        if (e instanceof Activity) {
                            Activity act = (Activity) e;
                            String id = baseId + counter;
                            counter ++;
                            Collection<StructuredActivityNode> nodes = act.
getStructuredNode();

                            // we simply attach the suffix '$node_' and
the node count
                            if (nodes != null && !nodes.isEmpty()) {
                                id += "$node_" + nodes.size();
                            }
                            idMap.put(e, id);
                        }
                    }
                }
            }
            // returning the mapping of elements and ids, so that the
framework can register them
            return idMap;
        }
    }
```

Example #2

A more complicated example can be found the OpenAPI packages *com.nomagic.magicdraw.autoid* and *com.nomagic.magicdraw.autoid.custo*. It shows an implementation that creates numbers for elements connected through UML Relationships. The UML diagram shows the *com.nomagic.magicdraw.autoid.custom.DirectedAssociationNumbering* class, which is customized to number element types that are connected by «DirectedAssociation» links.

The interested Reader should consult the source code, in order to learn how such an algorithm has been implemented.



Example of relationship numbering (fragment)