# Editor Actions

## Actions

MPS editor has quite sensible defaults in completion actions, node creation policy. But when you want to customize them, you have to work with the actions language. This language is also used to define Left and Right Transform actions (RT/LT-actions for short), which allows editing binary operations in a text-like way.

## Substitute Actions

Substitute actions are actions which are available when you press Ctrl+Space in the editor. MPS has the following default behavior for them:

- If your selection is inside of a position which allows concept A, then all enabled subconcepts of A will be available in the completion menu.
- All abstract concepts are excluded
- All concepts with dontSubstituteByDefault property are removed
- All concepts for which the 'can be a child' constraint returns false are excluded
- All concepts for which the 'can be a parent' constraint of a parent node returns false are excluded
- If a concept has a 1:1 reference, then it is not added to the completion menu. Instead, an item is added for each element of a scope for that reference. We use a name smart reference for such items.

When you want to customize this behavior, you have to create a node substitute actions root. Inside it you can create substitute actions builders. Each of them has a substitute node concept, where you type a name of a concept which you want to substitute. It has a condition; when this condition is satisfied, your actions will be added to the completion menu. Also it has an actions part where action behavior is specified. Each action has an associated concept which can be used for action filtering (see remove by condition).

### Add concept

If a concept isn't available because of default settings, you can add it with the 'add concept' construct.

### Remove defaults

Use this construct if you want to completely override defaults. It removes all default actions and adds only those actions which are specified in the actions language.

### Remove By Condition

Use this construct if you want to remove only some of the actions. It can be useful when you extend a language and want to remove some of its actions in a particular context.

### Custom items

If you aren't satisfied with default items, you can create your own. In this case you can override everything: output concept, matching text, description text, icon, and behavior on item invocation. When you create custom items, you have to specify output concept so it can be used to filter out your actions from extending language.

#### Simple

Simple item adds one item to substitute menu. You can specify all the properties of substitute action (matching text, description, icon etc). It can be useful for entering literals (boolean, integer, float, string, char).

#### Parametrized

This concept allows you to create an item in substitute menu based on a query. A query should return a list of something. For example, if you want to create completion for files in a directory, you can use it. This concept is similar to simple item but it has additional parameter parameterObject in all of its blocks.

### Wrapper

Sometimes we want to add all the completion items from a context of one type of concepts into a context of other concepts. Let's consider a couple of examples from baseLanguage. For example, we want to see all available items for expression in statement's context, since we may wrap all of them in ExpressionStatement. Or we can add all items of Type's completion menu, since we can create a local variable declaration. Wrapper blocks has a concept from whose context we want to add completion items. It also has a wrapper block with a nodeToWrap parameter, which the author of wrapper block should wrap.

### Concepts Menu

Sometimes you want to add items for subconcepts of a particular item but want to override its handler. Concepts menu block allows you to do so.

### Generic

If the above actions are not suitable, you can resort to generic item. It has a block which returns a list of INodeSubstitueAction items.

## Side Transform Actions

When you edit code in a text editor, you can type it either from left to right:

> 1 <caret> press +
> 1+<caret> press 2
> 1+2<caret>

or from right to left

> <caret>1 press 2
> 2<caret>1 press +
> 2+<caret>1

In order to emulate this behavior, MPS has side transform actions: left and right transforms. They allow you to create actions which will be available when you type on left or right part of your cell. For example, in MPS you can do the following:

> 1<caret> press + (red cell with + inside appears)
> 1+<caret> press 2 (red cell disappear)
> 1+2<caret>

or the following:

> <caret>1 press + (red cell with + inside appears)
> +<caret>1 press 2 (red cell disappear)
> 2<caret>+1

The first case is called right transform. The second case is called left transform.

In order to create transformation actions you have to create transform menu actions root. Inside it, you can create transform action builders. You can specify the following:

- whether it is left or right transform
- a concept which instance you want to transform
- a condition which defines where your actions will be applicable

## Add custom items

Custom items is similar to its counterpart in substitute actions part of a language. It allows you to add either one or many items to menu. Let's consider them in detail.

### Simple item

Simple item adds an item with matching text, description, icon, and substitute handler.

### Parametrized item.

Parametrized item adds a group of items based on a query which returning a list of something. It's similar to simple item but has additional parameterObject parameter in every block.

## Add concept

Add concept adds an item for every non-abstract subconcept of a specified concept. This item has a handler block where you can replace sourceNode with a newly created node. For example, this is useful when you want to create a transformation for each subconcept of a BinaryOperation, such as +, -, *, or /. The code for replacing sourceNode is the same in each of these cases - the only difference is the resulting concept.

## Include transform for

This construct allows you to include all the right transform action associated with a particular node.

## Remove By Condition

This allows you to filter actions in case of a language extension.

## Remove Concept

This allows you remove all the actions associated with a particular concept.

# Node Factories

When you have a baseLanguage expression selected, press Ctrl+Space on it and choose (expr). You will have your expressions surrounded by parenthesis. Node factories allows you to implement this and similar functionality by customizing instantiation of a new node. In order to create node factory, you first have to create a new node factories root node. Inside of this root you can create node factories for concepts. Each node factory consists of node creation block which has the following parameters: newNode (created node), sampleNode (currently substituted node; can be null), enclosing node (a node which will be a parent of newNode), and a model.

Previous Next