Editor language generation API

The editor language is supposed to be extended by numerous MPS users, so we designed the generator for the Editor language for ease of use - straightforward templates, human-readable generated code and use of meta-information more at the generation-time than at run-time. If any of your languages extend the editor language in order to provide new cell types, this document is for you.

API: EditorCell contract

The contract of EditorCell.setBig()/.getBig() methods was slightly changed. Please check the javadoc for details.

API: EditorCellFactory is now available only within UpdateSession

We made the EditorCellFactory instance controlled by the current UpdateSession. In the same time we put some caches inside the EditorCellFactory implementation, making the editor building process faster in some situations. The EditorContext.getCellFa ctory() method was deprecated and will be removed in the next release.

Language Runtime: AbstractEditorBuilder

AbstractEditorBuilder runtime class was introduced and should be used as a common super-class of any classes containing cell factory methods. This class implements common utility methods and provides access to generic contextual parameters of editor cell creation process like:

- editorContext
- node
- CellFactory
- UpdateSession

AbstractEditorBuilder is used to capture some context of cell creation process and execute consequent cell factory methods within this context.

Generator: EditorBuilder classes

A separate sub-class of AbstractEditorBuilder class will be generated now as a root class for each of available editor declaration hierarchies:

- ConceptEditorDeclaration.cellModel
- ConceptEditorDeclaration.inspectedCellModel
- EditorComponentDeclaration
- InlineEditorComponent

The MPS editor generator will continue creating classes, implementing ConceptEditor & ConceptEditorComponent. These classes were used earlier as containers for cell factory methods. In the new version of MPS these classes are used as descriptors providing access to the contextual hints information & instantiating actual EditorBuilders. Descriptor classes may be cached by the EditorCellFactory implementation.

Contextual parameters available for cell builders

The code, generated as a part of AbstractEditorBuilder sub-classes may access contextual parameters by using existing methods of AbstractEditorBuilder class. In addition to that, all available meta-information is used to generate private fields with more specific types than those available in the method signatures of AbstractEditorBuilder. For now each sub-class of AbstractE ditorBuilder will hold private node<TheConcept> myNode field, where TheConcept is the actual concept associated with this Ab stractEditorBuilder. This means that any cell factory method may use such a private field in order to get typed access to the contextual node and directly access available properties, links and other information from the contextual node using s-model language.

CellFactoryContextClass

CellFactoryContextClass is a handy utility class providing necessary context for templates, generating the code included into one of the generated AbstractEditorBuilder sub-classes. By using this class as a contextual class template authors will automatically obtain all available methods and fields, the code generation environment will be supported by MPS platform, and so it's not necessary to reconstruct it for each and every editor template anymore. At the same time, CellFactoryContextClass c an be used as a marker interface highlighting templates, which will generate code for one of the EditorBuilders, simplifying the process of locating such code & supporting it in the future.

GenericCellCreationContext

The GenericCellCreationContext interface provides limited subset of contextual information, which is always available for the code called either as a part of EditorBuilders or from a separate class, executed as a part of cell creation (editor update) process. This interface should be used as a template context instead of CellFactoryContextClass in those cases, when template authors are going to reuse the same template across the EditorBuilders generation process and some other places. For example, query methods which may be generated either inside EditorBuilders or within some style class.

New signature for createCell() methods

In the previous version of MPS, the cell factory methods were always generated with two additional parameters specifying the context of cell creation: EditorContext & node<>. From now on it's not necessary to specify these parameters any more - the generated code can always access this information (as well as any other contextual info) by calling methods from the containing EditorBuilder class. The new editor generator will generate cell factory methods without any parameters.

Automatic migration script for new createCell() methods

For compatibility with the existing generators, we provide a migration script patching available templates and introducing the new createCell() methods, which delegate to the old ones (with the two additional parameters) as a fallback. We recommend to execute this script first and then check all modifications and verify, if the modified generator still works correctly. The provided automatic migration supports only most frequent situations, so in some specific cases you may need to manually modify your generator in order to make it work again. The template, which generates the compatibility methods is called template_cellFactoryCompatibility. If you later modify your generator to generate directly the new createCellMethods, you should remove any calls to template_cellFactoryCompatibility. We do recommend to review all existing generators & patch obsolete templates generating the legacy createCell(...) methods in the scope of the current MPS release - we are going to drop the compatibility template in the next version.

Mapping labels

Several mapping labels were introduced into the Editor generator (MAPPING_main) and may be used to simplify code generation:

cellFactory.class.concept

cellFactory.class.concept : ConceptEditorDeclaration -> ClassConcept

This label expose a java class, generated for the EditorBuilder of ConceptEditorDeclaration.cellModel

cellFactory.class.inspector

cellFactory.class.inspector : ConceptEditorDeclaration -> ClassConcept

This label exposes a java class, generated for the EditorBuilder of ConceptEditorDeclaration.inspectedCellModel

cellFactory.class.component

cellFactory.class.component : EditorComponentDeclaration -> ClassConcept

This label exposes a java class, generated for the EditorBuilder of EditorComponentDeclaration

cellFactory.constructor

cellFactory.constructor : EditorCellModel -> ConstructorDeclaration

Used to mark the constructor of the generated EditorBuilder class.

cellFactory.factoryMethod

cellFactory.factoryMethod : EditorCellModel -> InstanceMethodDeclaration

The replacement for obsolete cellFactoryMethod label, containing the new cellFactory methods. This label should be used instead of cellFactoryMethod at the moment of modification of existing templates making them generating new cellFactorymethods.

generated.constructor

generated.constructor : <no input concept> -> ConstructorDeclaration

This label may be used together with existing generatedClass one to mark generated constructor instances. This label may be used to avoid the ugly code for locating first constructor instance inside node<ClassConcept>, returned from the generatedClas s mapping.

CellLayoutConstructor switch introduced

This template switch is used to instantiate proper cell layout while creating a collection cell. The previously used static createxx x() methods inside EditorCell_Collection class have been deprecated and will be removed.

New generator for RefCellCellProvider sub-classes

The generator for CellModel_RefCell has been modified. The newly generated anonymous inner classes for RefCellCellProvider d o not use the logic located inside RefCellCellProvider.createRefCell() runtime method. The meta-information, available at generation-time, are used in oder to create complete content of this method. If you do generate sub-classes of RefCellCellProvi der within your generators, you should consider reviewing such places and aligning your templates with the templates from MPS.

InlineCellProvider replaced with EditorBuilder sub-class

InlineCellProvider is not being used by the MPS generator anymore. MPS uses the generated sub-classes of AbstractEditorBuild er instead. Nevertheless, we modified some constraints inside InlineCellProvider in order to make the lifecycle more transparent. We recommend to check the javadoc for InlineCellProvider, if you are still using it.

Editor Styles generator

A separate static inner class will be generated for each entry inside StyleSheet & StyleKeyPack instances. The provided applySt yleClass template may be used to properly instantiate & call the new Style classes. Legacy static .applyxxx() methods should be removed in the next release.

StyleClassItem constraints modification

We removed the canBeChild constraints from the StyleClassItem concept. These constraints were replaced with canBeParent co nstraints of the node, containing the StyleClassItem. In addition to that isApplicableToCell(node<EditorCellModel> cellModel) b ehaviour method has been deprecated and is not used anymore. Instead we have introduced the following methods:

- isApplicableToCellConcept()
- isApplicableForLayout()
- isApplicableInLayout()

We recommend you to check the javadoc of StyleClassItem behavior methods, if you are implementing any custom StyleClassI tem in your language.