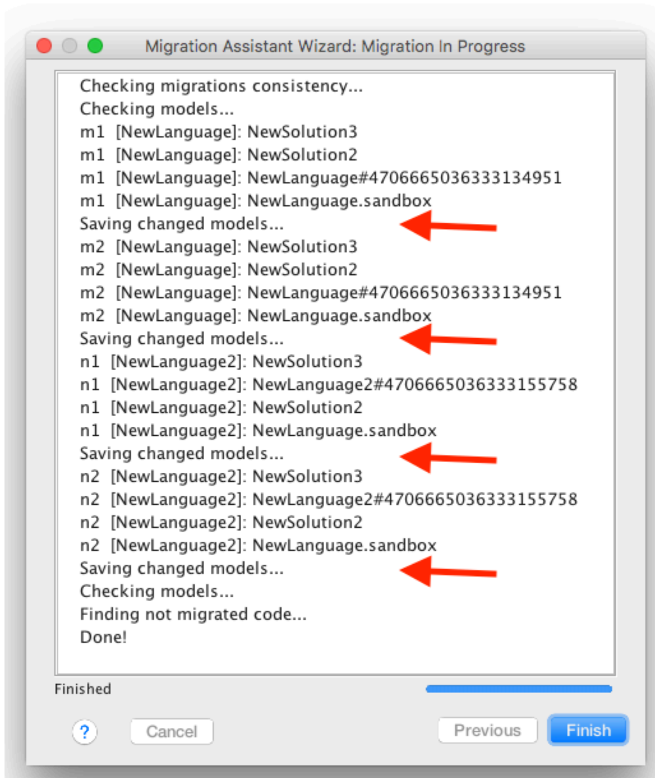
# What's new in MPS 2017.1


- EAP1
  - Changes made by migrations in Local History view
  - Migrations for conceptNode//, .asNode, .conceptNode etc
  - Modifications in the generator of the editor language
    - API: EditorCell contract
    - API: EditorCellFactory is now available only within UpdateSession
    - Language Runtime: AbstractEditorBuilder
    - Generator: EditorBuilder classes
    - Contextual parameters available for cell builders
    - CellFactoryContextClass
    - GenericCellCreationContext
    - New signature for createCell() methods
    - Automatic migration script for new createCell() methods
    - Mapping labels
      - cellFactory.class.concept
      - cellFactory.class.inspector
      - cellFactory.class.component
      - cellFactory.constructor
      - cellFactory.factoryMethod
      - generated.constructor
    - CellLayoutConstructor switch introduced
    - New generator for RefCellCellProvider sub-classes
    - InlineCellProvider replaced with EditorBuilder sub-class
    - Editor Styles generator
    - StyleClassItem constraints modification
  - HTTP Support Plugin
    - Node URLs
    - YouTrack and TeamCity Integration
    - Built-in server extensions
  - Fully-compiled languages
- EAP2
  - Support of the substitute and side transform actions in the aspect actions is dropped.
- RC1
  - Character Accent Menu supported in editor on Mac OS X
- RC2
  - Module Cloning supported
  - Cross-model generation support (experimental/work-in-progress functionality)
  - Context objects in TextGen
  - New handy operation in bl.collections
  - Find Usages for Mapping Configurations
  - Bootstrap Dependency in a Language Accessory model


# EAP1

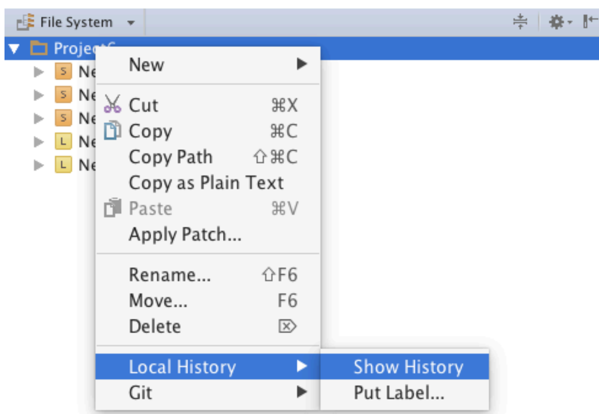
## Changes made by migrations in Local History view

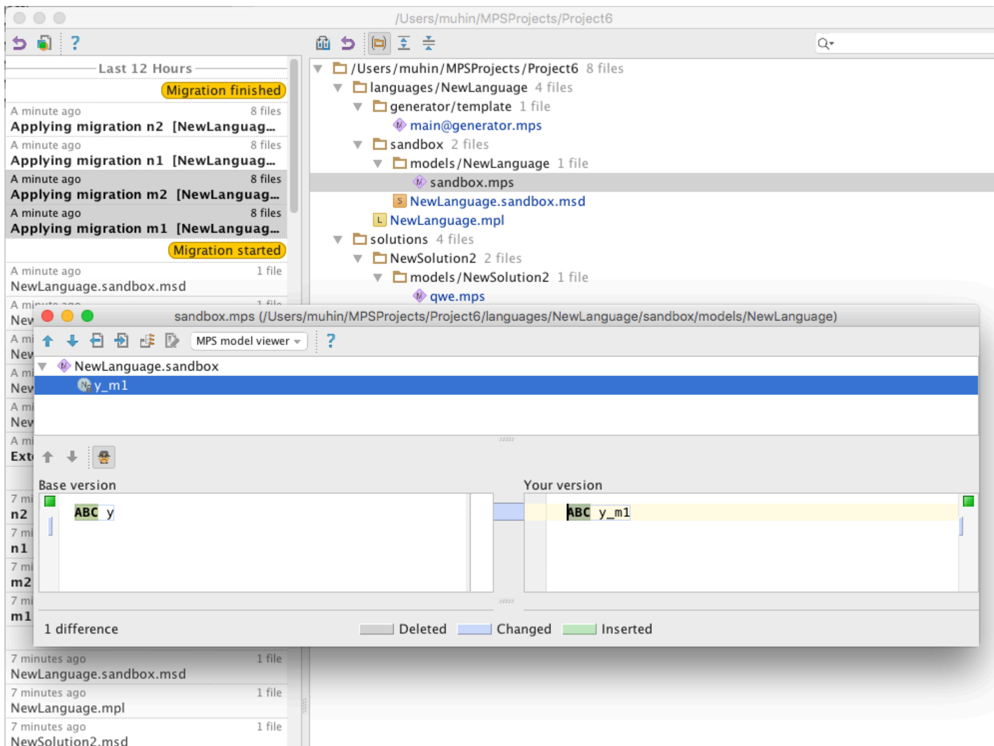
Migrations now cooperate with the Local History functionality. After running migrations, it's possible to see all the changes made to the project by each migration.




Open the Local History view for the project's folder, a module or a model, select any two changes and press Ctrl + D to see the difference.

It's also possible to revert a change or a group of those from the Local History view and the Diff dialogs






## Migrations for conceptNode//, .asNode, .conceptNode etc

//todo

## Modifications in the generator of the editor language

We decided to significantly re-design the generator of the editor language in this milestone. The editor language is supposed to be extended by the MPS users, so it makes sense to describe those changes we made in a single place. The new generator will simplify templates, make the generated code more human-readable and use more meta-information at the generation-time instead of the run-time. This item is relevant to those, who have developed languages extending the editor language. For projects not extending the editor language these changes should be transparent - you can just regenerate all editors in order to update your code.

If any of your languages extends the editor language in order to provide new cell types, we recommend you to carefully read through the following list of changes in order to be able to update your templates manually in cases, when it is required.

### 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.getCellFactory() 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_cellFacto ryCompatibility. 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 cellFactory met hods.

### 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 applyStyleClass 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 constraints of the node, containing the StyleClassItem. In addition to that isApplicableToCell(node<EditorCellModel> cellModel) behaviour 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 StyleClassItem in your language.

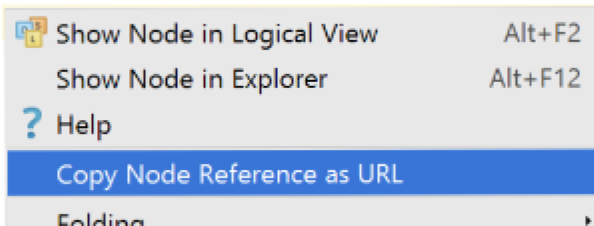
## HTTP Support Plugin

We introduced a new plugin that provides:

- co-operating via node URLs
- integration with YouTrack and TeamCity services
- a DSL for defining custom extensions to the IDEA Platform built-in server

### Node URLs

Now you can create URL references to your code via the context menu. The created URL will be copied to the clipboard and then can be pasted wherever you want. On clicking it, MPS will handle it and open the referenced code.




If you want to get a URL programmatically, you should use the .getURL operation defined in jetbrains.mps.ide.httpsupport language.

### YouTrack and TeamCity Integration

MPS listens for requests that come from YouTrack and TeamCity. On clicking the 'Open in IDE' button in a browser, MPS will open the requested file. Moreover, if you are trying to open a generated file, MPS will open its sources in a proper location.

### Built-in server extensions

The features above are implemented using the IDEA Platform built-in server. If you have any other necessities to handle HTTP requests in the IDE, you can define an extension to it server via jetbrains.mps.ide.httpsupport language. Note that the defined extensions should be placed in a plugin solution. See Plugin for more information

NodeOpener ×

```
request handler NodeOpener {
  query prefix: /node

  query parameters:
    ref     converter: default(SNodeReference) required
    project converter: default(Project)

  handle always

  handle(request)->void throws Exception {
    if (project != null) {
      project.getModelAccess().runWriteInEDT({ => HandlerUtil.openNode(request, project, ref); });
    } else {
      error "No project is available.";
      request.send response (HandlerUtil.FAILURE_STREAM : image/gif);
    }
  }
}
```

## Fully-compiled languages

//todo [MM]

work in progress a the moment. The idea behind this capability is to relax MPS's bond to the language declaration models. Namely, use of node.concept.getDeclarationNode as ConceptDeclaration assumes there's a structure model available. This might, however, not be always true, for example if there's a alternative model that declares concepts (or metamodel elements, in general)

Migrations to run:

- MigrateConceptFunctionParameters
- MigrateScopeProviders
- ??some constraints migration (radimir)
- MigrateCanBeCoerced

# EAP2

## Support of the substitute and side transform actions in the aspect actions is dropped.

Substitute and side transform actions from the actions aspect are no longer supported. Transformation and substitute menus should be used instead.

All commented out side transform and substitute actions will be removed by migration.

Corresponding commented out editor style SideTransformAnchorTagStyleClassItem will also be removed.

Instead of using SideTransformAnchorTagStyleClassItem in the specific cell one can attach the transformation menu with side transform section to that cell.

The instances of CellMenuPart_ApplySideTransforms will be removed as well.

CellMenuPart_ApplySideTransforms was designed to include items of the side-transform menu to the completion menu of the specific cell.

With the transformation menu language one can describe the same behaviour. She can attach the transformation menu with co mpletion section to the cell. That section will contain the with include menu part with the specified location of side transform. So the items of side-transform menu will be included to the completion.

You can find the example at the FieldDeclaration_ApplySideTransforms menu:

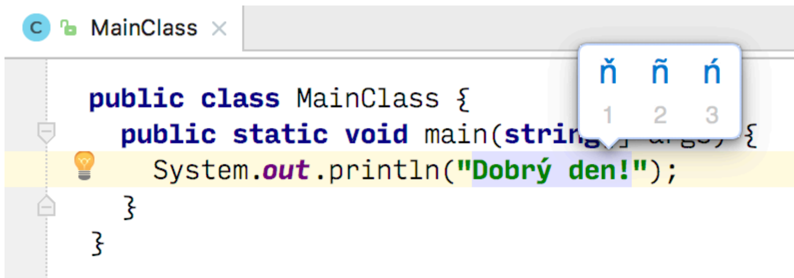
```
-----------
section(completion) {
  include default menu for FieldDeclaration location { side transform : right
}
```

Location of the included menu can be specified with the intention "Specify Location".

# RC1

## Character Accent Menu supported in editor on Mac OS X

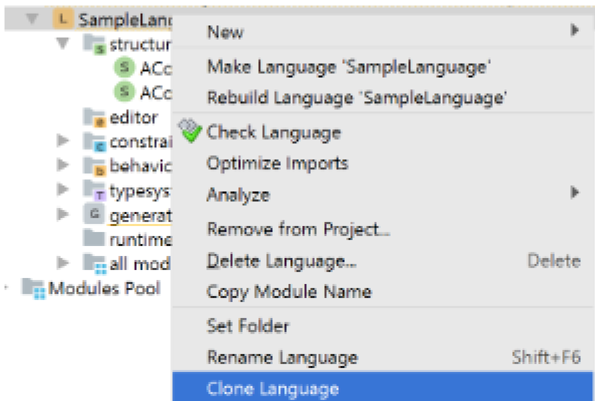
Same as in all other Mac OS X applications, in MPS editor you may now press & hold a button in oder to use Character Accent Popup for entering specific characters.




# RC2

## Module Cloning supported

We introduced a new action that provides module cloning facility. To clone a module you should select it in the project menu and click on `Clone Solution/Language` action in the context menu. You can see full documentation on clone module action her e.




## Cross-model generation support (experimental/work-in-progress functionality)

New language features in j.m.lang.generator.plan to address plan extensibility story:

- 'apply with extended' statement. Takes specified generators and those that extend these for the languages utilized in a model, and apply them as a single step. This mechanism allows plan designer to accommodate possible extensions to its languages/generators. It's expected that MPS would additionally respect priorities between involved mapping configuration in the upcoming releases.
- 'declare checkpoint <checkpoint>' and 'synchronize with <checkpoint>' statements. To share synchronization points between different plans, we captured different aspects of a checkpoint with distinct statements. Use 'declare checkpoint' to specify a token plans could share. Regular 'checkpoint <checkpoint>' statement may reference a checkpoint declared elsewhere (it's still possible to declare checkpoint in-place). This statement within a plan records/persists state of transformed model under designated checkpoint. And the last one, 'synchronize with <checkpoint>', tells generation

plans to look up target nodes in a persisted models of specified checkpoint. This statement doesn't introduce any new state (model being tranformed is not persisted for checkpoint we synchronize with) and always references a checkpoint declared elsewhere.

```
Plan1

apply   with extended
  Generator  jetbrains.mps.baseLanguage.checkedDots#4549711462050000009
  Generator  jetbrains.mps.baseLanguage#1129914002933
synchronize with  MyCheckpoint (of Plan2)
checkpoint XXX
```

```
Plan2

checkpoint YYY
checkpoint MyCheckpoint
declare checkpoint PureDeclaration
```

- 

Checkpoint models are denoted with a stereotype that matches name of a checkpoint the model has been created with. Models are persisted along with generated sources using naming scheme <plan-name>-<checkpoint name>.

## Context objects in TextGen

Although rarely needed, functionality to keep context information during M2T transformation of a text unit might be vital for certain scenarios (like BaseLanguage, where TextGen has to track model imports and qualified class names). In previous MPS versions, this has been approached with combersome and low-level code utilizing `buffer` object. Now, it's possible to specify necessary object as part of concept's textgen specification. At the moment, regular java class (with a no-arg or a single-arg constructor that takes concept instance) are supported as context objects.

Declare a context object and bind it to an accessor:

```
BaseLanguageTextGen  Plan1  ClassCreator_TextGen  ClassConcept_TextGen  Unit
    return importEntry.getName();
  }

  protected void appendClassName(string packageName, string fqName, node<> conte
    append ${getClassName(packageName, fqName, contextNode)};
  }

  ClassifierUnitContext : ctx  new  ClassifierUnitContext
}
```

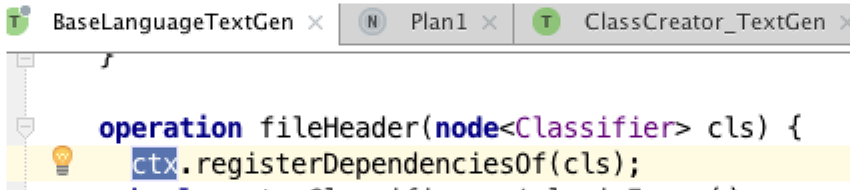
```
ClassifierUnitContext
/**
  Common context for any Classifier TextUnit (top-level), manages imports for sho
  to facilitate further creation of bl dependencies file
 */
public class ClassifierUnitContext implements RootDependencies.Source {
  private final node<Classifier> myClassifierNode;
  private final ImportsContext myImports;
  private final HashSet<String> myDepends;
  private final HashSet<String> myExtends;

  public ClassifierUnitContext(node<Classifier> topClassifierNode) {
```

Associate with a concept's textgen:

```
text gen component for concept ClassConcept {
file name : <Node.name>
extension : (node)->string {
  "java";
}
encoding : utf-8
text layout : Initial text area BODY
              HEADER
                << ... >>
              IMPORTS
                << ... >>
              SEPARATOR
                << ... >>
              BODY
                << ... >>
context objects : BaseLanguageTextGen.ctx : ClassifierUnitContext
```

Reference from code as a regular variable:




## New handy operation in bl.collections

To filter out null elements of a sequence, verbosity of `seq.where(it => it != null)` could be addressed with `seq.withoutNull`.

## Find Usages for Mapping Configurations

It's been a tedious work to find out what priority rules use given `MappingConfiguration`. Now, Find Usages looks up occurrences of a MC and shows them inside stubs for project modules:

BaseLanguageTextGen × typeof_NotNullOperation × GenMain × Plan1 × ClassCreator_Te

```
mapping configuration GenMain
top-priority group    true


mapping labels:
  label QueriesGenerated : <no input concept> -> ClassConcept

exports:
  Do not use, custom generation plans with regular mapping labels supersede Export

parameters:
```

jetbrains.mps.lang.generator ×

Warning: the node is in a read-only model. Your changes won't be saved

Text: all g   Match Case   Regex   Words

```
depends on generators:
  << ... >>

priority rule(
  generator/jetbrains.mps.lang.generator#1167163152317/ -> all local
  apply before or together (<=)
  all global
)
```

Usages

- Searched nodes
  - GenMain
- 2 usages found
  - jetbrains.mps.lang.project.modules (2)
    - module.jetbrains.mps.lang.generator@project_stub (2)
      - jetbrains.mps.lang.generator (2)
        - jetbrains.mps.lang.generator#1167163152317(role: *generator*, in: *jetbrains.mps.la*
          - all local(role: *innerRef*, in: *MappingConfigExternalRef*)
          - all global(role: *right*, in: *MappingPriorityRule*)

## Bootstrap Dependency in a Language Accessory model

MPS now treats accessory model that uses its own language as an error if the model could be generated ('Do not generate' flag not set). It's impossible to generate such model unless the language is generated, and as long as the model is part of the language, MPS is stuck. Previous MPS versions allow this scenario by silently assuming language has no generators (therefore, nothing would be generated), now MPS wants Language Designer to be explicit about own intentions and to denote model as 'Do not generate'.

If an accessory model doesn't use its hosting language, MPS still warns about generatable model as the primary scenario for accessory model is to supply nodes, not generated code.