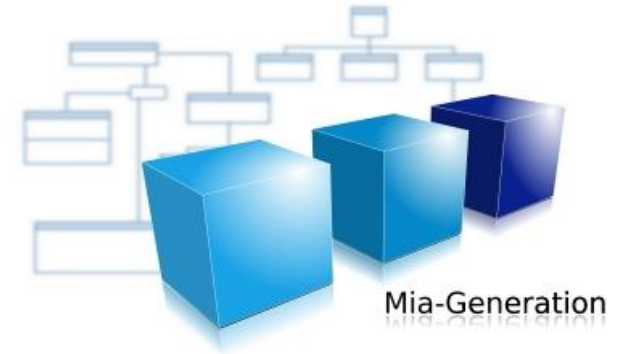


Génération avec Mia-Studio 9

Formation

1.0



A close-up photograph of a human eye with a light blue iris, looking slightly to the left. The eye is framed by dark eyelashes and skin.

Sommaire

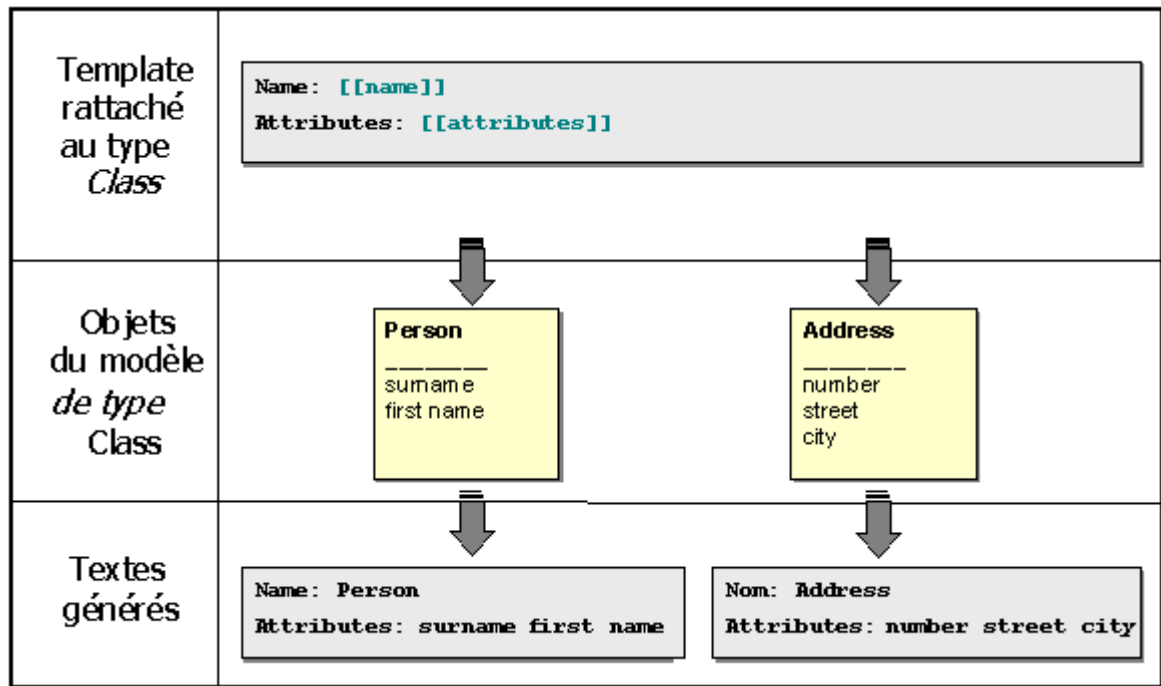
- Principes et concepts de base
 - Importation & consultation des modèles
 - Gestion et définition des scripts
 - Les variables et méthodes des scripts Java
 - Navigation dans les scripts
 - Scenario & Configuration de génération
 - Edition Developer & Integration IDE
 - Notions avancées

La suite Mia-Studio

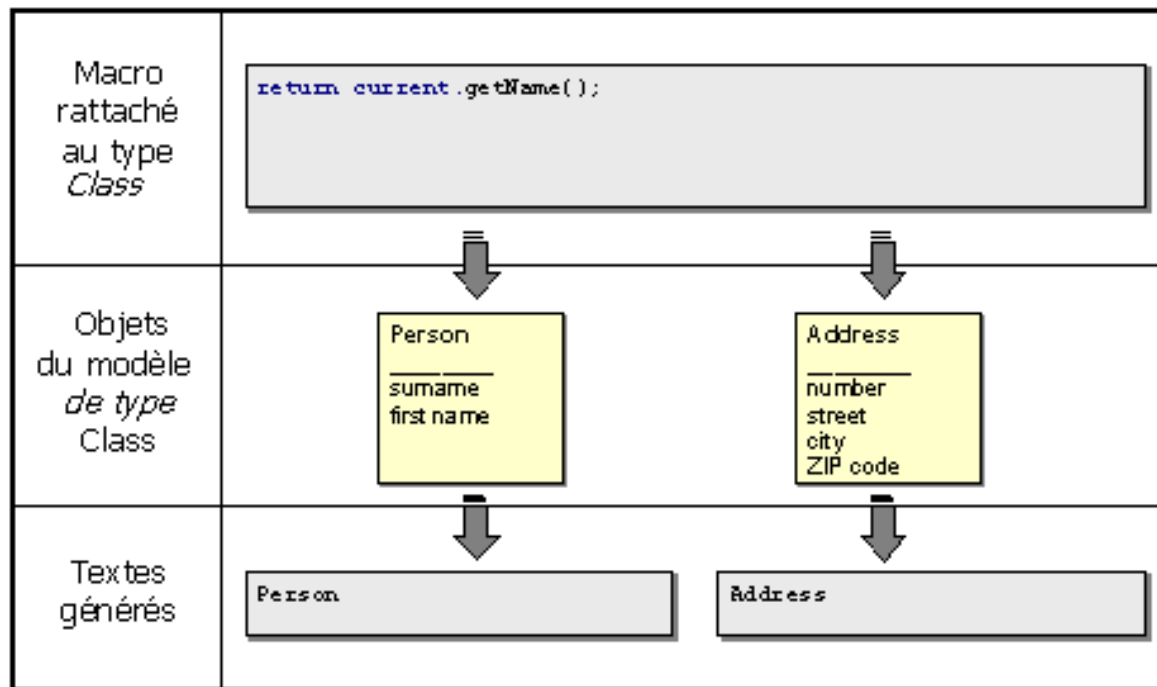


- Les règles de génération sont exprimées à travers des scripts évalués sur des objets d 'un modèle
 - 2 familles de scripts :
 - Template (WYSIWYG)
 - Code Java

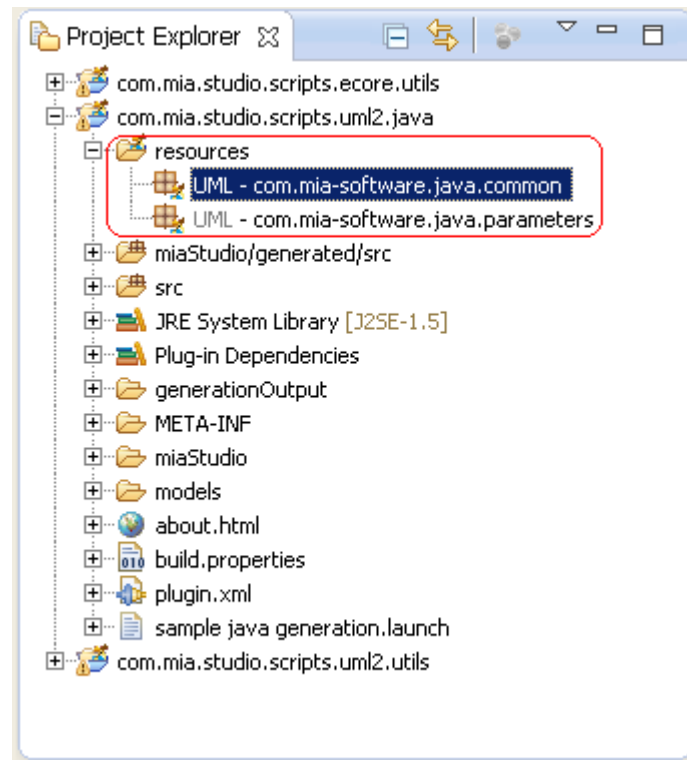
- Les templates



- Les scripts Java



- **Projet Mia-Studio de génération**
 - Un projet regroupe
 - Des *Package*
 - ensemble de scripts de génération
 - Des *Scénario*
 - descriptions d 'enchaînements de générations
 - Des *Profil*
 - personnalisation du méta-modèle



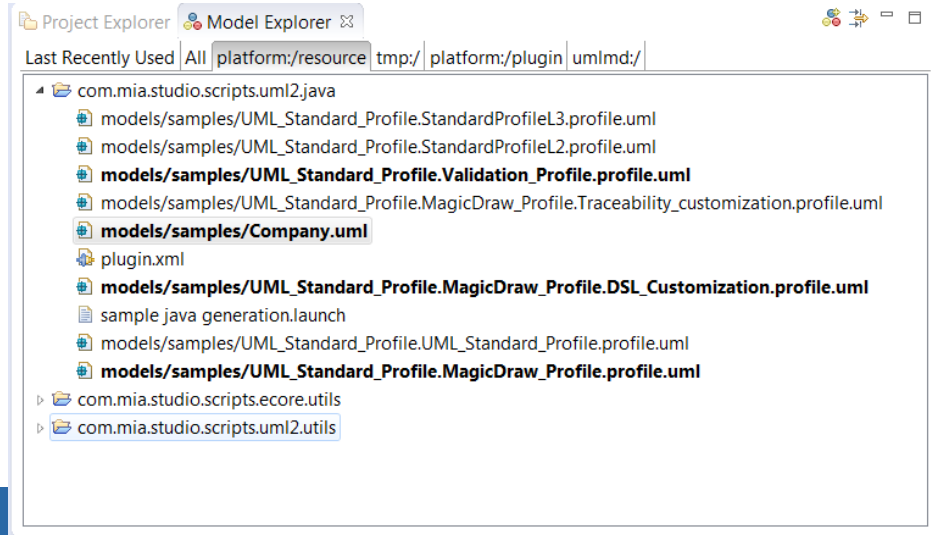
A close-up photograph of a human eye with a light blue iris, looking slightly to the left. The eye is framed by dark eyelashes and skin.

Sommaire

- Principes et concepts de base
- Importation & consultation des modèles
- Gestion et définition des scripts
- Les variables et méthodes des scripts Java
- Navigation dans les scripts
- Scenario & Configuration de génération
- Edition Developer & Integration IDE
- Notions avancées

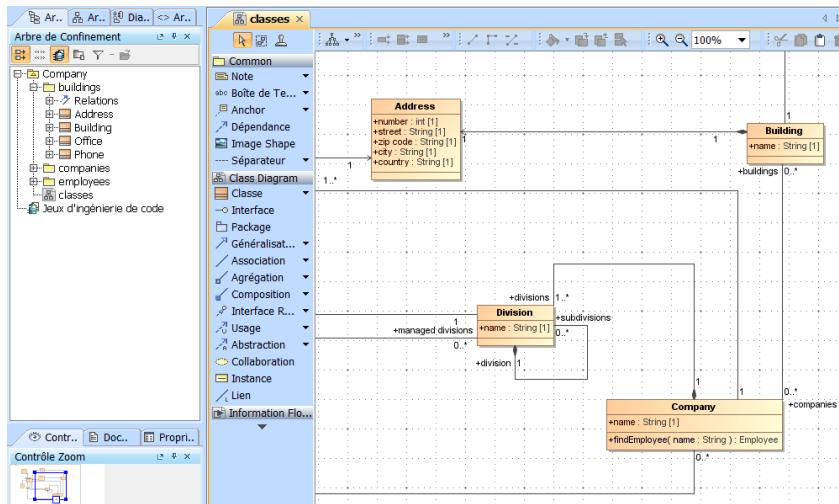
- Repose sur le framework EMF
- Plusieurs langages de modélisation préinstallés
 - UML14, UML2, ...
 - Possibilités de développer son propre langage (« DSL ») et ses propres lecteurs de format

La vue « Model Explorer »
indexe notamment les
fichiers XML de modèles du
workspace



Importation & consultation des modèles

- L'éditeur associé par défaut au « Model Explorer » permet de visualiser le contenu d'un modèle



De la vue d'un « modeleur UML »...

... à la vue dans Mia-Studio

Concepts du langage UML (ou « métamodèle » UML)

Elements du modèle UML (instances du métamodèle UML)

A close-up photograph of a human eye with a light blue iris, looking slightly to the left. The eye is framed by dark eyelashes and skin.

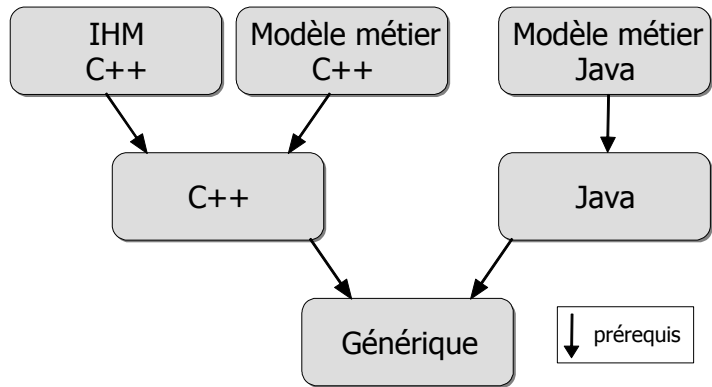
Sommaire

- Principes et concepts de base
- Importation & consultation des modèles
- Gestion et définition des scripts
- Les variables et méthodes des scripts Java
- Navigation dans les scripts
- Scenario & Configuration de génération
- Edition Developer & Integration IDE
- Notions avancées

- Les règles de génération sont exprimées à travers des scripts
 - **Un script = 1 texte**
 - stocké dans 1 **package** (ex : Java)
 - rattaché à 1 **concept du métamodèle** (ex : Property UML)
 - classé dans 1 **catégorie** (ex : accessors)
 - peut être rattaché à un autre script (propriété)
 - **2 familles de scripts :**
 - Template (WYSIWYG)
 - Code Java
- Polymorphisme = 1 script peut redéfinir 1 script sur concept parent
- Les templates et les scripts Java peuvent avoir des paramètres

Les packages

- Les packages permettent d'organiser les scripts.
- Entre deux exécutions de Mia-Studio, un script est persisté avec son package (fichier XML).
- Si le fichier est en lecture seule, les scripts ne seront pas modifiables dans Mia-Studio.
- Deux versions d'un même package peuvent être comparées au sein de Mia-Studio.



- L'éditeur de package permet la définition des scripts

The screenshot shows the 'com.mia-software.java.common' package editor. The interface is divided into several sections:

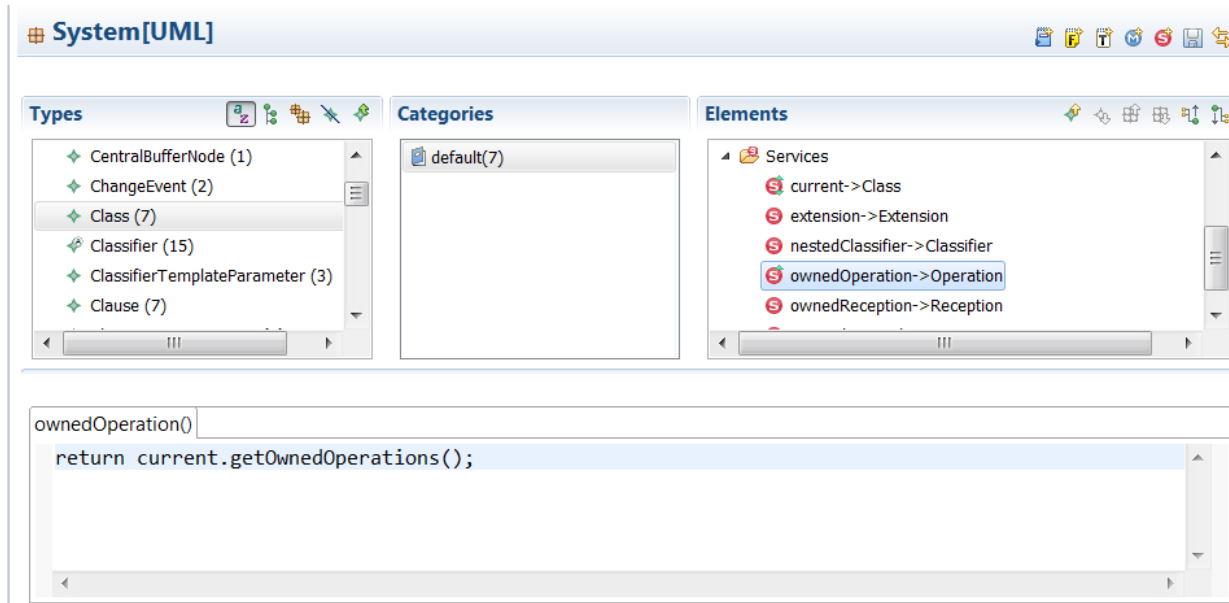
- Types:** A list of language concepts (UML metamodel) including Class (16), Classifier (11), Enumeration (2), EnumerationLiteral (1), EObject (3), Feature (3), and Interface (3).
- Categories:** A list of categories, currently showing 'default(16)'.
- Elements:** A tree view of package elements including Scenarios, File Templates, JavaClass, Text Templates, Macros, and Services.
- Script Editor:** A text editor at the bottom showing the body of a script for 'JavaClass()'. The script content includes package declarations, imports, class declarations, and nested class/attribute definitions.

Annotations with arrows point to the following elements:

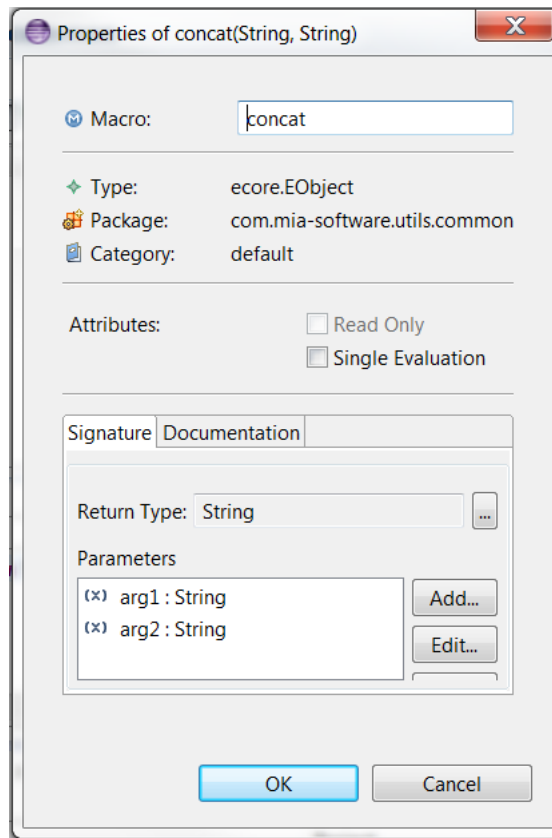
- Nom du Package:** Points to the package name 'com.mia-software.java.common' in the title bar.
- Concepts du langage (ou « métamodèle »), ici UML:** Points to the 'Types' list.
- Scripts du package:** Points to the 'Elements' tree view.
- Corps d'un script:** Points to the script body in the 'JavaClass()' editor.

```
JavaClass()  
  
[[javadocHeader]]package [[codingPackageQualifiedNames]];  
  
[[javadoc]] [[generationImport(referencedClassifiers)]]  
[[visibility]] [[modifiers]]class [[CodingName]] [[extendsDeclaration]]  
  
    // ----- NESTED CLASSES -----  
    [[userDefinition("Nested Classes")]]  
    [[nestedClassDeclaration]]  
  
    // ----- ATTRIBUTES -----  
    [[userDefinition("Attributes")]]
```

- Il existe un package « System » prédéfini qui définit des scripts utilitaires d'accès aux informations du modèle

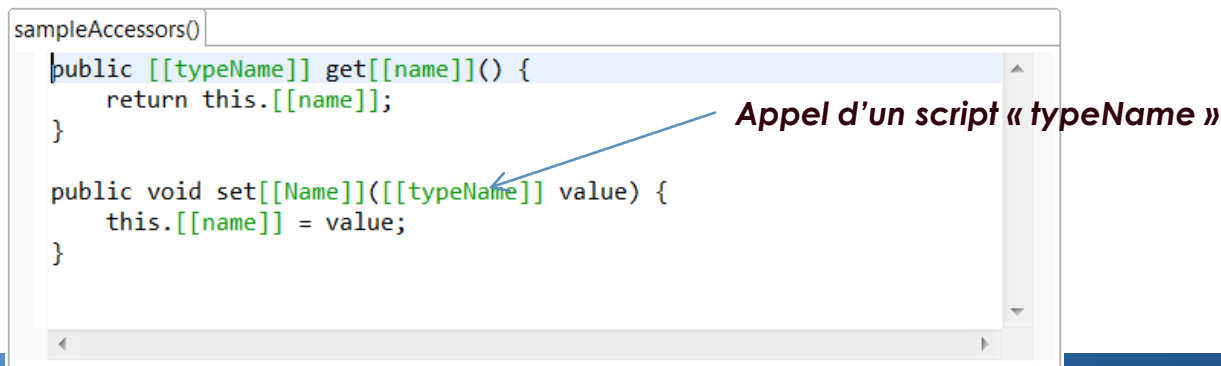


- Propriétés de script éditables via le menu "Properties"
 - Nom
 - Date de création
 - Date de modification
 - Evaluation unique
 - Signature
 - Documentation



Les scripts de type **Template**

- Portion de texte reproduite telle quelle au moment de la génération
- Contient des noms de scripts qui sont remplacés, au moment de la génération, par le résultat de leur évaluation
- Les scripts référencés doivent être définis sur le type courant (ex : Class) ou l'un de ses super-types



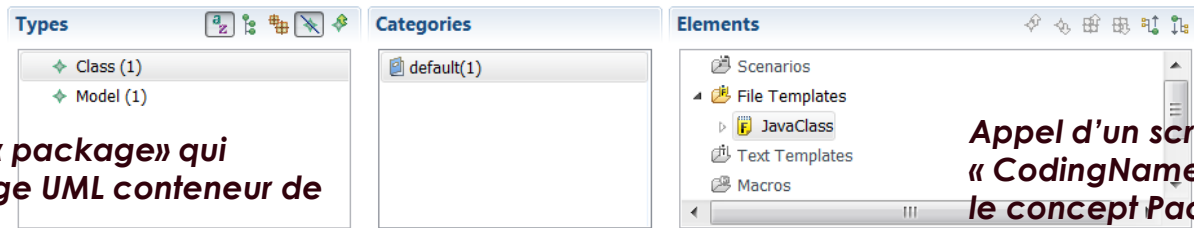
```
sampleAccessors()  
  
public [[typeName]] get[[name]]() {  
    return this. [[name]];  
}  
  
public void set[[Name]]([[typeName]] value) {  
    this. [[name]] = value;  
}
```

Appel d'un script « typeName »

Les scripts de type **Template**

- La **notation pointée** permet d'enchaîner l'appel à des scripts
 - Le premier script appelé retourne 1 ou n éléments du modèle
 - Le second script est évalué sur chacun des éléments obtenus

com.mia-software.java.helloworld



Appel d'un script « package » qui retourne le Package UML conteneur de la Class UML

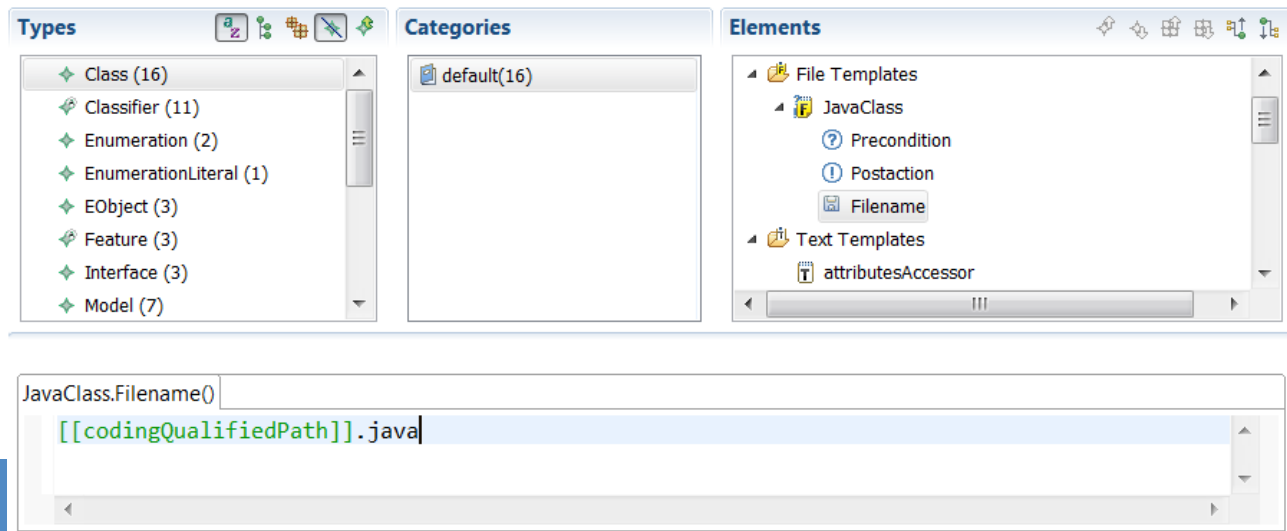
Appel d'un script « CodingName » qui est défini sur le concept Package UML

```
JavaClass()  
Hello World generated from [[name]] uml class, and from [[package.CodingName]] package! |
```

- Un template retourne toujours une instance de String
- Un template peut définir ses propres propriétés
 - Il y a sept propriétés possibles pour un template :
 - Filename
 - BeginTag
 - EndTag
 - Precondition
 - Preaction
 - Postaction
 - Postwrite action
 - Ce sont elles-mêmes des scripts rattachés au template.

- Le **Filename** est un Template générant le nom de fichier dans lequel est sauvegardé le résultat de la génération
 - Un template ayant une propriété Filename est un **FileTemplate**
 - Un template sans propriété Filename est un **TextTemplate**

Si le nom de fichier obtenu sur le FileTemplate n'est pas absolu, il sera résolu grâce au répertoire de génération spécifié à la génération.



Les **tags** permettent de délimiter une zone modifiable entre deux générations : le texte ne sera pas écrasé par la dernière génération.

BeginTag : template décrivant la balise de début

EndTag : template décrivant la balise de fin

Il peut y avoir autant de zones utilisateurs que vous voulez dans un fichier mais chaque zone doit être identifiée de manière unique. Pour rendre un tag unique, vous pouvez le « personnaliser » avec un nom d'objet du modèle, un identifiant unique, etc...

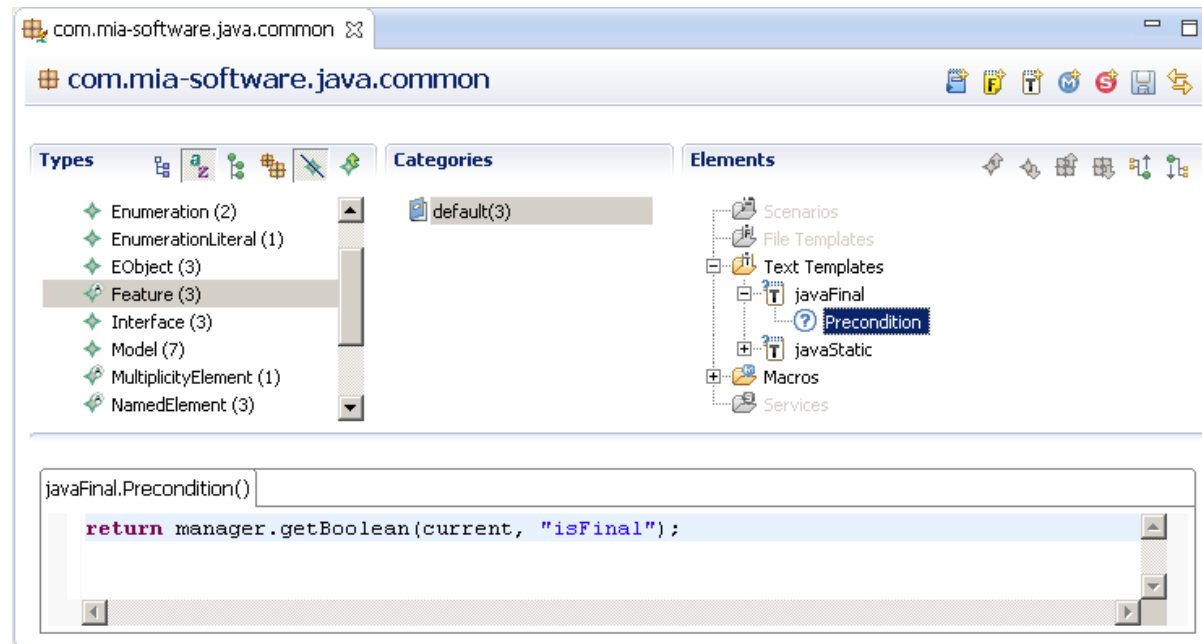
```
//Start of specific code for [[name]]|
```

Pour être prises en compte les deux propriétés doivent être référencées dans le texte du template

The screenshot displays an IDE interface with three panels at the top: 'Types', 'Categories', and 'Elements'. The 'Types' panel lists 'Class (17)', 'Classifier (11)', 'Enumeration (2)', 'EnumerationLiteral (1)', and 'EObject (3)'. The 'Categories' panel shows 'default(17)'. The 'Elements' panel shows a tree structure with 'superClassesAbstractOperationsDeclaration' and 'toString', which has sub-elements 'BeginTag' and 'EndTag'. Below these panels is a code editor showing the implementation of the 'toString()' method. The code is as follows:

```
toString()  
  
public String toString() {  
    [[BeginTag]]  
    return "instance of [[name]]";  
    [[EndTag]]  
}
```

- La **Précondition** permet de conditionner l'évaluation d'un template.
 - Exemple :
Génération de l'initialisation d'un texte uniquement si une condition est remplie



Les scripts de type Java

- Deux types de scripts
 - **Macro** : portion de code Java qui renvoie n 'importe quel type d 'objets Java (mais en général du texte)
 - **Service** : portion de code Java qui renvoie un(des) objet(s) du modèle.

The screenshot shows the Mia-Software interface with three main panels: 'Types', 'Categories', and 'Elements'. The 'Types' panel lists various UML types like Class (17), Classifier (12), Enumeration (2), etc. The 'Categories' panel shows 'associationEnds', 'attributes', and 'default(12)'. The 'Elements' panel shows a tree structure with 'Macros' and 'Services'. A script editor is open, showing a script for 'packageName()' that returns 'current.getPackage().getName()'. A dialog box is open, asking for the 'Script return Type', with 'Primitive' selected and 'String' chosen from the dropdown.

The screenshot shows the Mia-Software interface with three main panels: 'Types', 'Categories', and 'Elements'. The 'Types' panel lists various UML types like OperationTemplateParameter (1), OutputPin (1), Package (9), etc. The 'Categories' panel shows 'default(9)'. The 'Elements' panel shows a tree structure with 'URI' and 'Services'. A script editor is open, showing a script for 'packagedElement()' that returns 'current.getPackagedElements()'.

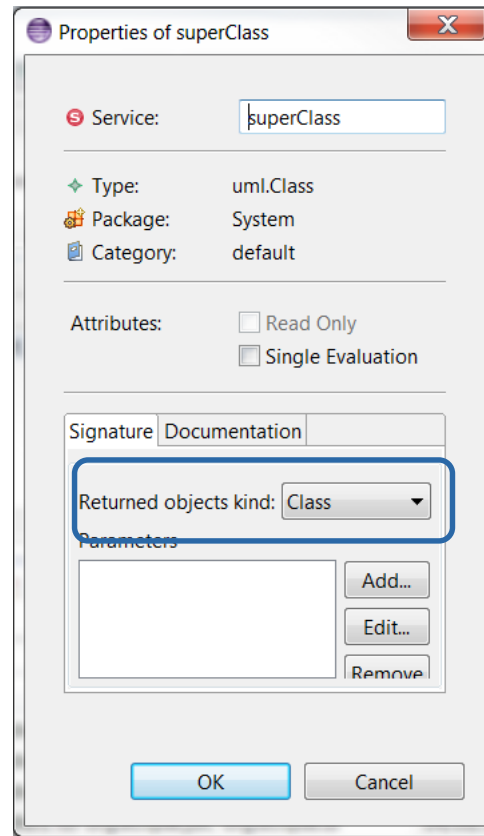
Les services : type des objets retournés

Un service a un type de retour déclaré Object.

Un service peut renvoyer réellement :

- un objet du modèle (ex : Class)
- une collection d 'objets (Collection)
- un itérateur (Iterator)
- un tableau d 'objets du modèle (Object[])

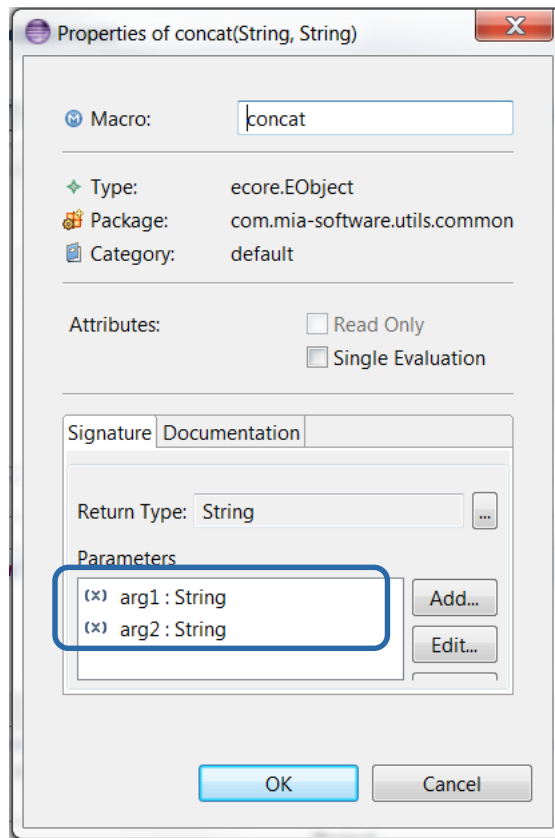
Pour utiliser les services, on définit le type des objets du modèle renvoyés par le service



Gestion et définition des scripts

Les templates, les macros et les services peuvent avoir des paramètres

- Le nombre de paramètres est quelconque.
- Un paramètre est nommé et typé.
- Un paramètre ne peut être un type primitif (int, boolean, etc.) : on ne passe que des objets.
- Le nom de chaque paramètre doit être un identifiant Java™ valide.
- Un script est identifié par son nom et le nombre de ses paramètres.
- Il est ainsi possible de définir deux scripts portant le même nom, sous réserve que leur nombre de paramètres soit différent.
- Un script ne peut déclarer deux paramètres portant le même nom.



A close-up, vertical photograph of a human eye with light blue irises, looking slightly to the left. The eye is framed by dark eyelashes and skin. The image is positioned on the left side of the slide, partially overlapping the title area.

Sommaire

- Principes et concepts de base
- Importation & consultation des modèles
- Gestion et définition des scripts
- Les variables et méthodes des scripts Java
- Navigation dans les scripts
- Scenario & Configuration de génération
- Edition Developer & Integration IDE
- Notions avancées

Les variables et méthodes des scripts Java

- **current**

Référence l'objet courant sur lequel le script est évalué

- **context**

Référence un dictionnaire global : sur cette variable peuvent être appliquées des méthodes pour gérer des variables globales

*String **getString** (String clé)*

*void **setString**(String clé, String valeur)*

*Object **getValue** (String clé)*

*void **setValue** (String clé, Object valeur)*

*void **clear**()*

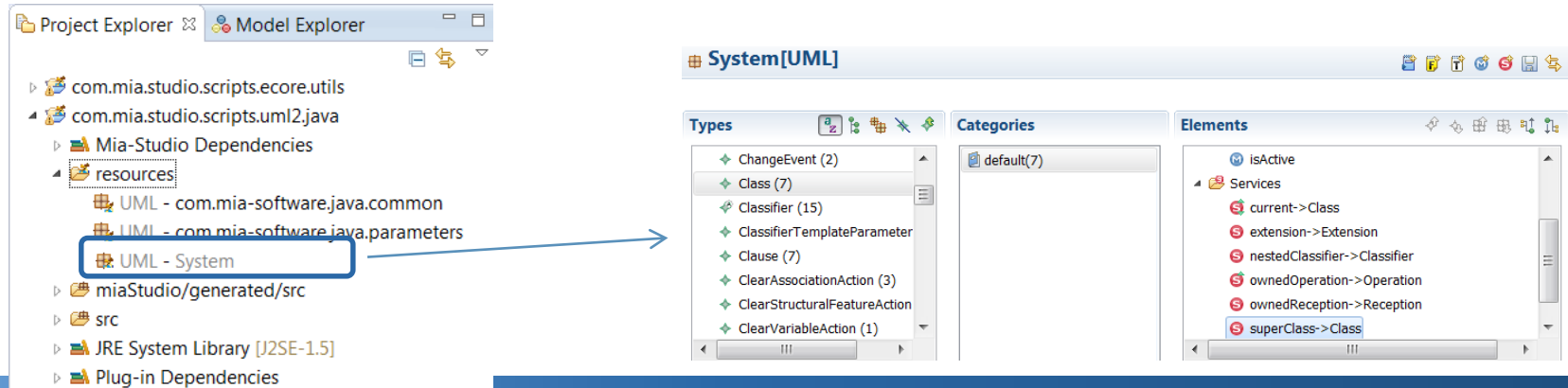
...

Donne accès à certaines fonctions génériques de Mia-Generation :

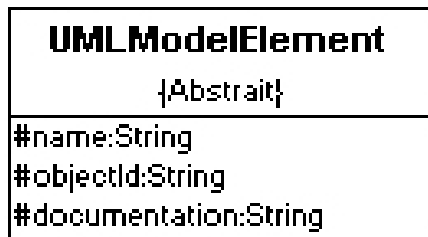
*void **cancelGeneration**()*

*boolean **isSilentMode**()*

- Pour chaque type d'objet du modèle, un certain nombre d'accesseurs est fourni :
 - Ces accesseurs correspondent aux liens et attributs définis dans le méta-modèle courant.
 - Le package « System » de Mia-Studio Generation contient les scripts (macros et services) correspondant à chaque accesseur.
 - Les objets héritent des accesseurs définis sur leurs types « Parent ».

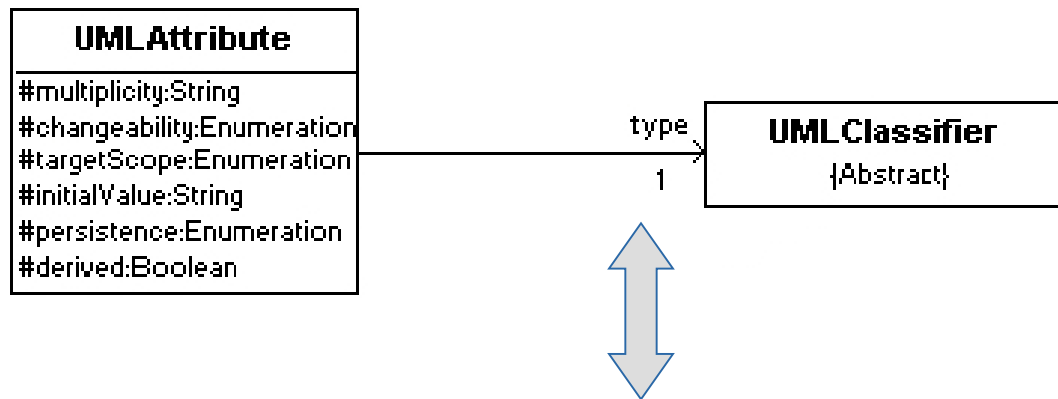


- pour chaque attribut **a** dans le métamodèle : une méthode **getA()** qui renvoie la valeur de l'attribut



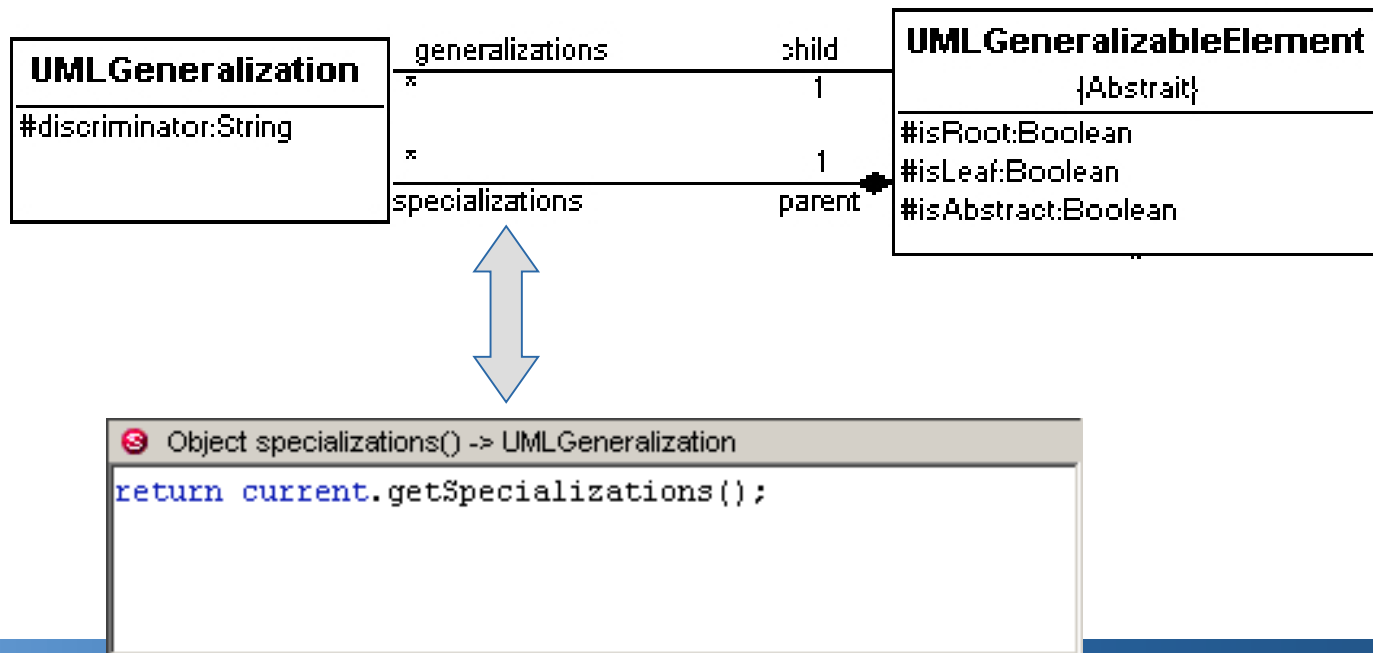
```
M String name()  
return current.getName();
```

- pour chaque rôle **r1** de multiplicité 1 dans le métamodèle : une méthode **getR1()** qui renvoie l'objet lié au rôle



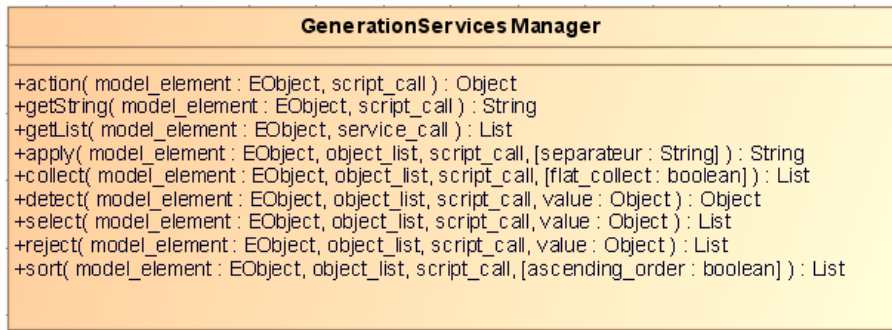
```
$ Object type() -> UMLClassifier  
return current.getType();
```

- pour chaque rôle **rN** de multiplicité n : une méthode **getrN()** qui renvoie, sous forme d'un tableau d'objets, les objets liés au rôle.



Les variables et méthodes des scripts Java

- Tous les objets du modèle manipulés dans les scripts Mia-Studio héritent de **EObject**.
- Une variable « **manager** » représente une instance de *GenerationServicesManager* et met à disposition des services pour l'application de scripts sur les instances EObject.



2 façons de référencer un appel de script (script_call) dans ces méthodes :

- un objet **String** : un script qui n'a aucun paramètre peut être référencé directement par son nom,
- un objet **Call** : l'objet Call doit être utilisé pour référencer les scripts qui attendent des paramètres.

Les listes d'objets du modèle (object_list) sont :

- des collections,
- des itérateurs,
- des tableaux,
- un appel de service (String ou Call).

Object **action** (model element, script call)

Exécute un script

(méthode générique, il faut en général lui préférer getString et getList)

String **getString** (model element, script call)

Exécute un script et renvoie une instance de String

(adapté pour appeler un template dans une macro)

List **getList** (model element, script call)

Exécute un script et renvoie une instance de List

*(**recommandé** pour appeler un service dans une macro car vous n 'avez pas à savoir si le service retourne une collection, un itérateur ou un tableau.)*

String **apply** ([model element], object list, script call, [separator])

Enchaîne l'exécution d'un script sur une liste d'objets, et renvoie la concaténation de chacun des résultats.

Il est possible de définir un séparateur qui sera inséré entre chaque concaténation.

Le paramètre 'object_list' peut être :

- une Collection*
- un Iterator*
- un tableau d'objets*
- un nom de service (dans ce cas, le paramètre 'model_element' est nécessaire pour désigner le receveur)*

Object **detect** ([model element], object list, script call, value)

Renvoie le premier objet dont l'évaluation du script est égal au critère.

Collection **select** ([model element], object list, script call, value)

Renvoie tous les objets dont l'évaluation du script est égal au critère.

Collection **reject** ([model element], object list, script call, value)

Renvoie tous les objets dont l'évaluation du script est différente du critère.

Collection **collect** ([model element], object list, script call, [flat collect])

Renvoie une collection contenant l'évaluation du script sur chacun des objets.

Le dernier argument (booléen) permet d'aplatir le résultat

(adapté si l'on récupère une collection de collections)

List **sort** ([model element], object list, script call, [ascendingOrder])

Renvoie une collection d'objets triée sur l'évaluation d'un script.

List **sort** (object list)

Renvoie une collection d'objets triée alphabétiquement d'après le champ 'name' si celui-ci existe, sinon d'après le résultat de 'toString()'

A close-up photograph of a human eye with a light blue iris, looking slightly to the left. The eye is framed by dark eyelashes and skin.

Sommaire

- Principes et concepts de base
- Importation & consultation des modèles
- Gestion et définition des scripts
- Les variables et méthodes des scripts Java
- Navigation dans les scripts
- Scenario & Configuration de génération
- Edition Developer & Integration IDE
- Notions avancées

- *TODO*
- *Status & Navigation & recherche*

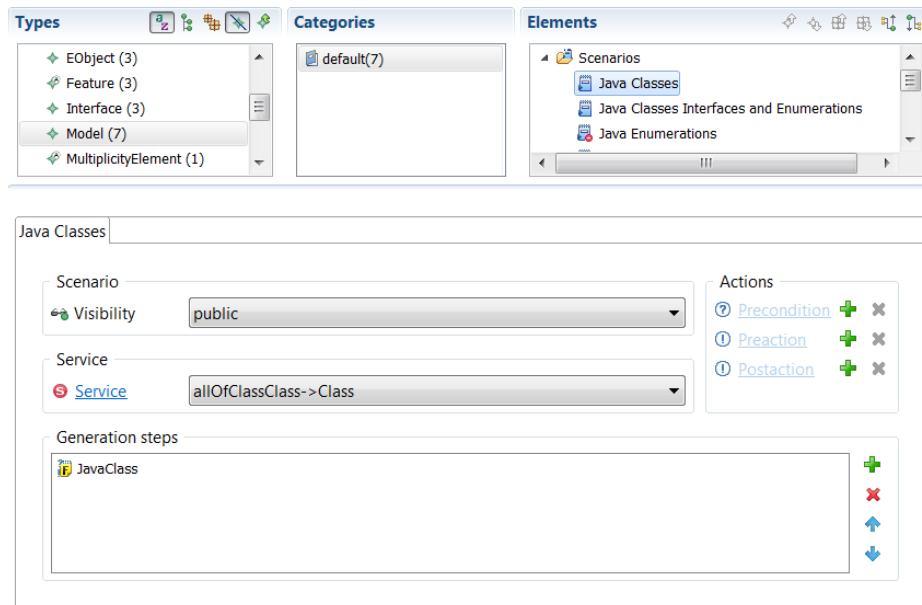


Sommaire

- Principes et concepts de base
- Importation & consultation des modèles
- Gestion et définition des scripts
- Les variables et méthodes des scripts Java
- Navigation dans les scripts
- Scenario & Configuration de génération
- Edition Developer & Integration IDE
- Notions avancées

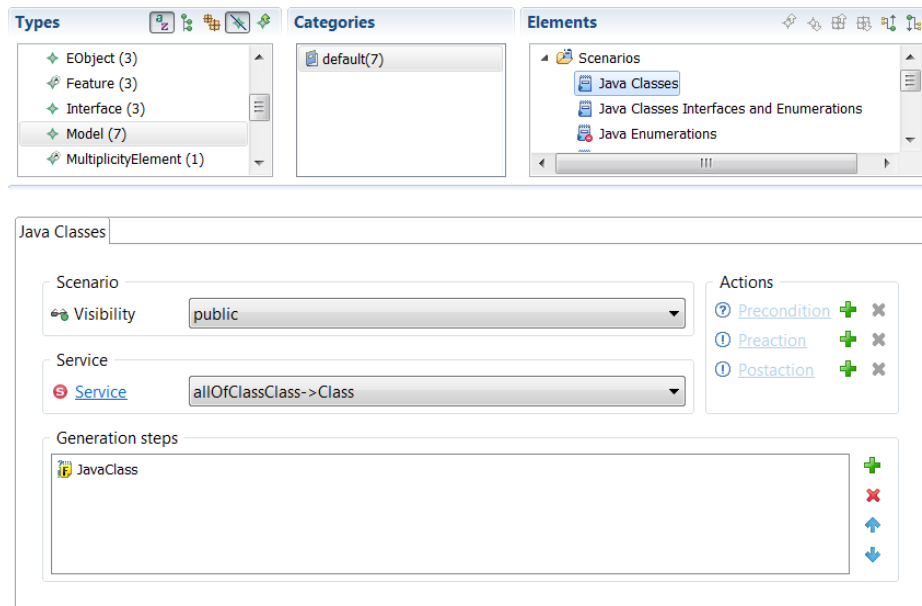
Scenario & Configuration de génération

- Un scénario décrit des enchaînements d'évaluations de scripts
 - Une pré-condition peut permettre de décider si un scénario doit être exécuté ou non.
 - On a la possibilité d'exécuter des actions avant et après l'exécution d'un scénario (PreAction et PostAction)
 - Référence un service (applicable sur les objets du type sur lequel est déclaré le scénario)
 - Référence un ensemble d'étapes de génération (**scénarios** ou **File templates**)
 - Un scénario peut avoir une visibilité publique ou privée (invisible après déploiement)



Pour chaque exécution de scenario, Mia-Studio

- évalue le service sur toutes les instances du type choisi
- applique chaque sous étapes de génération sur chacun des objets renvoyés par le service



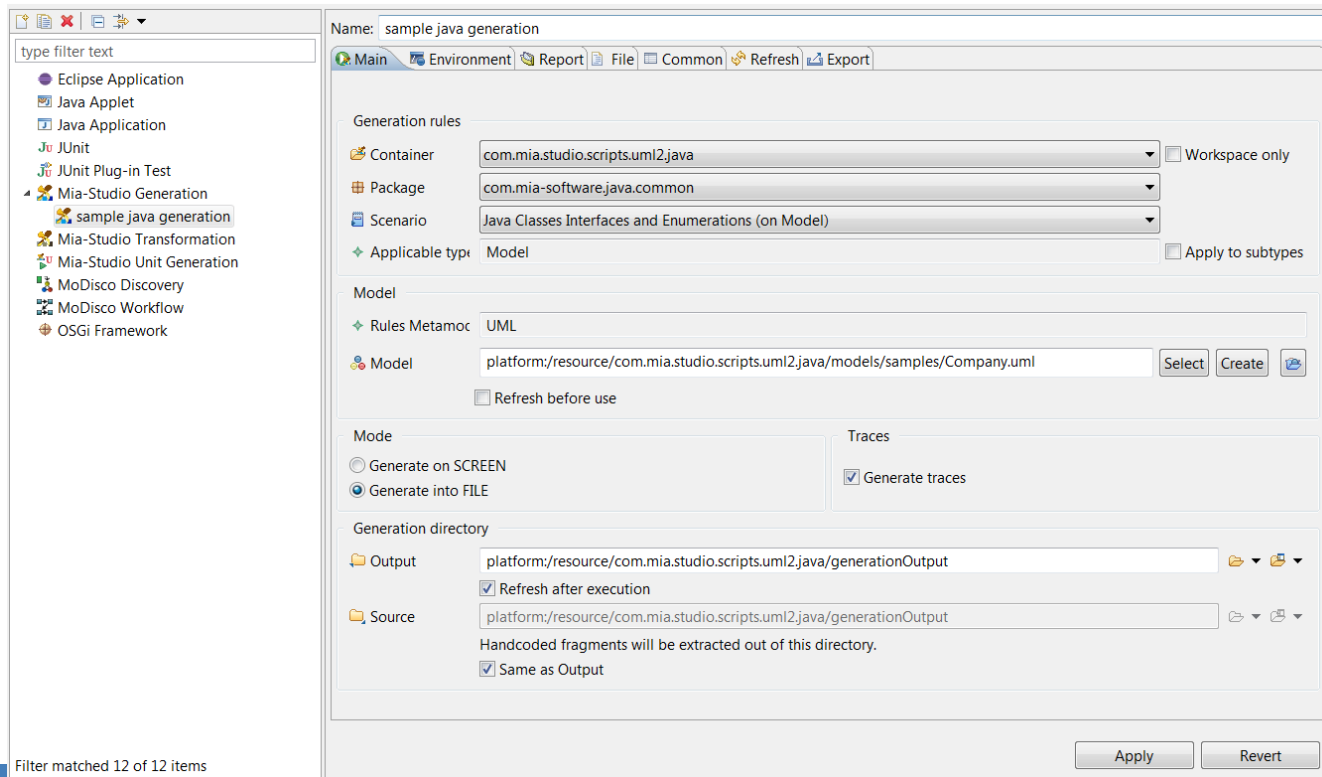
- Lancement d'une génération
 - Mode unitaire : mise au point de scripts
 - *Sélection des objets du modèle*
 - *Sélection des scripts applicables sur ces objets*
 - Mode par scénario : mise au point & mise en production
 - *Sélection d'un projet et d'un scénario « public »*
 - *Sélection d'un modèle*

Scenario & Configuration de génération

- Mode par scénario : type Run configuration par scénario

Paramétrages

- Choix du projet & scénario
- Mode « OnScreen/into File »
- Répertoire de génération
- Répertoire de référence
- Vidage du contexte global
- Option des fichiers
- Activation des traces
- ... etc

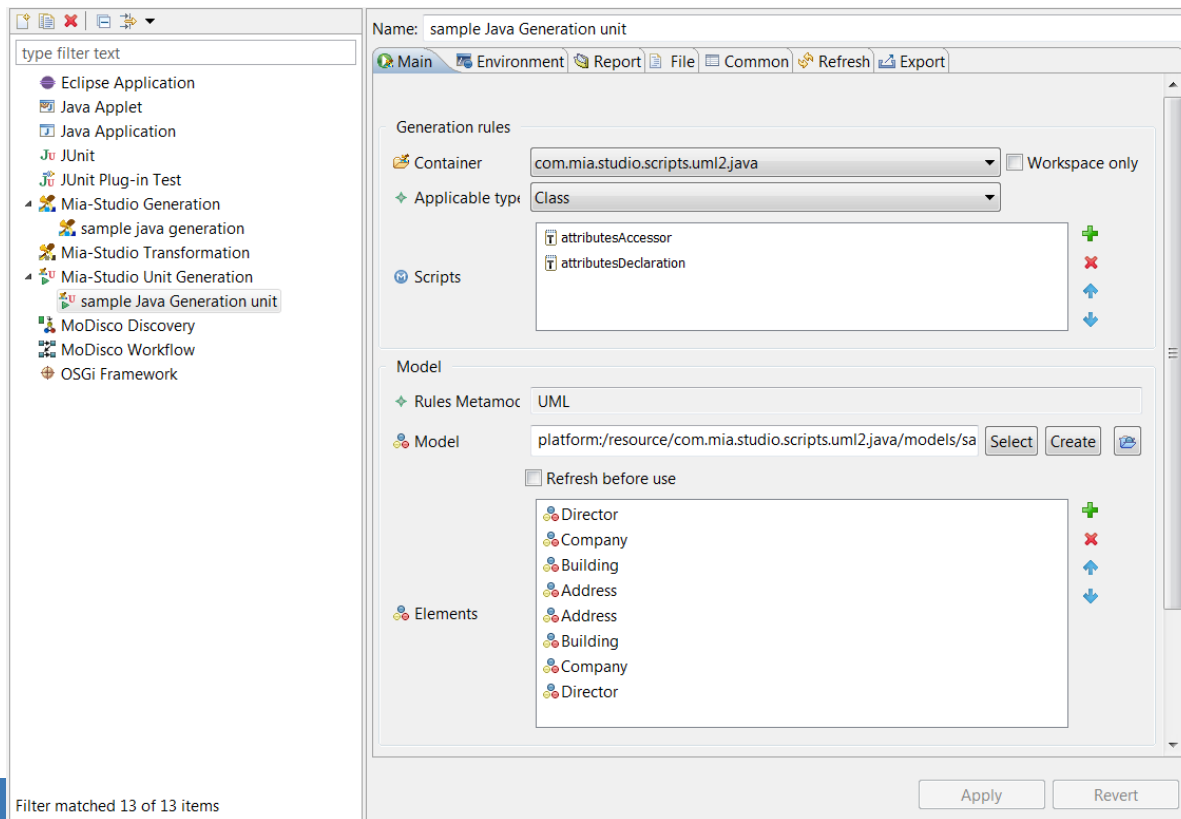


Scenario & Configuration de génération

- Mode unitaire : type Run configuration unitaire

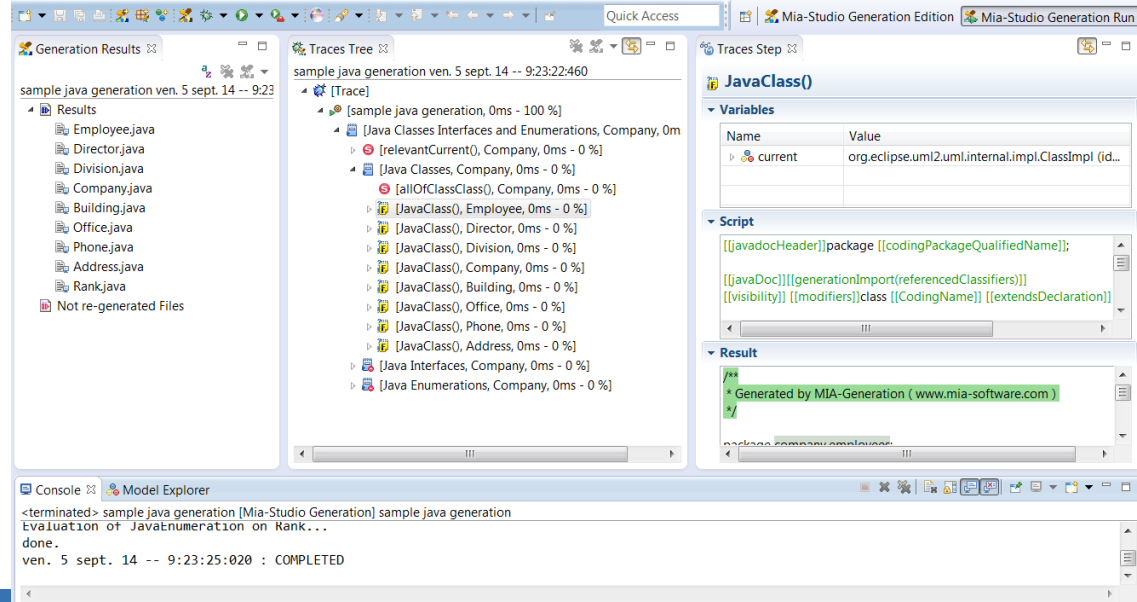
Paramétrages

- Choix des scripts
- Choix des éléments de modèle
- ... etc



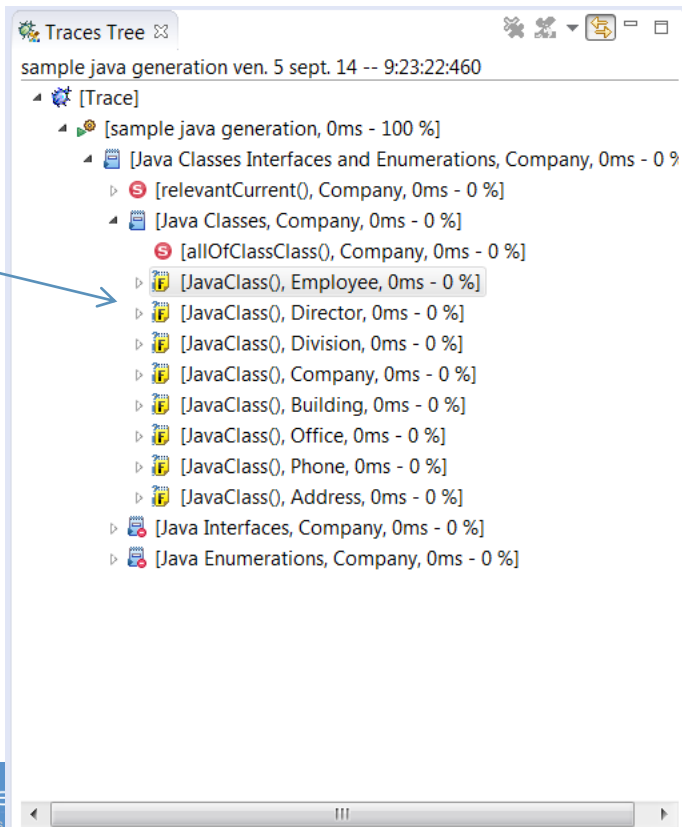
Scenario & Configuration de génération

- Résultats de génération
 - Production : Vue « Generation Results »
 - Arbre de fragments « code généré/code manuel balisé »
 - Mise au point : Perspective complète « Mia-Studio Generation Run »
 - Vue résultats
 - Console d'évaluation
 - Vue arbre de traces
 - Vue détail d'un nœud d'exécution

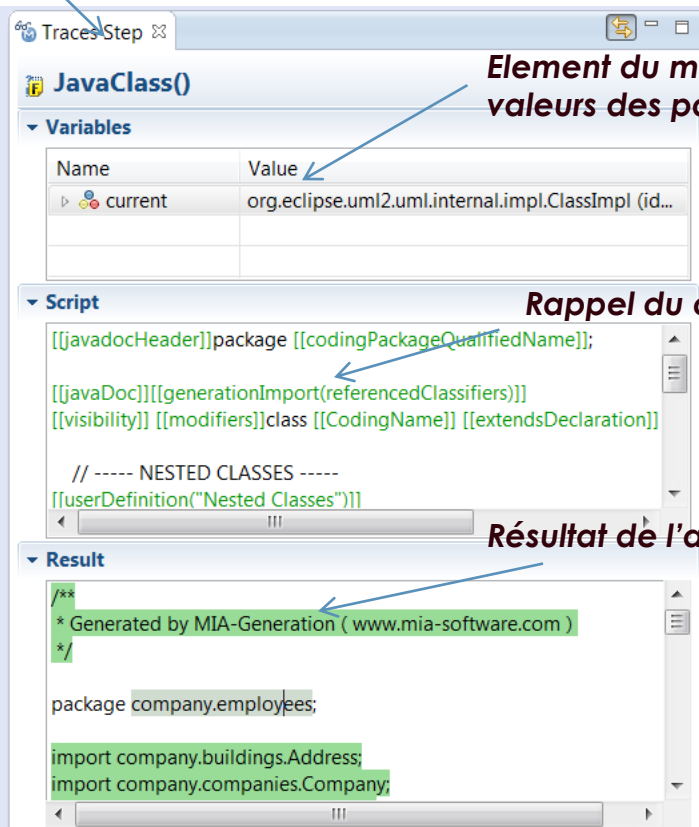


- Trace de génération

Arbre
d'appels
de scripts



Detail d'un appel de
script



A close-up photograph of a human eye with a light blue iris, looking slightly to the left. The eye is framed by dark eyelashes and skin.

Sommaire

- Principes et concepts de base
- Importation & consultation des modèles
- Gestion et définition des scripts
- Les variables et méthodes des scripts Java
- Navigation dans les scripts
- Scenario & Configuration de génération
- Edition Developer & Integration IDE
- Notions avancées

- 2 modes de fonctionnement Mia-Studio
 - Edition « Architect »
 - Droits d'écriture et exécution de generation/transformation
 - A destination des responsables de l'atelier/usine logiciel
 - Edition « Developer »
 - Seulement les droits d'exécution de generation/transformation
 - A destination des postes du développeur utilisateur
 - A destination des lancements silencieux en production

- Intégration « native » possible dans
 - Modeleur UML IBM RSA
 - Modeleur UML NoMagic MagicDraw (*incubation*)
- Caractéristiques
 - Utilisation en mémoire des modèles ouverts
 - Generation partielle depuis selection dans diagrammes (MagicDraw)
- Avantages
 - Performance par rapport à échange par XML
 - Limite la rupture d'outils
 - Permet la transformation de modèles sans perte des diagrammes



- Lancement silencieux de generation/transformation
 - Ligne de commande (.bat ou shell)
 - Api Java « Engine »
 - Plugin Maven **maven**

A close-up photograph of a human eye with a light blue iris, looking slightly to the left. The eye is framed by dark eyelashes and skin.

Sommaire

- Principes et concepts de base
- Importation & consultation des modèles
- Gestion et définition des scripts
- Les variables et méthodes des scripts Java
- Navigation dans les scripts
- Scenario & Configuration de génération
- Edition Developer & Integration IDE
- Notions avancées

- Intégration de profil UML sous forme de « Profile Mia »
 - Permet de voir des stereotypes comme de nouveaux types UML et y attacher des scripts

com.mia-software.cinematic.struts.jsp

Types

- Operation (2)
- Property (2)
- Property:inputCheckBox (1)
- Property:inputSecret (1)
- Property:inputText (1)
- Property:outputLabel (1)

Categories

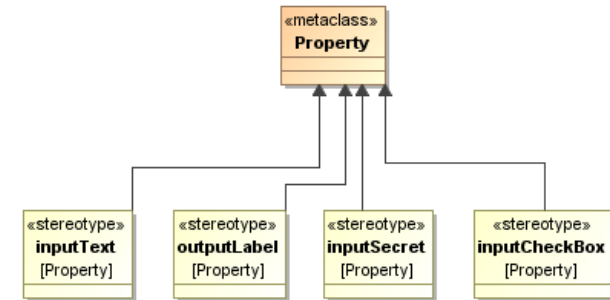
- default(1)

Elements

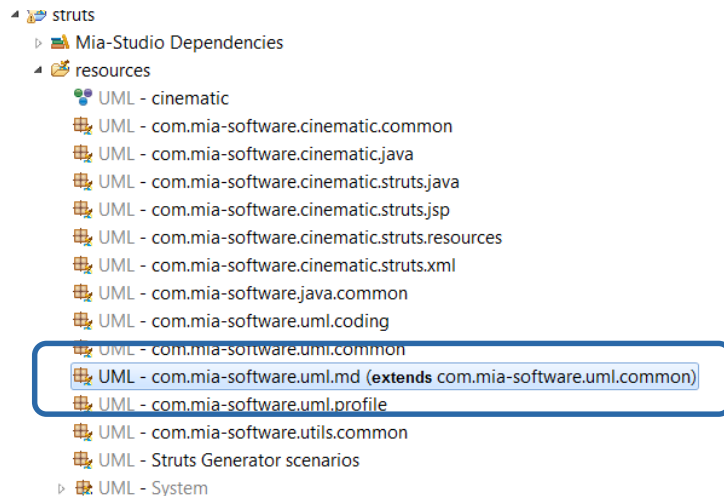
- Scenarios
- File Templates
- Text Templates
 - struts_field
- Macros
- Services

struts_field()

```
<html:checkbox property="[[codingName]]"/>
```



- Extension de package
 - **Principe** : permettre la redéfinition du script d'un package sans perdre, ni altérer son fonctionnement d'origine
 - **Intérêt**: Mettre en commun des packages de scripts qui ont un comportement commun pour une majorité de scripts, mais qui nécessitent une spécialisation pour un générateur donné



- Bonnes pratiques (1/2)

- **Principes hérités de l'approche objet : Scripts courts & Scripts sur la métaclasse la + adéquate**
 - métaclasse la plus proche du concept manipulé
 - métaclasse la plus générique (hiérarchie)⇒ Réutilisabilité + lisibilité
- **Répartition des scripts par packages ⇒ Réutilisabilité**
- **Privilégier au maximum des *templates* plutôt que macros (pour éviter des concaténations de chaînes) ⇒ Lisibilité (très souvent dans les macros, l'utilisation récurrente de *StringBuffer* est le signe que des *templates* auraient pu épargner un gros travail de concaténation)**
- Rattacher à *EObject* les scripts utilitaires hors modèle
- ~~Définir des macros "*apply(...)*" (cf boosters) qui seront utiles dans les *templates*~~
- Eventuellement utiliser le mécanisme de profil pour éviter des if/then
- Trier dans la mesure du possible les collections afin de générer toujours dans le même ordre
- Utiliser le *singleEvaluation* sur les scripts couteux en temps d'exécution
- Charger un modèle pour trouver les services de navigation à utiliser

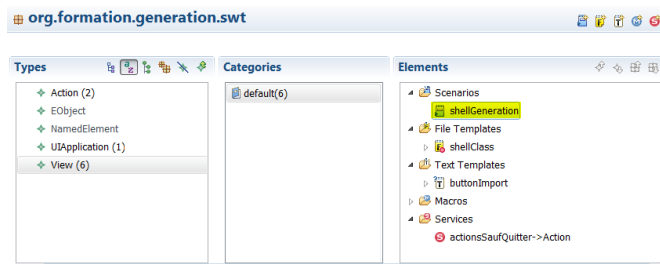
- Bonnes pratiques (2/2)

- Ne pas réinventer la roue : rechercher un script qui fait le traitement souhaité avant de l'écrire. Réutiliser les packages utils/uml fournis avec les boosters
- **Eviter les boucles Java, privilégier les apis MIA (*apply, collect, select, reject,...*)**
- **Eviter les macros et services de plus de 20 lignes**
- Ne pas définir de classes internes java dans les macros et services
- Eviter l'utilisation récurrente du contexte global : le contexte ne doit servir qu'à pallier des cas incontournables. Lorsqu'il s'agit d'un problème de performance / mémoire, l'utilisation de la propriété *singleEvaluation* peut suffire.

- Utilisation Générale de l'outil

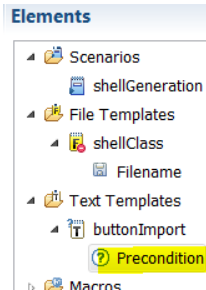
- Lire les « releases notes » à chaque version (notamment la section “compatibilité” à la fin), pour découvrir les nouvelles fonctionnalités et les bugs corrigés de l'outil à chaque version

Mia-Studio Generation : Mémo édition scripts

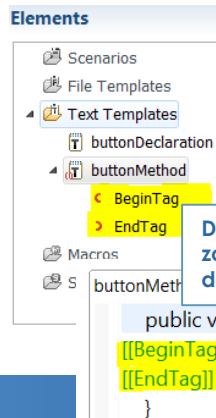


Un Scenario indique des appels de File Template, et est appelé depuis une runconfig Mia-Studio

File Template : permet la génération fichier avec indication emplacement (filename)



Precondition de template : permet de conditionner l'exécution d'un template



Appels de scripts depuis macro/service
manager.getString(..), manager.getList(..), manager.apply(..)

Des Tags délimitent une zone d'insertion code développeur

```
shellClass()
package com.mia.formation.generation.ui;

import org.eclipse.swt.widgets.Shell;
[[buttonImport]]
import org.eclipse.swt.SWT;
```

La notation pointée enchaîne les appels de scripts

```
public class [[name]] extends Shell {
    [[actions.buttonDeclaration]]

    for (Action anAction : (List<Action>) manager.getList(current, "actionsSaufQuitter")) {
        result += manager.getString(anAction, "buttonDeclaration");
    }

    result = manager.apply(current.getActions(), "buttonDeclaration");
    return result;
}
```

Dans une macro/service, « current » permet d'accéder à l'élément courant

