

# **Code Generation (ModelToText o M2T) with Acceleo**

Based on the OBEO manuals

available on the web

<http://www.acceleo.org/>

# Index

## Overview

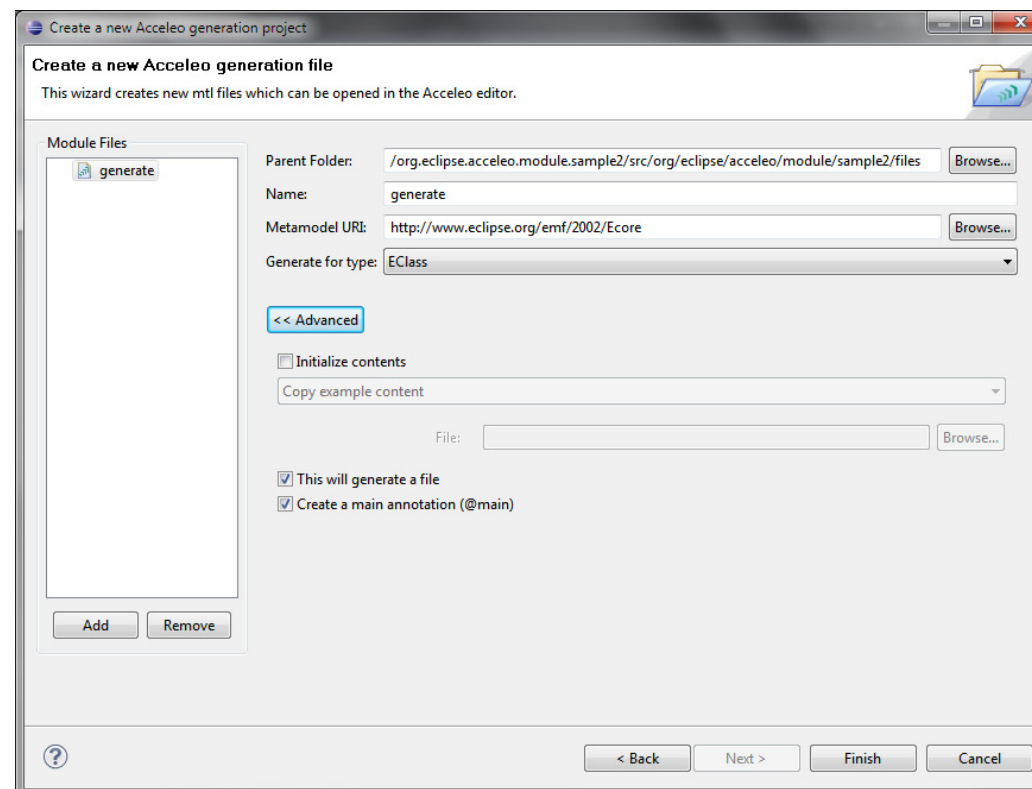
- Set up an acceleleo project
- The selection of metamodel, source model and target directory
- The Acceleleo template language
- An example

## Overview

- Acceleo is an Eclipse plugin
- It comes with its own perspective

## Step 1: Create a project

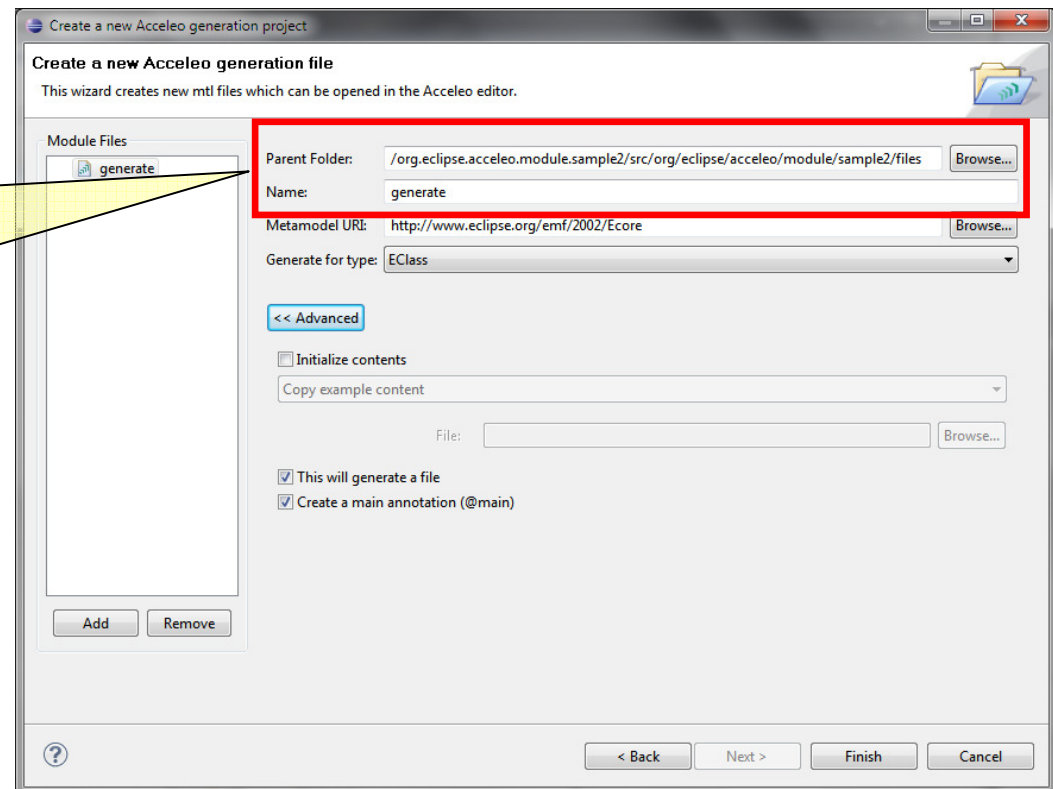
- Open the New project Eclipse wizard, and in the Model to Text Transformation category select Acceleo Project.
- On the next page, enter the project name. You can then click Finish or create one or several initial module files:



## Step 1: Create a project

- Open the New project Eclipse wizard, and in the Model to Text Transformation category select Acceleo Project.
- On the next page, enter the project name. You can then click Finish or create one or several initial module files:

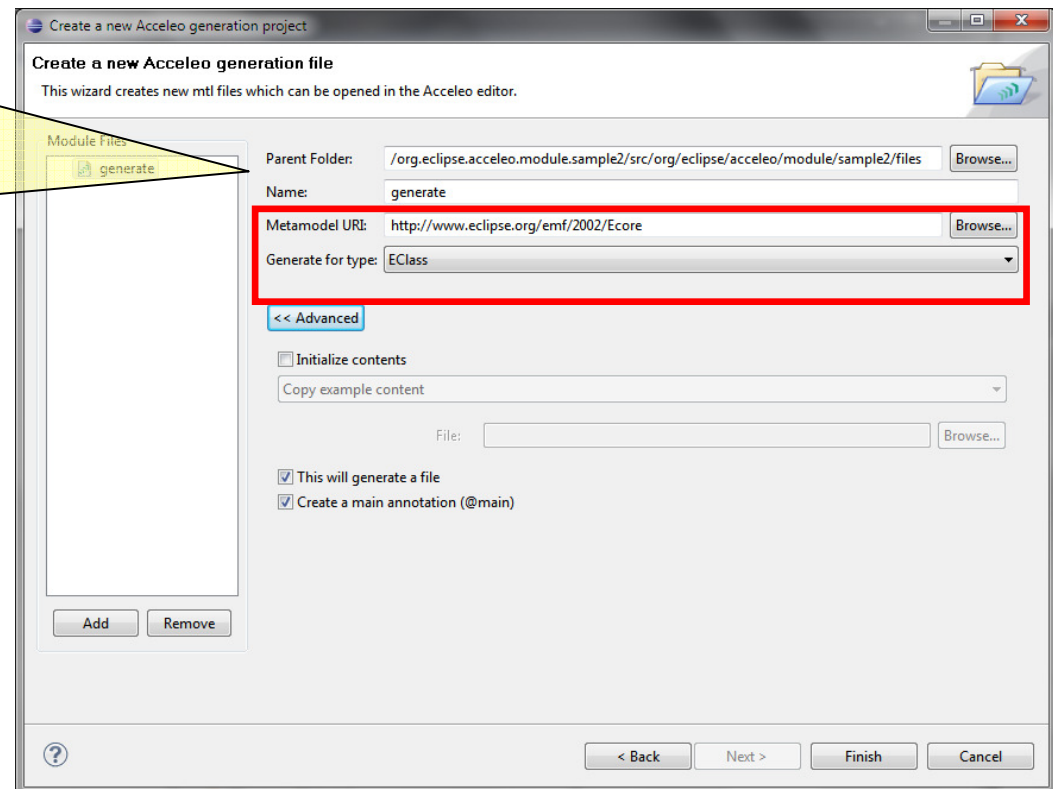
In the Module information section, you can specify the location and name of the new .mtl file to create.



## Step 1: Create a project

- Open the New project Eclipse wizard, and in the Model to Text Transformation category select Acceleo Project.
- On the next page, enter the project name. You can then click Finish or create one or several initial module files:

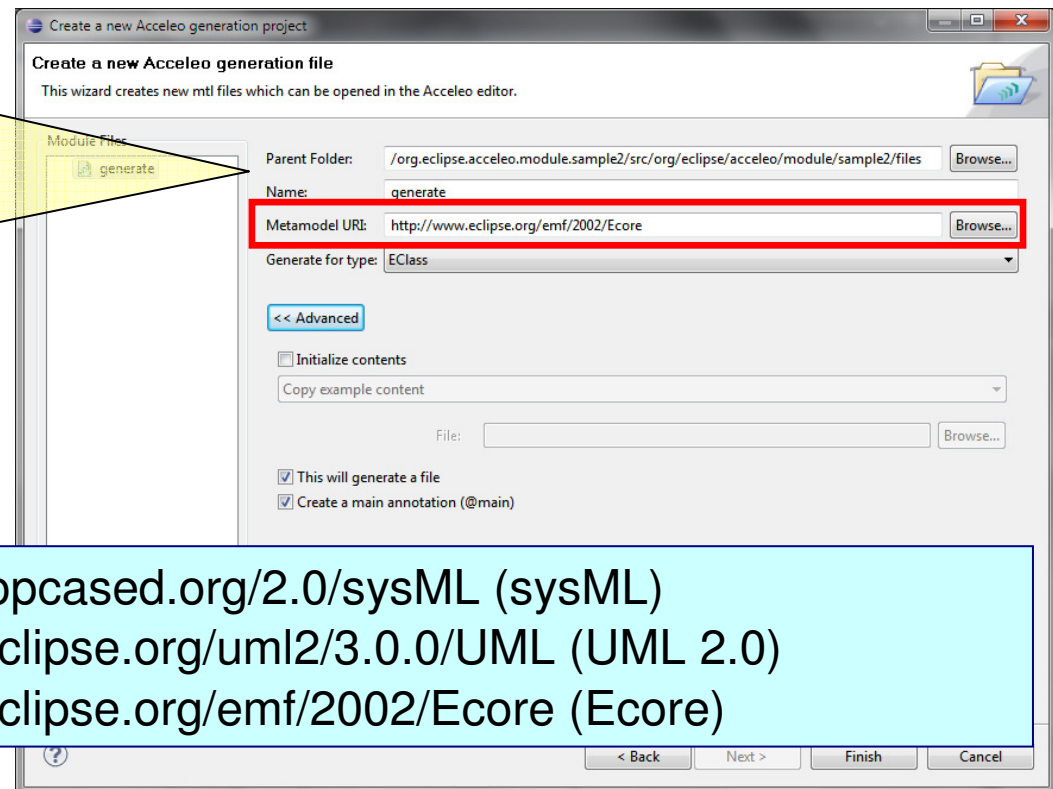
you can also initialize the contents of the new module file with either a fixed example file (Copy example content), or an existing Acceleo or Xpand template



## Step 1: Create a project

- Open the New project Eclipse wizard, and in the Model to Text Transformation category select Acceleo Project.
- On the next page, enter the project name. You can then click Finish or create one or several initial module files:

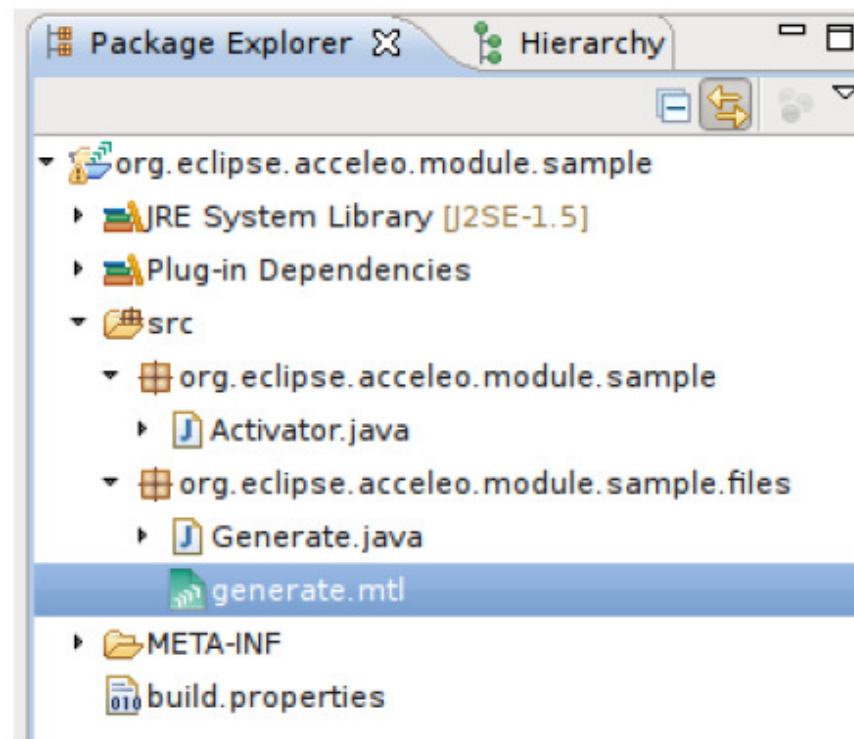
The Metamodel information section allows you to specify the input metamodel of your new module file, either from the list of registered metamodels or directly by URI.



Possible options: <http://www.topcased.org/2.0/sysML> (sysML)  
<http://www.eclipse.org/uml2/3.0.0/UML> (UML 2.0)  
<http://www.eclipse.org/emf/2002/Ecore> (Ecore)

## Step 1: Create a project

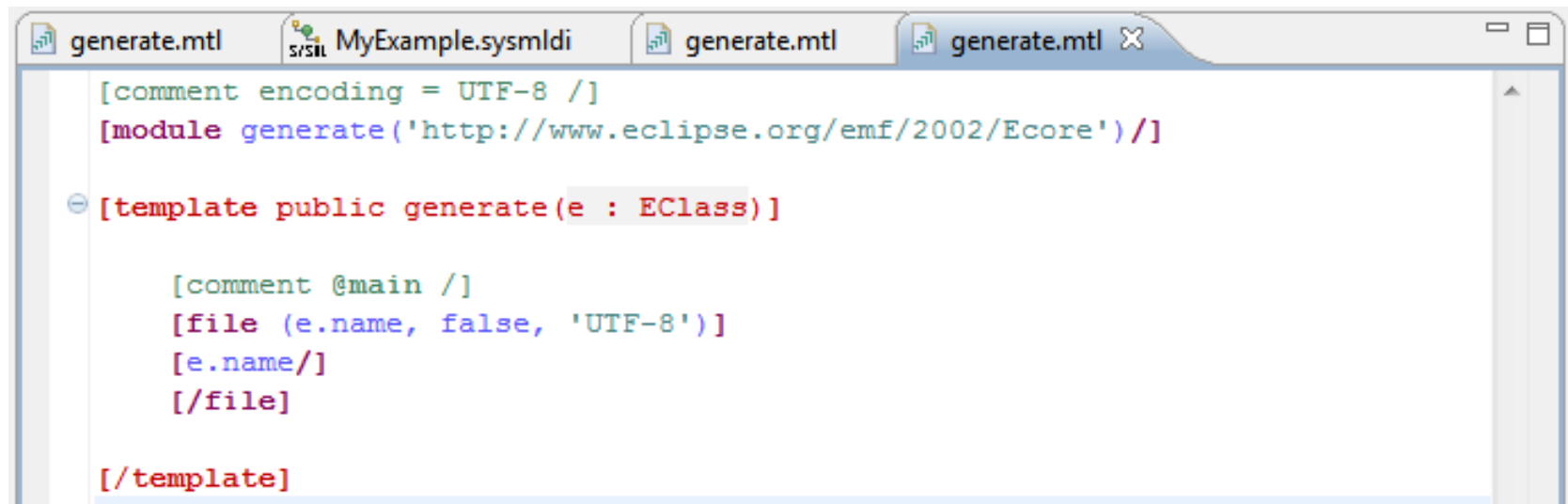
- The Acceleo project is created as an eclipse plugin!
- The wizard creates its own tree with a default template (generate.mtl) in the Acceleo template language





## Step 2: Editing the templates

- Acceleo comes with its own editor with several features ...
  - Syntax highlighting
  - Content assistant (ctrl + space)
  - Error detection
  - Dynamic outline
  - Code folding
  - Open declaration (either with 'ctrl + left click' or 'F3')
  - Search references (ctrl + shift + G)



```
[comment encoding = UTF-8 /]
[module generate('http://www.eclipse.org/emf/2002/Ecore')]

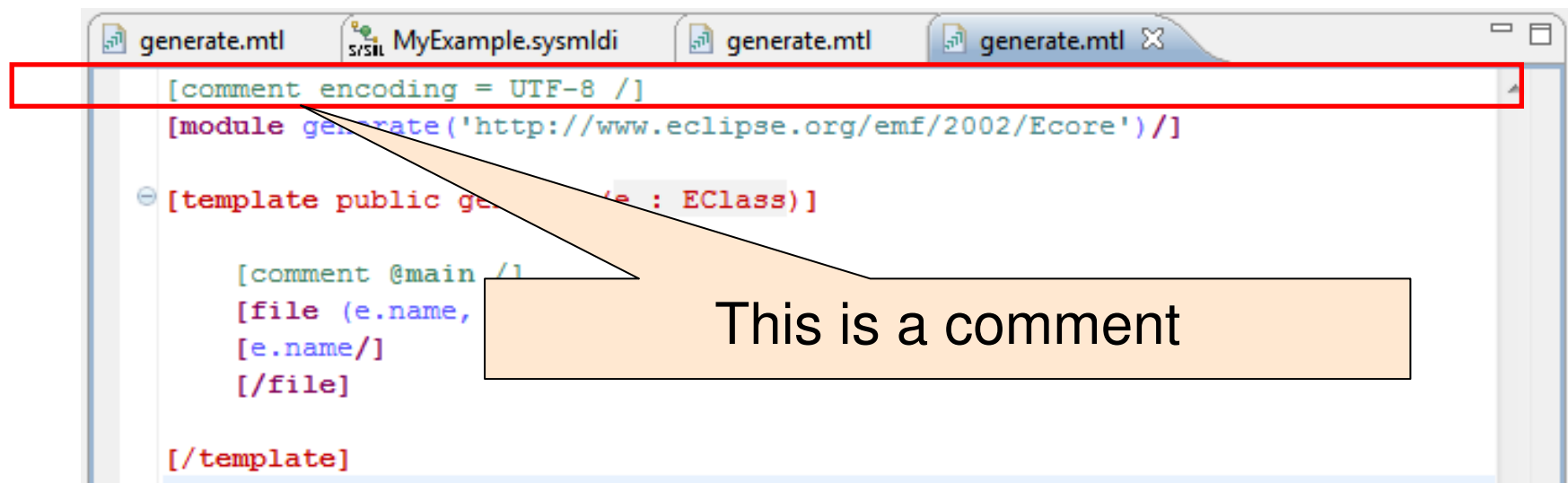
[template public generate(e : EClass)]

    [comment @main /]
    [file (e.name, false, 'UTF-8')]
    [e.name/]
    [/file]

[/template]
```

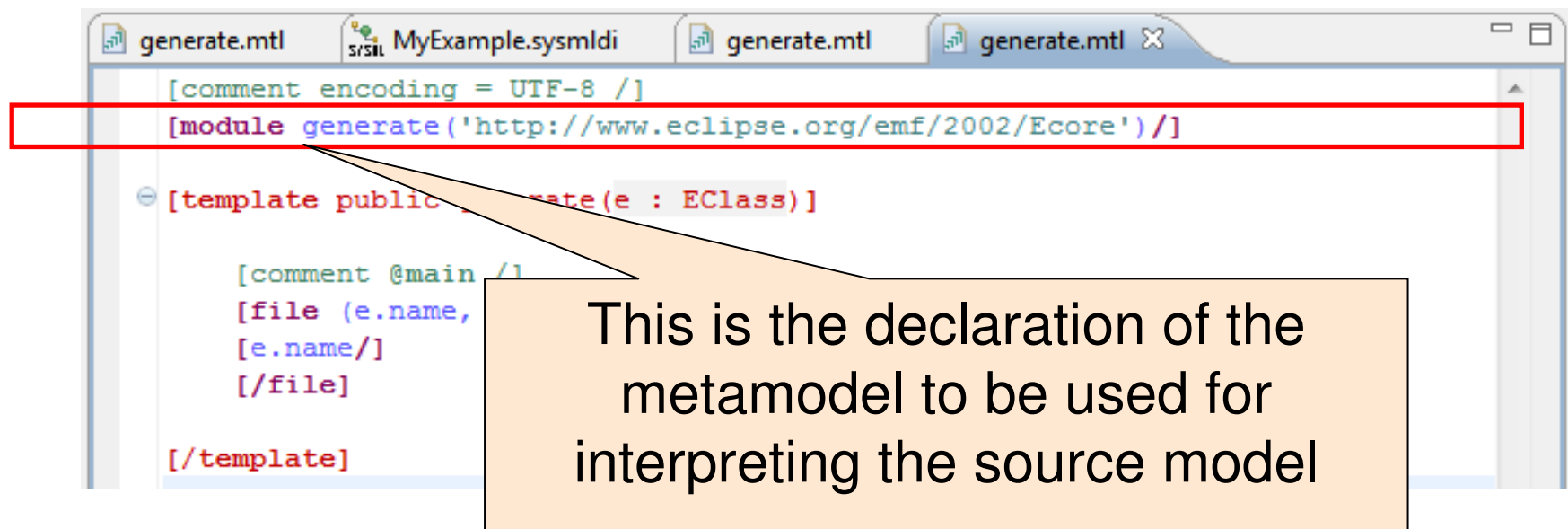
# The template language

- The language is intuitive, as in the first template module



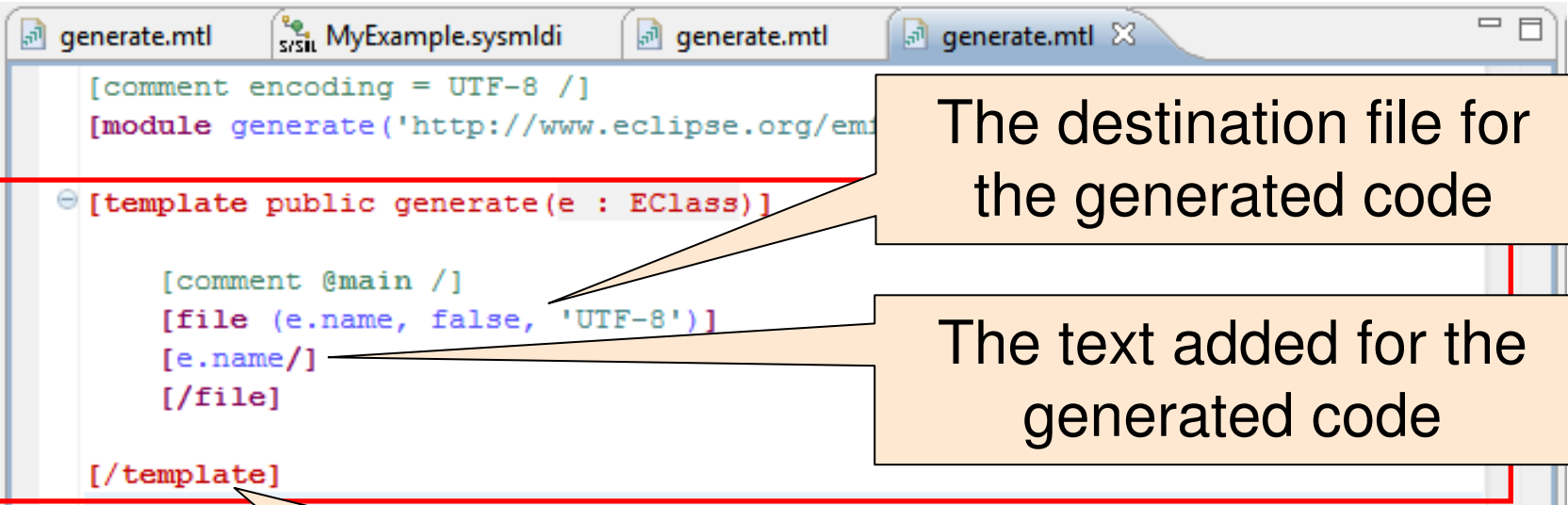
# The template language

- The language is intuitive, as in the first template module



# The template language

- The language is intuitive, as in the first template module



The screenshot shows an Eclipse IDE with a project named 'MyExample.sysmldi'. The editor displays a file named 'generate.mtl' containing the following code:

```
[comment encoding = UTF-8 /]  
[module generate('http://www.eclipse.org/emt')]  
[template public generate(e : EClass)]  
    [comment @main /]  
    [file (e.name, false, 'UTF-8')]  
    [e.name/]  
    [/file]  
[/template]
```

Three callouts explain the code:

- The destination file for the generated code**: Points to the `[file (e.name, false, 'UTF-8')]` line.
- The text added for the generated code**: Points to the `[e.name/]` line.
- This is the first template rule (there can be many of them)**: Points to the `[template public generate(e : EClass)]` line.

# The template language

- The language is intuitive, as in the first template module

The template rule operates on the **e** elements of type EClass

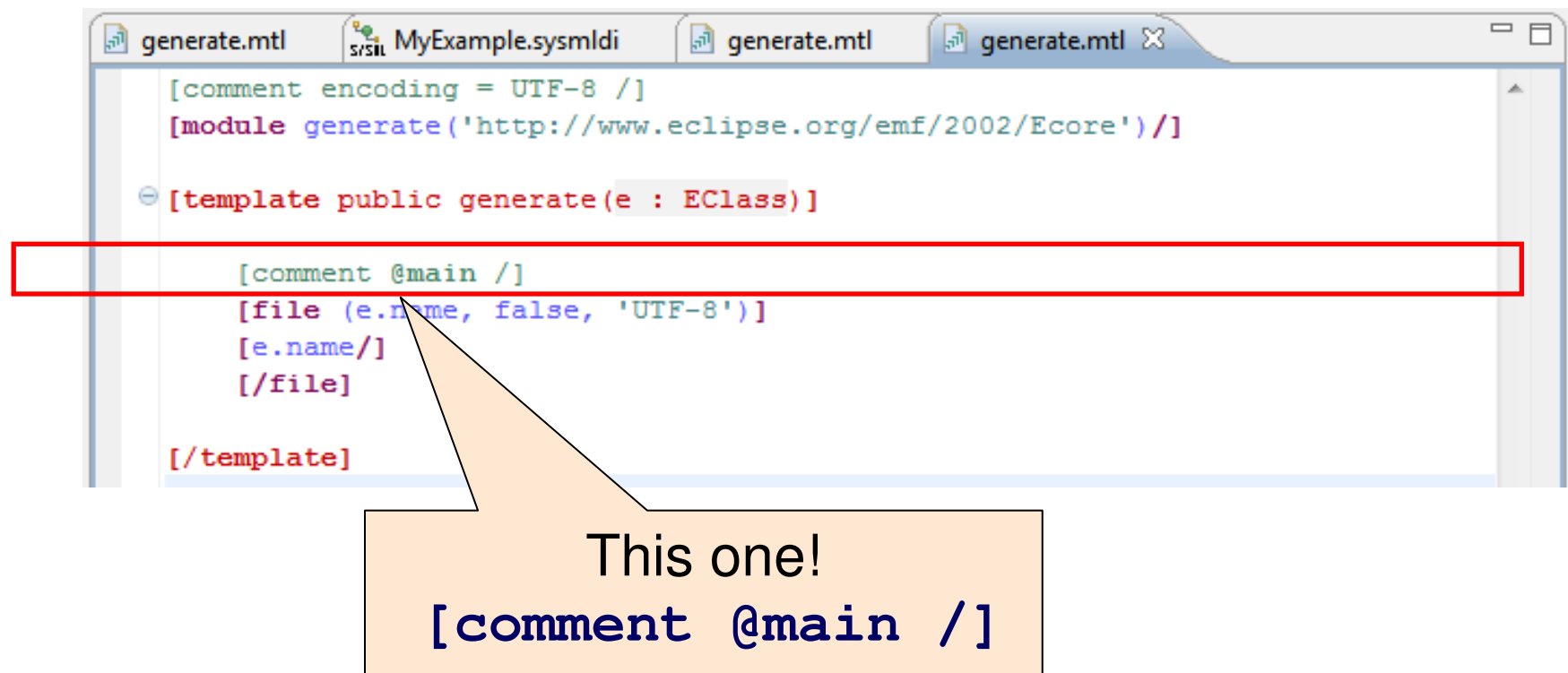
```
[module generate('http://www.eclipse.org/emt')  
  
  [template public generate(e : EClass)  
  
    [comment @main /]  
    [file (e.name, false, 'UTF-8')]  
    [e.name/]  
    [/file]  
  
  [/template]
```

The destination file for the generated code (name depends on **e.name**)

The text (code) to be produced inside it

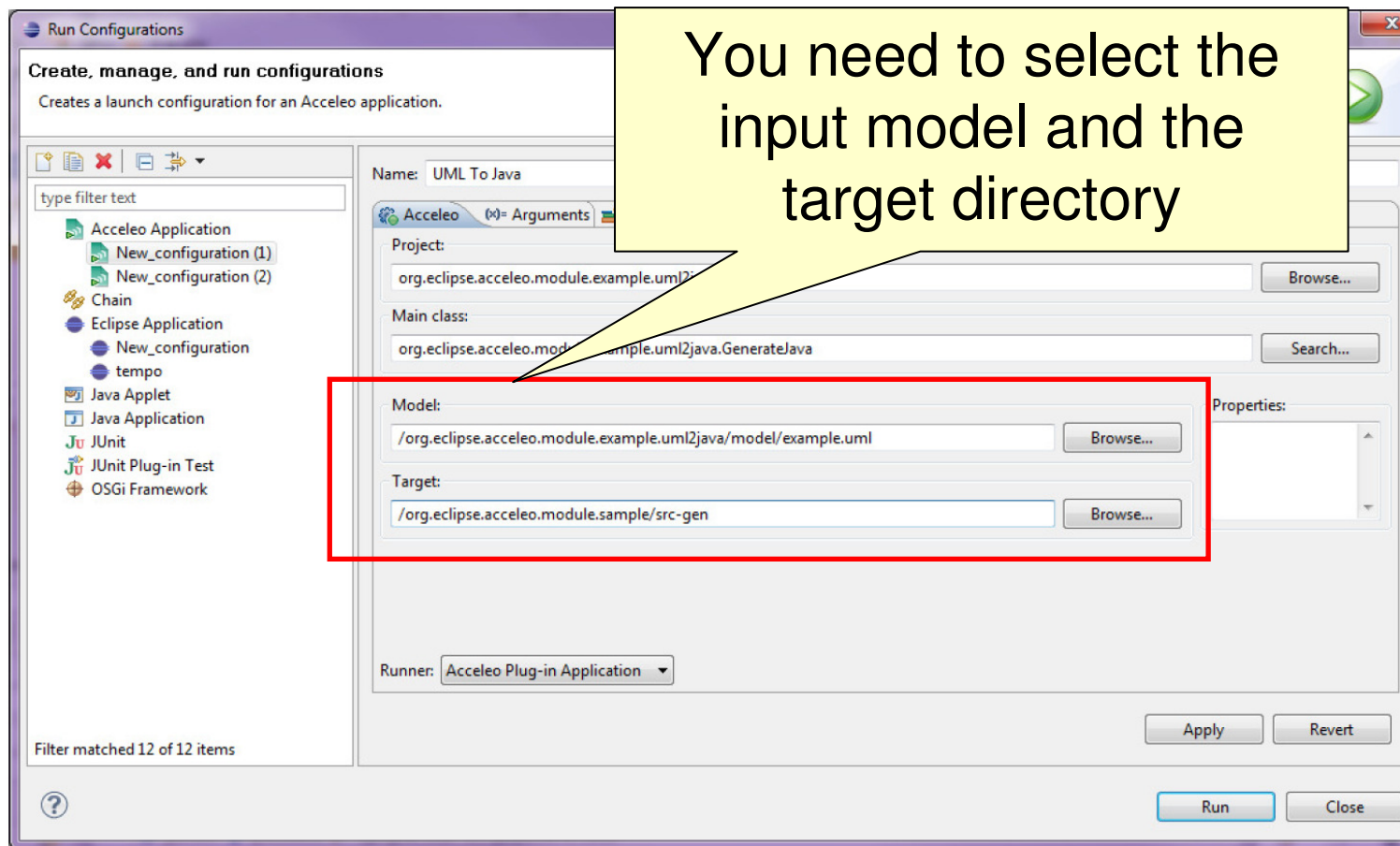
## The template language

- The acceleo module can be used as an entry point for text generation if it contains a special comment



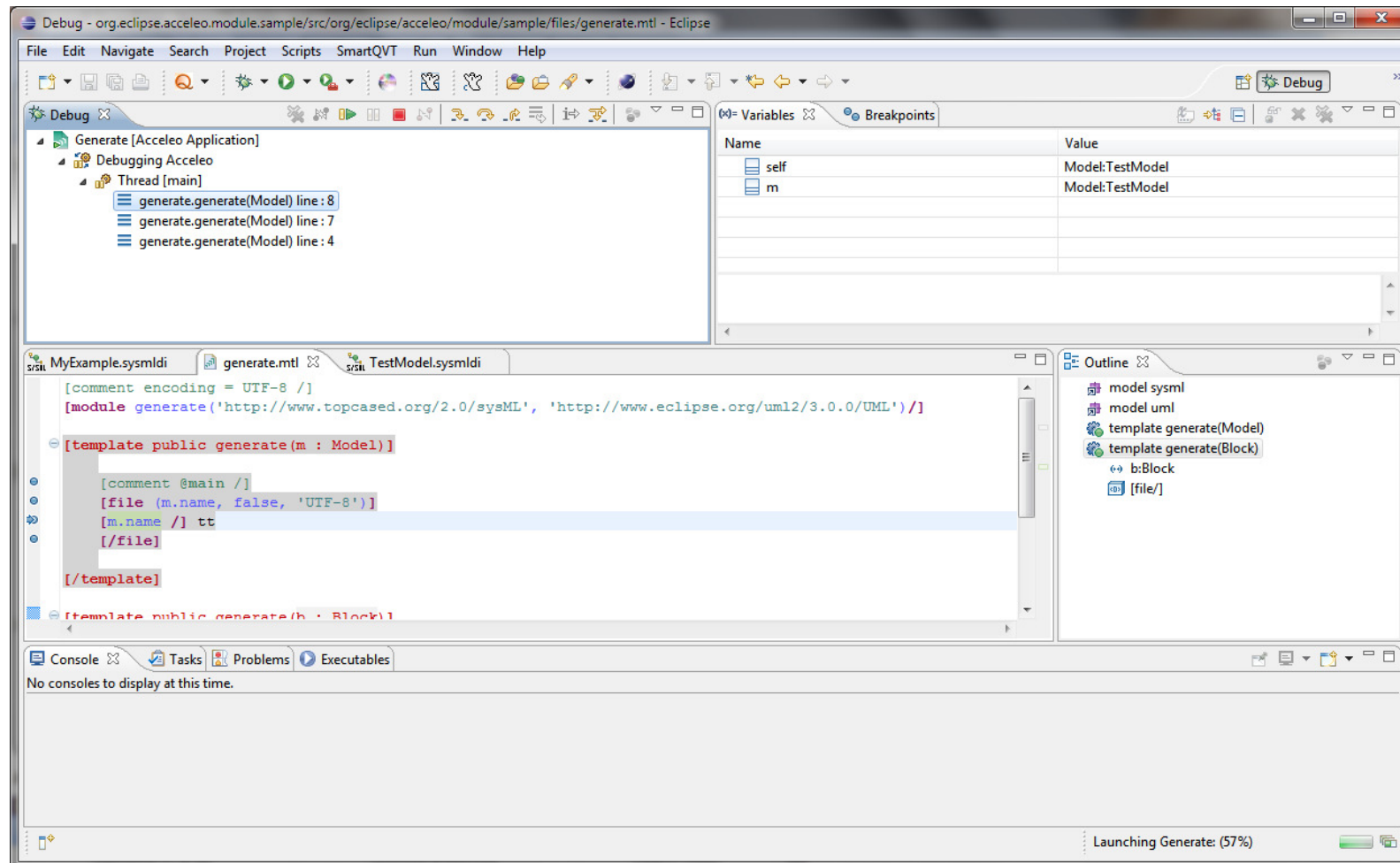
## Step 3: Running the Acceleo Converter

- Now, selecting Run Configurations, a selection window appears



## Step 4: Debugging an Acceleo Module

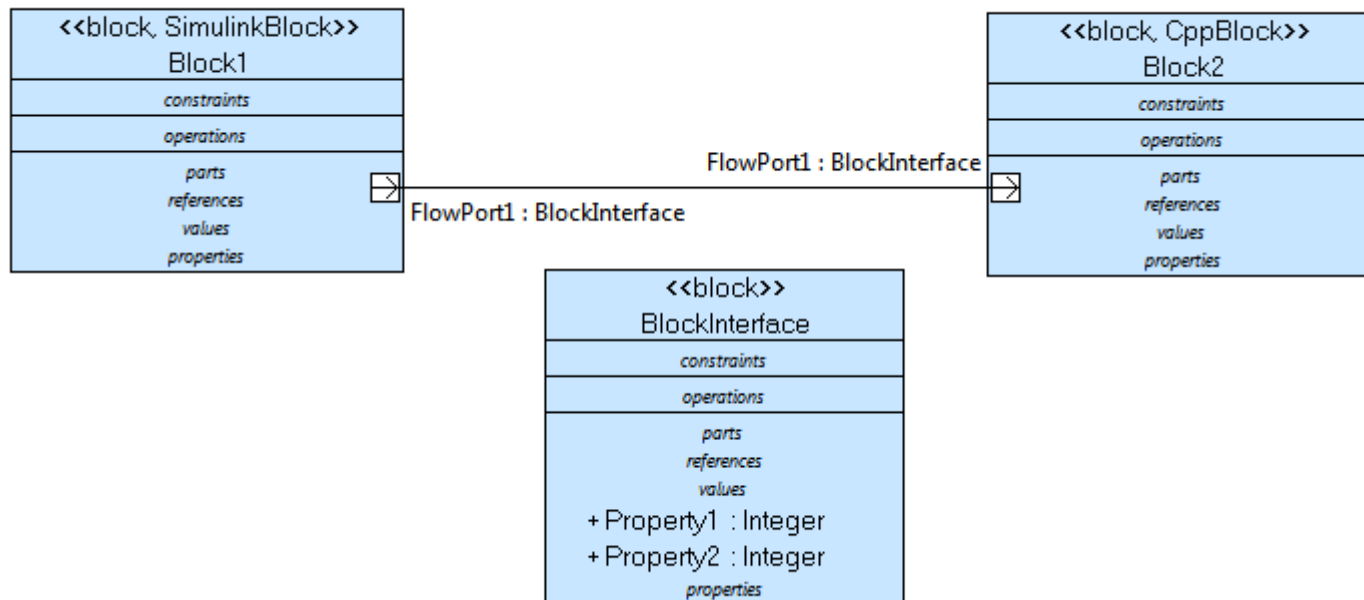
- You can select also the debug module





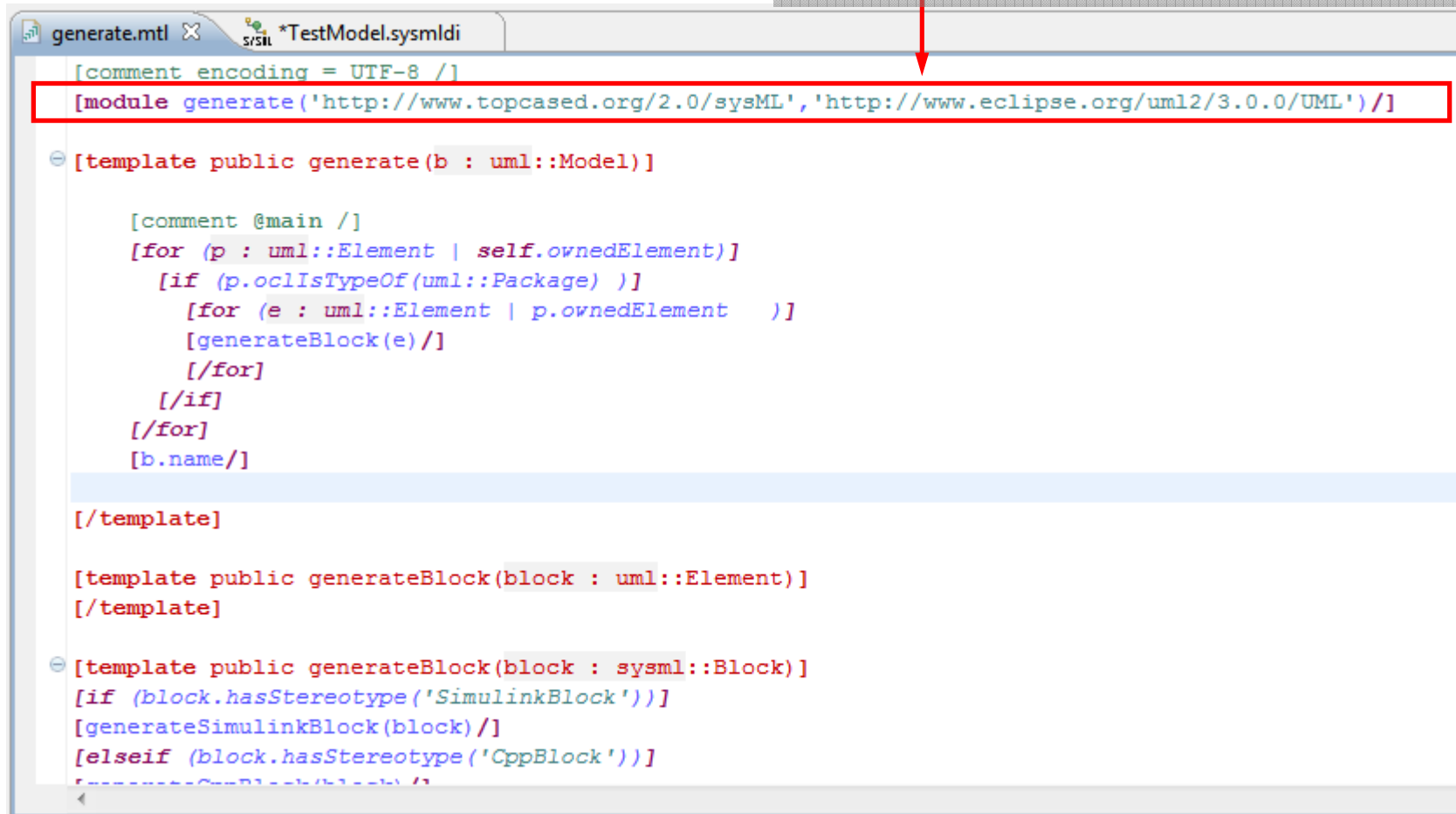
## An example (intro to language)

- A simple model with ports and stereotypes



# An example (intro to language)

**We are using the UML2.0  
and sysML metamodels**



```
generate.mtl  *TestModel.sysmldi
[comment encoding = UTF-8 /]
[module generate('http://www.topcased.org/2.0/sysML', 'http://www.eclipse.org/uml2/3.0.0/UML')/]

[template public generate(b : uml::Model)]

    [comment @main /]
    [for (p : uml::Element | self.ownedElement)]
        [if (p.oclIsTypeOf(uml::Package) )]
            [for (e : uml::Element | p.ownedElement )]
                [generateBlock(e)/]
            [/for]
        [/if]
    [/for]
    [b.name/]

[/template]

[template public generateBlock(block : uml::Element)]
[/template]

[template public generateBlock(block : sysml::Block)]
    [if (block.hasStereotype('SimulinkBlock'))]
        [generateSimulinkBlock(block)/]
    [elseif (block.hasStereotype('CppBlock'))]
        [generateCppBlock(block)/]
    [endif]
[/template]
```

## An example (intro to language)

**The first template: same name as module, it takes as argument the model itself**



```
[comment encoding = UTF-8 /]
[module generate('http://www.topcased.org/2.0/sysML', 'http://www.eclipse.org/uml2/3.0.0/UML')/]

[template public generate(b : uml::Model)]

    [comment @main /]
    [for (p : uml::Element | self.ownedElement)]
        [if (p.oclIsTypeOf(uml::Package) )]
            [for (e : uml::Element | p.ownedElement )]
                [generateBlock(e)/]
            [/for]
        [/if]
    [/for]
    [b.name/]

[/template]

[template public generateBlock(block : uml::Element)]
[/template]

[generateSimulinkBlock(block)/]
[elseif (block.hasStereotype('CppBlock'))]
[generateCppBlock(block)/]
```

**It is the main template**

## An example (intro to language)

**A for statement: iterates over all the ownedElements of self (the argument of the template – b).**

```
[comment encoding = UTF-8 /]
[module generate('http://www.topcased.org/2.0/sysML', 'http://www.eclipse.org/uml2/3.0.0/UML')/]

[template public generate(b : uml::Model)]

  [comment @main /]
  [for (p : uml::Element | self.ownedElement)]
    [if (p.oclIsTypeOf(uml::Package) )]
      [for (e : uml::Element | p.ownedElement )]
        [generateBlock(e)/]
      [/for]
    [/if]
  [/for]
  [b.name/]

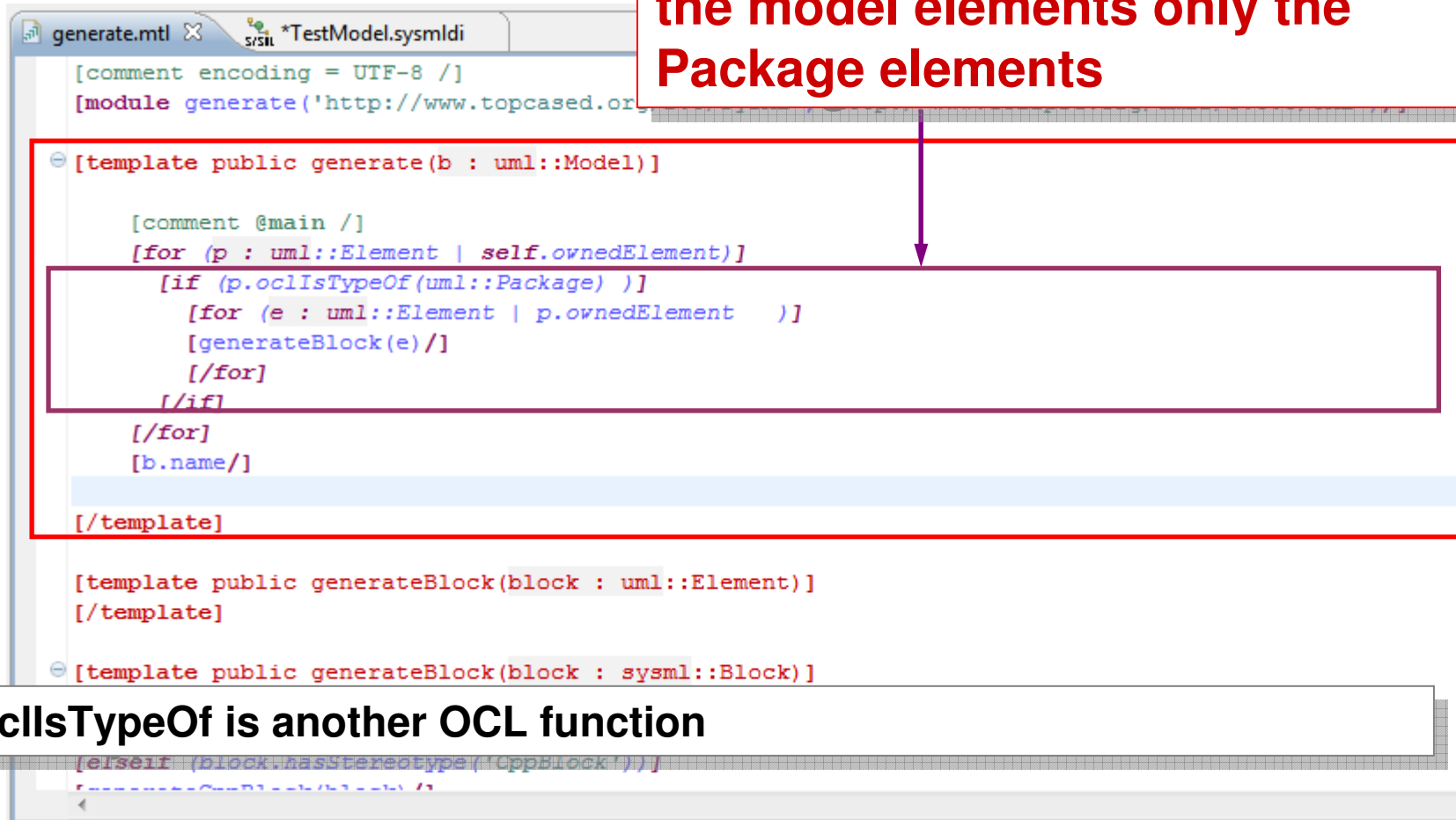
[/template]
```

**p is a generic UML Element (uml::Element refers to the uml metamodel namespace)**

**ownedElements is an OCL function – the Acceleo template language includes OCL functions for referring to the model structure**

## An example (intro to language)

**An if statement: selects over all the model elements only the Package elements**



```
[comment encoding = UTF-8 /]
[module generate('http://www.topcased.org')]

[template public generate(b : uml::Model)]

  [comment @main /]
  [for (p : uml::Element | self.ownedElement)]
    [if (p.oclIsTypeOf(uml::Package) )]
      [for (e : uml::Element | p.ownedElement )]
        [generateBlock(e)/]
      [/for]
    [/if]
  [/for]
  [b.name/]

[/template]

[template public generateBlock(block : uml::Element)]
[/template]

[template public generateBlock(block : sysml::Block)]

  [elseif (block.hasStereotype('OppBlock'))]
    [generateOppBlock(block) /]
  [/elseif]
[/template]
```

**oclIsTypeOf is another OCL function**

# Templates, Scripts and Services

- Templates
  - Main modules for text generation
  - They are polymorphic and can be overridden
  - The order of declaration of templates is important: The first template for which the guard condition is true is executed.
- Query
  - Functions that process model elements and extract information, that is values or Collections. (check boolean conditions)
  - They use OCL
  - Not polymorphic
- Services
  - Advanced processing custom written in Java

# Templates

(editor shortcut Ctrl-Space)

- Basic Syntax

```
[template public TemplateName() ]  
[/template]
```

- Additional features

- Overriding

```
[template public TemplateName() Overrides TemplateName ]
```

- Preconditions (guards)

```
[template public TemplateName() ? (bool_expr) ]
```

- Posttreatments

```
[template public TemplateName() post(postfun()) ]
```

- Variable initialization (possibly more than 1)

```
[template public TemplateName() {var:type = init;} ]
```

# Templates statements

## File tags

```
[file (out_file_uri, append_mode, encoding)] ... [/file]
```

## For loops

```
[for (iter_var : type | Collection)] ... [/for]
```

modifiers

`before()`

`separator()`

`after()`

## If condition

```
[if (condition)] ... [/if]
```

Let (assignment of final variables, value cannot change after init)

```
[let (var_name = init)] ... [/let]
```

## Comments

```
[comment] ... [/comment]
```



## Other postprocessing features (indentation)

### Post (postprocessing)

```
[template public name(pars) : post="trim()"]  
body  
[/template]
```

- `trim()` **removes** the leading and trailing spaces
- `indentSpace()`
- `indentTab()`

## Query format

```
[query public|private QueryName( par : type)
  : resType = queryBody /]
```

### Example

```
[query public hasStereotype( e : uml::Element, value
  : String) : Boolean =
  not e.getAppliedStereotypes()->select(e :
    uml::Stereotype | e.name = value)->isEmpty()
/]
```

# Acceleo Operators and OCL functions

## Categories

- ContextServices
- ENodeServices
- EObjectServices
- PropertiesServices
- RequestServices
- ResourceServices
- StringServices
- XpathServices

# Acceleo Operators and OCL functions

## ContextServices

`get (keyStr)` gets the object referenced by the key

`peek` gets the object on top of the context stack

`pop` removes the object on top of the context stack

`push` puts the object on top of the context stack

`put (keyStr)` adds the object to the context with the key

# Acceleo Operators and OCL functions

## ENodeServices

`toString`      cast to string if possible or provides object info

`nSort(expr)`    sorts the objects in a collection

`nReverse`

`nSize`            size of object

`nFirst`

`nGet(int)`

`nLast`

`nContains(ENode)`

`filter(typeStr)` returns the objects of the type indicated

`adapt(typeStr)`    type casting

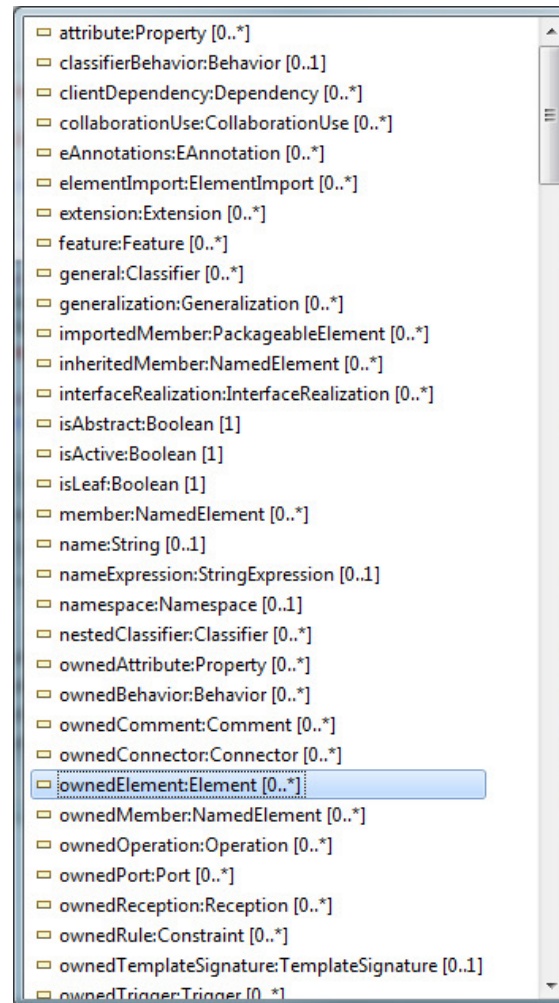
# Acceleo Operators and OCL functions

- Object attributes and operations can be easily found from the editor

Typing

[b.

(b of type  
uml::class)



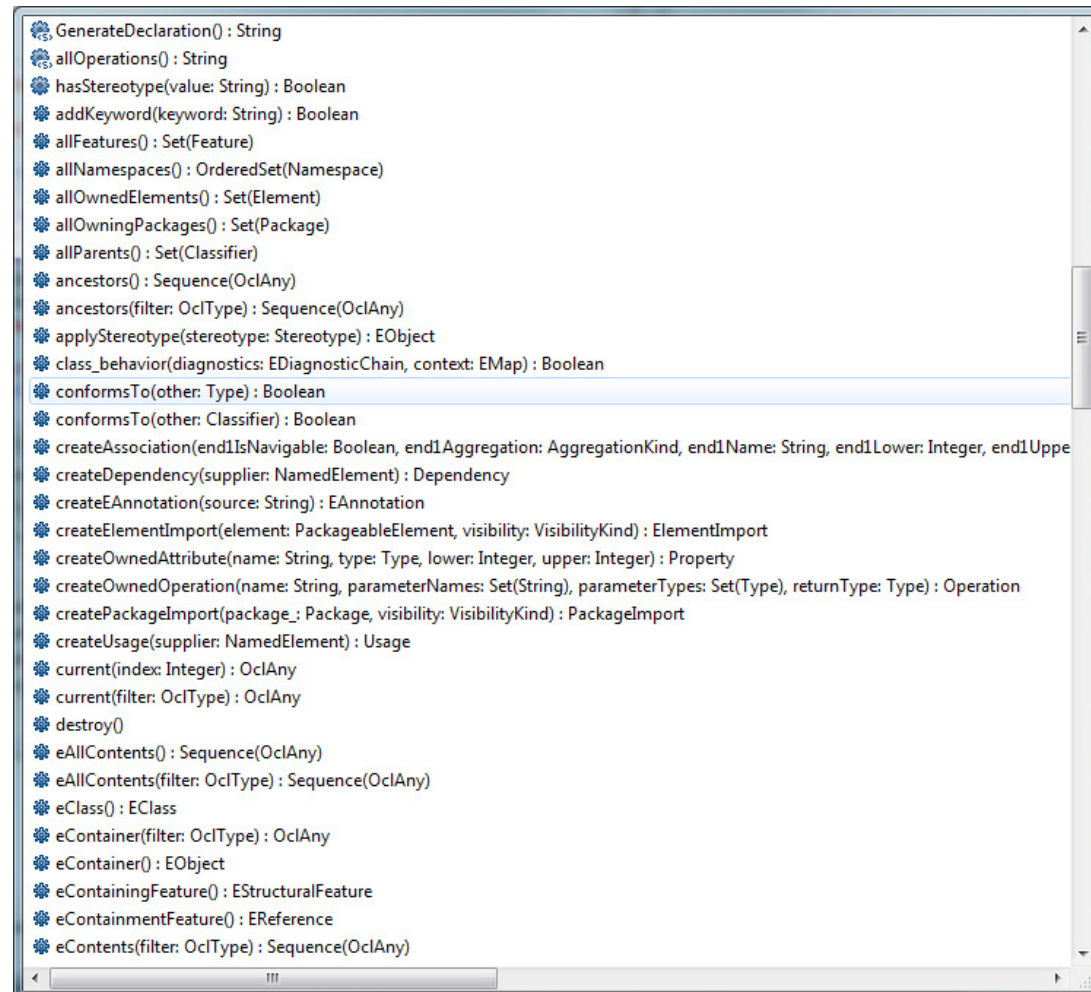
# Acceleo Operators and OCL functions

- Object attributes and operations can be easily found from the editor

Typing

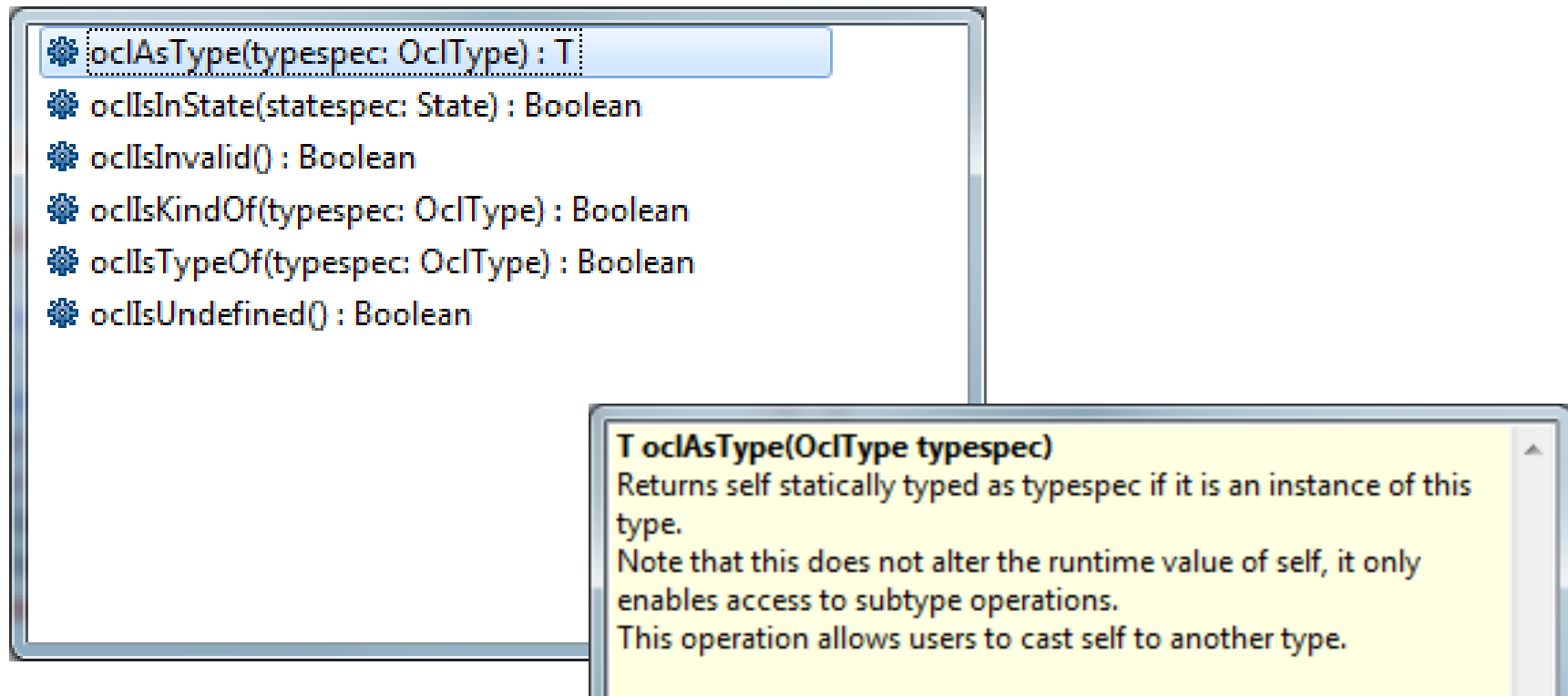
[b.

(b of type  
uml::class)



# Acceleo Operators and OCL functions

- Some are OCL functions (OCL is an OMG standard)



The image shows a screenshot of a software development environment. On the left, a list of OCL functions is displayed, each preceded by a gear icon. The first function, `oclAsType(typespec: OclType) : T`, is highlighted with a blue selection bar. To the right of this list, a tooltip window is open, providing details for the selected function. The tooltip has a yellow background and contains the function signature `T oclAsType(OclType typespec)`, followed by a description: 'Returns self statically typed as typespec if it is an instance of this type.' It also includes two additional notes: 'Note that this does not alter the runtime value of self, it only enables access to subtype operations.' and 'This operation allows users to cast self to another type.'

**oclAsType(typespec: OclType) : T**

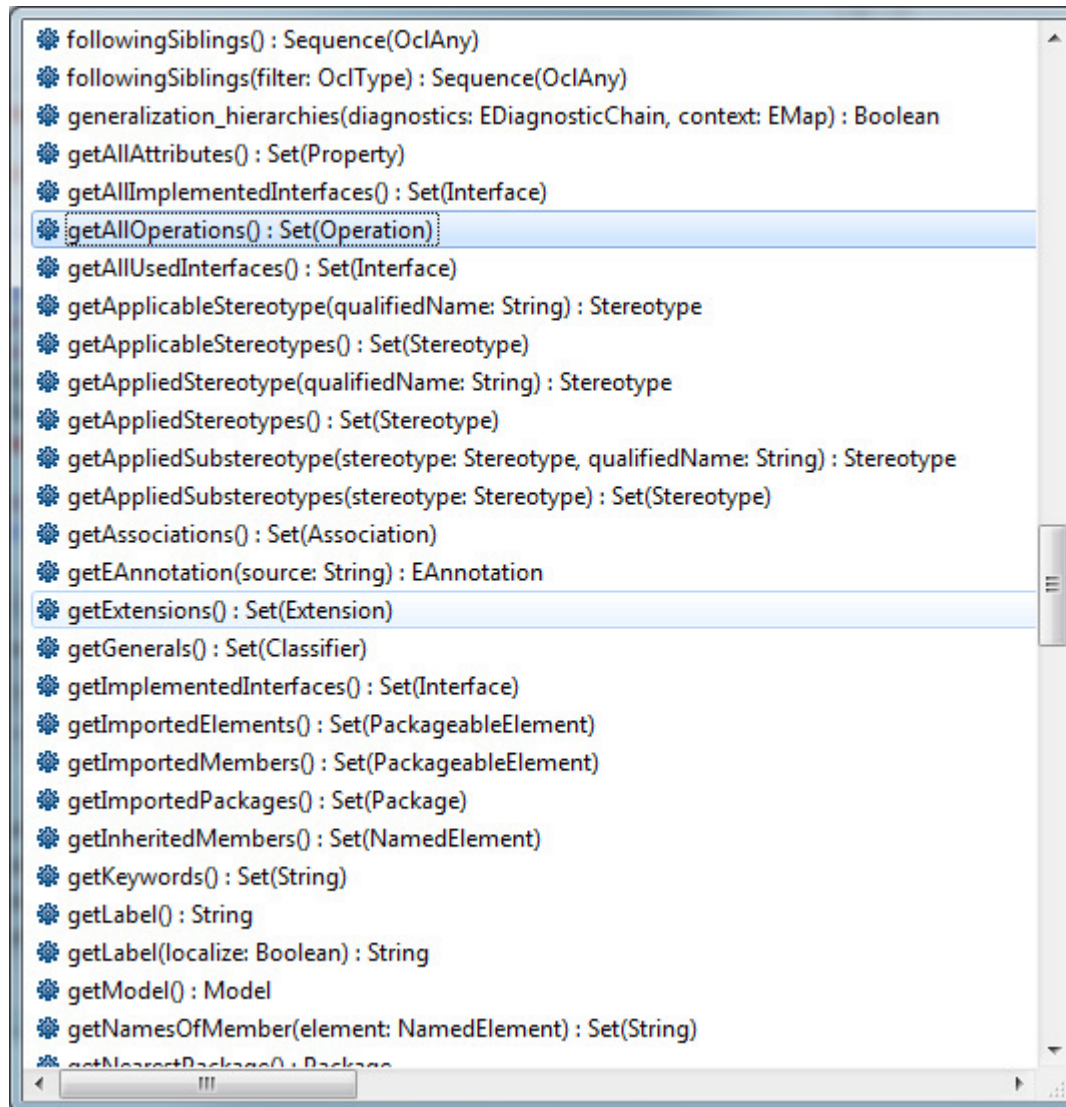
- oclIsInState(statespec: State) : Boolean
- oclIsInvalid() : Boolean
- oclIsKindOf(typespec: OclType) : Boolean
- oclIsTypeOf(typespec: OclType) : Boolean
- oclIsUndefined() : Boolean

**T oclAsType(OclType typespec)**  
Returns self statically typed as typespec if it is an instance of this type.  
Note that this does not alter the runtime value of self, it only enables access to subtype operations.  
This operation allows users to cast self to another type.



# Acceleo Operators and OCL functions

- Some are OCL functions (OCL is an OMG standard)



## Acceleo Operators and OCL functions

### EObjectServices

|                                     |  |
|-------------------------------------|--|
| <code>eAllContents</code>           | all contents   |
| <code>eAllContents (typeStr)</code> | all contents of the<br>given type                          |
| <code>eContents</code>              | the list of the immediate<br>children                      |
| <code>eContents (typeStr)</code>    | the list of the immediate<br>children<br>of the given type |
| <code>eContainer ()</code>          | the parent of the object                                   |
| <code>eClass ()</code>              | the class of the object                                    |
| <code>eContainer (typeStr)</code>   | the parent of the  |

# Acceleo Operators and OCL functions

## StringServices (examples)

`trim`

`toUpperCase`

`toLowerCase`

`toU1Case`

`substring(int, int)`

`startsWith(string)`

`split(string)`

`matches(regex)`

`replaceAll(string, string)`

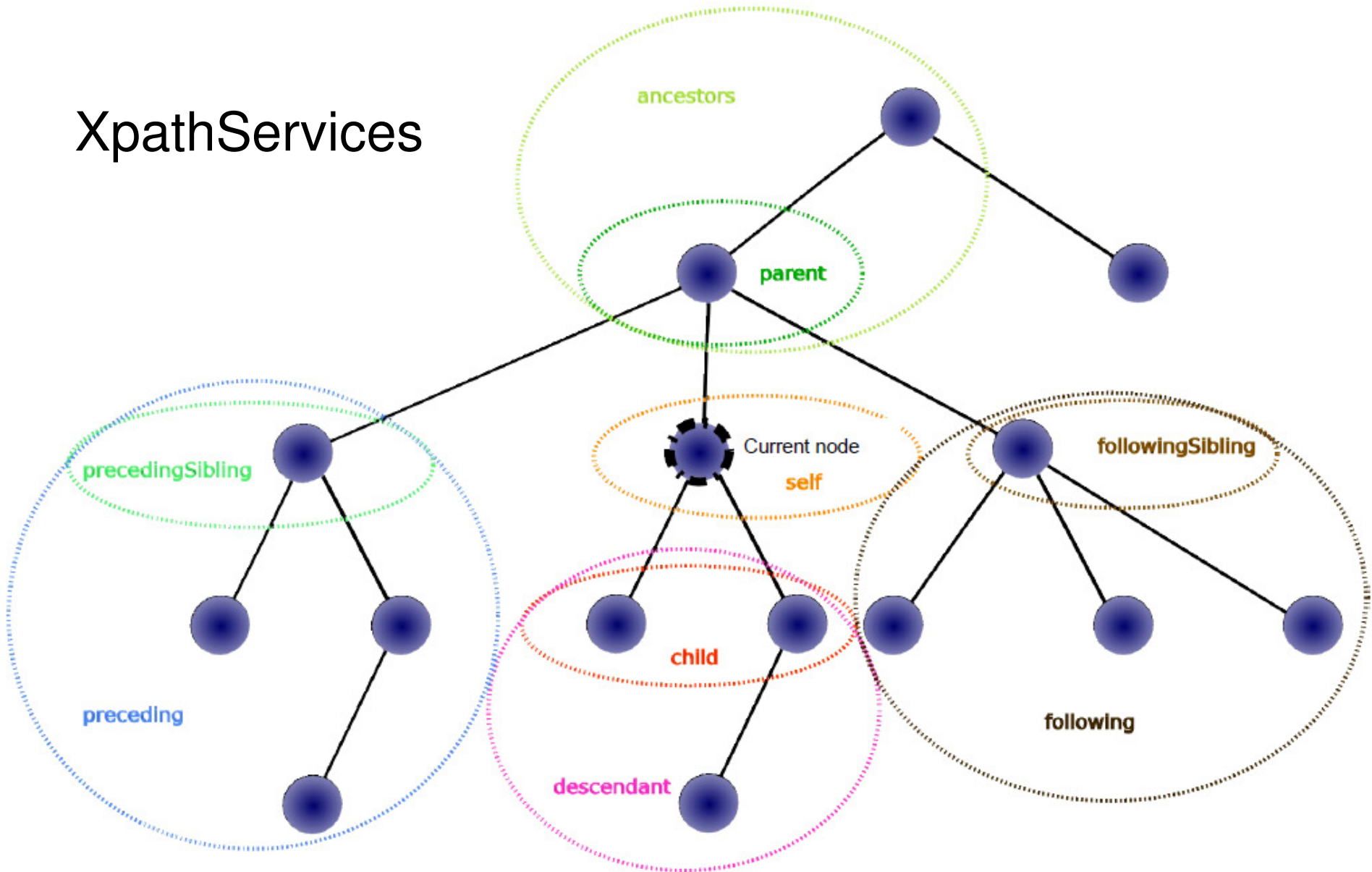
## Acceleo Operators and OCL functions

### XpathServices (examples)

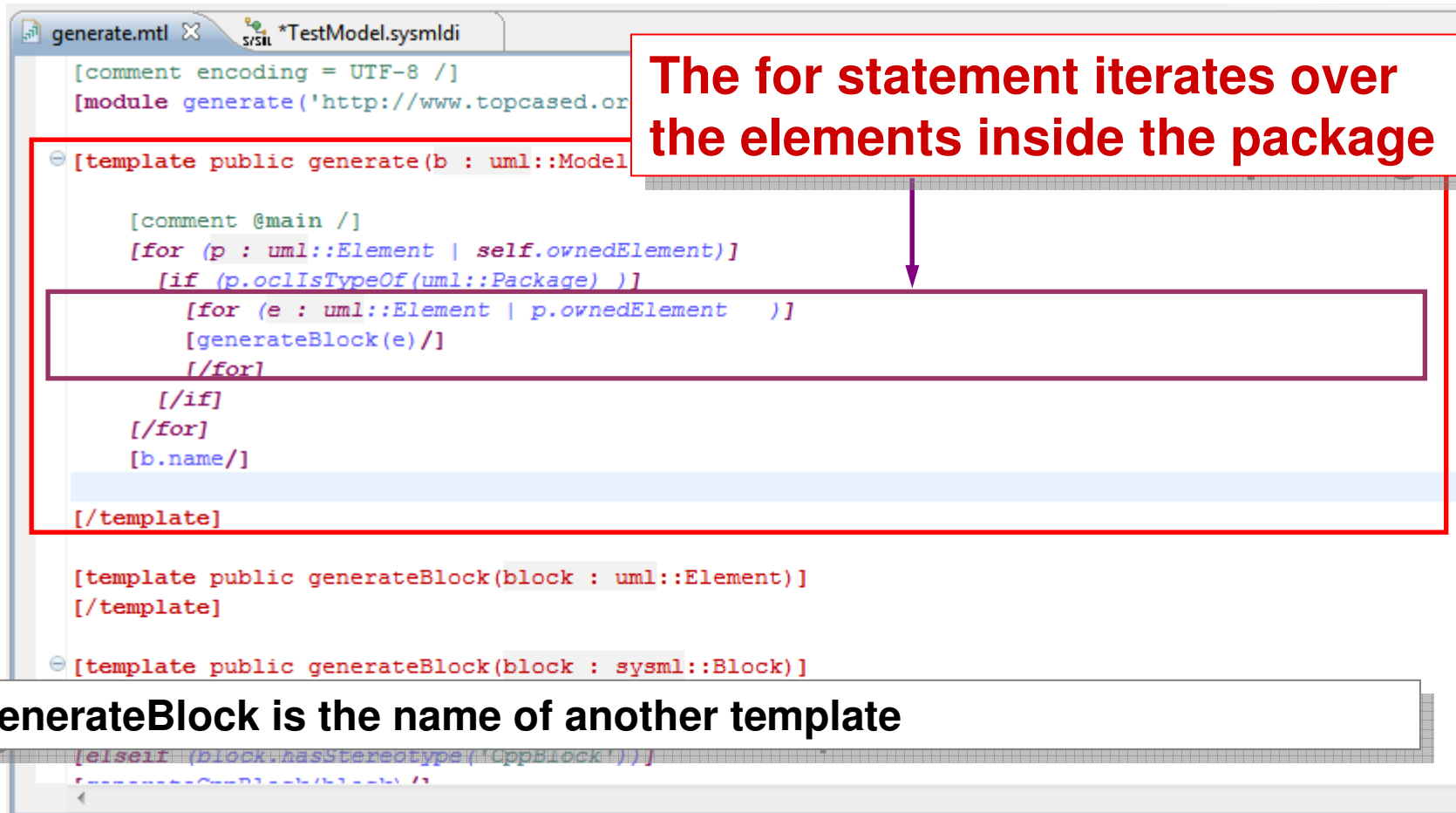
|                               |                                   |
|-------------------------------|-----------------------------------|
| <code>self</code>             | current object                    |
| <code>child</code>            | set of all the object children    |
| <code>descendant</code>       | set of all the object descendants |
| <code>parent</code>           | parent node                       |
| <code>followingSibling</code> | set of all brothers               |

# Acceleo Operators and OCL functions

## XpathServices



## An example (intro to language)



The screenshot shows a code editor with two tabs: 'generate.mtl' and '\*TestModel.sysmldi'. The code in 'generate.mtl' is as follows:

```
[comment encoding = UTF-8 /]
[module generate('http://www.topcased.org')]

[template public generate(b : uml::Model)]

  [comment @main /]
  [for (p : uml::Element | self.ownedElement)]
    [if (p.oclIsTypeOf(uml::Package) )]
      [for (e : uml::Element | p.ownedElement )]
        [generateBlock(e)/]
      [/for]
    [/if]
  [/for]
  [b.name/]

[/template]

[template public generateBlock(block : uml::Element)]
[/template]

[template public generateBlock(block : sysml::Block)]

  [elseif (block.hasStereotype('OppBlock'))]
    [generateOppBlock(block) /]
  [/elseif]
[/template]
```

Annotations on the code:

- A red box highlights the `[for (p : uml::Element | self.ownedElement)]` statement. A red text box to its right says: "The for statement iterates over the elements inside the package". A purple arrow points from this box to the nested `[for (e : uml::Element | p.ownedElement )]` statement.
- A purple box highlights the nested `[for (e : uml::Element | p.ownedElement )]` statement.
- A white box at the bottom says: "generateBlock is the name of another template".

# An example (intro to language)

This is executed for  
generic uml  
Elements

**There are two templates with the same  
name. Templates are polymorphic**

```
[template public generateBlock(block : uml::Element)]  
[/template]
```

```
[template public generateBlock(block : sysml::Block)]  
[if (block.hasStereotype('SimulinkBlock'))]  
[generateSimulinkBlock(block) /]  
[elseif (block.hasStereotype('CppBlock'))]  
[generateCppBlock(block) /]  
[else]  
[generateTypeBlock(block) /]  
[/if]  
[/template]
```

This is executed for  
sysML Blocks

**Inside, the generateBlock makes use of the OCL function  
hasStereotype and invokes other templates**

## An example (intro to language)

**The template generates the .h file for a special type of block (stereotyped as SimulinkBlock)**

```
[template public generateSimulinkBlock(block: sysml::Block)]  
[file ( 'Block'+ block.name + '.h', false, 'UTF-8')]  
#ifndef _BLOCK_[block.name.toUpper() /]  
#define _BLOCK_[block.name.toUpper() /]  
class Block[block.name /]  
{  
  public:  
  [for (port : uml::Port | block.ownedPort)]  
    [generatePort(port) /];  
  [/for]  
};  
  
#endif  
[/file]  
[/template]
```

Everything inside these two lines is text with destination in the given file

It is a mixture of free text (in black) that is copied “as is” in the file and template directives (between [ ] )



## An example (intro to language)

```
[template public generateSimulinkBlock(block: sysml::Block)]  
[file ( 'Block'+ block.name + '.h', false, 'UTF-8')]  
#ifndef _BLOCK_[block.name.toUpper() /]  
#define _BLOCK_[block.name.toUpper() /]  
class Block[block.name /]  
{  
    public:  
    [for (port : uml::Port | block  
        [generatePort(port) /];  
    [/for]  
};  
  
#endif  
[/file]  
[/template]
```

The template refers to attributes of the block object (its name)

# An example (intro to language)

```
[template public generateSimulinkBlock(block: sysml::Block)]  
[file ( 'Block'+ block.name + '.h', false, 'UTF-8')]  
#ifndef _BLOCK_[block.name.toUpper()/]  
#define _BLOCK_[block.name.toUpper()/]  
class Block[block.name/]  
{  
  public:  
  [for (port : uml::Port | block.ownedPort)]  
    [generatePort(port) /];  
  [/for]  
};  
  
#endif  
[/file]  
[/template]
```

• The template has a for cycle that iterates over the ports of the block (calling another template)

• For a flow port an attribute declaration is generated

```
[template public generatePort(port:uml::Port)]  
[/template]  
  
[template public generatePort(port:sysml::FlowPort)]  
  [port.getPortPrefix()/] [port.getPortTypeName()/] [port.name/];  
[/template]
```

## An example (intro to language)

```
#ifndef _BLOCK_BLOCK1
#define _BLOCK_BLOCK1
class BlockBlock1
{
    public:
        OutBlockInterface FlowPort1;
};

#endif
```

This is the  
generated file