



Document generation tutorial

Table of contents

1	Installation procedure	4
2	Default generation from a Papyrus model	4
3	Configure Document Generator in the Workbench	4
4	DOCX and ODT Document Generator.....	8
4.1	Creation of a document generator.....	8
4.2	Configure the generation : <config> tag	8
4.2.1	Define generation output	8
4.2.2	Define global parameters for the template.....	9
4.2.3	Pre-defined parameters	9
4.2.4	Use of variables inside parameters	10
4.3	Define script execution context : <context> tag.....	11
4.3.1	Dealing with specific models.....	11
4.4	Define script parts : <gendoc> tag :	12
4.4.1	Script language	12
4.4.2	Text generation.....	13
4.4.3	Images generation	14
4.4.4	Table generation	17
4.4.5	Papyrus and Sirius Tables.....	18
4.4.6	Bookmarks and hyperlinks generation	23
4.4.7	Rich text generation	24
4.4.8	Enclose the external document.....	25
4.4.9	Formatting.....	25
4.4.10	Listing elements.....	27
4.5	Reusing gendoc scripts inside the same document : <fragment> tag	28
5	XLSX Document Generator.....	29
5.1	Creation of a document generator.....	29
5.2	Configure the generation: <config> tag	29
5.2.1	Define generation output	30
5.2.2	Define global parameters for the template.....	30
5.2.3	Pre-defined parameters	31
5.2.4	Use of variables inside parameters	31
5.2.5	Variables stored in another file.....	32
5.2.6	Context with CDO models	32
5.3	Define script execution context : <context> tag.....	32
5.3.1	Dealing with specific models.....	33
5.4	Define script parts: <gendoc> tag :	33
5.4.1	Script language	34
5.4.2	Text generation.....	35
5.4.3	Images generation	35

5.4.4	Formatting.....	41
5.4.5	Reusing gendoc scripts inside the same document: <fragment> tag	45
6	PPTX Document Generator.....	46
6.1	Creation of a document generator.....	46
6.2	Configure the generation: <config> tag	47
6.2.1	Define generation output	47
6.2.2	Define global parameters for the template.....	47
6.2.3	Pre-defined parameters	48
6.2.4	Use of variables inside parameters	48
6.2.5	Variables stored in another file.....	49
6.2.6	Context with CDO models	49
6.3	Define script execution context : <context> tag.....	49
6.3.1	Dealing with specific models.....	50
6.4	Define script parts: <gendoc> tag :	50
6.4.1	Script language	51
6.4.2	Text generation.....	52
6.4.3	Images generation	53
6.4.4	Displaying diagrams.....	57
6.4.5	Formatting.....	58
6.5	Reusing gendoc scripts inside the same document: <fragment> tag	64
7	Command Line Interface	66
8	Gendoc bundles	66
8.1	Commons.....	66
8.1.1	Advanced services from bundle "commons"	67
8.2	HTML.....	68
8.3	Gmf	68
8.3.1	Advanced services concerning gmf diagrams	69
8.4	Papyrus	69
8.5	Capella.....	70
	APPENDIX: Overview of all Gendoc tags and attributes	70

1 Installation procedure

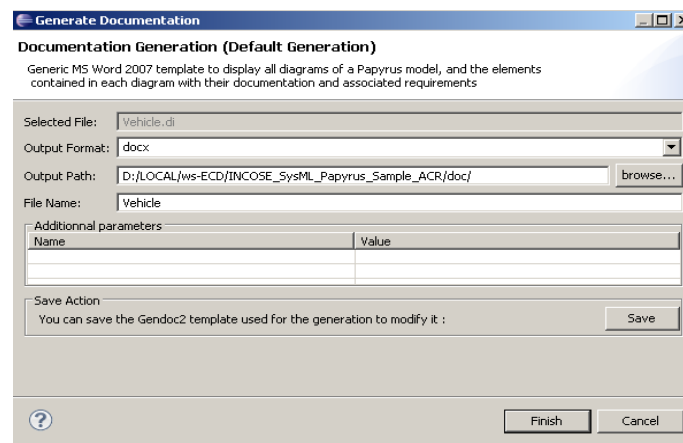
Get last Gendoc update site from [Gendoc downloads page](#).

Adding a software update site: see Eclipse.org [online help](#)

2 Default generation from a Papyrus model

A default simple template is available for any model

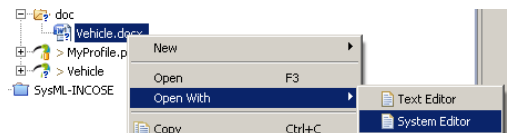
From the Papyrus model, right click > Generate documentation with Gendoc, adjust generation output and OK



Recommendation: use a dedicated “doc” directory

Refresh project (F5)

Open output doc with System editor



All diagrams are available with associated documentation. Each element of diagram that has associated documentation is listed in a paragraph with its documentation.

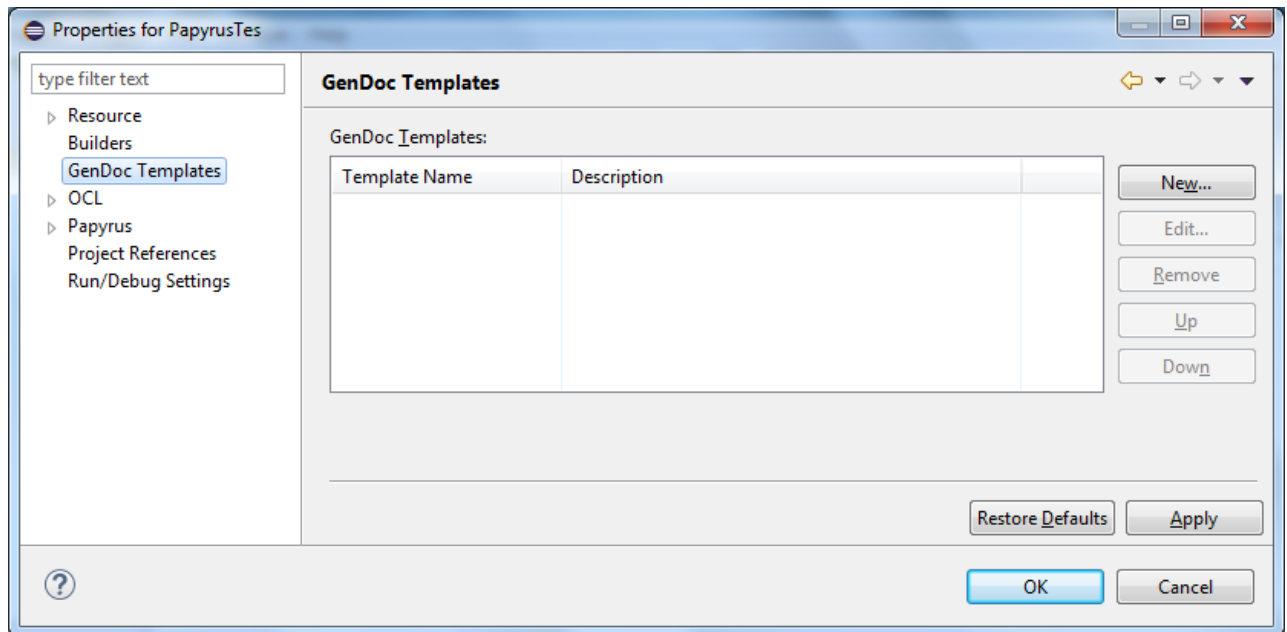
Notes : You have to manually update table of contents

3 Configure Document Generator in the Workbench

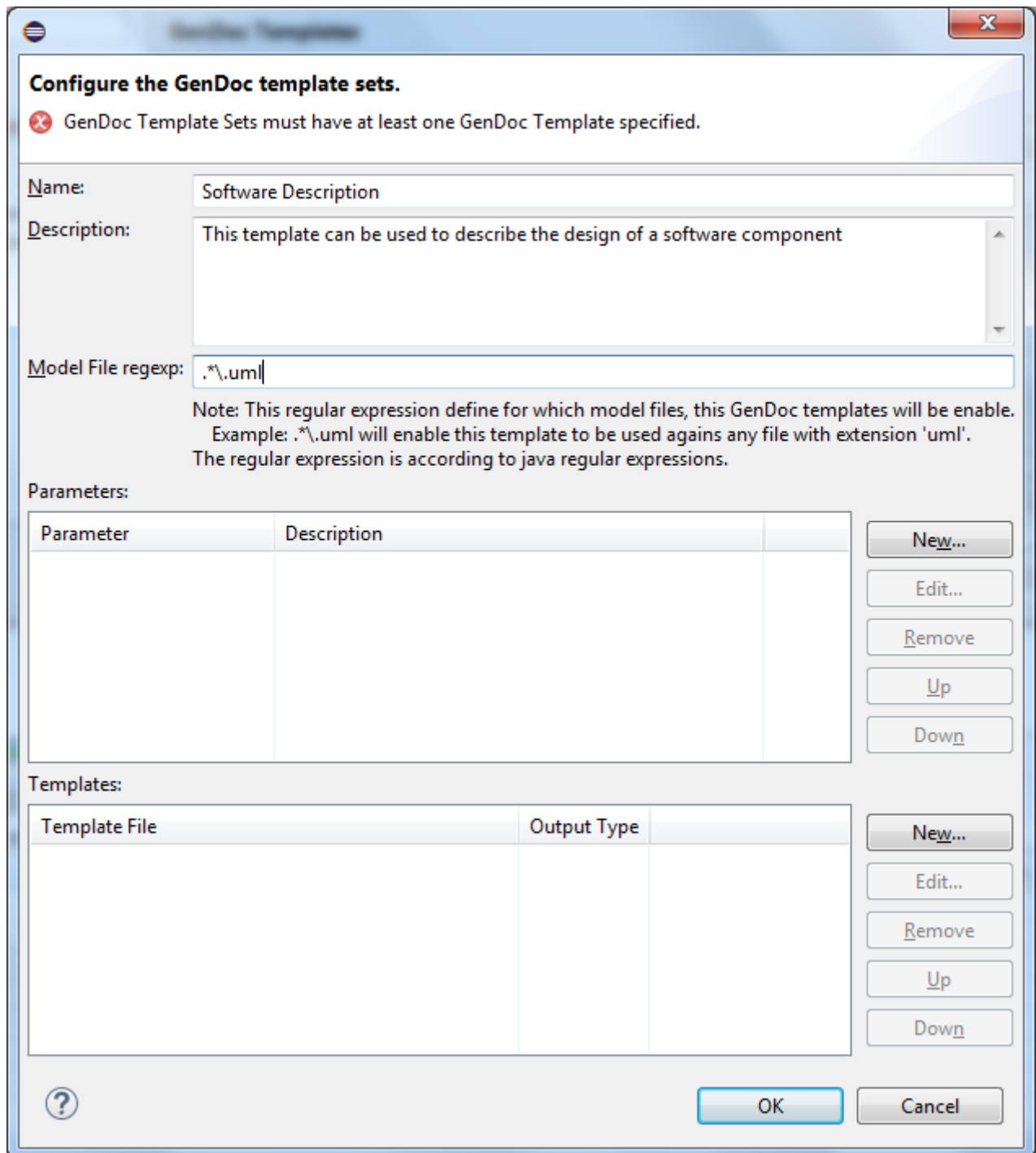
It is possible to configure local templates to be used in the Gendoc Wizard. Once the template is configured, the template will be available in the combo box of the Gendoc Wizard. The templates can be configured to be available for the entire workspace, or per project. The configuration specific to a project is stored together with the project file (.gendoc file in the project) and can be shared if using any source version control system. The workspace gendoc configuration is local to the workspace and it is done through the workspace preferences.

To configure templates in the workspace, open the preferences (**Window > Preferences**) and select GenDoc Templates.

To configure templates for a project, open the properties for the project (**Project > Properties**):



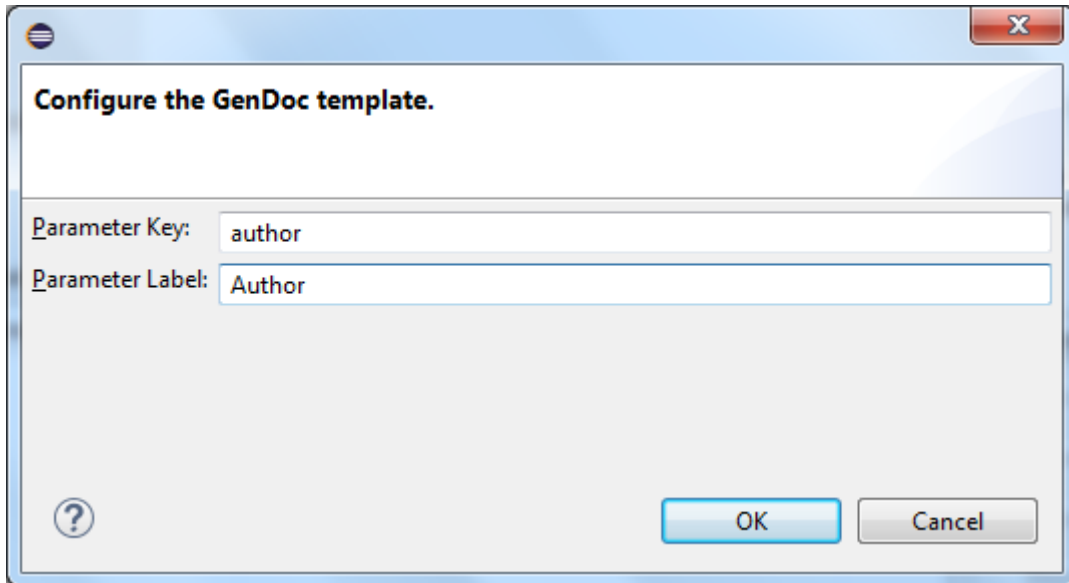
To add a new template click on the button **New...** and provide the information in the dialog:



- **Name:** A descriptive name for the template.
- **Description:** Information about the template, what it provides, what type of information will contain, in which case can the template be used for, etc...
- **Model File regexp:** A java regular expression that match the model files to which this template is applicable. It can be a filename also.
 - For papyrus files: `.*\\.uml`.
 - For a ecore model: `.*\\.ecore`

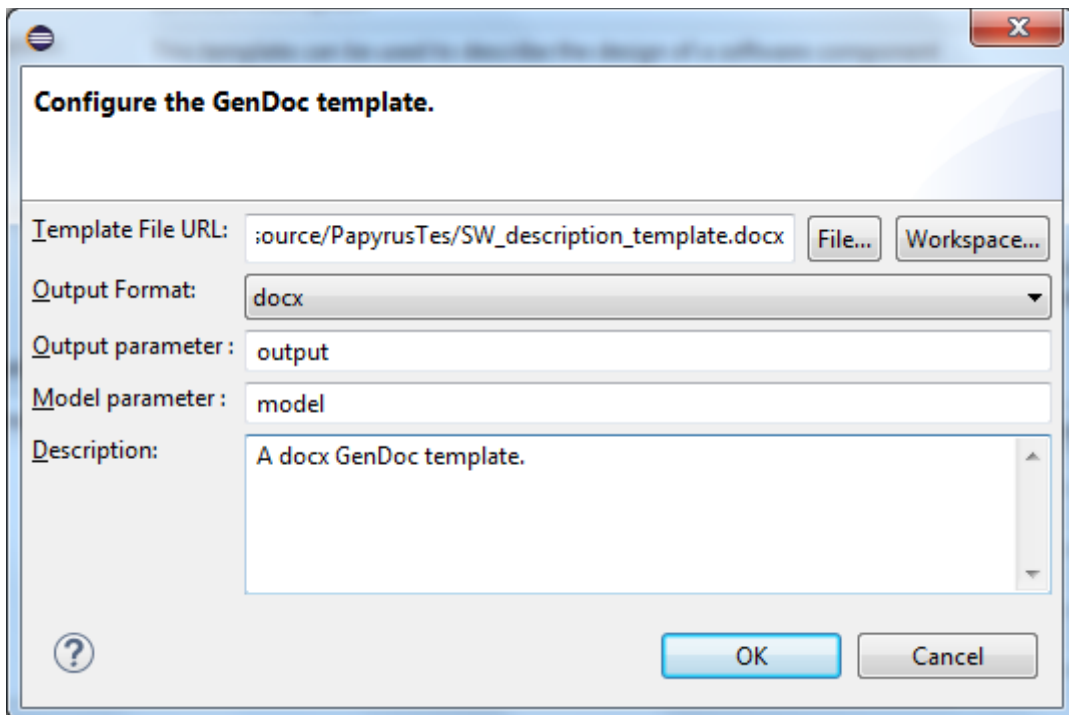
It is possible to configure parameters to be provided when launching the generation. The value for those parameter will be required by the wizard before launch the generation of the

document. To add a new parameter for this generator, click the button **New...** in the section **Parameters**. A dialog will be displayed to provide the new parameter information:



- **Parameter Key:** The name of the variable that will be available in the template scripts parts by using `${parameter_key}` to refer to the value of the parameter provided in the Gendoc wizard.
- **Parameter Label:** The label the Gendoc wizard will for this parameter.

The last information required, it is the template file itself. Click on the **New...** button in the **Templates** section. A dialog will request some information to catalog the template file.



- **Template File URL:** The URL to the template file. It could be a local file (`file:/...`), workspace file (`platform:/resource/`), or any other url kind supported by eclipse.

- **Output Format:** As gendoc support different output file formats, we indicate the output format of the selected template.
- **Output parameter:** The output file is required in the **<output>** tag. In order to be able to select the output file in the Gendoc wizard, a variable is used and the template should configure the output tag as: `<output path="${output_variable}">`. This *output_variable* is the one to set here.
- **Model parameter:** Similar as for the Output Parameter, but for the *model* attribute in the **<context>** tag, `<config model="${model_variable}"...>`.
- **Description:** A description for the template file.

Multiple template files can be added for different output formats. Which one to use, can be selected in the Gendoc Wizard.

Now the template should be available when launching the document generation for any model in the project (or in the workspace if it is a template configured in the gendoc workbench preferences.).

4 DOCX and ODT Document Generator

4.1 Creation of a document generator

- **Create a new document** in MS Office 2007+ (.docx) or OpenOffice Writer (.odt) format or get an existing document (with the company charter for example) in one of these formats
- Define **static parts** that can be : paragraphs with styles, images, document agenda
- **Identify dynamic parts** with `<gendoc>` tags
- Just adapt configuration parameters in template header :
 - Model path
 - Output file path
- Generate with a right click menu
- As it is an iterative process, you can do it whenever you want

Note: Generation can also be launched in batch mode .

4.2 Configure the generation : `<config>` tag

The tag **<config>** must be defined only once, on top of the template document.

This tag defines the path of the output document, and a list of global parameters for the template.

4.2.1 Define generation output

`<output>` tag is optional. If not present, the document is generated at template location, with suffix '_generated'

If defined, the syntax is the following:

<code><config></code>

```

        <output path=<<Absolute path of the document to be generated>>
/>
    ...
</config>

```

Global parameters can be used to define a relative path.

Example : The generated document will be located in D:/generatedFile.docx

```

<config>
    <output path='D:/generatedFile.docx' />
    ...
</config>

```

4.2.2 Define global parameters for the template

Global parameters for the template can be defined, for example to define model path, folders to use or any other static value to be used in template.

Parameters are defined in <config> tag with the following syntax:

```

<config>
    ...
    <param key=<<Parameter1_key>> value=<<Parameter1_value>> />
    <param key=<<Parameter2_key>> value=<<Parameter2_value>> />
    <param ... />
</config>

```

How to access parameters?

- `${paramKey}` inside <context> or other <param> ,
- `gGet(paramKey)` inside a <gendoc> tag

Example : creation of global parameters for model folder, model path, and path of a specific package inside model and example of usage in <context> tag.

```

<config>
    <param key='model_path' value='D:/Models/Model_v1/My_model.uml' />
    <param key='UC_package_path' value='/MyUMLModel/UseCases' />
</config>
<context model='${model_path}' element='${UC_package_path}' />

```

4.2.3 Pre-defined parameters

Some <param> are pre-defined in Gendoc and can be used directly in the template

- `${input}` is the name of the input template document

Example :

```

<param key='generation_folder' value='D:/Generated' />
<output path='${generation_folder}/${input}-generated.docx' />

```

If the input document is named `template1.docx`, the result file is named `template1-generated.docx`.

The following variables are also ready to be used by default:

- `${date}` is the date of the generation. The format of the date is 'yyyy-MM-dd-HH:mm:ss'
- `${input_directory}` location directory of the template

Example:

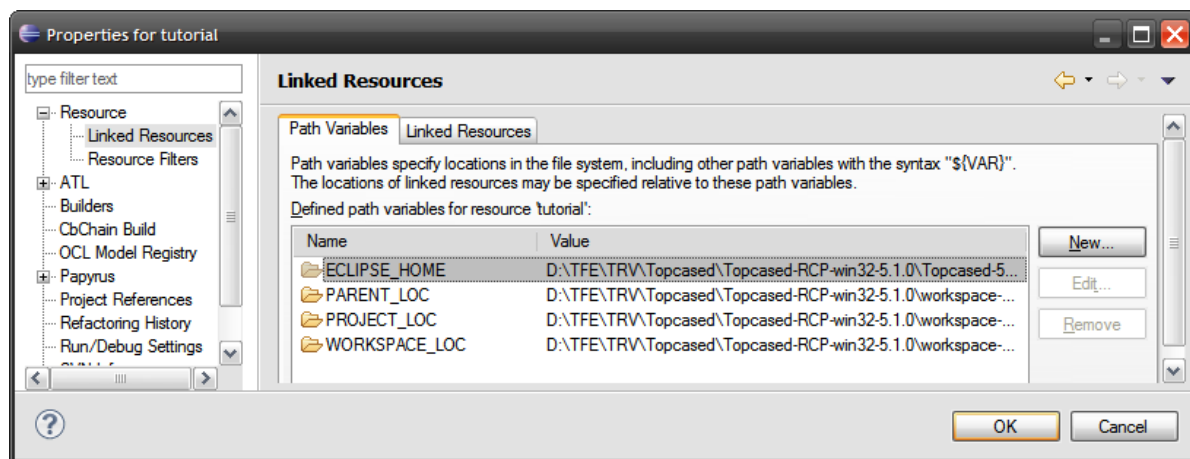
```
<output path='${input_directory}/${input}-generated-${date}.docx' />
```

Result file example: `template-generated-2014-08-02-093707.docx`

4.2.4 Use of variables inside parameters

It is also possible to use variables defined in project of the document.

From the project on Project Explorer view, right click > Properties > Resource > Linked Resources > Path Variables



Predefined variables or user variables can be used in the template. They are NOT case-sensitive.

Example :

```
<output path='${project_loc}/${input}-generated.docx' />
```

4.2.4.1 Variables stored in another file

As you may need to put many additional variables inside your project, in order to make the config tag more readable and more reusable, you can put the variables in a file with `.properties` extension. To access the content of this file you should add the `<properties>` tag in the following manner:

Example :

```
<properties path='${input_directory}/vars.properties' />
```

Where the `vars.properties` may have the content like this:

Example :

```
output_generation=${workspace_loc}/generated-${date}.docx
input_model_prop=${input}/model.uml
image_test=${project_loc}/company_logo.jpg
```

These variables can be used in Gendoc tags :

Example :

```
<output path='${output_generation}' />
```

4.2.4.2 Context with CDO models

You can use CDO URIs in context tags

Example :

```
<context
model='cdo.net4j.tcp://localhost:2036/repository/resource?transactional=true'
element='{0}' />
```

4.3 Define script execution context : <context> tag

Before a **<gendoc>** tag, a **<context>** must have been defined to determine the model and the element to use as starting context.

<context> tag can contain the following attributes :

- **model** : Model absolute path ([global parameters](#) can be used)
- **element** : Path to the model element to use as script context (path from model root)
- **importedBundles** : List of external bundles names (separated by ';' character)
- **searchMetamodels** : false(default)/true.

Set to “true” when your model uses element from external meta-models, for example with SysML, when both UML and SysML meta-models are used.

```
<context
  model='${model_folder}/model_1.uml'
  element='model/package1/subpackage1'
  importedBundles='gmf;papyrus'
  searchMetamodels='true'
/>
```

The context tag **must be defined at least once** in the document.

When executing a script, the **last defined** context is used.

Values of the attributes that are not updated are **kept from previous context**. For example, “importedBundles” attribute can be defined only once in the document and will be kept until a new value is indicated.

4.3.1 Dealing with specific models

4.3.1.1 Using multiple meta-models

If the model selected references another meta-model, set the following attribute to “true” in order for gendoc to analyse meta-models used as references.

```
searchMetamodels='true'
```

4.3.1.2 Meta-models where elements have no 'name' feature

With specific meta-models where elements have no 'name' property, context elements are defined:

- Using another property for all the elements in path
 - Ex : <context ... element='id1/id2/id3' labelFeature='id'/>
Property 'id' is used for all elements in path
 - Note : standard case is equivalent to <context ...
element='modelRoot/package1/subPackage1' labelFeature='name'/>
- Using another property only for some elements in path
 - Ex : <context ... element='modelRoot/id="id2"/subPackage1'/>
Property 'id' is used only for the package part of the element path
Property 'name' is used for others
- Using indexes of the position inside model tree (starting at 0 and not 1)
 - Ex: <context ... element='modelRoot/{1}/subPackage1'/>
If package2 is at the second place inside the model.

4.4 Define script parts : <gendoc> tag :

Each dynamic part corresponds to a <gendoc> tag.

A <context> tag must be present before, in order to set the execution context.

<gendoc> tag can contain:

- Acceleo script (see details on this language on <http://eclipse.org/acceleo/>)
- Static text
- Styles (colors / bullets / ...)
- [Tables](#)
- [Images and diagrams](#)
- [Rich text content](#)

4.4.1 Script language

The content of a gendoc tag corresponds to a script written in [Acceleo](#) language.

Example: display names of all packages

```
Acceleo syntax to display names of all packages
```

```
[for (p:Package | Package.allInstances())]  
[p.name/]  
[/for]
```

Gendoc syntax to display names of all packages

```
<context model='D:/.../myModel.uml' />
<gendoc>
[for (p:Package | Package.allInstances())]
[p.name/]
[/for]
</gendoc>
```

4.4.2 Text generation

Writing scripts inside a Text editor has a lot of inconveniences but the great advantage is to benefit from all text edition functionalities, mainly styles and formatting, from the Text editor.

4.4.2.1 Applying styles to the generation output

The style applied to the script inside template document is kept in the generation output (color, font, size, alignment,...).

Template content	Generation output
<pre><context model='...' /> <gendoc> [for (p:Package self.ownedElement- >filter(Package))] [p.name/] [/for] </gendoc></pre>	<p>Actors Use case view Logical view Deployment view</p>

4.4.2.2 Using bullets and numbering

With the same example as in the previous paragraph, other style information such as bullets or numbering can be used for generation.

Template content	Generation output
<pre><context model='...' /> <gendoc> [for (p:Package self.ownedElement- >filter(Package))] ❖ [p.name/] [/for] </gendoc></pre>	<p>❖ Actors ❖ Use case view ❖ Logical view ❖ Deployment view</p>

Template content	Generation output
<pre><context model='...' /> <gendoc> [for (p:Package self.ownedElement- >filter(Package))] 1. [p.name/] [for (p2:Package p.ownedElement- >filter(Package))] 1.1. [p2.name/] [/for] [/for] </gendoc></pre>	<p>1. Actors 2. Use case view 2.1. Data import 2.2. Data export 3. Logical view 4. Deployment view 4.1. Server side 4.2. Client side</p>

All other styles from document templates are kept during generation.

4.4.3 Images generation

`<image>` tag must be defined under a `<gendoc>` tag.

It shall define one of the following attributes:

- **object** for diagram generation, filled with an ID of the diagram. [See Diagram generation section](#).
- **filePath** for static image generation, filled with the image absolute path. [See static image generation section](#).

An empty drawing area inside start and end of tag:



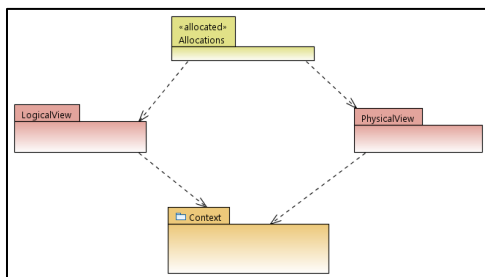
4.4.3.1 Customize image size

`<image>` tag provides the following attributes to handle image size customization : **keepW**, **keepH**, **maxW**, **maxH**. They are used in association with the dimensions of the drawing area inside `<image>` tag:

- **keepW** : output image width will be the same as drawing area width
- **keepH** : output image height will be the same as drawing area height
- **maxW** : output image width will not oversize drawing area width
- **maxH** : output image height will not oversize drawing area height


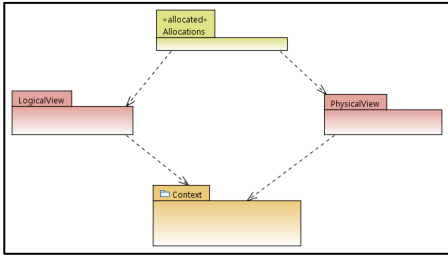
Possible values for these attributes: **false**(default), **true**.

For the following initial image:





- Fix image width, height is computed proportionally


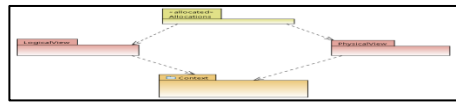
Template content	Output
<code><image object='...' keepW='true' keepH='false' ></code>	

 </image>	
--	---


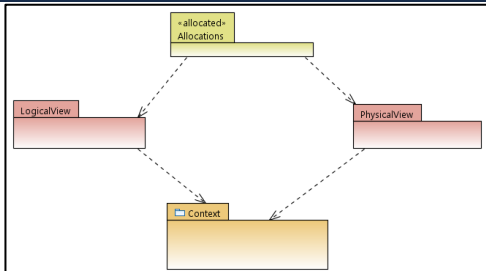
- Fix image height, width is computed proportionally

Template content	Output
<pre><image object='...' keepW='false' keepH='true' ></pre>  </image>	

- **TO BE AVOIDED** : Fix image height and width


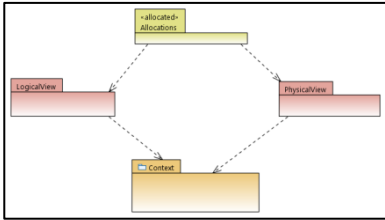
Template content	Output
<pre><image object='...' keepW='true' keepH='true' ></pre>  </image>	

- Ensure the image will not oversize a specified width
 - Case 1 : Image is smaller than the drawing area
Output corresponds to origin image dimensions


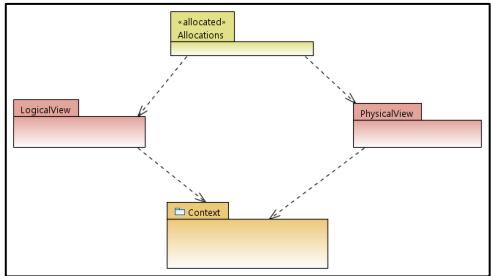
Template content	Output
<pre><image object='...' maxW='true' ></pre>  </image>	

- Case 2 : Image is larger than the drawing area
Output corresponds to drawing area dimension:


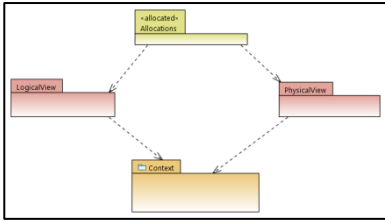
Template content	Output
<pre><image object='...' maxW='true' ></pre>	

 </image>	
--	---

- Ensure the image will not oversize a specified height
 - Case 1 : Image is smaller than the drawing area
Output corresponds to origin image dimensions

Template content	Output
<pre><image object='...' maxH='true' ></pre>  <pre></image></pre>	

- Case 2 : Image is larger than the drawing area
Output corresponds to drawing area dimension:

Template content	Output
<pre><image object='...' maxH='true' ></pre>  <pre></image></pre>	

4.4.3.2 Displaying diagrams

Attribute object shall be filled by an ID of the diagram. Diagram ID can be generated by [service getDiagram from bundle gmf](#), called on the diagram:

```
<context model='${model}' importedBundles='gmf;papyrus' />
<gendoc>
  [for (diag : Diagram| self.getPapyrusDiagrams())
    <image object='[diag.getDiagram() /]' maxW='true' keepH='false'>
```




```

</image>
[/for]
</gendoc>



```

4.4.3.3 Displaying static images

<image> tag can also be used for static image generation, with the following content:

- attribute **filePath** shall contain the absolute path of the static image.
The following image formats are supported : GIF, JPG, JPEG, BMP, PNG, SVG
- **<image>** tag shall contain an empty drawing area (alignment, text adaptation, ...)
- size attributes can be used : **keepW**, **keepH**, **maxW**, **maxH**

The following example shows the display of a static image:

Template content	Generation output
<pre> <context model='\${model_path}' /> <gendoc> Project logo is displayed below : <image filePath='D:/gendoc_logo.jpg' maxW='true'>  </image> </gendoc> </pre>	<p>Project logo is displayed below :</p> 

4.4.4 Table generation

<table> tag must be defined under a **<gendoc>** tag

The purpose of this tag is to merge all tables found inside tag content into one global table.

Template content

```
<context model='${project_loc}/Models/TrafficLightManager.uml'
element='TrafficLightManager/LogicalView'
importedBundles='gmf;papyrus' />
```

The following elements are described in the Logical view :

```
<gendoc>
  <table>
```

Name	Attributes
------	------------

```
[for (c:Class|self.ownedElement->filter(Class)->sortedBy(name)) ]
```

[c.name/]	<pre>[for (p:Property c.ownedAttribute)]<drop/> • [p.name/] : [p.type.name/] [/for]</pre>
------------------	--

```
[/for]
  </table>
</gendoc>
```

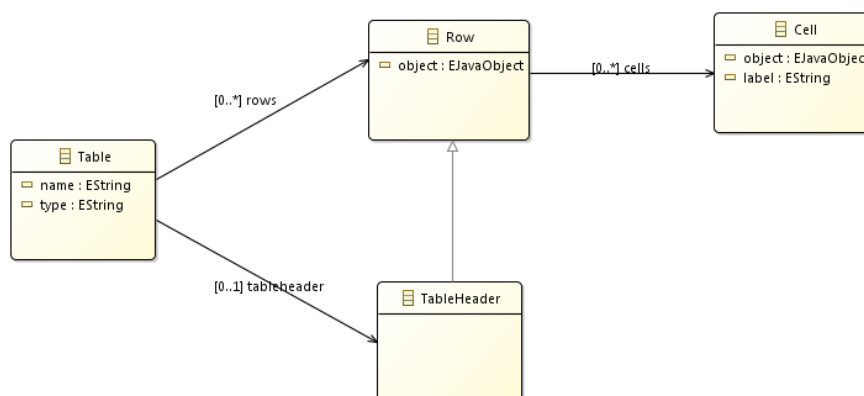
Output	The following elements are described in the Logical view :	
	Name	Attributes
	SystemLauncher	<ul style="list-style-type: none"> • x roads ctrl : XRoadsControler • tl group : TrafficLightGroup • tl : TrafficLight • red fire : RedFire • orange fire : OrangeFire • green fire : GreenFire
	TrafficLight	<ul style="list-style-type: none"> • traffic light id : Integer • operator maintenance : Operator maintenance • xroadscontroler : XRoadsControler • road : Road
	TrafficLightGroup	<ul style="list-style-type: none"> • tl group id : Integer • nb tl : Integer • traffic light : TrafficLight
	XRoadsControler	<ul style="list-style-type: none"> • nb tl group : Integer • crossroads domain : Crossroads Domain • operator maintenance : Operator maintenance • trafficlight : TrafficLight

If table styles are used in the template document inside a **<table>** tag, the style of the output table will be the style of the first table inside **<table>** tag.

4.4.5 Papyrus and Sirius Tables















As tables exist in Papyrus and Sirius we want to provide a way to integrate them as Word tables or LibreOffice tables.

Gendoc tag already exists, so the solution proposed is to have a pivot generic Table object, offered by a dedicated Gendoc metamodel, in order to generate all kinds of table the same way. Dedicated acceleo bundles already existing for Papyrus and Sirius have a new service to transform their table to the Gendoc generic table. An overview of the Gendoc table very simple metamodel



4.4.5.1 Papyrus table generation

- From a Papyrus model containing this table:

		A	B	C
		◦ id : String [1]	◦ name : String [0..1]	◦ text : String [1]
0	 Automated al...	REQ_001	Automated alarm clock	The alarm clock radio shall wak...
1	 Radio manage...	REQ_002	Radio management	The user shall be able to modif...
2	 Clock manage...	REQ_003	Clock management	The user shall be able to updat...
3	 Radio station ...	REQ_004	Radio station management	The user shall be able to modif...
4	 Volume mana...	REQ_005	Volume management	The user shall be able to modif...
5	 Radio frequen...	REQ_006	Radio frequency modes	The alarm clock radio shall pro...
6	 Radio frequen...	REQ_007	Radio frequencies	FM frequencies shall be betwee...
7	 Backup	REQ_009	Backup	A backup battery shall keep the...
8	 Voltage	REQ_010	Voltage	The alarm clock voltage shall b...
9	 Backup battery	REQ_011	Backup battery	The backup battery shall be a 9...
10	 Snooze	REQ_012	Snooze	Hitting the snooze button duri...
11	 Power	REQ_008	Power	The alarm clock radio shall be ...
12	 Listen to radio	REQ_013	Listen to radio	The user shall be able to listen t...
13	 Display time	REQ_014	Display time	The right time shall be displaye...

- Gendoc template fragment :

```

<config><drop/>
<param key='modelPath' value='${project_loc}/model.um' />
<output path='${project_loc}/Documents/Generated/result${date}.docx' />
</config><drop/>

<context model='${modelPath}' importedBundles='gmf;papyrus'
searchMetamodels='true' />

<context element='RootElement/Requirements' />

<gendoc id='requirements'><drop/>

[for (table : table::Table | self.getPapyrusTables())]
<table>

```

[table.name/] :

```

[table.tableheader.cell s -> at(1).label/] [table.tableheader.cell s -> at(2).label/] [table.tableheader.cell s -> at(3).label/]

[for (row : table::Row | table.rows)]

[row.cells -> at(1).label/] [row.cells -> at(2).label /] [row.cells -> at(3).label /]

[/for]
</table>

[/for]

</gendoc><drop/>

```

In this example the service “getPapyrusTables()” will return the list of tables contained in the context element.

It is possible to navigate in a table using relations “tableheader” and “row”. The first relation returns the headers of a table and the latter a list of row. From each of these objects we use the relation “cells” to access to the list of cells composing a row or the table header.












- Result in the word document produced by Gendoc :

RequirementTable0 : PapyrusSysMLRequirementTable

id	name	text
REQ_001	Automated alarm clock	The alarm clock radio shall wake up the user automatically at the right time, through radio or buzzer.
REQ_002	Radio management	The user shall be able to modify easily the radio station and the volume.
REQ_003	Clock management	The user shall be able to update easily the time displayed by the clock or the alarm.
REQ_004	Radio station management	The user shall be able to modify easily the radio station.
REQ_005	Volume management	The user shall be able to modify easily the volume.
REQ_006	Radio frequency modes	The alarm clock radio shall provide a way to select either AM or FM frequencies for radio.
REQ_007	Radio frequencies	FM frequencies shall be between 88MHz and 108MHz and AM frequencies shall be between 530kHz and 1600kHz.

4.4.5.2 Sirius table generation

- From a Sirius model containing this table:

	Father	Mother
 Woman Léa	Paul	Isa
 Man Elias	Paul	Isa
 Woman Fiona	Elias	
 Woman Clara	Elias	
 Man Dave	Elias	
 Man Bryan	Elias	
 Man Alain	Dave	Katell
 Woman Katell		
 Woman Isa		
 Man Paul		
 Man Albert		

- Gendoc template fragment :

```

<config><drop/>
<param key='modelPath' value='${project_loc}/example.basicfamily'/>
<output path='${project_loc}/Documents/Generated/result${date}.docx' />
</config><drop/>

<context model='${modelPath}' importedBundles='gmf;sirius'
searchMetamodels='true'/>

<gendoc id='requirements'><drop/>

[for (table : table::Table | self.getSiriusTables())]
<table>

    [table.name/] :



| [table.tableheader.cell<br>s -> at(1).label/] | [table.tableheader.cell<br>s -> at(2).label/] | [table.tableheader.cell<br>s -> at(3).label/] |
|-----------------------------------------------|-----------------------------------------------|-----------------------------------------------|
| [for (row : table::Row   table.rows)]         |                                               |                                               |
| [row.cells -><br>at(1).label/]                | [row.cells -> at(2)<br>.label /]              | [row.cells -> at(3)<br>.label /]              |



[/for]
</table>

[/for]

</gendoc><drop/>

```

This example is very similar to the precedent. The only notable difference is how we access to the list of tables. In this case as we are in a Sirius model, we use the “getSiriusTables()” service.

- Result in the word document produced by Gendoc :

Family :

	Father	Mother
Man Paul		
Woman Isa		
Man Elias	Paul	Isa
Woman Léa	Paul	Isa
Man Dave	Elias	
Man Alain	Dave	Katell
Man Bryan	Elias	
Woman Fiona	Elias	
Woman Katell		
Woman Clara	Elias	
Man Albert		

4.4.5.3 Automated table generation

From a Papyrus model containing several tables on a package the following gendoc fragment allows a full table generation.

Note: Automated mode also works with Sirius

- Gendoc code :

```

<config><drop/>
<param key='modelPath' value=' ${project_loc}/model.uml' />
<output path=' ${project_loc}/Documents/Generated/result${date}.docx' />
</config><drop/>

<context model='${modelPath}' importedBundles='gmf:papyrus'
searchMetamodels='true' />

<context element= 'RootElement/Requirements' /><drop/>
<gendoc id='requirements'><drop/>
[for(t : table::Table |self.getPapyrusTables())]
    [t.name/]:[t.type/]
<table object='[t.getId()]/'><drop/>

```

```

</table><drop/>

[/for]
</gendoc><drop/>

```

The table tag has a new attribute “object” for the automatic generation. This attribute has to be set to the identifier of the table. The service “getId()” is used to retrieve this identifier.

Note: in this mode all cells will have the same style.

- Gendoc result :

RequirementTable0:PapyrusSysMLRequirementTable

id : String [1]	name : String [0..1]	text : String [1]
REQ_001	Automated alarm clock	The alarm clock radio shall wake up the user automatically at the right time, through radio or buzzer.
REQ_002	Radio management	The user shall be able to modify easily the radio station and the volume.
REQ_003	Clock management	The user shall be able to update easily the time displayed by the clock or the alarm.
REQ_004	Radio station management	The user shall be able to modify easily the radio station.
REQ_005	Volume management	The user shall be able to modify easily the volume.

AllocationTable0:PapyrusSysMLAllocationTable

name : String [0..1]
Allocate1
Allocate2
Allocate3
Allocate4
Allocate5

TableOfViews0:PapyrusViewsTable


name : EString [0..1]	/context : EObject [1]	/isOpen : EBoolean [1]	type : EString [1]
RequirementTable0	Requirements	false	PapyrusSysMLRequirementTable
AllocationTable0	Requirements	false	PapyrusSysMLAllocationTable
TableOfViews0	Requirements	false	PapyrusViewsTable

4.4.6 Bookmarks and hyperlinks generation

The complexity of generating bookmarks and hyperlinks in an output document is the dynamicity of both bookmarks and hyperlinks.

The idea is to find a generated or not unique ID that will link source (hyperlink) and target (bookmark) location in the document.

The following example shows how to create dynamic bookmarks and hyperlinks in templates for a UML model containing classes, with references to other classes inside attributes.

	Template content	Output details
Step 1	<p>Display classes and their attributes and types :</p> <pre><context model='\${project_loc}/Models/TrafficLightManager.uml' element='TrafficLightManager/LogicalView' /> <gendoc><drop/> [for (c:Class self.ownedElement->filter(Class)->sortedBy(name))] [c.name/] [for (a:Property c.ownedAttribute->filter(NamedElement)- >sortedBy(name))] -[a.name/]: [a.type.name/] [/for] [/for] </gendoc></pre>	<p>TrafficLight</p> <ul style="list-style-type: none"> -green fire: GreenFire -orange fire: OrangeFire -red fire: RedFire <p>GreenFire</p> <p>OrangeFire</p> <p>RedFire</p>
Step 2	<p>Add a (static) bookmark on the class name :</p> <pre><context model='\${project_loc}/Models/TrafficLightManager.uml' element='TrafficLightManager/LogicalView' /> <gendoc><drop/> [for (c:Class self.ownedElement->filter(Class)->sortedBy(name))] [c.name/] Add a bookmark : - On MS Word :  Insert > Links > Bookmark - On OpenOffice / LibreOffice Writer :  /  Insert > Bookmark Name of the bookmark (must be unique in document) : c_name_bookmark [for (a:Property c.ownedAttribute->filter(NamedElement)- >sortedBy(name))] -[a.name/]: [a.type.name/] [/for] [/for] </gendoc></pre>	<p>TrafficLight</p> <p>bookmark c_name_bookmark</p> <ul style="list-style-type: none"> -green fire: GreenFire -orange fire: OrangeFire -red fire: RedFire <p>GreenFire</p> <p>bookmark c_name_bookmark</p> <p>OrangeFire</p> <p>bookmark c_name_bookmark</p> <p>RedFire</p> <p>bookmark c_name_bookmark</p>

Step 3	<p>Add dynamicity on the bookmark :</p> <p>Indicate in a dedicated tag on top of document how to generate a dynamic ID at bookmark location to make bookmark become dynamic.</p> <p>What will the bookmark point to : class c</p> <p>How to generate a unique Id for class c : use service getId() from bundle commons.</p> <pre><context model='\${project_loc}/Models/TrafficLightManager.uml' element='TrafficLightManager/LogicalView' /> <bookmarks> <alias source='c_name_bookmark' target=' [c.getId() /]' /> </bookmarks> <gendoc><drop/> [for (c:Class self.ownedElement->filter(Class)->sortedBy(name))] [c.name/] Bookmark named : c_name_bookmark [for (a:Property c.ownedAttribute->filter(NamedElement) - >sortedBy(name))] -[a.name/]: [a.type.name/] [/for] [/for] </gendoc></pre>	<div>TrafficLight bookmark : TrafficLight class ID</div> <div>-green fire: GreenFire -orange fire: OrangeFire -red fire: RedFire</div> <div>GreenFire bookmark : GreenFire class ID</div> <div>OrangeFire bookmark : OrangeFire class ID</div> <div>RedFire bookmark : RedFire class ID</div>
Step 4	<p>Add hyperlinks to the (future) bookmark location :</p> <p>The hyperlink must also be dynamic and point to the future bookmark location, here the generated unique ID for the class.</p> <p>So the hyperlink must no point on c_name_bookmark (it would be replaced by c.getId() and point to current class), but to the id of the property type class : a.type.getId().</p> <pre><context model='\${project_loc}/Models/TrafficLightManager.uml' element='TrafficLightManager/LogicalView' /> <bookmarks> <alias source='c_name_bookmark' target=' [c.getId() /]' /> </bookmarks> <gendoc><drop/> [for (c:Class self.ownedElement->filter(Class)->sortedBy(name))] [c.name/] Bookmark named : c_name_bookmark [for (a:Property c.ownedAttribute->filter(NamedElement) - >sortedBy(name))] -[a.name/]: [a.type.name/] Add an hyperlink (Insert > Hyperlink ...) to the ID of the property type : #[a.type.getId()]/ [/for] [/for] </gendoc></pre>	<div>TrafficLight bookmark : TrafficLight class ID</div> <div>-green fire: GreenFire hyperlink to GreenFire class ID -orange fire: OrangeFire hyperlink to OrangeFire class ID -red fire: RedFire hyperlink to RedFire class ID</div> <div>GreenFire bookmark : GreenFire class ID</div> <div>OrangeFire bookmark : OrangeFire class ID</div> <div>RedFire bookmark : RedFire class ID</div>
Final output	<div>GreenFire OrangeFire RedFire TrafficLight</div> <div>-green fire: GreenFire -orange fire: OrangeFire -red fire: RedFire</div>	

4.4.7 Rich text generation

<richText> tag must be defined under a **<gendoc>** tag and allows to display rich text content (HTML or RTF) content inside the generated document.

It can contain the following attributes:

- **format** (optional) : describes file format (RTF, HTML).
Default value : HTML.
- **filePath** (optional) : full path of the rich text file to import

Rich text content can come:

- from an external file
 - in HTML format :

```
<richText filePath='D:/file.html' />
```

equivalent to :

```
<richText filePath='D:/file.html' format='HTML' />
```

- in RTF format :

```
<richText filePath='D:/file.rtf' format='RTF' />
```

- from model content



<richText> tag must contain only the script to access the rich text content :
no additional spaces or line breaks .

For example for UML comment contents in HTML format in the model:

```
<richText format='HTML'>[comment._body/]</richText>
```

It is equivalent to:

```
<richText>[comment._body/]</richText>
```

4.4.8 Enclose the external document

Gendoc offers the possibility of importing the content of an external document inside output, **for Microsoft Word templates only**, through tag **<include>**.

<include> tag must be contained in a **<gendoc>** tag.

The absolute path of the file to be imported is defined in **filePath** attribute. The following file formats are supported: docx, txt, html.

```
<gendoc>
  <include filePath='C:/myFolder/anotherFile.docx' />
</gendoc>
```

4.4.9 Formatting

4.4.9.1 Removing extra lines

All characters inside scripts are used for generation output, including spaces, line breaks, or carriage return characters.

Template content	Actual output	Expected output
<pre> <gendoc>¶ ..[for (p:Package self.ownedElement- >filter (Package) ->sortedBy(name))] ·¶ [p.name/] ·¶ ..[/for] ·¶ </gendoc>¶ </pre>	<pre> ¶ ...¶ Allocations·¶ ...¶ Context·¶ ...¶ LogicalView·¶ ...¶ PhysicalView·¶ ...¶ UseCases·¶ ...¶ ¶ </pre>	<pre> Allocations·¶ Context·¶ LogicalView·¶ PhysicalView·¶ UseCases·¶ </pre>

<drop/> tag allows to remove extra lines.

Document generation is internally performed in two steps and **<drop/>** tag removes the **WHOLE** paragraph in which it is contained so it must be handled with care.

First step is to analyze the lines to get as output to understand where the extra lines come from in the template and where the **<drop/>** tags should be located.

Template content	Output	
<pre> <gendoc>¶ ..[for (p:Package self.ownedElement- >filter (Package) ->sortedBy(name))] ·¶ [p.name/] ·¶ ..[/for] ·¶ </gendoc>¶ </pre>	<pre> ¶ ...¶ Allocations·¶ ...¶ Context·¶ ...¶ LogicalView·¶ ...¶ PhysicalView·¶ ...¶ UseCases·¶ ...¶ ¶ </pre>	
<pre> <gendoc><drop/>¶ ..[for (p:Package self.ownedElement- >filter (Package) ->sortedBy(name))] <drop/>¶ [p.name/] ¶ ..[/for] ·<drop/>¶ </gendoc><drop/>¶ </pre>	before <drop/> handling <pre> <drop/>¶ ...<drop/>¶ Allocations·¶ ...<drop/>¶ Context·¶ ...<drop/>¶ LogicalView·¶ ...<drop/>¶ PhysicalView·¶ ...<drop/>¶ UseCases·¶ ...<drop/>¶ <drop/>¶ </pre>	Final output <pre> Allocations·¶ Context·¶ LogicalView·¶ PhysicalView·¶ UseCases·¶ </pre>

4.4.9.2 Removing lines with empty content

Tag **<dropEmpty/>** drop a paragraph if the tag content is empty.

The two following examples are equivalent:

```

<context model='${model_path}' />
<gendoc>
All comments on packages:

```

```

    [for (p:Package|Package.allInstances()->sortedBy(name))]<drop/>
    [for (c:Comment| p.ownedComment)]<drop/>
- Comment for package [p.name/]: <dropEmpty>[c._body/]</dropEmpty>
    [/for]<drop/>
  [/for]<drop/>
</gendoc>

```

```

<context model='${model_path}' />
<gendoc>

```

All comments on packages:

```

    [for (p:Package|Package.allInstances()->sortedBy(name))]<drop/>
    [for (c:Comment| p.ownedComment)]<drop/>
    [if (not(c._body.oclIsUndefined()))]<drop/>
- Comment for package [p.name/]: [c._body/]
    [/if]<drop/>
  [/for]<drop/>
[/for]<drop/>
</gendoc>

```

4.4.9.3 Removing line breaks

Using tag `<nobr/>` allows to make template scripts easier to maintain, because code can be written on several paragraphs without displaying line breaks in output document, such as in the following example.

Template content	Output
<pre> <gendoc> [for (p:Package ...)]<drop/>¶ Name: <nobr/>¶ [if (...)]<drop/>¶ [p.name/]¶ [else]<drop/>¶ Not found¶ [/if]<drop/> ¶ <<Other info on package>>¶ ¶ [/for]<drop/>¶ </gendoc>¶ </pre>	<pre> Name: Actors Name: DeploymentView Name: Actors¶ <<Other info on package>>¶ ¶ Name: LogicalView¶ <<Other info on package>>¶ ¶ Name: UseCaseView¶ <<Other info on package>>¶ ¶ Name: Not found¶ <<Other info on package>>¶ ¶ </pre>

4.4.10 Listing elements

By using the tag `<list>` you can also easily list the model elements.

Template content	Output
<pre> <gendoc>¶ <list>¶ [for (p:NamedElement self.ownedElement- >filter(NamedElement))] ¶ o [p.name.clean()/] [/for] ¶ </pre>	<pre> o Model2 o secondly o Package2 </pre>

<code></list>¶</code> <code></gendoc>¶</code>	
--	--

4.5 Reusing gendoc scripts inside the same document : `<fragment>` tag

If a script section is used several times in a same document template, tag `<fragment>` can be used to define the script section and its attributes.

It can then be called from `<gendoc>` tags inside the same template document


`<fragment>` tag can contain the following attributes :

- **name** (Mandatory) : the name to call to use the fragment
- **importedBundles** (Optional) : the [external bundles](#) needed in the fragment code (separated by ;)
- **removeClosingLine** (Optional) : when this attribute is set to 'true' the line containing the closing tag (`</fragment>`) is removed during the generation

From **Gendoc 0.7.0** the use of fragments does **not** have any **limitations**. They can be recursive and it may contains circular references.

```

<fragment name='displayDiagram' importedBundles='commons;gmf;papyrus'>
  <arg name='element' type='uml::Element' />
  [for (d:Diagram|element.getPapyrusDiagrams())]<drop/>
Diagram [d.name/] :

  <image object='[d.getDiagram() /] ' maxW='true'><drop/>

  </image>
[/for]<drop/>
</fragment>

```

```

<gendoc>
[for (p:Package|Package.allInstances())]
  [p.name/]

  [p.displayDiagram() /]
[/for]
</gendoc>

```

5 XLSX Document Generator

5.1 Creation of a document generator

- Create a new document in MS Office 2007+ (.xlsx) format or get an existing document (with the company charter for example) in one of these formats.
- Define static parts that can be : cells with styles, text, some data, formulas, etc.
- Identify dynamic parts with <gendoc> tags.
- Just adapt configuration parameters in template header:
 - Model path
 - Output file path
- Generate with a right click menu
- As it is an iterative process, you can do it whenever you want

Tables and Pivot Tables are not supported in this release.

Note: Generation can also be launched in batch mode.

5.2 Configure the generation: <config> tag

The tag <config> must be defined **only once**, on top of the template document, in a cell in the first rows, in the first worksheet, before any other Gendoc tag.

This tag defines the path of the output document, and a list of global parameters for the template.

5.2.1 Define generation output

`<output>` tag is optional. If not present, the document is generated at template location, with suffix '_generated'

If defined, the syntax is the following:

```
<config>
  <output path=<<Absolute path of the document to be generated>> />
  ...
</config>
```

Global parameters can be used to define a relative path.

Example: The generated document will be located in D:/generatedFile.xlsx

```
<config>
  <output path='D:/generatedFile.xlsx' />
  ...
</config>
```

5.2.2 Define global parameters for the template

Global parameters for the template can be defined, for example to define model path, folders to use or any other static value to be used in template.

Parameters are defined in `<config>` tag with the following syntax:

```
<config>
  ...
  <param key=<<Parameter1_key>> value=<<Parameter1_value>> />
  <param key=<<Parameter2_key>> value=<<Parameter2_value>> />
  <param .../>
</config>
```

How to access parameters?

- `${paramKey}` inside `<context>` or other `<param>`,
- `gGet(paramKey)` inside a `<gendoc>` tag

Example: creation of global parameters for model folder, model path, and path of a specific package inside model and example of usage in `<context>` tag.

```
<config>
  <param key='model_path' value='D:/Models/Model_v1/My_model.uml' />
  <param key='UC_package_path' value='/MyUMLModel/UseCases' />
</config>
<context model='${model_path}' element='${UC_package_path}' />
```

5.2.3 Pre-defined parameters

Some `<param>` are pre-defined in Gendoc and can be used directly in the template.

- `${input}` is the name of the input template document

Example:

```
<param key='generation_folder' value='D:/Generated' />
<output path='${generation_folder}/${input}-generated.docx' />
```

If the input document is named `template1.docx`, the result file is named `template1-generated.docx`.

The following variables are also ready to be used by default:

- `${date}` is the date of the generation. The format of the date is 'yyyy-MM-dd-HHmms'.
- `${input_directory}` location directory of the template.

Example:

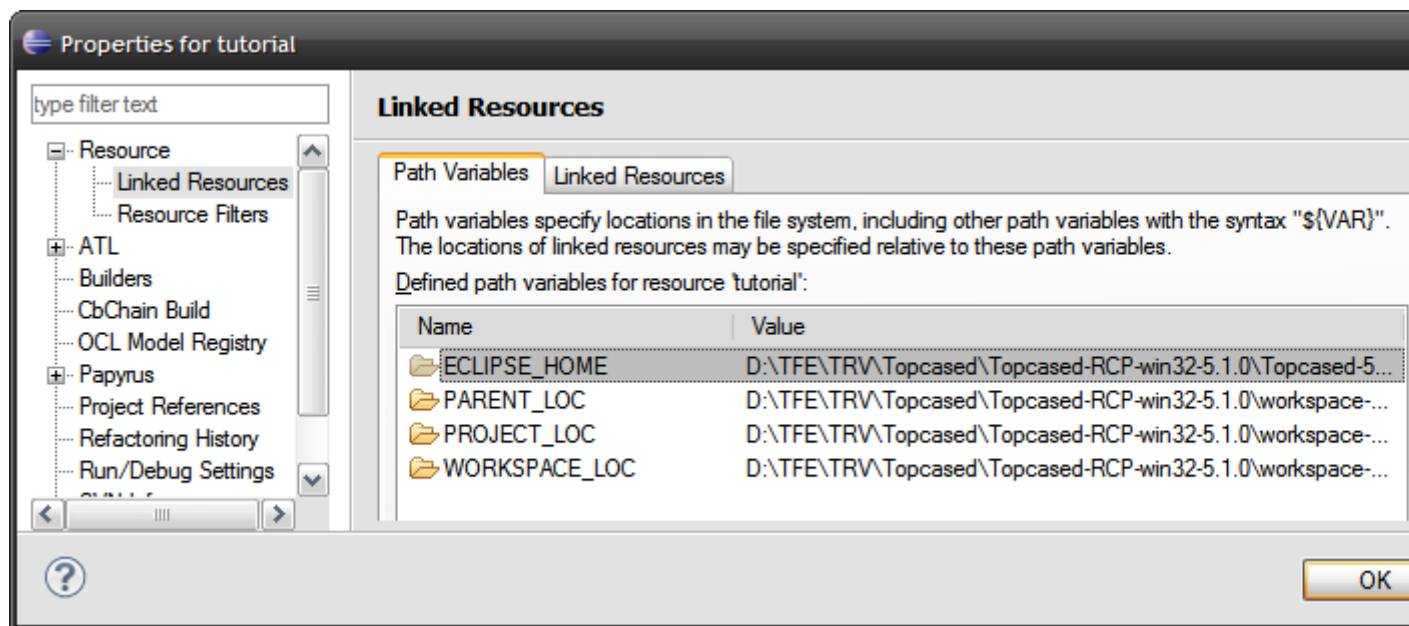
```
<output path='${input_directory} /${input}-generated-${date} .docx' />
```

Result file example: `template-generated-2014-08-02-093707.docx`

5.2.4 Use of variables inside parameters

It is also possible to use variables defined in project of the document.

From the project on Project Explorer view, right click > Properties > Resource > Linked Resources > Path Variables



Predefined variables or user variables can be used in the template. They are NOT case-sensitive.

Example :

```
<output path='${project_loc}/${input}-generated.docx' />
```

5.2.5 Variables stored in another file

As you may need to put many additional variables inside your project, in order to make the config tag more readable and more reusable, you can put the variables in a file with .properties extension. To access the content of this file you should add the `<properties>` tag in the following manner:

Example :

```
<properties path='${input_directory}/vars.properties' />
```

Where the vars.properties may have the content like this:

Example :

```
output_generation=${workspace_loc}/generated-${date}.docx
input_model_prop=${input}/model.uml
image_test=${project_loc}/company_logo.jpg
```

These variables can be used in Gendoc tags :

Example :

```
<output path='${output_generation}' />
```

5.2.6 Context with CDO models

You can use CDO URIs in context tags

Example :

```
<context
model='cdo.net4j.tcp://localhost:2036/repository/resource?transactional=true'
element='{0}' />
```

5.3 Define script execution context : `<context>` tag

Before a `<gendoc>` tag, a `<context>` must have been defined to determine the model and the element to use as starting context.

`<context>` tag can contain the following attributes :

- **model** : Model absolute path ([global parameters](#) can be used)

- **element** : Path to the model element to use as script context (path from model root)
- **importedBundles** : List of external bundles names (separated by ';' character)
- **searchMetamodels** : false(default)/true.
Set to "true" when your model uses element from external meta-models, for example with SysML, when both UML and SysML meta-models are used.

```
<context
  model='${model_folder}/model_1.uml'
  element='model/package1/subpackage1'
  importedBundles='gmf;papyrus'
  searchMetamodels='true'
/>
```

The context tag **must be defined at least once** in the document.

When executing a script, the **last defined** context is used.

Values of the attributes that are not updated are **kept from previous context**.
For example, "importedBundles" attribute can be defined only once in the document and will be kept until a new value is indicated.

5.3.1 Dealing with specific models

5.3.1.1 Using multiple meta-models

If the model selected references another meta-model, set the following attribute to "true" in order for gendoc to analyse meta-models used as references.

```
searchMetamodels='true'
```

5.3.1.2 Meta-models where elements have no 'name' feature

With specific meta-models where elements have no 'name' property, context elements are defined:

- Using another property for all the elements in path
 - Ex : `<context ... element='id1/id2/id3' labelFeature='id' />`
Property 'id' is used for all elements in path
 - Note : standard case is equivalent to `<context ... element='modelRoot/package1/subPackage1' labelFeature='name' />`
- Using another property only for some elements in path
 - Ex : `<context ... element='modelRoot/id="id2"/subPackage1' />`
Property 'id' is used only for the package part of the element path
Property 'name' is used for others
- Using indexes of the position inside model tree (starting at 0 and not 1)
 - Ex : `<context ... element='modelRoot/{1}/subPackage1' />`
If package2 is at the second place inside the model.

5.4 Define script parts: <gendoc> tag :

Each dynamic part corresponds to a <gendoc> tag.

A **<context>** tag must be present before, in order to set the execution context.

<gendoc> tag can contain:

- [Acceleo script](#) (see details on this language on <http://eclipse.org/acceleo/>)
- Static text
- [Styles \(colors / bullets / ...\)](#)
- [Images and diagrams](#)

Note: Rich text content is not supported in this release

Note: Worksheet Data Tables and Pivot tables are not supported in this release.

Note: Bookmark are not supported in this release.

Note: External document inclusion is not supported in this release.

Note: Excel does not support listing or tables in the cell text formatting

5.4.1 Script language

The content of a gendoc tag corresponds to a script written in [Acceleo](#) language.

Acceleo syntax to display names of all packages

Example: display names of all packages

```
[for (p:Package | Package.allInstances())]  
  [p.name/]  
[/for]
```

The script is is written directly in the sheet cells. When the the scrip is executed, rows and cells will be added or removed. Below there is a example of how to display names of all packages, one per row.

	A	B
1	<context model='D:/.../myModel.uml' />	
2	<gendoc> [for (p:Package Package.allInstances())]	
3	[p.name/]	
4	[/for] </gendoc>	

5.4.2 Text generation

Writing scripts inside a worksheet has a lot of inconveniences but the great advantage is to benefit from all edition functionality, mainly column, row and cell formats.

5.4.2.1 Applying styles to the generation output

The style applied to the cell, row or column containing the script inside the template document is kept in the generation output (color, font, size, alignments).

Template content			Generation output		
	A	B		A	B
1	<code><context model='D:/.../myModel.uml' /> <gendoc> [for (p:Package Package.allInstances())]</code>		1		
2	<code>[p.name/]</code>		2	Actors	
3	<code>[/for] <gendoc></code>		3	Use case view	
			4	Logical view	
			5	Deployment view	
			6		

All other styles from document templates are kept during generation.

5.4.3 Images generation

`<image>` tag must be defined under a `<gendoc>` tag.

It shall define one of the following attributes:

- **object** for diagram generation, filled with an ID of the diagram. [See Diagram generation section.](#)
- **filePath** for static image generation, filled with the image absolute path. [See static image generation section.](#)

The drawing are correspond to the entire cell:

	A	B
1	<code><context model='D:/.../myModel.uml' /> <gendoc></code>	

2	<code><image ... /></code>	
3	<code><gendoc></code>	

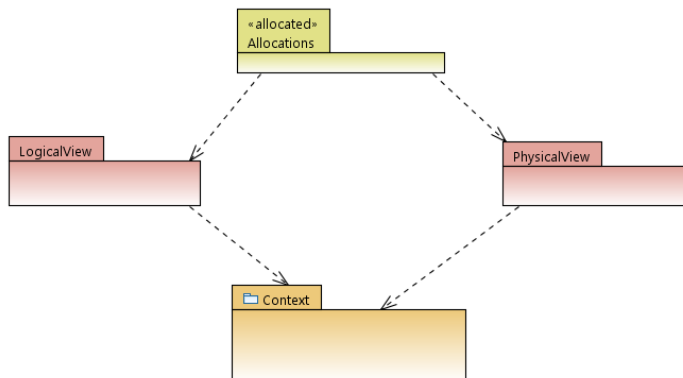
5.4.3.1 Customize image size

`<image>` tag provides the following attributes to handle image size customization: `keepW`, `keepH`, `maxW`, `maxH`. They are used in association with the dimensions of the drawing area inside `<image>` tag:

- **keepW**: output image width will be the same as drawing area width
- **keepH**: output image height will be the same as drawing area height
- **maxW**: output image width will not oversize drawing area width
- **maxH**: output image height will not oversize drawing area height

Possible values for these attributes: **false**(default), **true**.

For the following initial image:

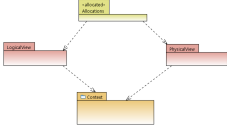


- Fix image width, height is computed proportionally, the image overlap partially the next rows.

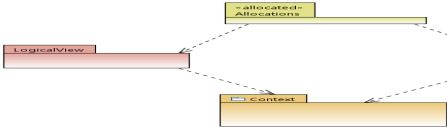
Template content			Output		
	A	B		A	B
1	<code><gendoc></code>		1		
2	<code><image object='...' keepW='true' keepH='false' /></code>		2		

3	</gendoc>		3		
---	-----------	--	---	--	--

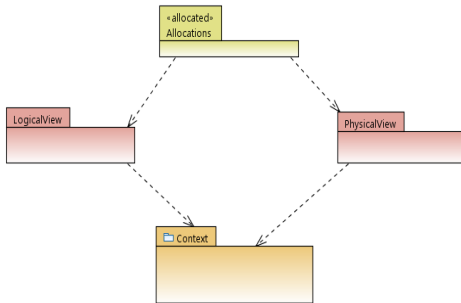
- Fix image height, width is computed proportionall

Template content			Output		
	A	B		A	B
1	<gendoc>		1		
2	<image object='...' keepW='false' keepH='true' />		2		
3	</gendoc>		3		

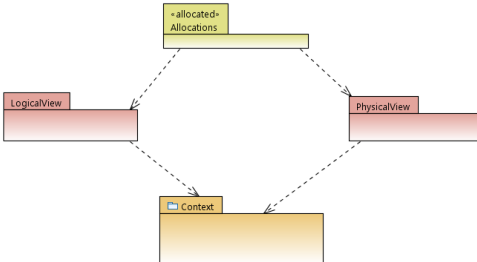
- TO BE AVOIDED : Fix image height and width

Template content			Output		
	A	B		A	B
1	</gendoc>		1		
2	<image object='...' keepW='true' keepH='true' />		2		
3	</gendoc>		3		

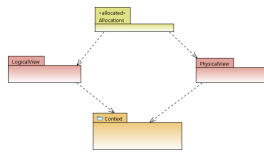
- Ensure the image will not oversize a specified width
 -
 - Case 1 : Image is smaller than the drawing area
Output corresponds to origin image dimensions, and the picture may overlap with the following rows.

Template content			Output		
	A	B		A	B
1	<code></gendoc></code>		1		
2	<code><image object='...' maxW='true' /></code>		2		
3	<code></gendoc></code>		3		

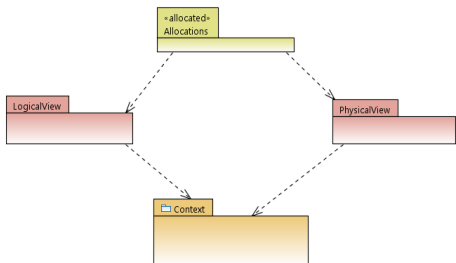
- Case 2 : Image is larger than the drawing area
Output corresponds to drawing area dimension :

Template content			Output		
	A	B		A	B
1	<code></gendoc></code>		1		
2	<code><image object='...' maxW='true' /></code>		2		
3	<code></gendoc></code>		3		

- Ensure the image will not oversize a specified height
 - Case 1 : Image is smaller than the drawing area
Output corresponds to origin image dimensions

Template content			Output		
	A	B		A	B
1	<code></gendoc></code>		1		
2	<code><image object='...' maxH='true' /></code>		2		
3	<code></gendoc></code>		3		

- Case 2 : Image is larger than the drawing area
Output corresponds to drawing area dimension :

Template content			Output		
	A	B		A	B
1	<code></gendoc></code>		1		
2	<code><image object='...' maxH='true' /></code>		2		
3	<code></gendoc></code>		3		

5.4.3.2 Displaying diagrams

Attribute object shall be filled by an ID of the diagram. Diagram ID can be generated by [service getDiagram from bundle gmf](#), called on the diagram:

Template content


	A	B
1	<pre><context model='\${model}' importedBundles='gmf;papyrus' /> <gendoc> [for (diag : Diagram self.getPapyrusDiagrams())]</pre>	
2	<pre> <image object='[diag.getDiagram() /]' maxW='true' keepH='false' /></pre>	
3	<pre> [/for] </gendoc></pre>	

5.4.3.3 Displaying static images

<image> tag can also be used for static image generation, with the following content:

- attribute **filePath** shall contain the absolute path of the static image.
The following image formats are supported : GIF, JPG, JPEG, BMP, PNG, SVG
- **<image>** tag shall contain an empty drawing area (alignment, text adaptation, ...)
- size attributes can be used : **keepW**, **keepH**, **maxW**, **maxH**

The following example shows the display of a static image:

Template content			Output		
	A	B		A	B
1	<pre><context model='\${model_path}' /> <gendoc> Project logo is displayed below :</pre>		1	Project logo is displayed below :	
2	<pre><image filePath='D:/gendoc_logo.jpg' ' maxW='true' /></pre>		2		
3	<pre></gendoc></pre>		3		

5.4.4 Formatting

5.4.4.1 Removing extra lines

All characters inside scripts are used for generation output, including spaces, line breaks, or carriage return characters.

Template content			Actual output			Expected output		
	A	B		A	B		A	B
1	<gendoc>¶		1	¶		1	Allocations .¶	
2	..[for (p:Package self.ownedElement->filter(Package)->sortedBy(name))].¶		2	...¶		2	Context.¶	
3	[p.name/].¶		3	Allocations .¶		3	LogicalView .¶	
4	..[/for].¶		4	...¶		4	PhysicalView. w.¶	
5	</gendoc>¶		5	Context.¶		5	UseCases.¶	
			6	...¶				
			7	LogicalView .¶				
			8	...¶				
			9	PhysicalView. w.¶				
			10	...¶				
			11	UseCases.¶				
			12	...¶				
			13	¶				

<drop/> tag allows to remove extra **rows**.

Document generation is internally performed in two steps and <drop/> tag removes the **WHOLE** row in which it is contained so it must be handled with care.

First step is to analyze the lines to get as output to understand where the extra lines come from in the template and where the <drop/> tags should be located.

Template content			Output		
	A	B		A	B
1	<gendoc>¶		1	¶	
2	<pre> ..[for (p:Package self.ownedElement->filter(Package)->sortedBy(name))].¶ </pre>		2	...¶	
3	[p.name/].¶		3	Allocations.¶	
4	..[/for].¶		4	...¶	
5	</gendoc>¶		5	Context.¶	
			6	...¶	
			7	LogicalView.¶	
			8	...¶	
			9	PhysicalView.¶	
			10	...¶	
			11	UseCases.¶	
			12	...¶	
			13	¶	
Template content			before <drop/> handling		Final output
	A	B		A	B
1	<gendoc><drop/>¶		1	<drop/>¶	1 Allocations.¶
2	<pre> ..[for (p:Package self.ownedElement->filter(Package)->sortedBy(name))].<drop/>¶ </pre>		2	...<drop/>¶	2 Context.¶
3	[p.name/].¶		3	Allocations.¶	3 LogicalView.¶
4	..[/for].<drop/>¶		4	...<drop/>¶	4 PhysicalView.¶
5	</gendoc><drop/>¶		5	Context.¶	5 UseCases.¶
			6	...<drop/>¶	

	7	LogicalView .¶	
	8	...<drop/>¶	
	9	PhysicalView. ¶	
	10	...<drop/>¶	
	11	UseCases.¶	
	12	...<drop/>¶	
	13	<drop/>¶	

5.4.4.2 Removing lines with empty content

Tag `<dropEmpty/>` drop a paragraph if the tag content is empty.

The two following examples are equivalent:

	A	B
1	<code><context model='\${model_path}' /> <gendoc><drop/></code>	
2	All comments on packages:	
3	<code>[for (p:Package Package.allInstances()->sortedBy(name))] [for (c:Comment p.ownedComment)]<drop/></code>	
4	- Comment for package [p.name/]: <code><dropEmpty>[c._body/]</dropEmpty></code>	
5	<code>[/for] [/for] </gendoc><drop/></code>	

	A	B
1	<code><context model='\${model_path}' /> <gendoc><drop/></code>	
2	All comments on packages:	
3	<code>[for (p:Package Package.allInstances()->sortedBy(name))] [for (c:Comment p.ownedComment)] [if (not(c._body.oclIsUndefined()))] <drop/></code>	
4	- Comment for package [p.name/]: [c._body/]	
5	<code>[/if] [/for] [/for] </gendoc><drop/></code>	

5.4.5 Reusing gendoc scripts inside the same document: <fragment> tag

If a script section is used several times in a same document template, tag <fragment> can be used to define the script section and its attributes.

It can then be called from <gendoc> tags inside the same template document

<fragment> tag can contain the following attributes:

- **name (Mandatory)** : the name to call to use the fragment
- **importedBundles** (Optional) : the [external bundles](#) needed in the fragment code (separated by ;)
- **removeClosingLine**(Optional) : when this attribute is set to 'true' the line containing the closing tag (</fragment>) is removed during the generation

From **Gendoc 0.7.0** the use of fragments does **not** have any **limitations**. They can be recursive and it may contains circular references.

	A	B
1	<code><fragment name='displayDiagram' importedBundles='commons;gmf;papyrus'> <arg name='element' type='uml::Element' /> [for (d:Diagram element.getPapyrusDiagrams())]<drop/></code>	
2	<code>Diagram [d.name/]:</code>	
3	<code><image object=' [d.getDiagram() /]' maxHeight='true' /></code>	
4	<code>[/for] </fragment><drop/></code>	
5		
6	<code><gendoc> [for (p:Package Package.allInstances())]<drop/></code>	
7	<code>[p.name/]</code>	
8	<code>[p.displayDiagram() /]</code>	
9	<code>[/for] </gendoc> <drop/></code>	

6 PPTX Document Generator

6.1 Creation of a document generator

- **Create a new document** in MS Office 2007+ (.pptx) format or get an existing document (with the company charter for example) in one of these formats.
- Define **static parts** in the slides that can be : images, text, some data, formulas, etc.
- **Identify dynamic parts** with <gendoc> tags inside text boxes in the slides.
- Just adapt configuration parameters in template header:
 - Model path
 - Output file path
- Generate with a right click menu
- As it is an iterative process, you can do it whenever you want

Note : Animations inside of <gendoc> tags are not supported. RichText is not supported.

Note: Generation can also be launched in batch mode.

6.2 Configure the generation: <config> tag

The tag <config> must be defined **only once**, on top of the template document, in a text box in one of the first slides, before any other Gendoc tag.

This tag defines the path of the output document, and a list of global parameters for the template.

6.2.1 Define generation output

<output> tag is optional. If not present, the document is generated at template location, with suffix '_generated'

If defined, the syntax is the following:

```
<config>
  <output path=<<Absolute path of the document to be generated>> />  ...
</config>
```

Global parameters can be used to define a relative path.

Example: The generated document will be located in D:/generatedFile.xlsx

```
<config>
  <output path='D:/generatedFile.xlsx' />
  ...
</config>
```

6.2.2 Define global parameters for the template

Global parameters for the template can be defined, for example to define model path, folders to use or any other static value to be used in template.

Parameters are defined in <config> tag with the following syntax:

```
<config>
  ...
  <param key=<<Parameter1_key>> value=<<Parameter1_value>> />
  <param key=<<Parameter2_key>> value=<<Parameter2_value>> />
  <param .../>
</config>
```

How to access parameters?

- `${paramKey}` inside <context> or other <param> ,
- `gGet(paramKey)` inside a <gendoc> tag

Example: creation of global parameters for model folder, model path, and path of a specific package inside model and example of usage in <context> tag.

```
<config>
  <param key='model_path' value='D:/Models/Model_v1/My_model.uml' />
```

```
<param key='UC_package_path' value='/MyUMLModel/UseCases' />
</config>
<context model='${model_path}' element='${UC_package_path}' />
```

6.2.3 Pre-defined parameters

Some `<param>` are pre-defined in Gendoc and can be used directly in the template.

- `#{input}` is the name of the input template document

Example:

```
<param key='generation_folder' value='D:/Generated' />
<output path='${generation_folder}/${input}-generated.docx' />
```

If the input document is named `template1.docx`, the result file is named `template1-generated.docx`.

The following variables are also ready to be used by default:

- `#{date}` is the date of the generation. The format of the date is 'yyyy-MM-dd-HH:mm:ss'.
- `#{input_directory}` location directory of the template.

Example:

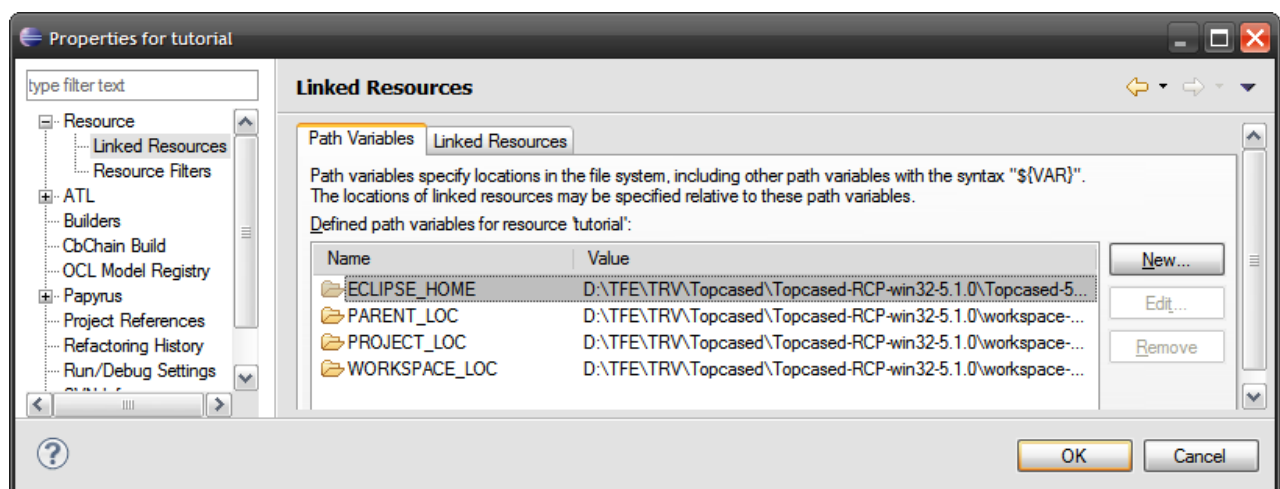
```
<output path='${input_directory} /${input}-generated-#{date} .docx' />
```

Result file example: `template-generated-2014-08-02-093707.docx`

6.2.4 Use of variables inside parameters

It is also possible to use variables defined in project of the document.

From the project on Project Explorer view, right click > Properties > Resource > Linked Resources > Path Variables



Predefined variables or user variables can be used in the template. They are NOT case-sensitive.

Example :

```
<output path='${project_loc}/${input}-generated.docx' />
```

6.2.5 Variables stored in another file

As you may need to put many additional variables inside your project, in order to make the config tag more readable and more reusable, you can put the variables in a file with .properties extension. To access the content of this file you should add the `<properties>` tag in the following manner:

Example :

```
<properties path='${input_directory}/vars.properties' />
```

Where the vars.properties may have the content like this:

Example :

```
output_generation=${workspace_loc}/generated-${date}.docx
input_model_prop=${input}/model.uml
image_test=${project_loc}/company_logo.jpg
```

These variables can be used in Gendoc tags :

Example :

```
<output path='${output_generation}' />
```

6.2.6 Context with CDO models

You can use CDO URIs in context tags

Example :

```
<context
model='cdo.net4j.tcp://localhost:2036/repository/resource?transactional=true'
element='{0}' />
```

6.3 Define script execution context : `<context>` tag

Before a `<gendoc>` tag, a `<context>` must have been defined to determine the model and the element to use as starting context.

`<context>` tag can contain the following attributes :

- **model** : Model absolute path ([global parameters](#) can be used)
- **element** : Path to the model element to use as script context (path from model root)
- **importedBundles** : List of external bundles names (separated by ';' character)
- **searchMetamodels** : false(default)/true.

Set to "true" when your model uses element from external meta-models, for example with SysML, when both UML and SysML meta-models are used.

```
<context
  model='${model_folder}/model_1.uml'
  element='model/package1/subpackage1'
  importedBundles='gmf;papyrus'
  searchMetamodels='true'
/>
```

The context tag **must be defined at least once** in the document.

When executing a script, the **last defined** context is used.

Values of the attributes that are not updated are **kept from previous context**.

For example, “importedBundles” attribute can be defined only once in the document and will be kept until a new value is indicated.

6.3.1 Dealing with specific models

6.3.1.1 Using multiple meta-models

If the model selected references another meta-model, set the following attribute to “true” in order for gendoc to analyse meta-models used as references.

```
searchMetamodels='true'
```

6.3.1.2 Meta-models where elements have no ‘name’ feature

With specific meta-models where elements have no ‘name’ property, context elements are defined:

- Using another property for all the elements in path
 - Ex : `<context ... element='id1/id2/id3' labelFeature='id' />`
Property ‘id’ is used for all elements in path
 - Note : standard case is equivalent to `<context ... element='modelRoot/package1/subPackage1' labelFeature='name' />`
- Using another property only for some elements in path
 - Ex : `<context ... element='modelRoot/id="id2"/subPackage1' />`
Property ‘id’ is used only for the package part of the element path
Property ‘name’ is used for others
- Using indexes of the position inside model tree (starting at 0 and not 1)
 - Ex : `<context ... element='modelRoot/{1}/subPackage1' />`
If package2 is at the second place inside the model.

6.4 Define script parts: `<gendoc>` tag :

Each dynamic part corresponds to a `<gendoc>` tag.

A `<context>` tag must be present before, in order to set the execution context.

`<gendoc>` tag can contain:

- [Acceleo script](http://eclipse.org/acceleo/) (see details on this language on <http://eclipse.org/acceleo/>)
- Static text
- [Styles \(colors / bullets / ...\)](#)
- [Images and diagrams](#)

Note: Rich text content is not supported in this release

Note: Animations inside <gendoc> tags are not supported in this release.

Note: Bookmark are not supported in this release.

Note: External document inclusion is not supported in this release.

Note: List and tables are not supported in this release.

6.4.1 Script language

The content of a gendoc tag corresponds to a script written in [Acceleo](#) language.

Acceleo syntax to display names of all packages

```
Example: display names of all packages
[for (p:Package | Package.allInstances())]
  [p.name/]
[/for]
```

The script is written inside a text box in the slide. When the the scrip is executed, the textboxes are considered ordered by the position of its top-left position. Below there is an example of how to display names of all packages, one in each slide.

Template content	Generation output
<pre><context model='D:/.../myModel.uml' /> <gendoc></pre> <pre>[for (p:Package Package.allInstances())]</pre>	<div></div> <div></div>
<pre>[p.name/] [/for] </gendoc></pre>	<div>Package 1</div> <div>...</div> <div>Package n</div> <div></div>

6.4.2 Text generation

Writing scripts inside a worksheet has a lot of inconveniences but the great advantage is to benefit from all edition functionality, mainly column, row and cell formats.

6.4.2.1 Applying styles to the generation output

The style applied to the text box containing the script inside the template document is kept in the generation output (color, font, size, alignments).

Template content	Generation output
<pre><context model='D:/.../myModel.uml' /> <gendoc></pre>	
<pre>[for (p:Package Package.allInstances())]</pre>	
<pre>[p.name/]</pre>	Actors
<pre>[/for] </gendoc></pre>	Use case View
	Logical View
	Deployment View

All other styles from document templates are kept during generation.

6.4.3 Images generation

`<image>` tag must be defined under a `<gendoc>` tag.

It shall define one of the following attributes:

- **object** for diagram generation, filled with an ID of the diagram. [See Diagram generation section.](#)
- **filePath** for static image generation, filled with the image absolute path. [See static image generation section.](#)

The drawing are correspond to the entire text box:

<code><context model='D:/.../myModel.uml' /> <gendoc></code>
<code>[for (p:Package Package.allInstances())]</code>
<code><image ... /></code>
<code></for></code>

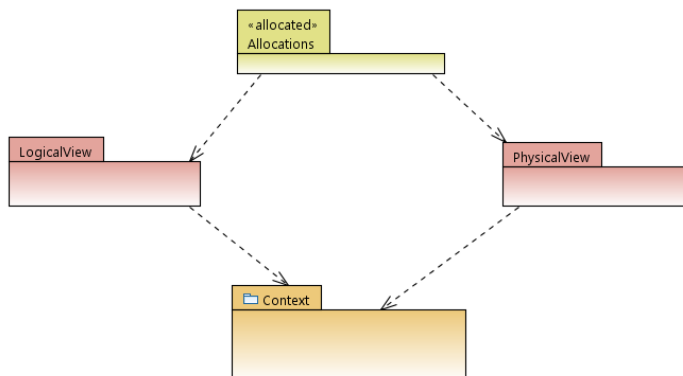
6.4.3.1 Customize image size

`<image>` tag provides the following attributes to handle image size customization: `keepW`, `keepH`, `maxW`, `maxH`. They are used in association with the dimensions of the drawing area inside `<image>` tag:

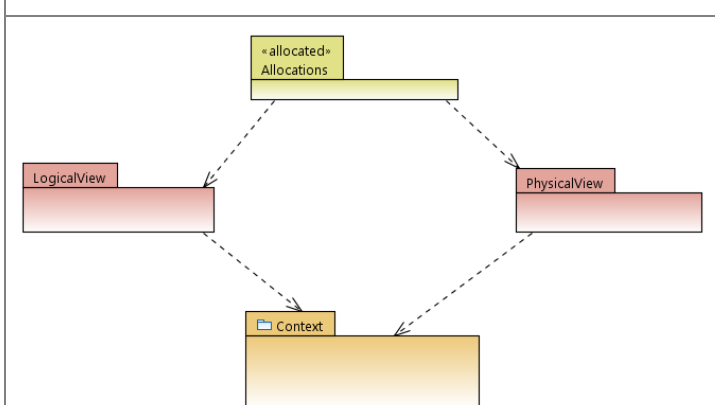
- **keepW**: output image width will be the same as drawing area width
- **keepH**: output image height will be the same as drawing area height
- **maxW**: output image width will not oversize drawing area width
- **maxH**: output image height will not oversize drawing area height

Possible values for these attributes: **false**(default), **true**.

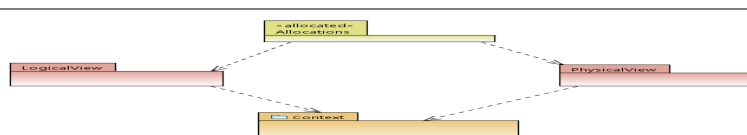
For the following initial image:



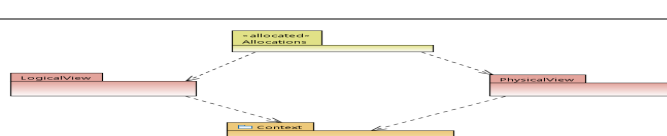
- Fix image width, height is computed proportionally, the image overlap partially the next boxes.

Template content	Output
<pre> <gendoc> <image object='...' keepW='true' keepH='false' /> </gendoc> </pre>	 <p>The diagram shows three boxes: LogicalView (red), PhysicalView (red), and Context (yellow). They are connected by dashed arrows in a diamond shape: LogicalView to PhysicalView, LogicalView to Context, and PhysicalView to Context. Above the diamond is a yellow box labeled '«allocated» Allocations'.</p>

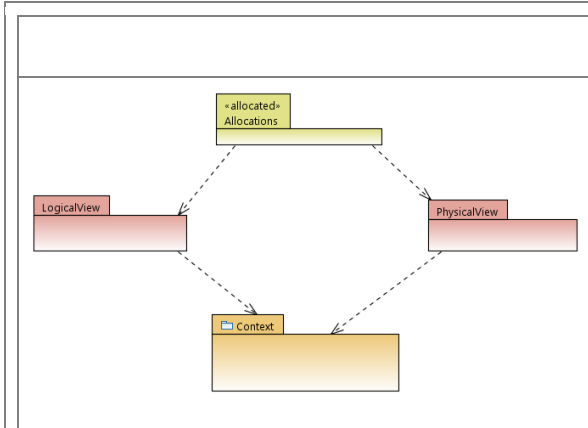
- Fix image height, width is computed proportionally

Template content	Output
<pre> <gendoc> <image object='...' keepW='false' keepH='true' /> </gendoc> </pre>	 <p>The diagram shows three boxes: LogicalView (red), PhysicalView (red), and Context (yellow). They are connected by dashed arrows in a diamond shape: LogicalView to PhysicalView, LogicalView to Context, and PhysicalView to Context. Above the diamond is a yellow box labeled '«allocated» Allocations'.</p>

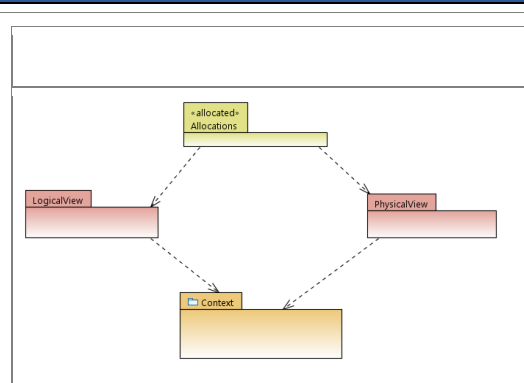
- TO BE AVOIDED : Fix image height and width

Template content	Output
<pre> <gendoc> <image object='...' keepW='true' keepH='true' /> </gendoc> </pre>	 <p>The diagram shows three boxes: LogicalView (red), PhysicalView (red), and Context (yellow). They are connected by dashed arrows in a diamond shape: LogicalView to PhysicalView, LogicalView to Context, and PhysicalView to Context. Above the diamond is a yellow box labeled '«allocated» Allocations'.</p>

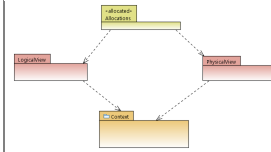
- Ensure the image will not oversize a specified width
 - Case 1 : Image is smaller than the drawing area
Output corresponds to origin image dimensions, and the picture may overlap with the following rows.

Template content	Output
<pre><gendoc> <image object='...' maxW='true' /> </gendoc></pre>	 <p>The diagram shows three boxes: 'LogicalView' (red), 'PhysicalView' (red), and 'Context' (yellow). 'LogicalView' and 'PhysicalView' are connected by a dashed arrow. 'PhysicalView' and 'Context' are connected by a dashed arrow. 'LogicalView' and 'Context' are connected by a dashed arrow. The diagram is smaller than the drawing area.</p>

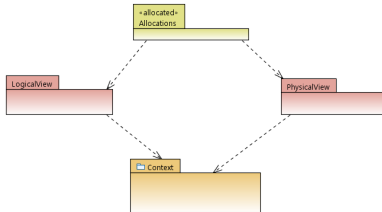
- Case 2 : Image is larger than the drawing area
Output corresponds to drawing area dimension:

Template content	Output
<pre><gendoc> <image object='...' maxW='true' /> </gendoc></pre>	 <p>The diagram shows three boxes: 'LogicalView' (red), 'PhysicalView' (red), and 'Context' (yellow). 'LogicalView' and 'PhysicalView' are connected by a dashed arrow. 'PhysicalView' and 'Context' are connected by a dashed arrow. 'LogicalView' and 'Context' are connected by a dashed arrow. The diagram is scaled to fit the drawing area.</p>

- Ensure the image will not oversize a specified height
 - Case 1 : Image is smaller than the drawing area
Output corresponds to origin image dimensions

Template content	Output
<pre><gendoc> <image object='...' maxH='true' /> </gendoc></pre>	

- Case 2 : Image is larger than the drawing area
Output corresponds to drawing area dimension:

Template content	Output
<pre><gendoc> <image object='...' maxH='true' /> </gendoc></pre>	

6.4.4 Displaying diagrams

Attribute object shall be filled by an ID of the diagram. Diagram ID can be generated by [service getDiagram from bundle gmf](#), called on the diagram:

Template content

```
<context model='${model}' importedBundles='gmf;papyrus' />
<gendoc>
  [for (diag : Diagram| self.getPapyrusDiagrams())

    <image object=' [diag.getDiagram() /]' maxW='true' keepH='false' />


  [/for]
</gendoc>
```

6.4.4.1 Displaying static images

`<image>` tag can also be used for static image generation, with the following content:

- attribute **filePath** shall contain the absolute path of the static image.
The following image formats are supported : GIF, JPG, JPEG, BMP, PNG, SVG
- `<image>` tag shall contain an empty drawing area (alignment, text adaptation, ...)
- size attributes can be used : **keepW**, **keepH**, **maxW**, **maxH**

The following example shows the display of a static image:

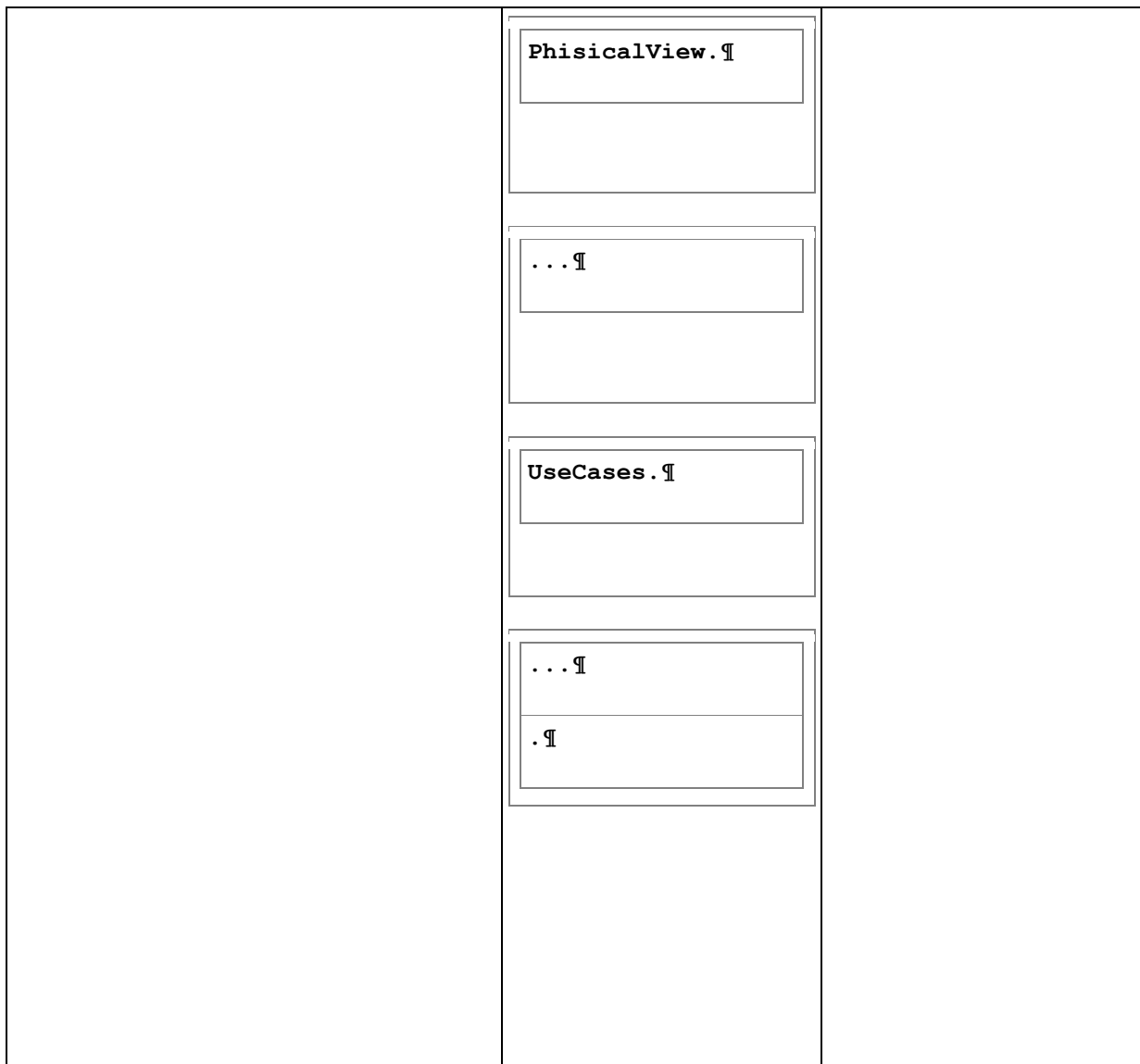
Template content	Output
<pre><context model='\${model_path}' /> <gendoc> Project logo is displayed below : <image filePath='D:/gendoc_logo.jpg' maxW='true' /> [/for] </gendoc></pre>	<p>Project logo is displayed below :</p>  <p>[/for] </gendoc></p>

6.4.5 Formatting

6.4.5.1 Removing extra lines

All characters inside scripts are used for generation output, including spaces, line breaks, or carriage return characters.

Template content	Actual output	Expected output
<div><gendoc>¶</div> <div> <pre> ..[for (p:Package self.ownedElement->filter(Package)->sortedBy(name))].¶ </pre> </div>	<div>¶</div> <div>...¶</div>	<div>Allocations.¶</div>
<div>[p.name/].¶</div>	<div>Allocations.¶</div>	<div>Context.¶</div>
<div>..[/for].¶</div> <div></gendoc>.¶</div>	<div>...¶</div>	<div>LogicalView.¶</div>
	<div>Context.¶</div>	<div>PhysicalView.¶</div>
	<div>...¶</div>	<div>UseCases.¶</div>
	<div>LogicalView.¶</div>	
	<div>...¶</div>	



`<drop/>` tag allows to remove extra **lines**. If after applying `<drop/>` tag, the textbox containing it is empty, the empty text box is removed also.

`<dropSlide/>` tag allows to remove the whole slide.

Document generation is internally performed in two steps and `<drop/>` tag removes the **WHOLE** paragraph in which it is contained and `<dropSlide/>` tag removes the whole slide in which it is contained, so it must be handled with care.

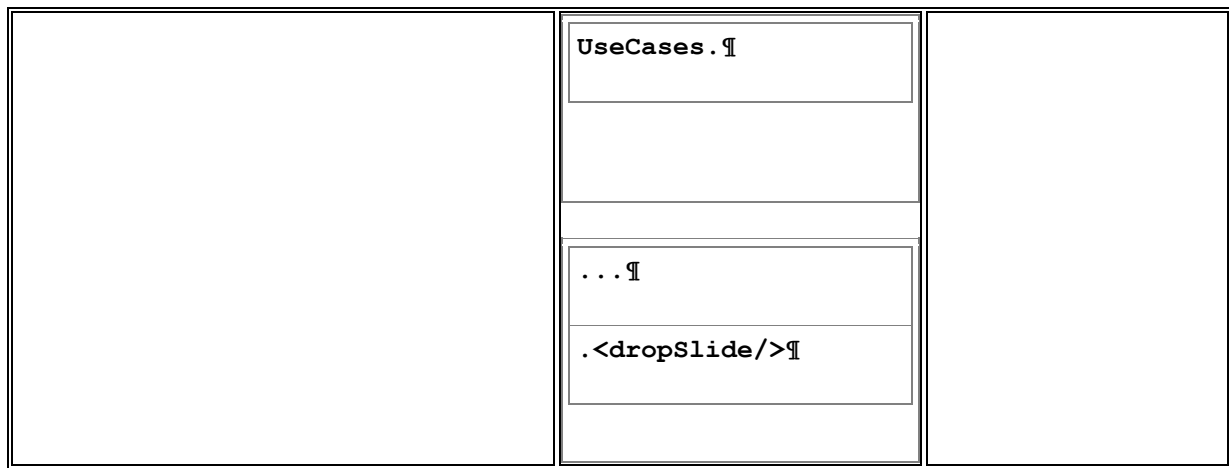
First step is to analyze the lines to get as output to understand where the extra lines come from in the template and where the `<drop/>` tags should be located.

Template content	Output
<div><gendoc>¶</div> <div>..[for (p:Package self.ownedElement->filter(Package)->sortedBy (name))].¶</div>	<div>¶</div> <div>..¶</div>
<div>[p.name/]¶</div>	<div>Allocations.¶</div>
<div>..[/for]¶</div> <div></gendoc>¶</div>	<div>..¶</div> <div>Context.¶</div> <div>..¶</div> <div>LogicalView.¶</div>

	<div> <div> <div>..</div> <div> <div></div> <div></div> </div> </div> </div>
	<div>PhysicalView.</div>

Template content	before <drop/> and <dropSlide/> handling	Final output
<div> <div><gendoc></div> <div> <div>..[for (p:Package self.ownedElement->filter(Package) ->sortedBy(name))].<dropSlide/></div> </div> </div>	<div> <div></div> <div> <div>...<dropSlide/></div> </div> </div>	<div> <div>Allocations.</div> </div>

[p.name/].¶	Allocations.¶	Context.¶
..[/for].¶	.<dropSlide/>..¶	LogicalView.¶
</gendoc>.<dropSlide>¶		
	Context.¶	PhisicalView.¶
	.<dropSlide/>..¶	UseCases.¶
	LogicalView.¶	
	.<dropSlide/>..¶	
	PhisicalView.¶	
	.<dropSlide/>..¶	



6.4.5.2 Removing lines with empty content

Tag `<dropEmpty/>` drop a paragraph if the tag content is empty.

6.5 Reusing gendoc scripts inside the same document: `<fragment>` tag

If a script section is used several times in a same document template, tag `<fragment>` can be used to define the script section and its attributes.

It can then be called from `<gendoc>` tags **inside the same template document**

`<fragment>` tag can contain the following attributes:

- **name (Mandatory)** : the name to call to use the fragment
- **importedBundles** (Optional) : the [external bundles](#) needed in the fragment code (separated by ;)
- **removeClosingLine**(Optional) : when this attribute is set to 'true' the line containing the closing tag (`</fragment>`) is removed during the generation

```
<dropSlide/>
<fragment name='displayDiagram'
importedBundles='commons;gmf;papyrus'>
  <arg name='element' type='uml::Element' />

<dropSlide/>
[for (d:Diagram|element.getPapyrusDiagrams())]
```

```
Diagram [d.name/]:
```



```
<image object=' [d.getDiagram() /]' maxHeight='true' />
```

```
[/for]  
<dropSlide/>  
</fragment>  
<dropSlide/>
```

```
<gendoc>  
[for (p:Package|Package.allInstances())]  
<dropSlide/>
```

```
[p.displayDiagram() /]
```

```
<dropSlide/>
  [/for]
</gendoc>
```

7 Command Line Interface

To generate the documentation from the command line use the example presented below.

```
java -cp
"ECLIPSE_PATH\plugins\org.eclipse.equinox.launcher_XXXX.jar"
org.eclipse.core.launcher.Main -application
org.eclipse.gendoc.batch.GendocBatchMode -data
d:\workspace_directory -idt dir="D:\your_template.docx"
```

Where:

- ECLIPSE_PATH stands for the Eclipse directory
- org.eclipse.equinox.launcher_XXXX.jar – the launcher version
- d:\workspace_directory is the directory of the workspace you use
- d:\your_template.docx is your template path

All the directory paths are absolute.

8 Gendoc bundles

A set of additional services is provided to Gendoc, to be used inside scripts defined in **<gendoc>** tags:

- **commons**: provides some facilities (for special characters, splitting lines, ID generation...)
- **gmf** : provides some services for GMF diagrams generation
- **papyrus**: provide services dedicated to MDT Papyrus models (diagram export, ...)

8.1 Commons

Name: **commons**

This bundle is installed by default and provides the following services, available from **<gendoc>** tags:

- **clean(stringWithSpecialCharacters : String) : String**

Cleans special characters inside the given String. This method needs to be used if a string to be displayed can contain special characters such as: <, >, &, \, " .

- **cleanAndFormat(stringWithFormattingCharacters : String) : String**

Format of the parameter string to display line break or carriage return characters as line breaks, and tabulation characters as tabulations in output document.

- **splitNewLine(stringWithMultipleLines : String) : Sequence(String)**

Split the specified String on the line separator and return a Set of each line to manage manually new lines in a text.

- **gPut (paramKey : OclAny, paramValue : OclAny) : String**

Link a value to a specific key. This variable can be used all over the document (including other gendoc parts) using gGet .

- **gGet (paramKey : OclAny) : OclAny**

Get a value already stored in Gendoc (by gPut or defined in param tags).

- **getText (modelElement : OclAny) : String**

Returns a generic String for the given model element.

- **getId (modelElement : OclAny) : String**

Get a unique id associated to the given model element (to be used for bookmarks for example).

- **getPluginImage(pluginId : String, path : String) : String**

Load an image located in another plugin in order to generated it in the output document. This method should be used inside <image> tag with the following syntax:

```
<image object='[getPluginImage('org.mycompany.myplugin.id',  
'/resources/images/myimage.png')/]' ...>  
...  
</image>
```

8.1.1 Advanced services from bundle “commons”

Several services allow to load other models in order to be able to use their elements inside scripts.

- **load(modelElement : OclAny, extensionReplacement : String) : String**

Load a model located next to the model where the modelElement comes from. For example, for a Package p contained in the file located at file://c:/test/file.uml, the call: p.load('notation') will load the file located at file://c:/test/file.notation

- **loadRelative(modelElement : OclAny, relativePath : String) : String**

Load a model located with a relative path to the model where the modelElement comes from. For example, for a Package p contained in the file located at file:///c:/test/file.uml, the call: p.load('../file2.notation') will load the file located at file:///c:/file2.notation.

- **loadURI(uri : String) : String**

Load a model from its URI. For example, the call : p.load('file:///c:/test/file.notation') will load the file located at file:///c:/test/file.notation.

Loading another model in order to be able to use its elements in script.

8.2 HTML

- **isHtml(String) : Boolean**

Return if the given string contains HTML tags.

- **stripHtmlTags(String) : String**

Return a string without html tags.

- **htmlToText(String) : String**

Return a plain text string for the given html string. This provide simple indentation and list marks.

- **textToHtml(String) : String**

Return a html string for the given plain string. It consider list marks and space and tabs to provide indentation in the HTML string, trying to keep the simple format of the provided plain text.

8.3 Gmf

This bundle provides a set of services available with all GMF models. This bundle is **NOT** configured by default.

It must be referenced inside attribute [importedBundles](#) from **<context>** tag:

```
<context ... importedBundles='gmf' />
```

- **getDiagram(diag : Diagram) : String**

Get diagram ID for a given diagram : element to use inside <image> tag with the following syntax:

```
<image object='[diag.getDiagram() /]' ...>
```

```
...
```

</image>

- **getDiagram(diagram : Diagram, modelElementsToDisplay : Sequence(OclAny)) : String**

Get diagram ID for a diagram: element to use inside <image> tag. User can provide a list of elements in the diagram to display.

- **getElementsInDiagram(diagram : Diagram) : Sequence(EObject)**

Get all elements contained in the diagram.

- **isDiagramEmpty(diagram : Diagram) : boolean**

Indicates if the diagram passed as a parameter is empty or not.

8.3.1 Advanced services concerning gmf diagrams

8.3.1.1 Customize image generation format

By default, images are generated in PNG format. On Unix OS, they are generated in JPG format.

The following services allow customizing the generation format among the following: PNG, JPEG, GIF, BMP, JPG, SVG.

- **getDiagramExt(diagram : Diagram, imageExtension : String) : String**

Get diagram ID for a diagram: element to use inside <image> tag. The second parameter allows choosing the extension of the exported image.

- **getDiagramExt(diagram : Diagram, imageExtension : String, modelElementsToDisplay : Sequence(OclAny)) : String**

Get diagram ID for a diagram: element to use inside <image> tag. The second parameter allows choosing the extension of the exported image. User can provide a filtered list of elements in the diagram to display.

8.3.1.2 Getting diagrams from models

- **getDiagramsInModel(modelElement : OclAny) : Sequence(Diagram)**

Get all diagrams found in the model file resource the modelElement belongs to.

8.4 Papyrus

- **getPapyrusDiagrams(EObject) : Sequence(Diagram)**

Get all the diagrams with the object as root object.

- **getPapyrusOwnedDiagrams(EObject) : Sequence(Diagram)**

Get all the diagrams owned by the object.

- **getDocumentation(EObject) : String**

Get the documentation of an object.

- **getDocumentationResources(EObject) : Sequence(String)**

Get the documentation resources of an object.

- **replaceLinksByNameOrLabel(String, EObject) : String**

Replace the links in the string with the name or the label of the object, using the Eobject as context to resolve the link.

- **getPapyrusTables (EObject) : Sequence(Table)**

Get the tables of an object.

- **getAppliedComment(Element) : Bag(String)**

Get the applied comments.

8.5 Capella

- **getSiriusDiagrams(EObject) : Sequence(Diagram)**

Get the diagrams of an object.

- **getSiriusDiagramName(Diagram) : String**

Get the name of the diagram.

- **getSiriusTables(EObject) : Sequence(Table)**

Get the tables of an object

APPENDIX: Overview of all Gendoc tags and attributes

```
[0..1] <config>
  [0..1] <output
    [1..1] path='absolutePath'>
  [0..*] <param
    [1..1] key='uniqueParamKey'
    [1..1] value='paramValue'>
</config>

[0..1] <bookmarks>
  [0..*] <alias
    [1..1] source='uniqueAliasKey'
    [1..1] target='replacementValue' /> ...
</bookmarks>

[1..*] <context
```

```

[1..1] model='full_model_file_path'
[0..1] element='package_name/element_name'
[0..1] importedBundles='bundle1;bundle2;bundle3'
[0..1] searchMetamodels='true' (default='false')/>

[1..*] <gendoc>
..
[0..*] <image
    [0..1] object='image_id'
    [0..1] filePath='absolute_path'
    [0..1] keepW='true' (default='false')
    [0..1] keepH='true' (default='false')
    [0..1] maxW='true' (default='false')
    [0..1] maxH='true' (default='false')>
    Drawing area
</image>
[0..*] <table
    [0..1] object='image_id'>
    Rows or tables
</table>
[0..*] <include filePath='absolute_path' />
[0..*] <richText
    [0..1] format='RTF' (default:'HTML')
    [0..1] filePath='absolute_path'>
</richText>
[0..*] <drop/>
[0..*] <dropSlide/>
[0..*] <dropEmpty>..</dropEmpty>
[0..*] <nobr/>
</gendoc>

[0..*] <fragment
[1..1] name='unique_fragment_name'
[0..1] importedBundles='bundle1;bundle2'
[0..1] importedFragments='fragment1;fragment2'>
[0..1] removeClosingLine='true' (default='false')>

[0..*] <arg
    [1..1] name='argument1'
    [0..1] type='full_metamodel_element_name' />
..
</fragment>

```