

# M2Doc User Guide

---

## Template authoring

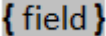
The M2Doc technology adopts an approach where the document authoring tools (Libre Office, Open Office, MS Word) are leverage as much as possible. What other tool is more adapted to style and static part authoring?

Furthermore, these tools are quite common and widely adopted so that there's no necessity to learn yet another document authoring tool. Last but not least, there's a great deal of document models legacy all over the places that should be reused as easily as possible.

Templates are made of static parts and dynamic parts. Static parts are produced in the generated document as they are in the templates while dynamic parts are replaced by some text which depends on the provided input models. Dynamic parts are provided in fields so that there's always a clear separation between static and dynamic parts.

## How to cope with field codes

The edition of template must be made in a mode where field codes are visible. In MS Word, you can toggle this mode on/off by pressing Alt-F9. When the mode is on, you should see this **Erreur ! Signet**

**non défini.** like the next picture shows: 

To insert a new field, press Ctrl-F9. You obtain an empty field like this:



That you must edit to provide the code and instructions.

The next section gives all the details that are necessary to edit M2Doc generation tags. Alternatively, if you start your template from the provided model (templateModel.dotx), you can use the Insert>QuickPart>AutoText to insert M2Doc generation tags. Once you know the tag, it might be faster to just edit them.

You must toggle field codes on to read this document.

## M2Doc static generation tags

M2Doc provides right now 4 generation tags that have fixed formats. Those tags are described below:

- {m:<query>} : the directive is replaced with the string representation of the query's evaluation result
- { m:for <var> | <query>} iterated body { m:endfor } : the directive is replaced by the iterated generation of the body over a collection of values. The specified variable is successively bound to the values of the evaluated collection so that it is accessible from queries inside the body.
- { m:if <expression1>} true branch { m:elseif <expression2> } ... { m:elseif <expression\_n>} ... { m:else } ... {m:endif } : conditional generation. The first branch among the if and elseif directives which expression evaluates to true is processed. If no expression evaluates to true, the else branch is processed (if present).

- { m:image file:"image file path" width:"image insertion width" height:"image insertion height" legend:"image legend" legendPos:"above/below"} : inserts the image which file is specified (through a path relative to the eclipse project where the generation model is placed). The insertion will have the specified width, height and a legend will be inserted at the specified position if one is specified (default is below).

## M2Doc dynamic generation tags

M2Doc provides a way to extend its information retrieval for a tag by registering a provider by an extension point. Each provider uses its own options in addition of the one generically provided par M2Doc core. Many providers can be associated to a same kind of tag and any one of those can be choose to retrieve information in a tag.

### Sirius diagram provider

M2doc gives a way to insert Sirius diagrams: you have to install the Sirius provider feature coming with M2Doc installation.

The diagram tag has the following tag: {m:diagram}, here the different options:

- title: AQL expression, mandatory if you want to get a diagram by its name. Be careful, as it is an AQL expression, if you just want to put a string, you have to add some" around the string: title:""diagram name"",
- width: integer, optional, by default the diagram export real width,
- height: integer, optional, by default the diagram export real height,
- provider: Java class qualified name, optional. Use to define your own way to get a Sirius diagram.

The default way to get a diagram is by its name. If several diagrams have the same name, it will get the first.

However, if the behavior is not enough, you can define your own diagram provider and refers it in "provider" option tag. See the developer guide to add one.

Some examples:

- m:diagram title:""Diagram name"": get the first diagram named "Diagram name" in the modeling project. The diagram size is the export image size.
- m:diagram title:"self.name": if "self.name" is "My Diagram", get the first diagram named "My Diagram" in the modeling project.
- m:diagram title:""Diagram name"" width:"200" height:"200": get the first diagram named "Diagram name" in the modeling project. The diagram picture will be resize to 200x200.

## Document generation and style

The style of the fragments of generated documents is determined by the style of the templates parts.

- Iteration : there's no text style in the tag itself. The style of the body is reproduced as is. The style of the paragraph holding the opening **m:for** tag, however, is reproduced throughout the iterations.
- Conditional : there's no text style in the tag itself. The style of the branch's bodies are reproduced as is.
- Image : there's no text style associated to this style. The paragraph style is reproduced thought.
- Queries : the style of the first run of the expression is used to generate the text that replaces the query's field. For instance, {m :table.comments} will produce comments in orange while {m :table.comments} will produce comments in black.

## How to create dynamic tables

There's no specific tag required to create a dynamic table. Here is an example of a dynamic table description to provide a database table's details:

Name	Description
{ m:for table   db.tables }	
{ m:table.name }	{ m:table.comments }
{ m:endfor }	

### Table { SEQ Tableau \ \* ARABIC } : tables description

It is sufficient to enclose the dynamic part in a generation tag. Here, we have a simple iteration tag. We could have a combination of iteration and conditional tags or whatever other combinations is necessary.

Note : invisible characters might sneak in between the two table fragments (the header and the iterated body). Toggle on the mode where these characters are shown and remove them, if there are any, after the for tag.

## How to create dynamic lists

Creation of dynamic lists is quite similar to dynamic tables : there's no specific tag required. As bulleted or numbered lists are style attributes the style is carried from the template to the generated doc as is :

Tables for database { m:db.name }

{ m:for table | db.tables }

- { m:table.name } :
  - { m:for col | table.columns } { m:col.name } : { m:col.comments }
  - { m:endfor }
- { m:endfor }

**Note** : for the bullets to be correctly spaced and so that there's no spurious carriage return introduced, the endfor tags must be on bullets at the same level as the corresponding tag (as above).

## How to create dynamic hyperlinks

Hyperlinks are WORD fields looking like the following example {HYPERLINK "http://www.obeo.fr"}.

If you want to dynamically retrieve the link with an AQL expression, it is possible. Just adds the request in a WORD field in the double quote like that:

```
{HYPERLINK "m:" http://www.obeo.fr "}"
```

**Warning** : This feature is currently an experimental one. In some context it may be not working.

## Headings and tables of contents

Headings are just treated like tables and lists are : the style makes it all.

Here's how we would create headings that corresponds to a database tables:

**{ m:for table |db.tables } Table { m:table.name }**

**Description** : { m:table.comments }

**{ m:for col |table.columns }Column { m:col.name }**

Description : { m:col.comments }

{ m:endfor }

{ m:endfor }

Insertion of table of contents has no interactions with the templating. The table of content is just a field which is processed by Word which collects all the headings. The only requirement is to make an update of the field right after document generation.

## Define variables

To write AQL expressions, variables are useful to simplify the syntax and to have an easy access to model elements.

Variables can be defined in the template document with template properties:

The screenshot shows the 'Personnalisation' tab of a configuration window. It includes a list box for 'Nom' with options: Assistante, Bureau, Client, Date enregistrement, De la part de, and Destination. There are 'Ajouter' and 'Supprimer' buttons next to it. Below is a 'Type' dropdown set to 'Texte', a 'Valeur' text field, and a checkbox 'Lier au contenu'. The 'Propriétés' section features a table with the following data:

Nom	Valeur	Type
m:var:author	String	Texte
m:var:db	database::Database	Texte

With this example, we can write AQL expressions like 'm:author' or 'm:db.name'.

The variables names have to be set with 'm:var:' prefix: "m:var:author" will define 'author' variable. The variables values defines the variable type (author is a string and db a model element database::Database). The possible basic types are 'int', 'real', 'string', 'boolean', 'date'. To type an object, you have to set its domain class like in AQL syntax: prefix::ClassName (database::Database).

These variables are automatically imported in the configuration model where you can value them. If you do not define the variable in the template document, you will have to do it in the configuration model (see Configure variables definitions paragraph) .

### Miscelaneous points

- Adding dynamic content in headers and footers is simply done by inserting generation tags in the header and footer.
- Dynamic content in text area isn't supported yet. Any generation tag that appears in a text area will just be reproduced as is and won't be processed as a generation tag.

### Documentation generation

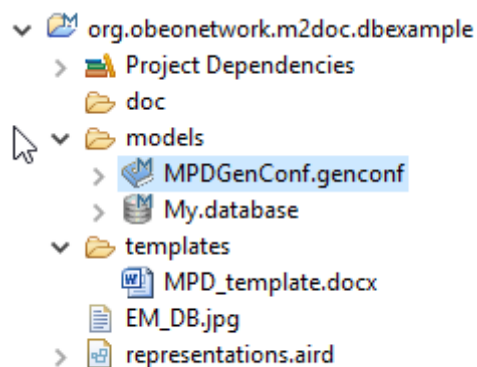
As of now, custom properties are not implemented and a generation configuration model must be used to generate a document from a template.

Document generation is done in three steps :

- Template authoring (in a modeling project)
- Generation configuration with the creation of a configuration model which binds values to variables
- Generation itself

We provide a step by step tutorial that explains how to generate a document from a database model. To start with, M2Doc and the database DSL must be installed in your bundle. Then, import the example modeling project (org.obeonetwork.m2doc.dbexample) in your workspace. (File>Import>General>Existing Project into Workspace). The example project is on github: <https://github.com/ObeoNetwork/M2Doc/tree/master/doc/example/org.obeonetwork.m2doc.dbexample>.

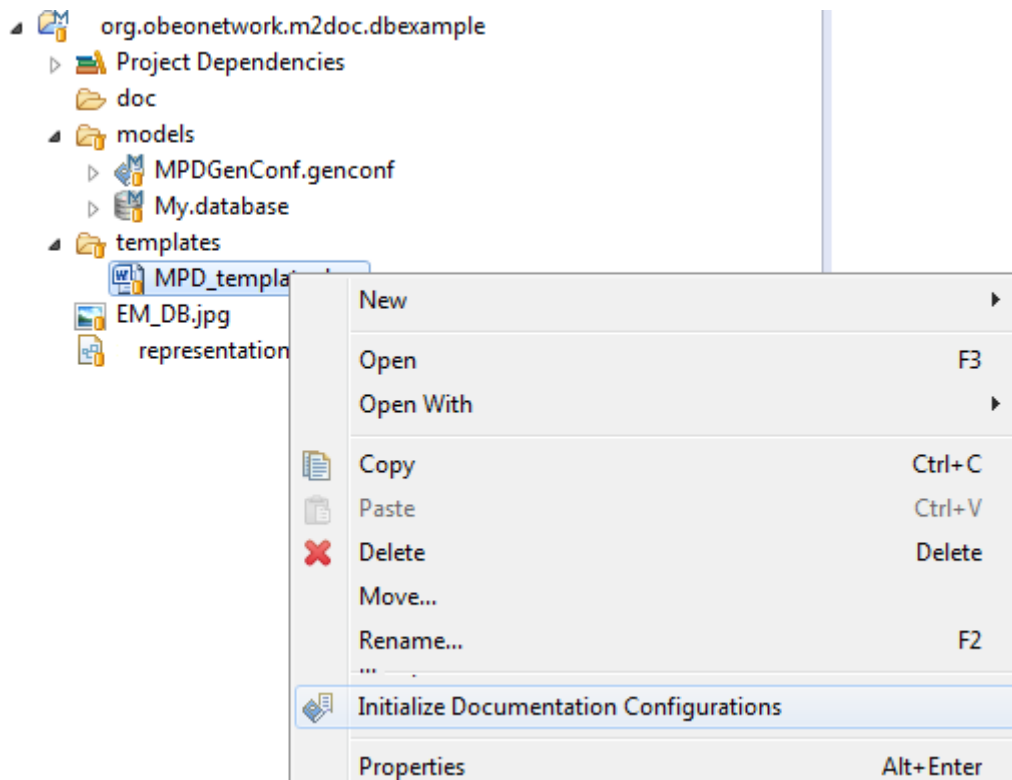
Open the project. You should have this folder layout :



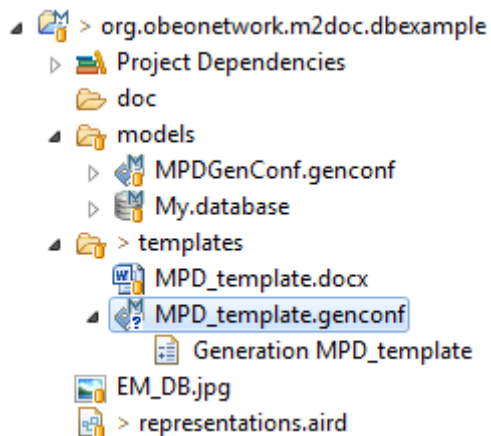
We will keep the existing configuration model (.genconf) and create another one to document the procedure.

## Initialize the configuration model

First, initialize the configuration model: right click on the template file >Initialize Documentation Configurations:



The configuration model is created by default near the template.



You will have to set those two URIs in the Package NSURI attribute:

- <http://www.obionetwork.org/dsl/database/1.0>
- <http://www.obionetwork.org/dsl/typeslibrary/1.0>

and the result file name.

The resulting property sheet should look like this:

Properties Problems Error Log <> Interpreter	
Property	Value
Definitions	author, db
Name	
Packages NSURI	http://www.obeonetwork.org/dsl/database/1.0, http://www.obeonetwork.org/dsl/typeslibrary/1.0
Result File Name	doc/result.docx
Services Tokens	
Template File Name	templates/MPD_template.docx
Time Stamped	true

## Configure variables definitions

Now, you have to configure the variables definitions. If these variables are set in the template file, they are automatically added to the configuration model and you just need to value them.

Otherwise, you have to create these variables definitions.

## Manage variables with M2Doc Sirius viewpoint

A Sirius viewpoint “M2Doc” is activated on the “org.obeonetwork.m2doc.dbexample” example project. In the configuration diagram on Generation element, you can manage the variables definitions.

Two kind of variable definitions can be created:

- string definition: to add one, use the palette tool and set the key and the value (direct edit or properties view)

The screenshot shows the M2Doc Sirius viewpoint. The top part is the Configuration Diagram, which contains two variable definition boxes. The first box is labeled 'author: John Doe' and the second is labeled 'db: DataBase[TRANSIENT]'. Below the diagram is the Properties view for the selected 'author: John Doe' element. The Properties view has a table with the following data:

Semantic	Property	Value
Style	author: John Doe	
Appearance	Key	author
	Value	John Doe

- model definition: to add one, select the model element value in the model explorer and drag n drop it into the diagram and set the key name (direct edit or property view). You can also use the palette tool and select the model element value.



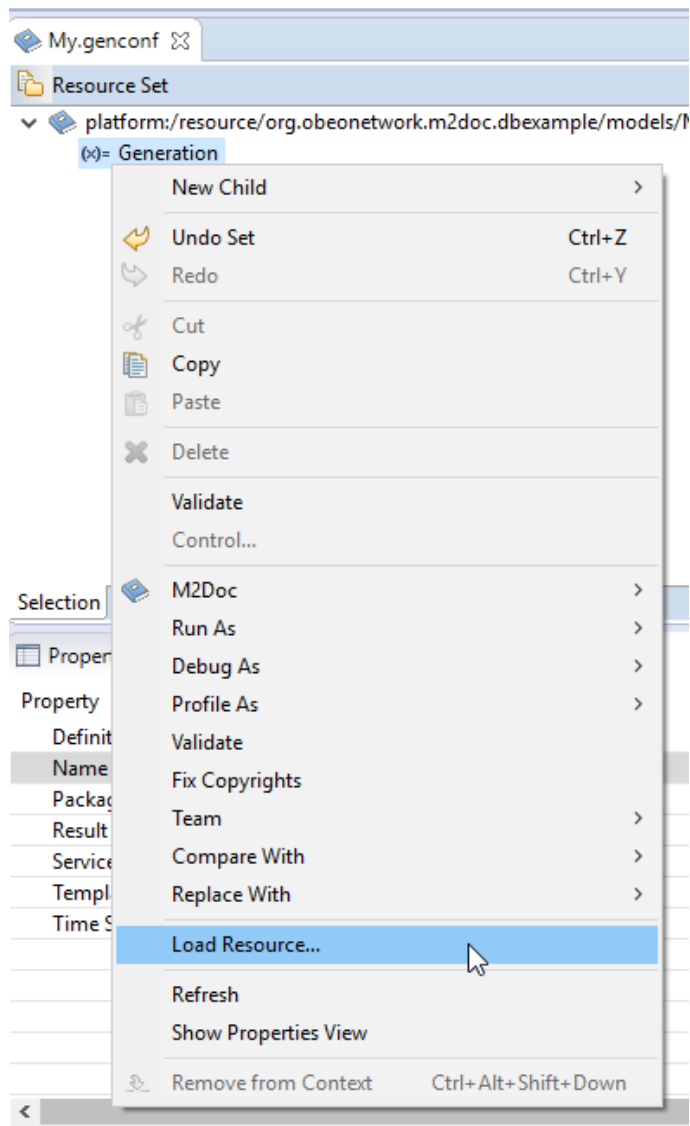
The screenshot shows the Sirius EMF Editor interface. The main workspace displays a Configuration Diagram with two nodes: a light blue box labeled 'author: John Doe' and a darker blue box labeled 'db: DataBase[TRANSIENT]'. The 'db' node is currently selected, indicated by a black border. Below the workspace, the Properties view is open, showing the semantic properties of the selected node.

Property	Value
db: DataBase[TRANSIENT]	
Key	db
Type	DataBase -> Tab
Value	Data Base EM_D

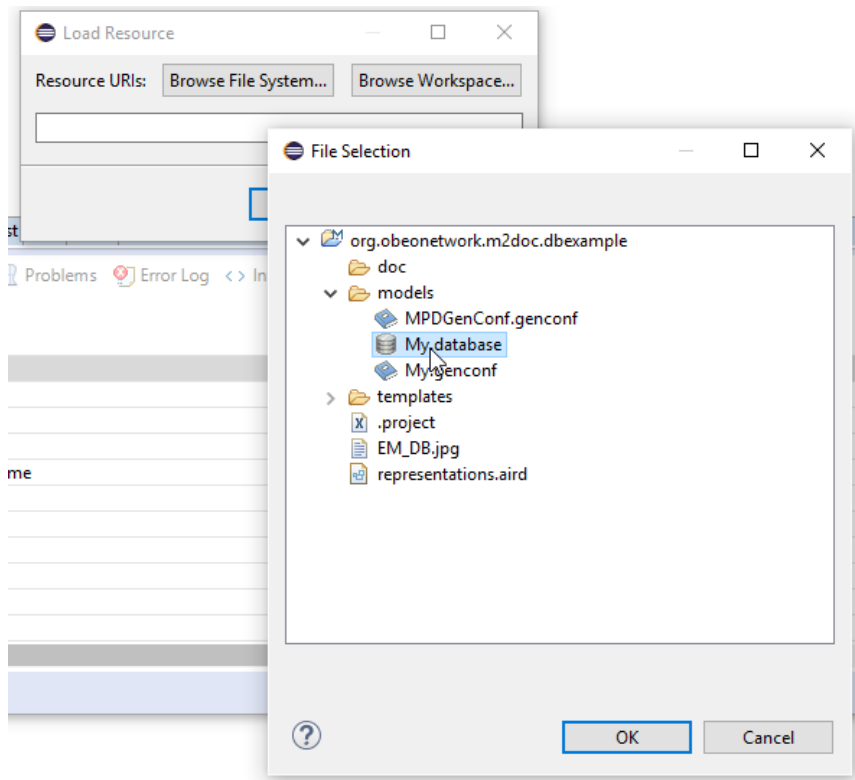
### Create variable with EMF Editor

If you do not have Sirius, you have to set the variable via the genconf EMF Editor:

First, you have to load the main database resource :

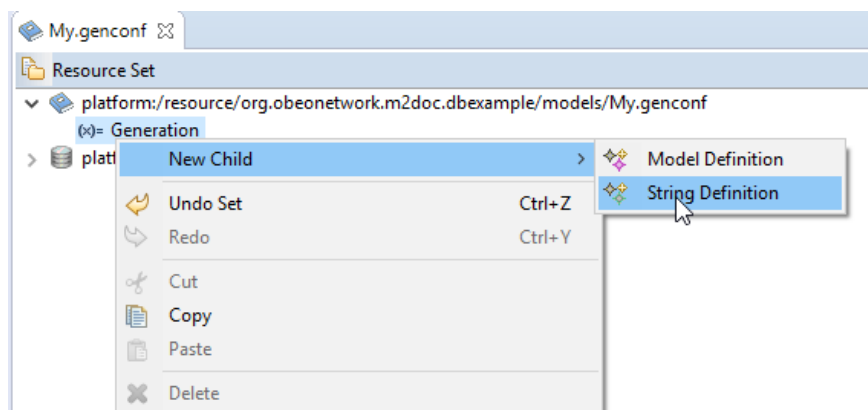


Select the database model that is in the model directory :

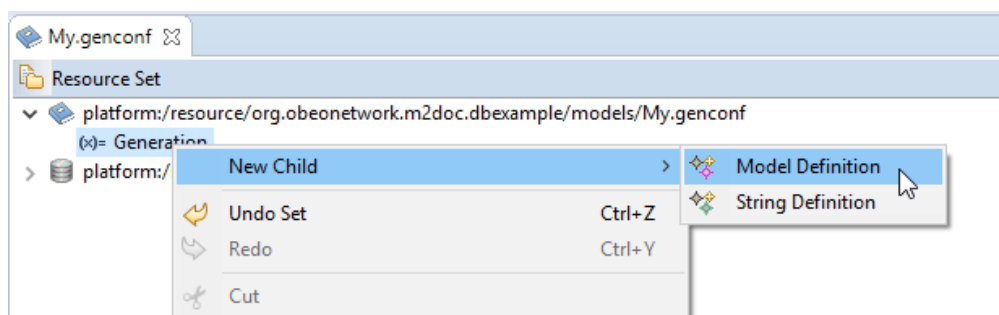


Now that the database resource is loaded, we will be able to use model element in it as variable's values.

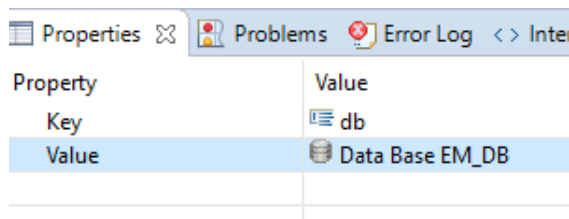
We first create a string variable, key='author', value='John Doe':



And a model variable :

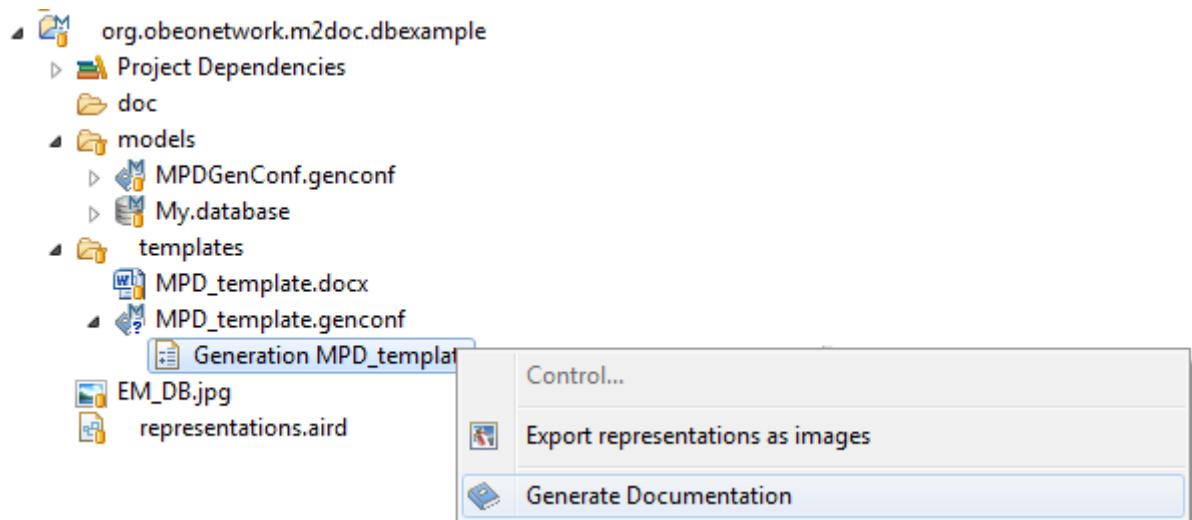


Set the values of the variables in the property view. The model variable must be defined like follows :

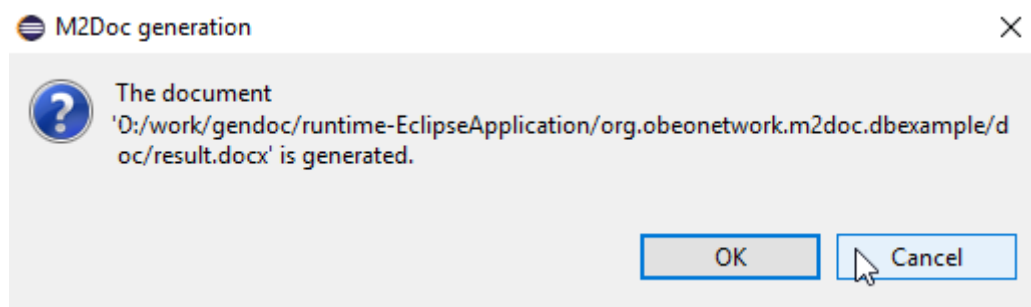


## Generate documentation

You just have to invoke the generation by right clicking on the generation model element :



A message indicates everything went well :

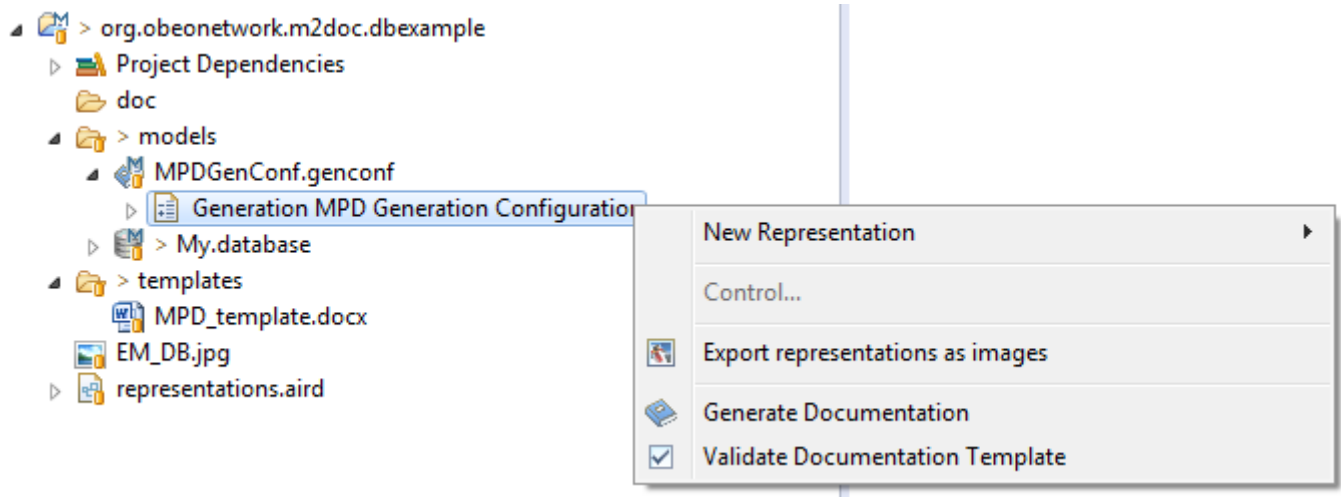


Refresh the 'doc' directory if needed. The result should be in it.

## Template validation

When the generation is launched, we first run a template validation. If this validation gets errors, an error file will be generated near the generated result file.

The template validation can also be started alone, if it gets errors, an error file will be generated near the template file.



The template validation checks :

- M2Doc syntax,
- AQL expression syntax and typing,
- Feature and services called in AQL expressions exist,
- Variables syntax and typing.