MBEE DocGen

DocGen 3 Manual

MBEE DocGen: DocGen 3 Manual

Publication date Sat May 17 11:50:26 PDT 2014

Table of Contents

List of Figures

List of Tables

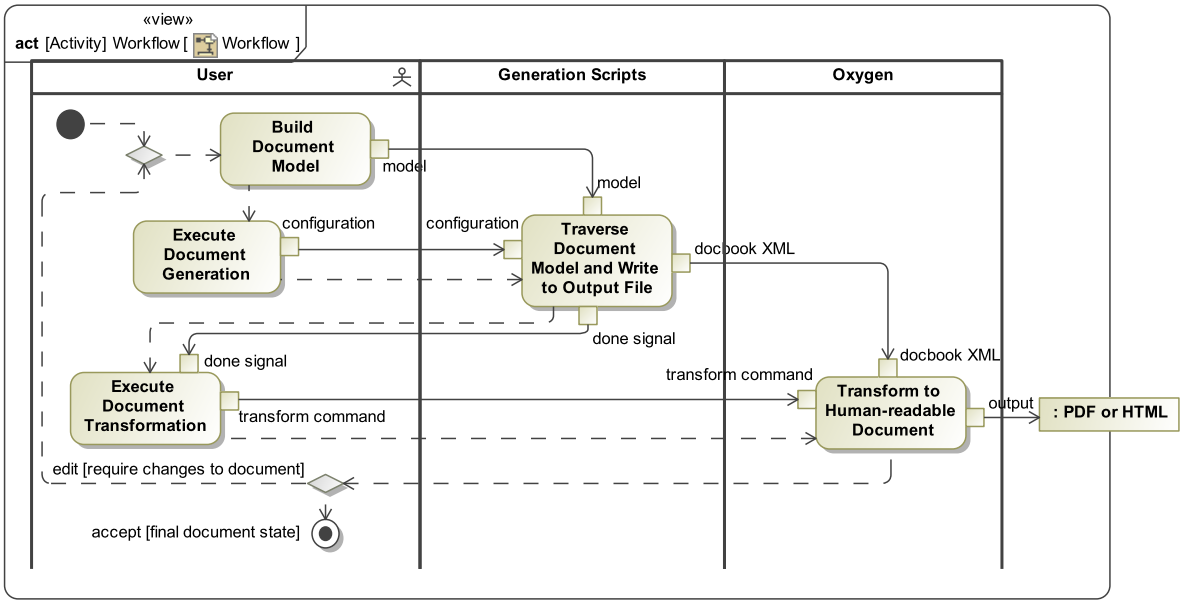
1. Introduction

The Document Generator provides capabilities for generating formal documents from UML/SysML models in MagicDraw. For the purposes of this tool, a "document" is a view into a model, or a representation of model data, that may be structured in sequential and nesting pieces. More concretely, a document is a collection of paragraphs, sections, and analysis where the order and depth of the content is important. The Document Generator (DocGen) operates within MagicDraw, traversing a document "outline" and collecting information, performing analysis and writing the output to a file. The Document Generator produces a DocBook XML file, which may be fed into transformation tools to produce the document in PDF, HTML, or other formats.

This is an example document generated using the Document Model profile in a test project. This document and project as a set demonstrate the application of the current set of queries (reusable analysis functions) provided by the Document Model to generate views as text, tables and images of data from a model. This document captures the current state of DocGen; one of the strengths of DocGen is that it may be extended and queries may be added as needed to perform any analysis desired.

DocGen consists of a UML profile with elements for creating a document framework; a set of scripts for traversing document frameworks, conducting analysis, and producing the output; and a set of tools to help users validate the correctness and completeness of their documents. Users will have to invest the time to create the document framework; however, once this is done, the document can be produced from a button click whenever the model data is updated. The user never has to waste time numbering sections or fighting with reluctant formatting, as this is all performed automatically during the transformation to PDF, HTML, etc.

Figure 1.1. Workflow



2. Installation

DocGen is distributed as a MagicDraw plugin zip file and is part of the MDK plugin. The current version is DocGen 3 for MD 17.0.2. DocGen 2 is no longer supported. If you happen to have an old version of MagicDraw that still has DocGen 2, please see https://docweb.jpl.nasa.gov/app/link/4/ for the DocGen 2 manual.

Install:

1. Get the latest SSCAE Magicdraw prebuilt distribution. **The SSCAE download is preferred** unless you're told otherwise by an MDK developer or something.
2. If you got the SSCAE prebuilt, there's no need to do the following steps. If you got the plugin:
3. Do not unzip the file. Start MagicDraw, go to Help->Resource/Plugin Manager
4. Click on Import and choose the zip file. Restart MagicDraw.

Use:

1. Open or create a project. Go to Options->Modules.
2. Click on Use Module and selected the SysML Extensions.mdxml file (this should be in the md.install/profiles/MDK directory)
3. Now you can use the stereotypes from the Document Profile 3, inside Architecture Framework

3. Using DocGen 3

Using Viewpoints and Views (For Beginners)

DocGen 3's new view and viewpoint semantics makes it a lot easier to put together a default document quickly. For beginners, this is the simplest way to get started.

Figure 3.1. Creating DocGen 3 View Diagram

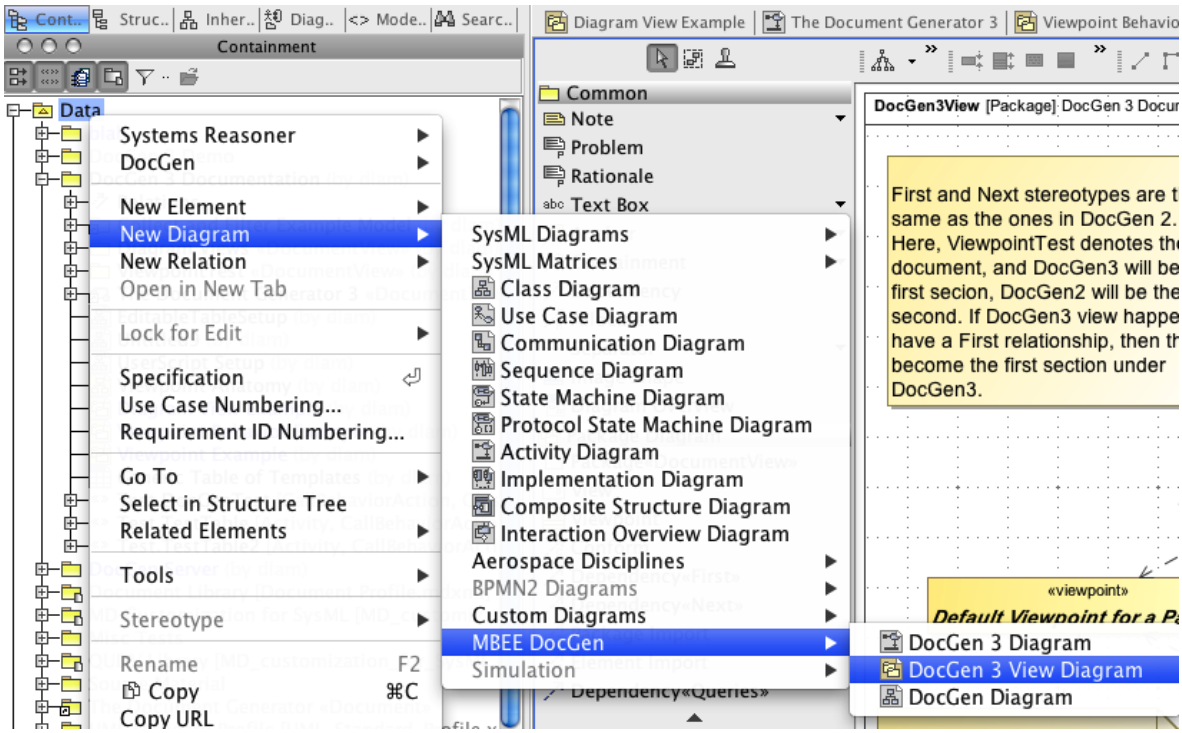
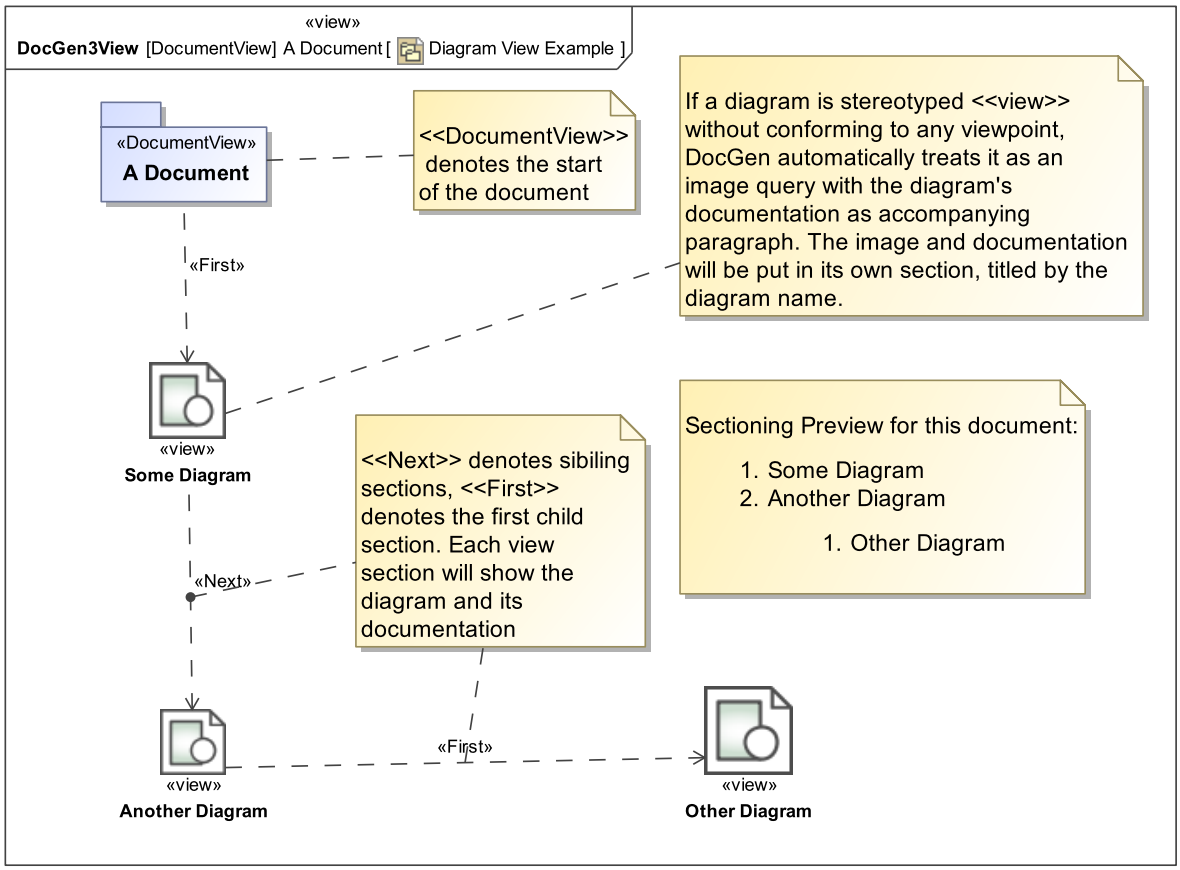


Diagram Views

The simplest degenerate case for a document is if you have diagrams and their documentations. The following shows how you can string your diagrams together to create a simple document. For each diagram (and its documentation) you want to show, stereotype it <<view>>. The sectioning is applied automatically for each diagram.

Figure 3.2. Diagram View Example



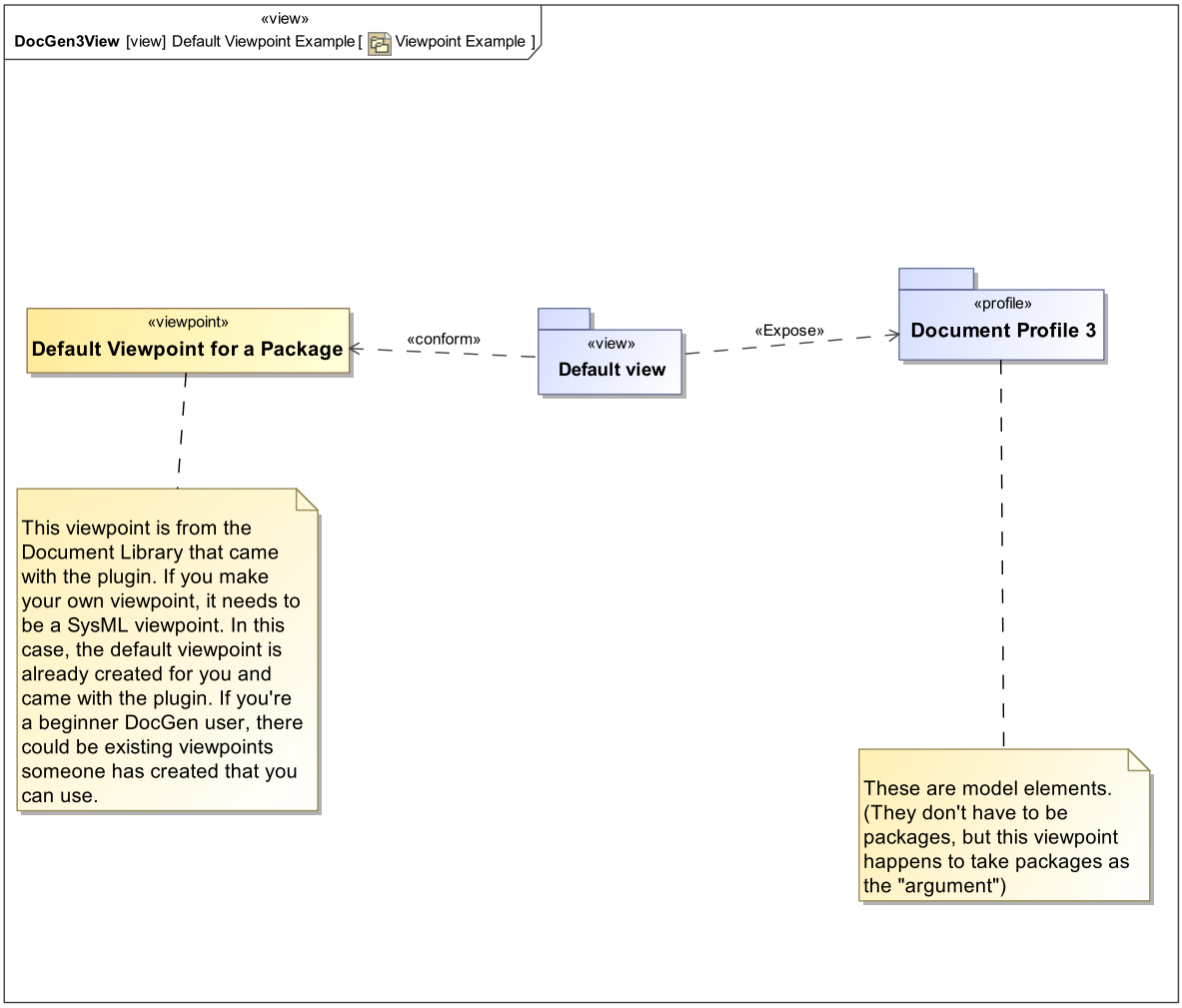
Only use <<view>> if your diagram will not be in multiple documents. If it does, DocGen will not know which first or next relationship belongs to which document. There is another way of using viewpoints and views, which will be shown in the next section. Refer to View Equivalence

Default Viewpoint Example

This is an example of how to use the default package viewpoint that came with the plugin, and to show a more normal way to use viewpoints and views. This viewpoint takes a package, and recursively prints out images and documentation tables for things inside the package, creating section hierarchies that mimic the package structure. To see what this document actually generates, go to https://docweb.jpl.nasa.gov/app/link/217/

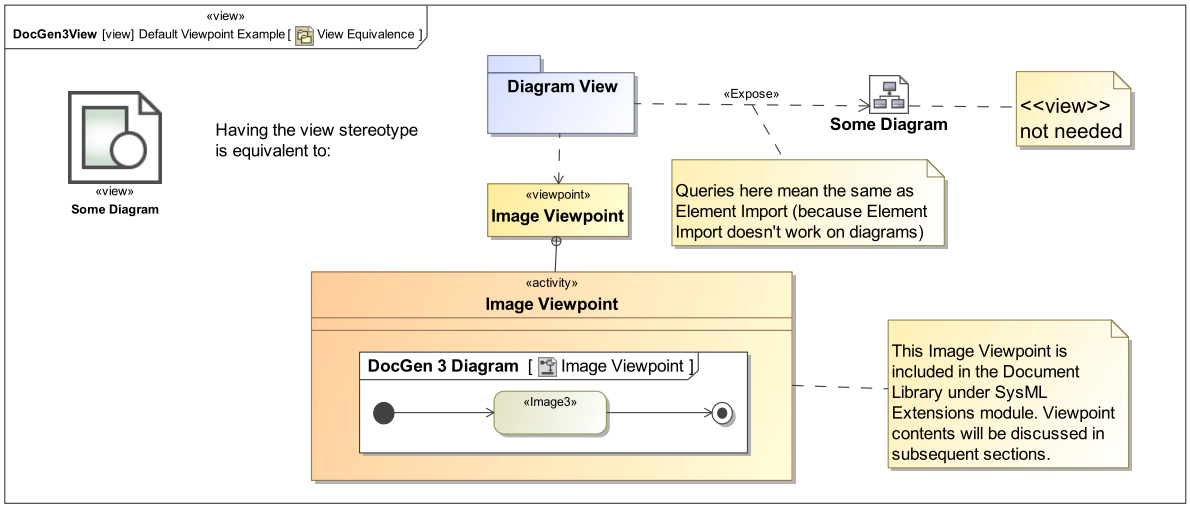
Note: The document generated is not any kind of manual. It's for demonstration purposes only.

Figure 3.3. Viewpoint Example



Given this, the previous diagram view document is really just a shortcut. You can expand the previous document consisting of only diagrams and have them conform to viewpoints that print out the image and documentation, and the result would be the same.

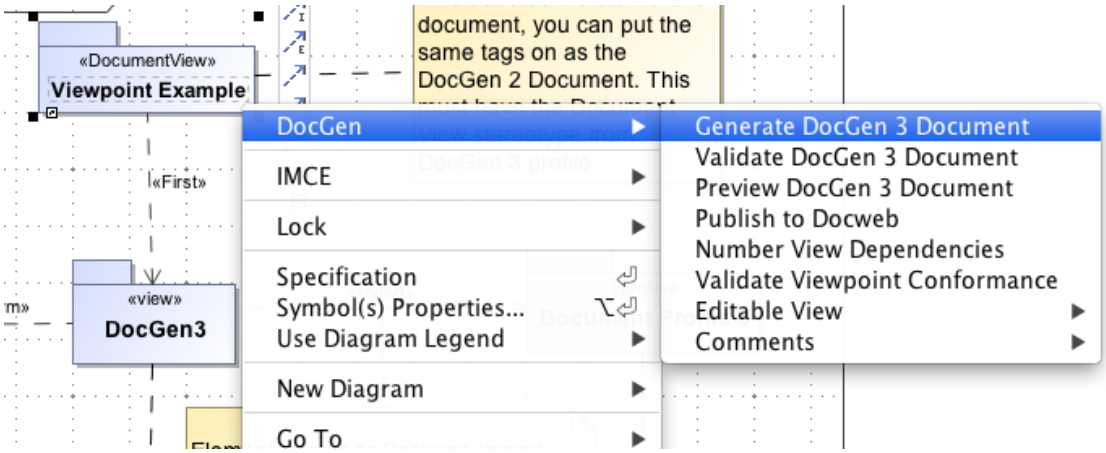
Figure 3.4. View Equivalence



Also, if the view does not query/import any element, but does have a <<conforms>> relationship, then the view element itself would be given to the viewpoint.

To generate the document, right click on the Viewpoint Example (the one with DocumentView stereotype), go to the DocGen submenu, and hit "Generate". It'll prompt for a file to save to. Once it's finished, open it using Oxygen.

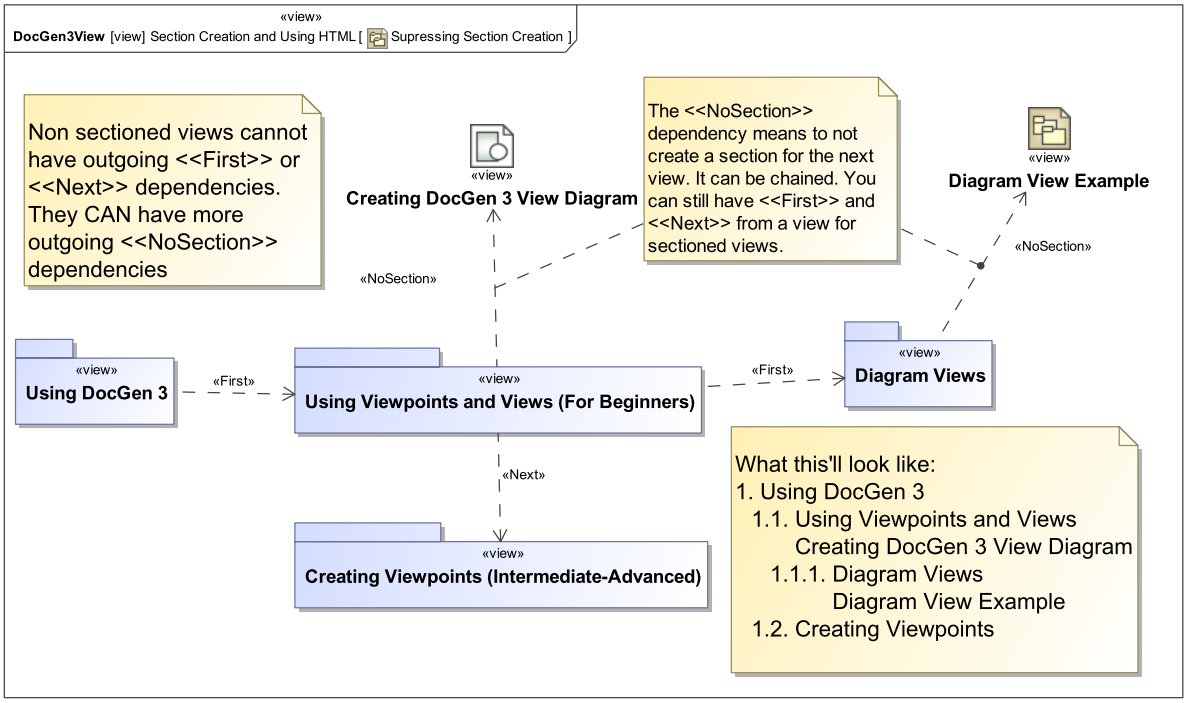
Figure 3.5. Generate



Section Creation and Using HTML

By default, each <<view>> denotes a new section in the document. If you don't want a new section to be created, use <<NoSection>> dependency in place of <<First>>. (You can still have a <<First>> to denote the first child section after the contents from <<NoSection>> dependencies)

Figure 3.6. Supressing Section Creation



HTML Documentations

For text in documentations, you can use plain text or html.

For plain text:

If all you have is a simple sentence or paragraph, this is enough. Newlines will be ignored.

If you know docbook or want specialized docbook tags to be included, this is also the way to go. Simply type in the docbook markup (make sure you have valid docbook tags!)

For html:

Check the html checkbox in magicdraw on any text or documentation field. DocGen will convert html tags to docbook. For the html conversion, there are some limitations to what html tags you can use. In the Magicdraw Advance HTML Editor, you can look at the HTML source tab to see if your html source can be successfully converted into DocBook. Below are a list of tags converted:

- p
- ul
- ol
- li
- b
- i
- u
- a
- sub
- sup
- pre

Examples:

A paragraph

- unordered list item
  - nested list item

1. ordered list item

Bold *Italic* **underline** or hyperlinked text [http://docweb.jpl.nasa.gov], email [mailto:dlam@jpl.nasa.gov], and $_{\text{subscripts}}$ and $^{\text{superscripts}}$.

```
Other preformmatted text like code and xml (the < chars in xml still have to be
<xml>
      <sometag>
      hellllloo and other stuff <>
      </sometag>
</xml>
```

All other styles or font setting will be stripped off as docgen processes it.

Tables made in the Advanced HTML Editor may also be supported, but there must be a <caption></caption> as the first child of the table. The caption will be used as the title of the table generated. You'll have to add this in the HTML source tab.

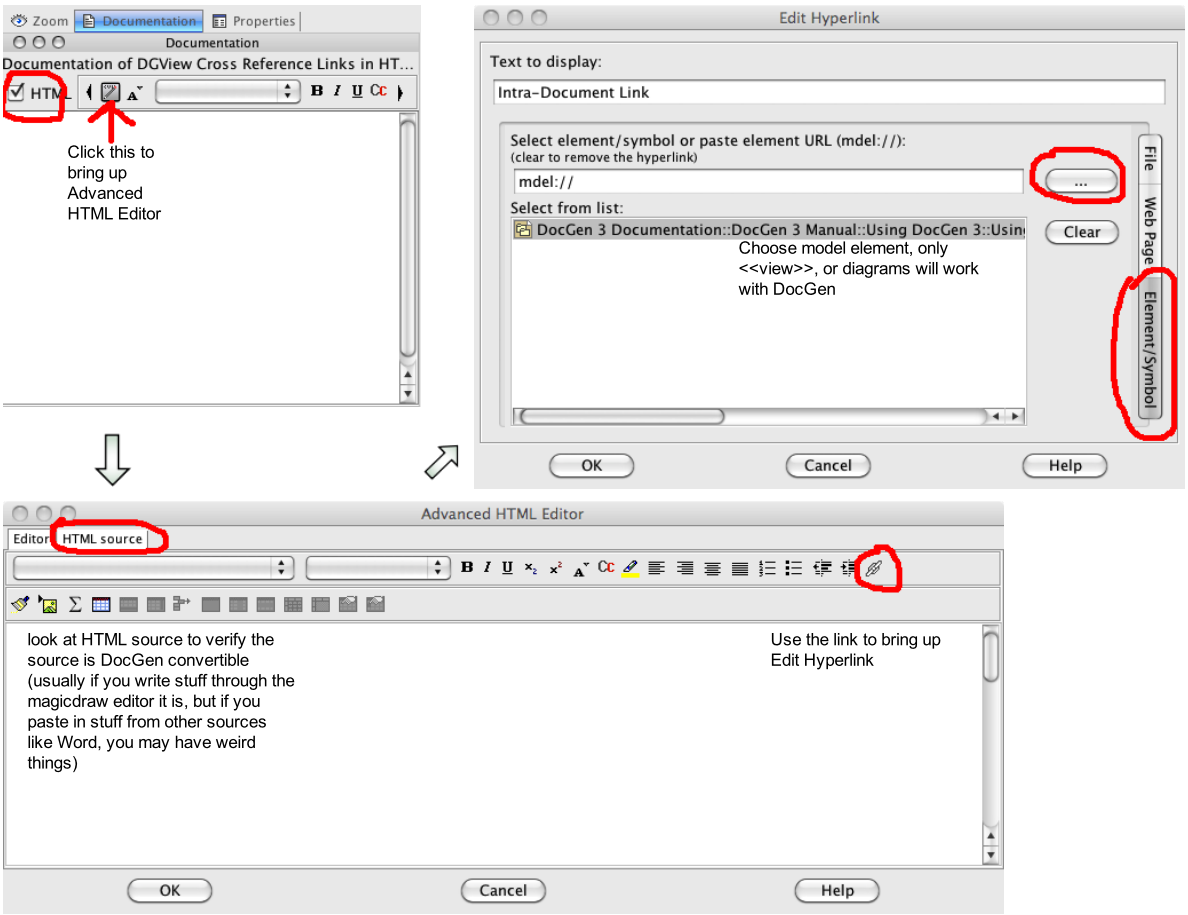
Table 3.1. HTML Table

example	html
table	in html editor

If you don't want a title for your table, make it an informal table. In the HTML source tab, put in <table class="informal" at the table tag. Don't put in the caption. This also means this table won't get an entry in the table of contents.

html	table
without	caption (title)

Figure 3.7. Cross Reference Links in HTML



Cross reference links in html are <a> tags, but their href starts with mdel. It can look something like this:

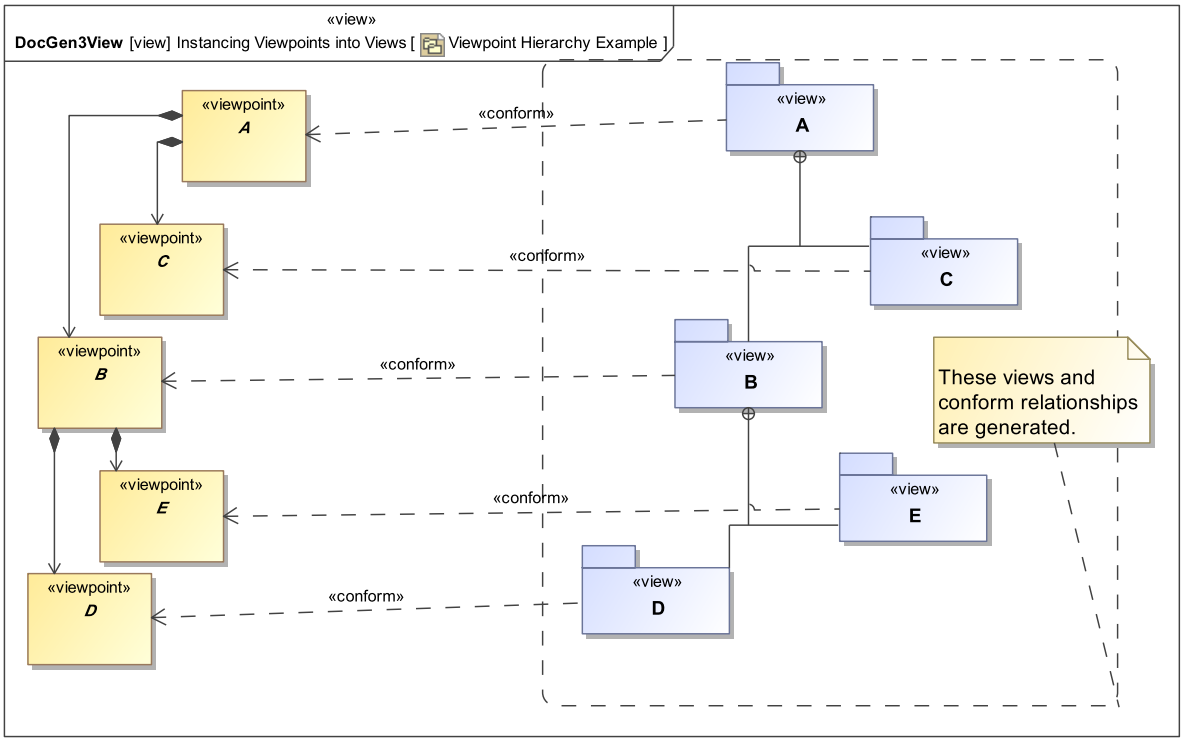
link text

or something to that effect that has the magicdraw element id. When processing a view based document, DocGen 3 will use the view package or diagram element ids as ids in the document. So only link to diagrams or views! (Choose the diagram or view package as the element to link to) It is also assumed that no view or diagram will be used twice in a document so the ids should be unique. If a duplicate id would result, the first element in the document to get the id will win. If you have a view that's the target of a <<NoSection>> dependency, the view's id will be used for the first table in that view, so you can directly reference tables that are in NoSectioned views.

Instancing Viewpoints into Views

If you built a viewpoint hierarchy, you can automatically instance your viewpoints into a view tree with the "conforms" relationship set for you. (But you'll still need to order the views yourself.) Below is an example.

Figure 3.8. Viewpoint Hierarchy Example



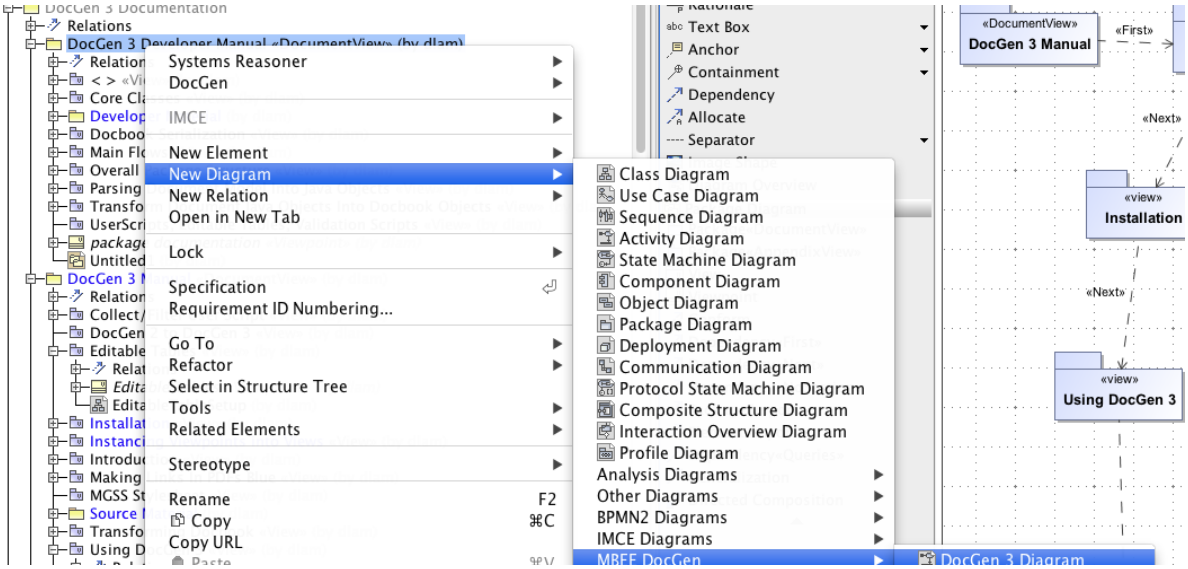
To instance a viewpoint hierarchy, right click on any viewpoint, go to DocGen submenu and choose "Instance Viewpoint". You can choose where to put the generated views.

The viewpoints must be connected by composition relationships. This tool will not order views for you.

Creating Viewpoints (Intermediate-Advanced)

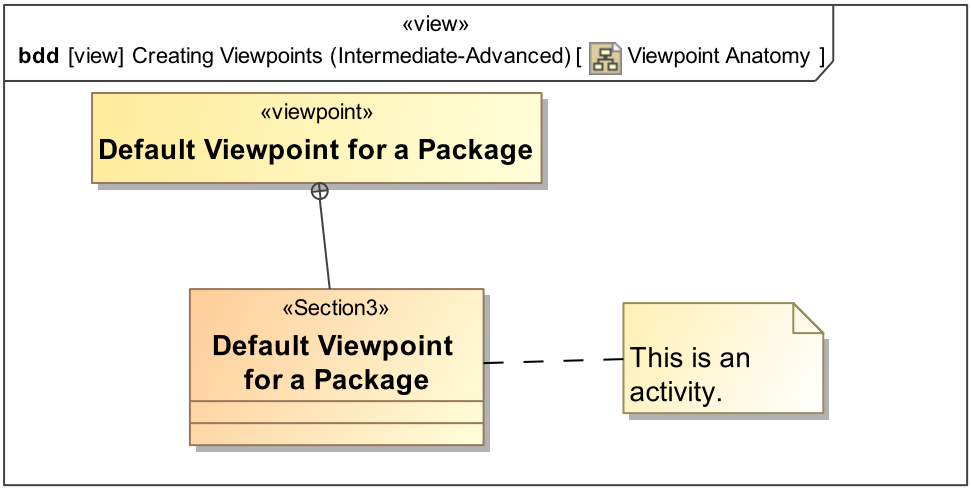
To create a custom viewpoint, make a SysML Viewpoint block first, then create an activity under it (so it becomes an owned behavior). The stereotype of the activity doesn't really matter, but it's best to stereotype it "StructuredQuery". You can choose the DocGen 3 Diagram when you make the activity to get easy access to all DocGen relevant elements

Figure 3.9. Creating DocGen 3 Activity Diagram (For advanced users)



The activity inside each viewpoint is the behavior of the viewpoint.

Figure 3.10. Viewpoint Anatomy



Collecting, Filtering, and Sorting

DocGen 3 separated out the query collection and filtering out into individual "actions". Let's assume that there's an existing context of model elements available for collection and filtering. The examples below show what kinds of model collection constructs are available and how they can be combined. A sequence of these actions can also be made reusable if the activity they're in is stereotyped "CollectAndFilterGroup". Similarly, each collect or filter action can be stereotyped to an activity with predefined tags for reusability (drag them onto activity diagrams to instance them as call behavior actions).

Figure 3.11. Collection and Filtering Actions

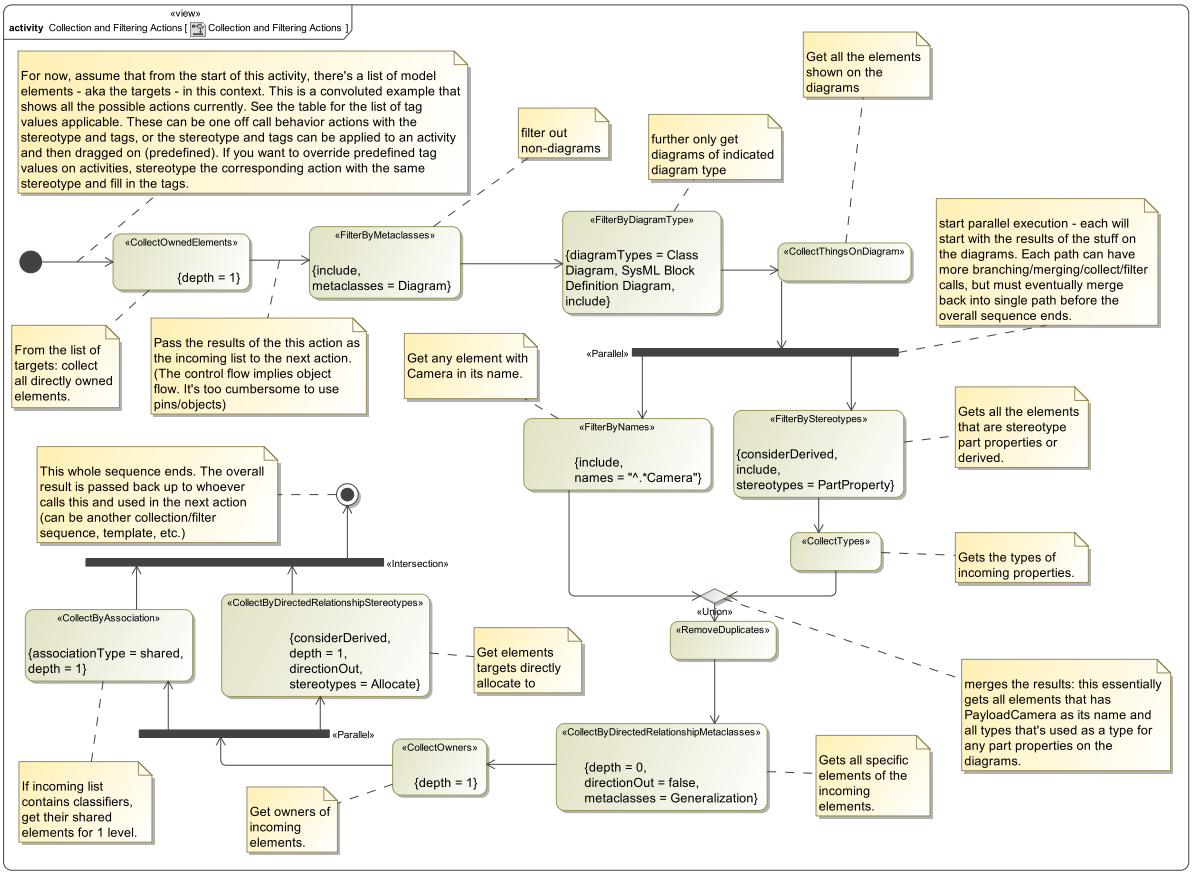


Table 3.2. Collection and Filter Actions

Name	Description
Collect/Filter UserScript	Similar to UserScript, but for collections only. This separates out collection/filter user scripts with presentation template user scripts so they can be mixed and matched or you can use a custom collection then pass to a predefined template.
CollectByAssociation	Get the associated elements (associations are not directed relationships, use this instead of the directed relationship collectors). A similar method would be to get owned properties and then get their types (if you don't care about the type of composition).
CollectByDirectedRelationshipMetaclasses	Collect the ends of the directed relationships whose metaclass are the metaclasses indicated.
CollectByDirectedRelationshipStereotypes	Collect the ends of the directed relationships that are stereotyped by stereotypes indicated.
CollectByExpression	Collect elements by ocl expression.
CollectByStereotypeProperties	Get the value of the stereotype properties indicated of the targets (if the property values are other model elements)
CollectClassifierAttributes	This is only for input types being Classifiers. Will get all the attributes of the classifier with optional inherited attributes. Inherited attributes will be pruned based on redefinition.
CollectOwnedElements	Self explanatory.
CollectOwners	Get the owner of the targets.
CollectThingsOnDiagram	Self explanatory.

Name	Description
CollectTypes	If the given targets are typed elements, gets the type of those elements. If target is call behavior action or call operation action, gets the behavior or operation.
FilterByDiagramType	Self explanatory.
FilterByExpression	Filter elements by ocl expression.
FilterByMetaclasses	Self explanatory.
FilterByNames	Given a list of regular expression patterns, include or exclude the given list of elements. (see java.util.regex.Pattern for how to use regex patterns.) (Unnamed elements will not be included regardless.)
FilterByStereotypes	Self explanatory.
Intersection	From multiple incoming list of elements, take the intersection of the lists and output the resulting element list.
Parallel	During collection and filtering, make "copies" of the incoming list as needed and execute in parallel the outgoing actions. The parallel strings must eventually result in a merge or join node during the collection/filtering sequence.
RemoveDuplicates	Outputs a list of elements with all duplicate elements removed from incoming list.
SortByAttribute	Sort by name, documentation, or value. Value only applies to a collection of value properties or slots.
SortByExpression	Sort elements based on the result of applying an ocl expression to each element.
SortByName	[Deprecated: Use Sort by Attribute] Self explanatory. (Result WILL include unnamed elements and their order is undefined) (This is superceded by SortByAttribute)
SortByProperty	Sort by the value of a property. A property can be a stereotype property or a value property of a classifier.
Union	From multiple incoming lists of elements, take the union of the elements and output the resulting list. (there will be no duplicates)
XOR	Given 2 inputs of a collection of elements, outputs the symmetric difference between them.

These actions can have several options. Since many of the same options can appear on different actions, the table below show all possible options. If you see an option show up in the Tags, that means it's applicable to the particular action.

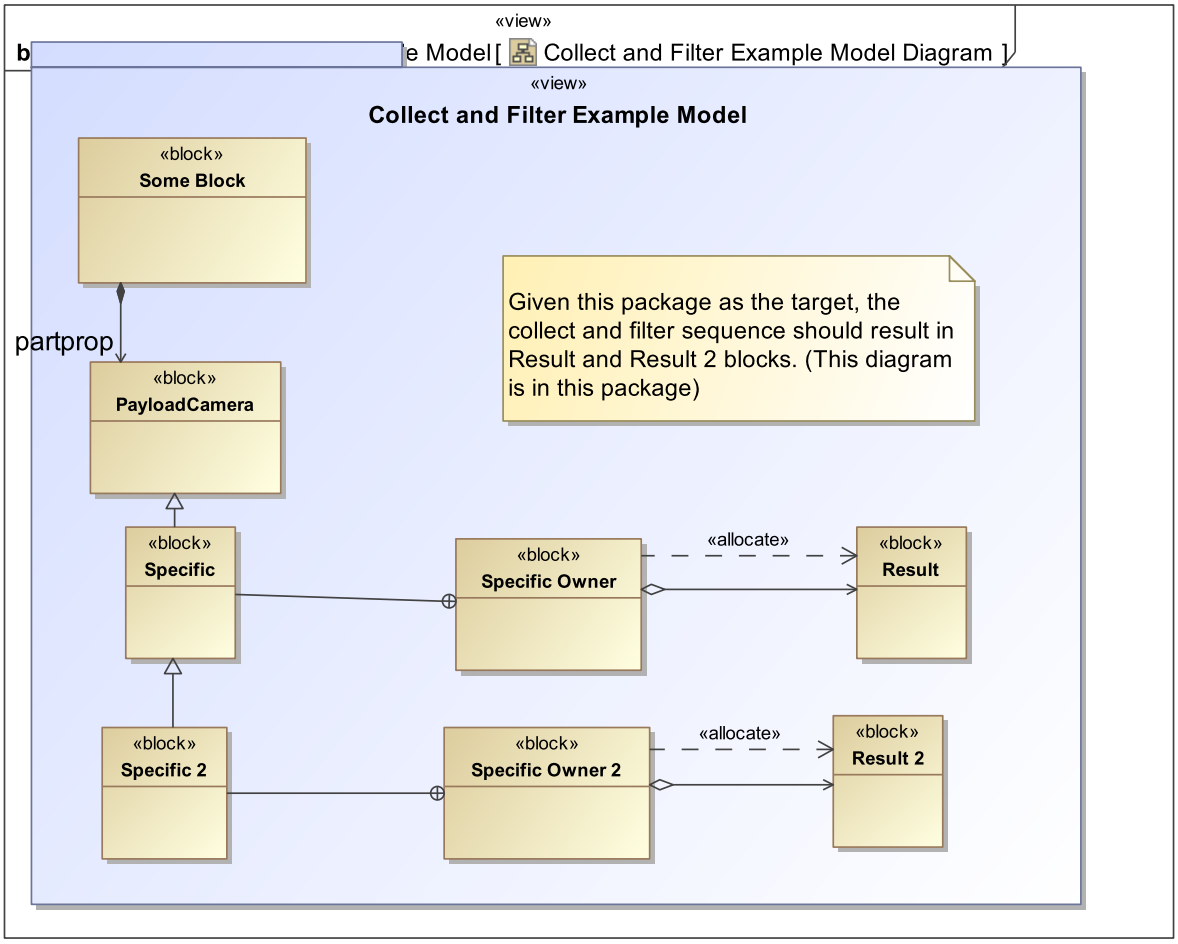
Table 3.3. Applicable Tags

Name	Description
associationType	Same choices as magicdraw's choices for associations.
considerDerived	When collecting or filtering based on stereotypes, choose whether more specific stereotypes should be included.
depth	When collecting by relationships, associations, owned elements, or owner, indicate the depth to crawl. 0 means infinite. 1 means 1 step out, etc.
desiredAttribute	The attribute can be name, documentation, or value (in the case of property or slots)
desiredProperty	Property can be a stereotype property or value property of a class.
diagramTypes	Choose a diagram type.
directionOut	When collecting by relationships, indicate whether the starting point target is the client or supplier of the relationships. (true means to collect "outward")

Name	Description
expression	An ocl expression.
include	Applicable for filtering, choose whether the indicated filter criteria is for inclusion or exclusion.
metaclasses	List of metaclasses for filtering. These are "classes" that come from the UML Standard Profile that comes with Magicdraw, inside the UML 2 Metamodel. In the selection dialog that pops up, if you can't find these classes, make sure a little button with the metamodel icon is "checked".
names	List of regular expression strings. See java.util.regex.Pattern for how regex patterns work.
reverse	Reverse the sort.
stereotypeProperties	Given a list of stereotype properties, will return their value where applicable. This applies to certain tables, bulleted list, and collection action. For templates, they'll print out the property value(s). For collection, returns the values that are model elements. The stereotype properties can also be derived properties in customizations.
stereotypes	List of stereotype for filtering.

Collect and Filter Example Model Diagram

Figure 3.12. Collect and Filter Example Model Diagram

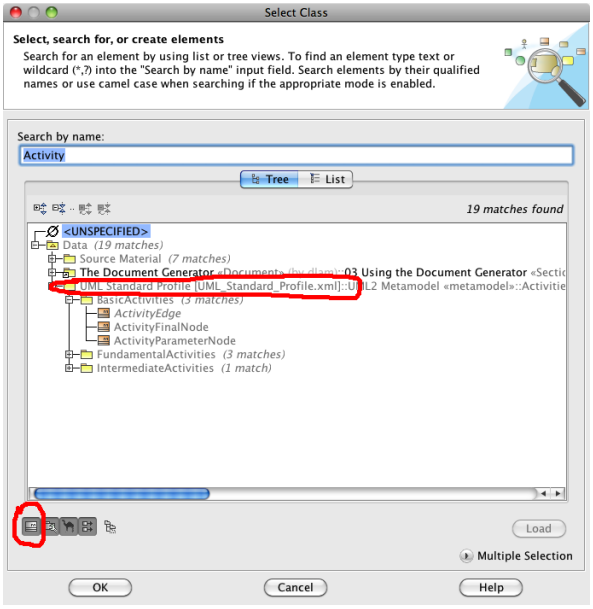


Using the previous collection and filtering activity example, we give the package "Collect and Filter Example Model" to it as the target and output the results as a bulleted list.

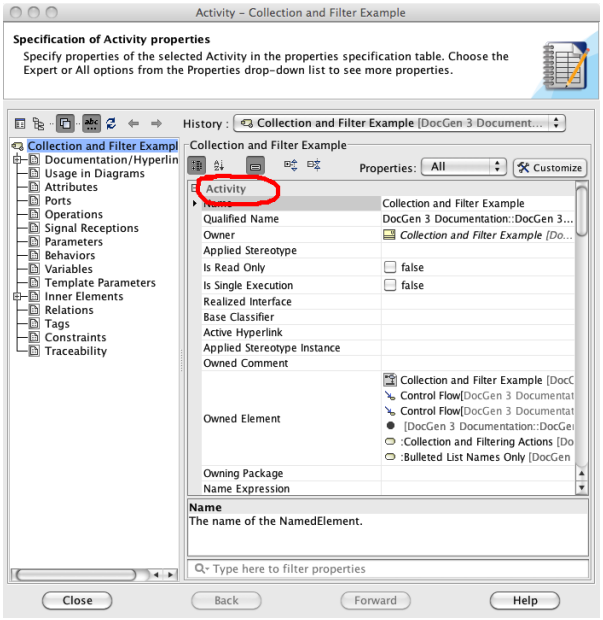
- Result 2
- Result

Note About Metaclasses

Figure 3.13. Note About Metaclasses



Metaclass refers to the type of model element (NOT stereotypes). To choose a metaclass, make sure the little button that looks like class with an m is clicked, then look for the model element with desired metaclass name under UML Standard Profile

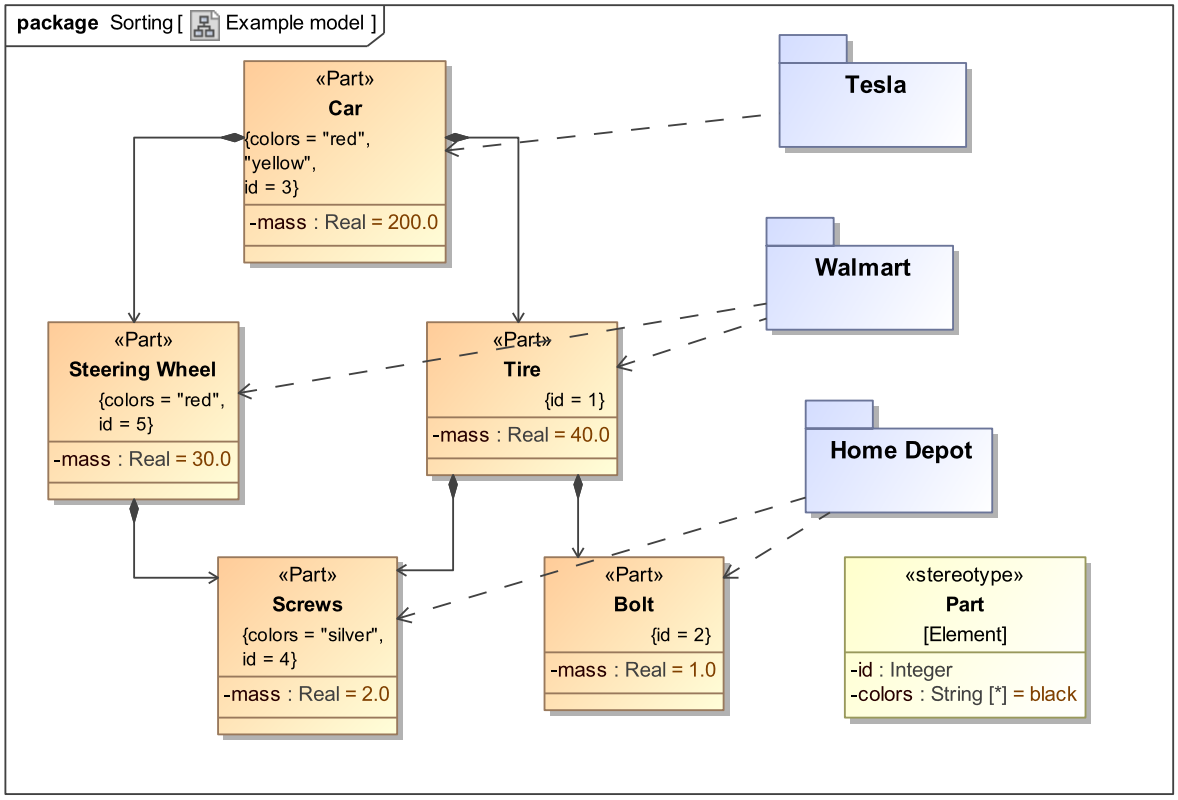


To find out what type a particular model element is, open the spec window and look at the upper left corner. Usually that's the element's metaclass. If your element is in sysml however, like <<Block>>, you'll have to look at the metaclass for the sysml stereotype to find out what it is (this is because sysml masks the metaclass name)

Sorting

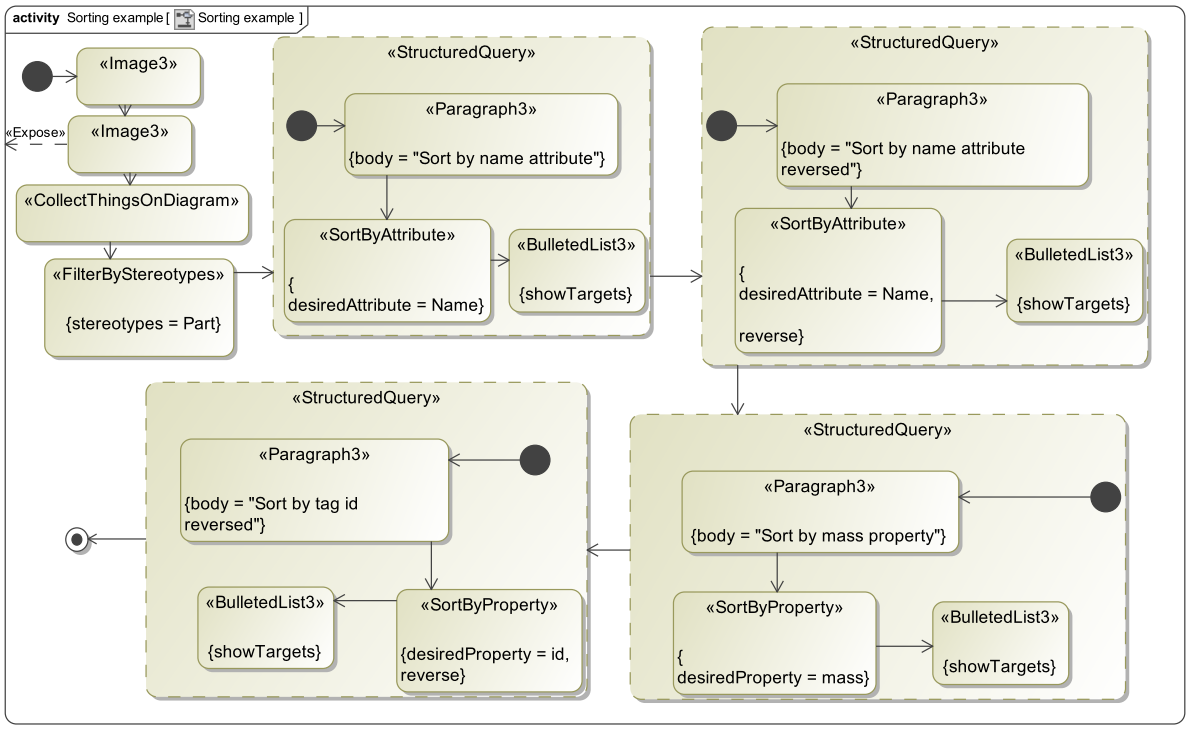
Sorting is used to sort elements before they get passed to presentation templates. Some templates, like list and tables, are naturally ordered, and the output of the sort becomes the input element order of the next DocGen action. Examples follow.

Figure 3.14. Example model



Example Model.

Figure 3.15. Sorting example



This shows how to use the actions.

Sort by name attribute

- Bolt
- Car
- Screws
- Steering Wheel
- Tire

Sort by name attribute reversed

- Tire
- Steering Wheel
- Screws
- Car
- Bolt

Sort by mass property

- Bolt
- Screws
- Steering Wheel
- Tire
- Car

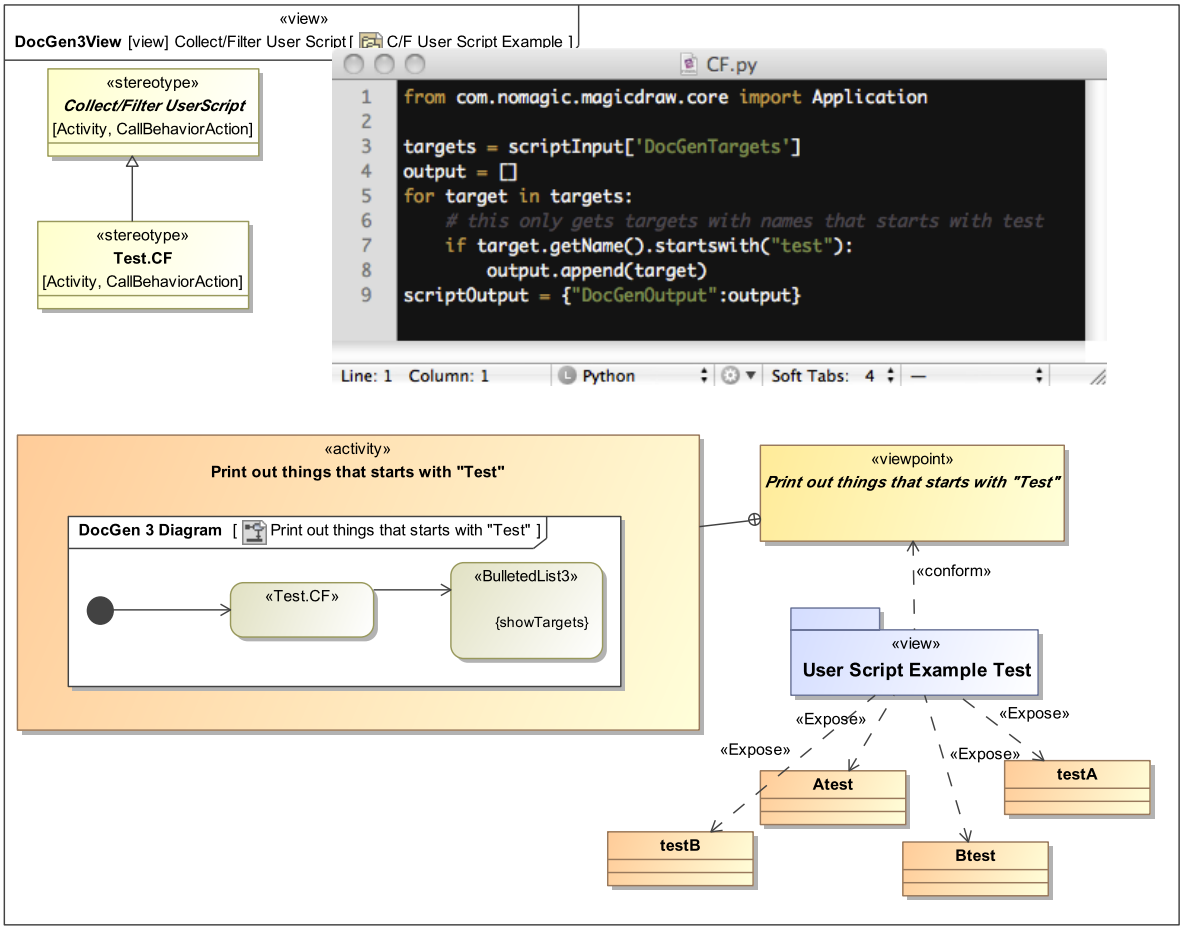
Sort by tag id reversed

- Steering Wheel
- Screws
- Car
- Bolt
- Tire

Collect/Filter User Script

First, look at the UserScript Section. Using the Collect/Filter User Script is exactly the same, except you return a list of model elements instead of docbook elements, and you specialize your stereotype from Collect/Filter User Script instead of just UserScript. This user script can be used anywhere you would use a predefined collect/filter action.

Figure 3.16. C/F User Script Example



Output:

- testA
- testB

Display Templates

Display templates are predefined output formats with some settable options. They take some targets (model elements) and output text, images, lists, or tables. Like collect and filter actions, all template stereotypes can be applied to activities or call behavior actions on activity diagrams.

Options (tags) on the templates can be predefined on the activity and instanced as actions, or filled in on call behavior actions. You can also override tag values on activities by resetting them on the action. The priority is this: action, activity, context (more on this later). (undefined tags will default to the next available). Like in DocGen 2, you can set the targets directly on the tag, or use the Queries dependency. (Because activity diagrams are now used, most things have to set through tags since they can't be shown on the diagram.)

Like collect and filter tags, there are some tags that are common to all (or most) templates. You will see them shown if it's applicable to the current template. Here's a list of them and what they mean.

Table 3.4. Common Tags on All (Most) Templates

Name	Description
captions	The field takes ordered list of captions. Applicable to images and tables.

Name	Description
headers	Applies to certain tables where headers can be customized. As of now - CombinedMatrix and GenericTable
includeDoc	Where application, includes the documentation of the element. For tables, this will be shown in the column right after the element. For lists, it'll be shown right below.
showType	Where application, shows the type of an element (ex. attributes in properties table)
skipIfNoDoc	Skips a row element if the element has no documentation. Applies to GenericTable and CombinedMatrix
targets	This is what is given to the template for output. If this is empty, the current context of targets is used instead. (for example inside structured queries)
titlePrefix	String prepended to title as is (you will need to put in spaces if you want it)
titles	This field stores the title or titles of the template output. Applicable for most tables and structured queries.
titleSuffix	String appended to title as is (you'll need to put in a space if you want it)
useSectionNameAsTitle	When a title is otherwise empty or undefined, this will use the containing section title instead. The title taken will be the title without any prefix or suffix added. This is generally used in viewpoints to auto generate titles.

In the upcoming sections, each section will introduce a new display template with an explanation of its unique tags. Then a child section called [Template] Example will be shown that's itself made in an activity, and its source activity diagrma will be shown.

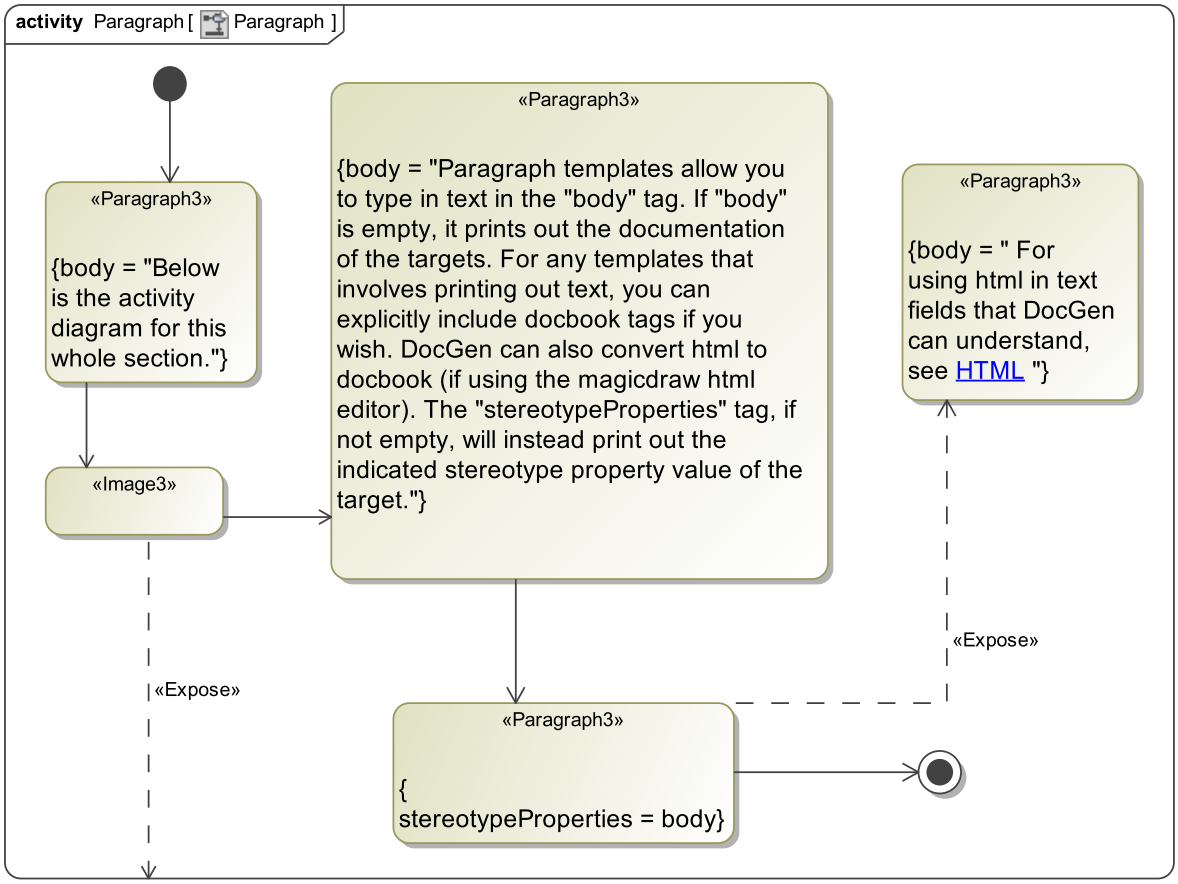
Paragraph

Paragraph templates allow you to type in text in the "body" tag. If "body" is empty, it prints out the documentation of the targets. For any templates that involves printing out text, you can explicitly include docbook tags if you wish. DocGen can also convert html to docbook (if using the magicdraw html editor). The "stereotypeProperties" tag, if not empty, will instead print out the indicated stereotype property value of the target. For an explanation of using html, see HTML

Paragraph Example

Below is the activity diagram for this whole section.

Figure 3.17. Paragraph



Paragraph templates allow you to type in text in the "body" tag. If "body" is empty, it prints out the documentation of the targets. For any templates that involves printing out text, you can explicitly include docbook tags if you wish. DocGen can also convert html to docbook (if using the magicdraw html editor). The "stereotypeProperties" tag, if not empty, will instead print out the indicated stereotype property value of the target.

For using html in text fields that DocGen can understand, see HTML

Image

Images are easy. Just fill in diagrams as the target or point Queries dependencies to diagrams. DocGen will output the diagram as an image in both SVG and PNG format (SVG is used for PDF, and PNG is used for HTML). The title will be the diagram's name and if the diagram has documentation, that'll become a paragraph. Below is an example that prints out the DocGen 3 Profile diagram.

Image Example

Figure 3.18. Image Template Example

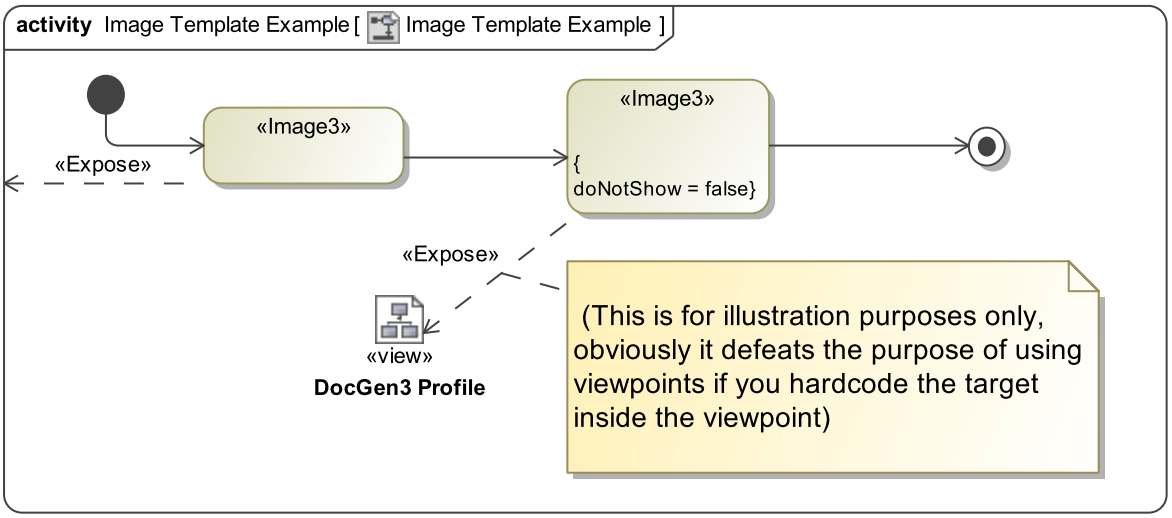
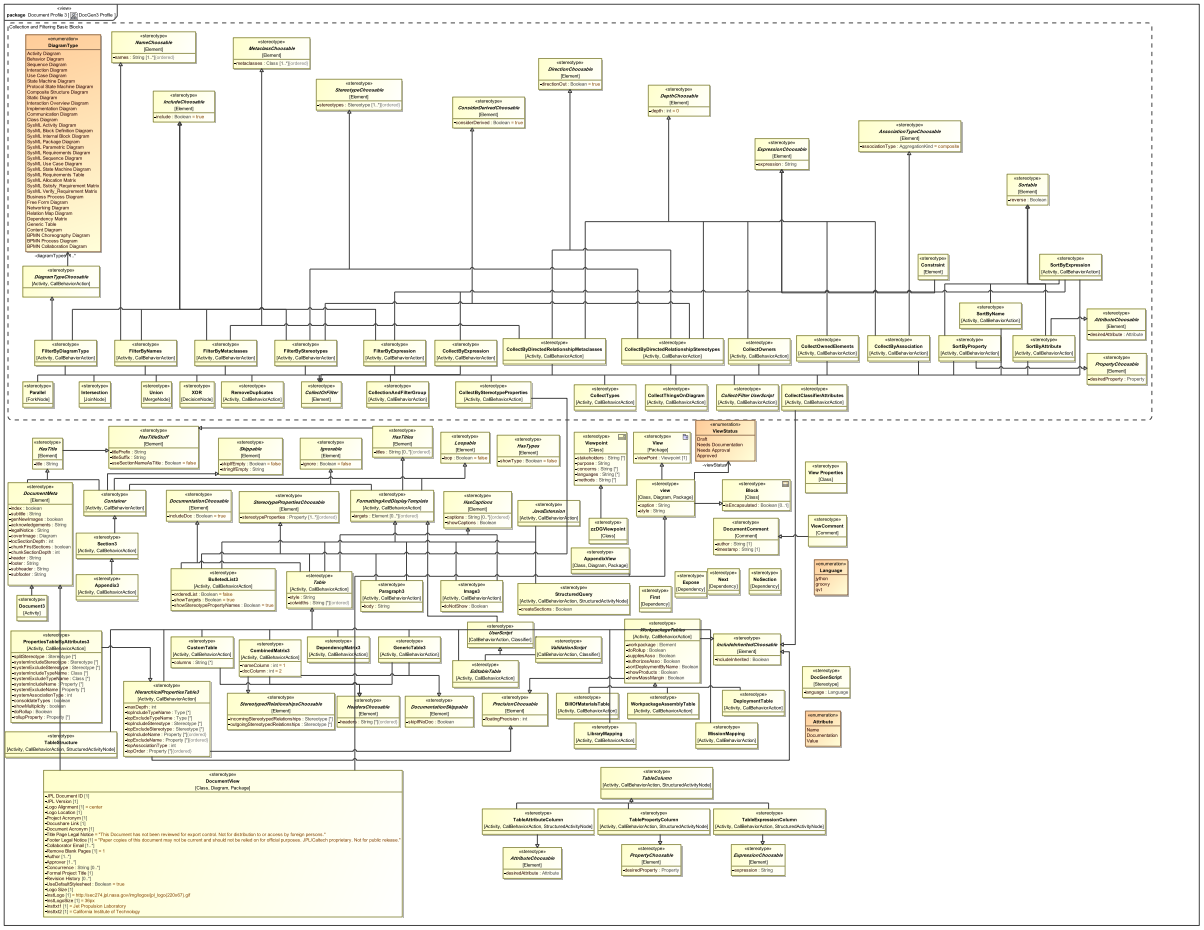


Figure 3.19. DocGen3 Profile



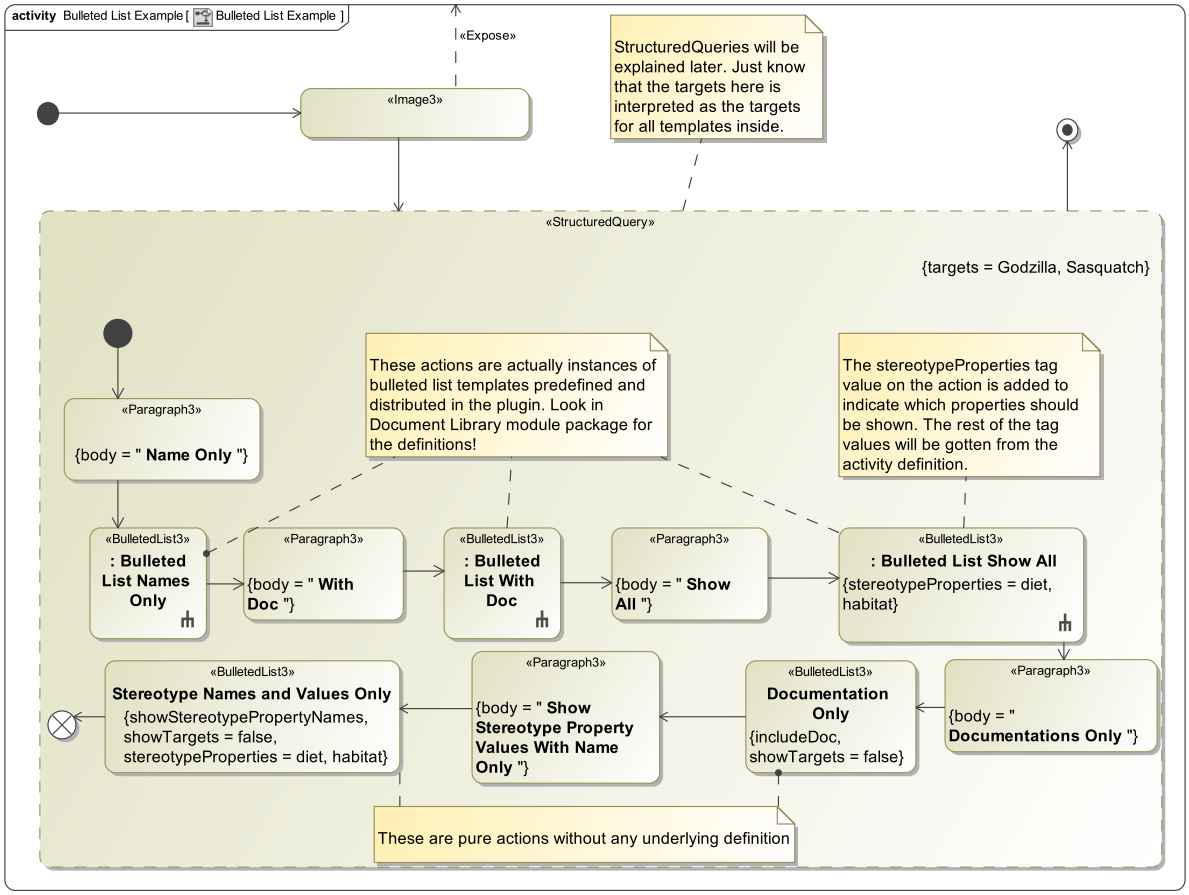
Profile for DocGen 3

Bulleted List

This prints out a list obviously. Given some targets, this template can print out a list of their names, documentation, stereotype property values, or a combination. You can optionally choose to not show the target names or property names.

Bulleted List Example

Figure 3.20. Bulleted List Example



Name Only

- Godzilla
- Sasquatch

With Doc

- Godzilla

  A very large and very angry dinosaur
- Sasquatch

  Gentle wood-ape

Show All

- Godzilla

  A very large and very angry dinosaur

    - diet
        - People
        - Skyscrapers
        - Cars
    - habitat
        - Can be found destroying major urban areas
- Sasquatch

  Gentle wood-ape
    - diet
        - Unknown
    - habitat
        - Behind rocks and out-of-focus areas

Documentations Only

- A very large and very angry dinosaur
- Gentle wood-ape

Show Stereotype Property Values With Name Only

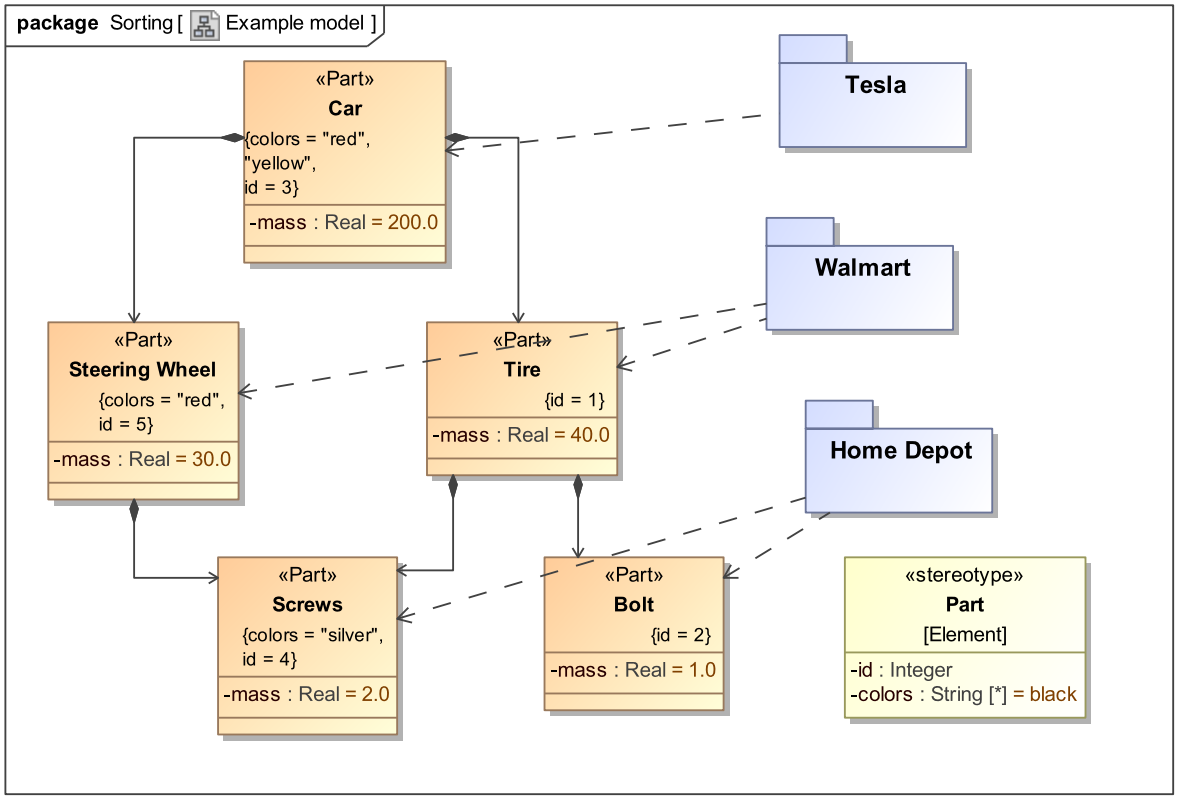
- diet
    - People
    - Skyscrapers
    - Cars
- habitat
    - Can be found destroying major urban areas
- diet
    - Unknown
- habitat
    - Behind rocks and out-of-focus areas

Table Structure

Table Structure is a generic table making template that supercedes the Generic Table and Combined Matrix. It allows users to add in behaviors (the current collect/filter/sort options) for each column of the table and indicate which attribute or property to display. The display options are the same as SortByAttribute or SortByProperty ones - name, documentation, value of properties or stereotype properties.

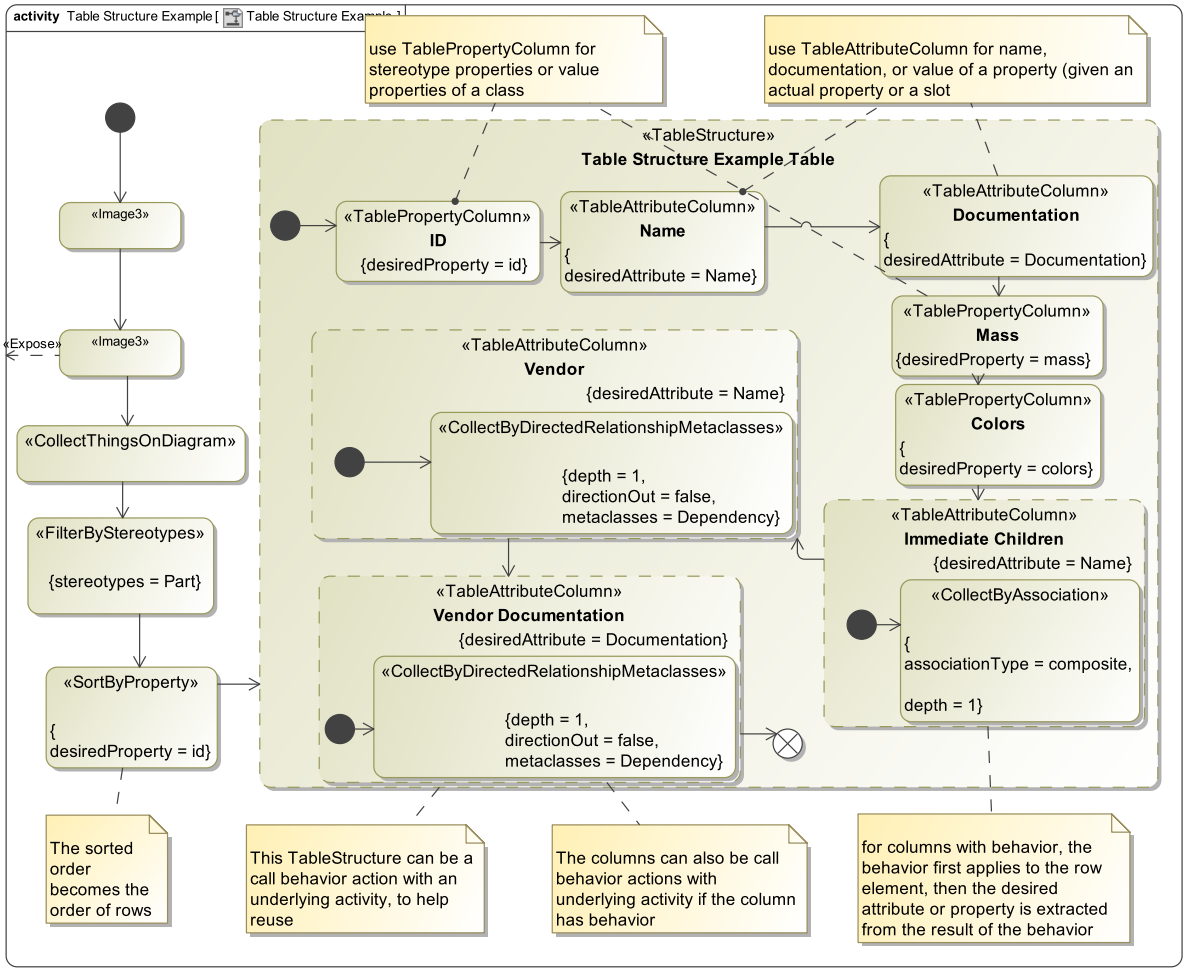
Table Structure Example

Figure 3.21. Example model



Example Model.

Figure 3.22. Table Structure Example



The Table Structure viewpoint behavior.

Table 3.5. Table Structure Example Table

ID	Name	Documentation	Mass	Colors	Immediate Children	Vendor	Vendor Documentation
1	Tire	Spinning thing.	40.0	black	Bolt Screws	Walmart	Where you see poorly dressed people.
2	Bolt	Animated Dog.	1.0	black		Home Depot	Where people buy stuff.
3	Car	Will be self driving in 2020.	200.0	red yellow	Steering Wheel Tire	Tesla	Elon Musk's electric car company.
4	Screws	Inclined plan wrapped around pointed cylinder.	2.0	silver		Home Depot	Where people buy stuff.

ID	Name	Documentation	Mass	Colors	Immediate Children	Vendor	Vendor Documentation
5	Steering Wheel	Spinning thing for people.	30.0	red	Screws	Walmart	Where you see poorly dressed people.

Generic Table [Deprecated]

The generic table template prints out the magicdraw Generic Table as an actual table and not some messed up image (although Magicdraw may have fixed the mess-up in 17.0.1). Still, the table output lets your highlight and copy text in the report. You can also define your own custom headers. The first column (with the numbers) is not shown.

Generic Table Example

Figure 3.23. GenericTable Example

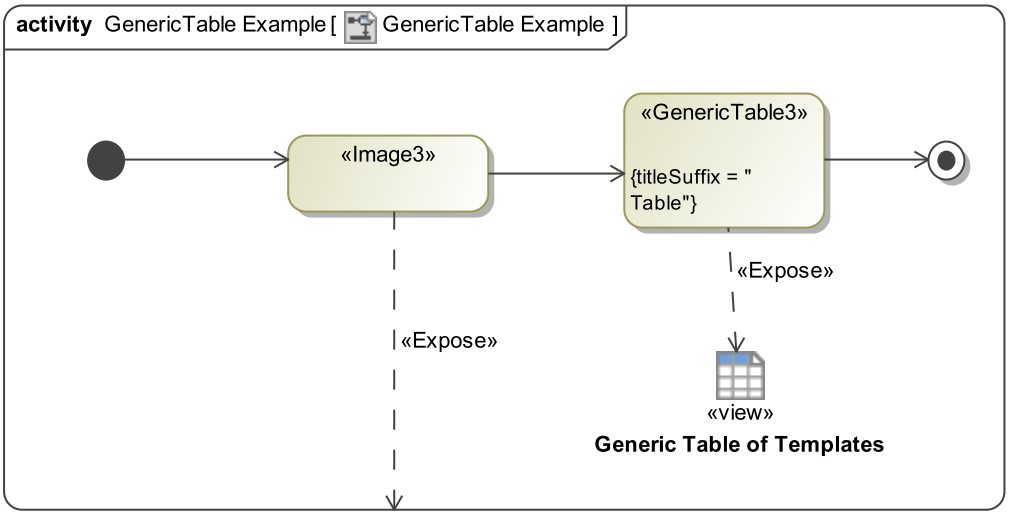


Table 3.6. Generic Table of Templates Table

Name	Documentation	Owned Attribute
BulletedList3	The bulleted list template is designed to print out a bulleted or numbered list of things, including element names, documentation, or stereotype properties.	base_Activity base_CallBehaviorAction orderedList showTargets showStereotypePropertyNames
CombinedMatrix3	This template outputs a table that consist of target elements' name, possible documentation, specific stereotype properties (including derived properties), or their stereotyped relationships with other elements.	base_Activity base_CallBehaviorAction nameColumn docColumn
DependencyMatrix3	not implemented	base_Activity

Name	Documentation	Owned Attribute
		base_CallBehaviorAction
GenericTable3	Given a Generic Table, prints out the table as an actual table and not some kind of messed up diagram. Table elements that are NamedElements will have their name outputted, else just a string representation.	base_Activity base_CallBehaviorAction
Image3	This produces one or more diagram images and associated titles and captions. If the targets given are images, it will print them out with the name of the diagram as the title and documentation of the diagram as captions.	base_Activity base_CallBehaviorAction doNotShow
Paragraph3	Denotes a literal paragraph. Put the text in the body attribute. If body property is filled it will use that. Else it will print out the documentation of the targets as paragraphs.	base_Activity base_CallBehaviorAction body
PropertiesTableByAttributes3	Show a table, with class compositional hierarchies down the rows, and hierarchical properties on top.	splitStereotype systemIncludeStereotype systemExcludeStereotype systemIncludeTypeName systemExcludeTypeName systemIncludeName systemExcludeName systemAssociationType consolidateTypes showMultiplicity base_Activity base_CallBehaviorAction doRollup rollupProperty
UserScript	This is the way for users to extend available templates by creating their own using jython scripts. New templates must be new stereotypes that extend this UserScript stereotype, and the new stereotype's name must reflect the folder and script file structure under [md install]/DocGenUserScripts.	base_CallBehaviorAction base_Classifier

Name	Documentation	Owned Attribute
	See the DocGen 3 manual for more information.	
HierarchicalPropertiesTable3	A set of hierarchical tables where the rows going down depict some hierarchical organization (for example, block compositions, activity call levels), and the columns are properties such as value properties on blocks, stereotype properties, or documentation. The properties can also be hierarchical in the case of blocks. • If no filtered are applied, all possible properties will be discovered and displayed (does not include stereotypeProperties). This include/exclude semantics also applies to any options specific to each table type. • If only include filters are specified, only properties that match those filters will be displayed (and all their children properties if any). • If only exclude filters are specified, all properties will be displayed except for those that match the exclude filters and their hierarchical children. • If both include and exclude filters are specified, properties that match includes will be displayed (and their children) until the excludes match. Because the targets for this family of tables are top level elements and most table entries are discovered, the flag loop controls whether the top level element and generated table are shown in one table or separated into different tables.	maxDepth topIncludeTypeName topExcludeTypeName topIncludeStereotype topExcludeStereotype topIncludeName topExcludeName topAssociationType topOrder base_Activity base_CallBehaviorAction

Combined Matrix [Deprecated]

The combined matrix takes the targets and outputs a table with their name, documentation, stereotype property values, related elements (where relationships are stereotyped), or a combination of the above. The most common use case is a documentation table, which is also distributed as a predefined template in the Document Library module package. Below are some examples and the diagram that generated this section.

Table 3.7. Tag Options Unique to Combined Matrix

Name	Description
docColumn	Set the column where documentation should appear (includeDoc is true). Default is 2.
incomingStereotypedRelationships	Choose stereotyped relationships coming in to the target(s). The output would be the opposite end of the relationship.
nameColumn	Set the column where name of element should appear. Default is 1.
outgoingStereotypedRelationships	Choose stereotyped relationships going out from the target(s). The output would be the opposite end of the relationship.
stereotypeProperties	Given a list of stereotype properties, will return their value where applicable. This applies to certain tables, bulleted list, and collection action. For templates, they'll print out the property value(s). For collection, returns the values that are model elements. The stereotype properties can also be derived properties in customizations.

Combined Matrix Example

Figure 3.24. Combined Matrix

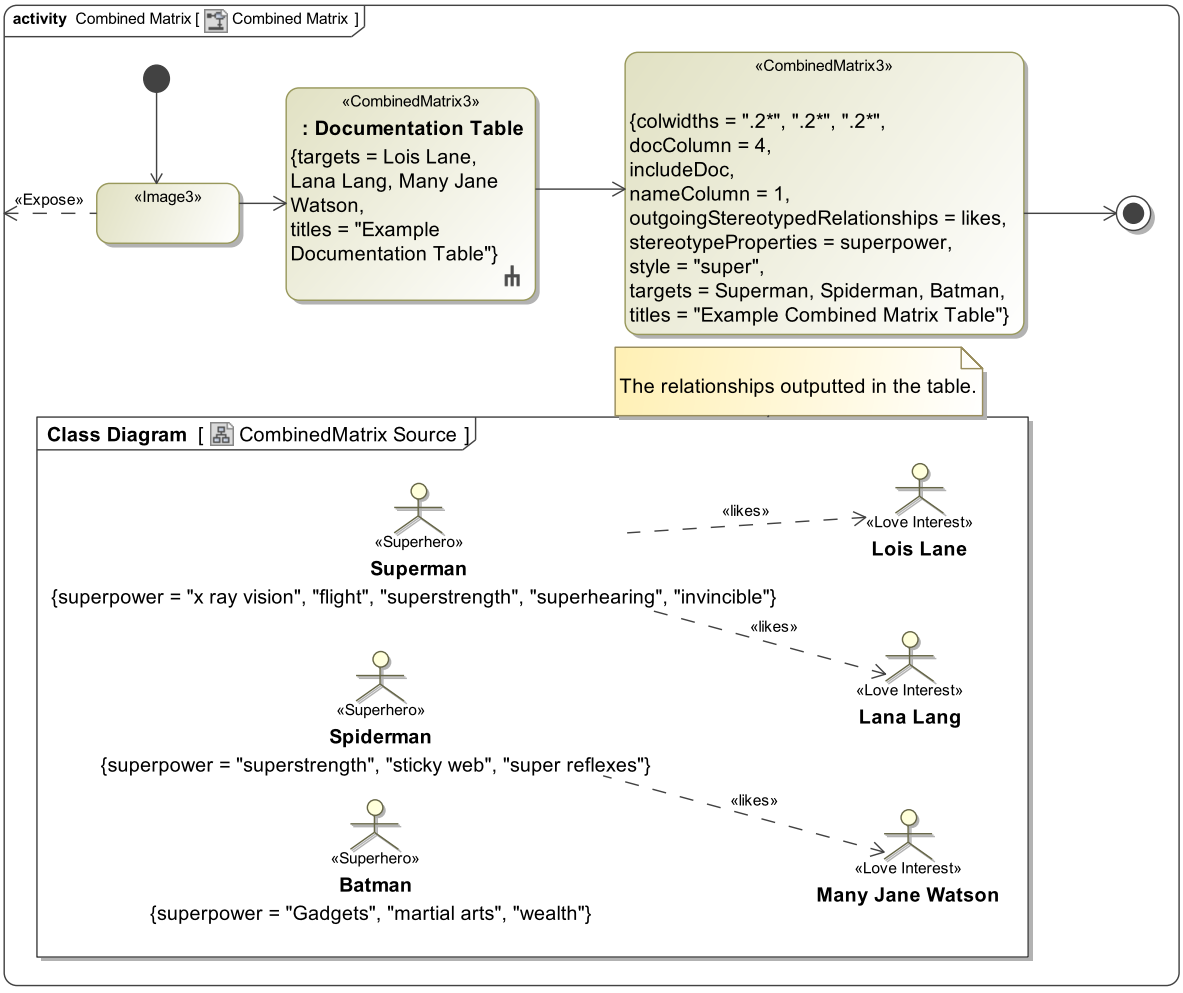


Table 3.8. Example Documentation Table

Name	Description
Lana Lang	Teenage superman's love interest in Smallville.
Lois Lane	I have no idea what happened to Lois Lane since I got tired of Smallville and don't read the comics.
Many Jane Watson	Something to do with Spidey. Her name seems to be spelled wrong.

Table 3.9. Example Combined Matrix Table

Name	superpower	likes	Description
Batman	Gadgets martial arts wealth		Fights crime in Gotham City
Spiderman	superstrength sticky web super reflexes	Many Jane Watson	Teenage science whiz bitten by a radioactive spider
Superman	x ray vision flight superstrength superhearing invincible	Lois Lane Lana Lang	Alien that takes energy from the sun

Properties Table By Attributes

This is part of a category called hierarchical properties tables:

A set of hierarchical tables where the rows going down depict some hierarchical organization (for example, block compositions, activity call levels), and the columns are properties such as value properties on blocks, stereotype properties, or documentation. The properties can also be hierarchical in the case of blocks.

- If no filtered are applied, all possible properties will be discovered and displayed (does not include stereotypeProperties). This include/exclude semantics also applies to any options specific to each table type.
- If only include filters are specified, only properties that match those filters will be displayed (and all their children properties if any).
- If only exclude filters are specified, all properties will be displayed except for those that match the exclude filters and their hierarchical children.
- If both include and exclude filters are specified, properties that match includes will be displayed (and their children) until the excludes match.

Because the targets for this family of tables are top level elements and most table entries are discovered, the flag loop controls whether the top level element and generated table are shown in one table or separated into different tables.

The amount of options available for this table is huge, so if you don't understand how to use this table, let me know so I can add more examples, or just experiment a bit.

Table 3.10. Unique tags for Properties Table

Name	Description
consolidateTypes	Instead of showing properties down the left, properties with the same type are consolidated and the type block is shown instead.
floatingPrecision	Optional. If a property is a number, this specifies to how many decimal places it will be rendered.
maxDepth	Optional. Maximum depth of row hierarchy. 0 means infinite and is the default.
showMultiplicity	Shows a column right after documentation that shows the multiplicity of the property, or if consolidateTypes is checked, the overall number of units of that block (under its parent block). Note this number is the count of what would have been shown if it wasn't checked, the count occurs after all the filtering are done.
splitStereotype	Optional. This specifies the stereotypes on blocks that indicates they should be treated as properties in the hierarchy. From the system hierarchy, everything underneath a block with this stereotype will be moved up top (with its own property hierarchy). If empty, this means all properties across the columns will be leaf properties.
systemAssociationType	Optional. Specifies what kind of association to follow for hierarchical systems. 1 for composite, 2 for shared, 0 for both (default is 0).
systemExcludeName	Optional. Exclude system whose role has these names
systemExcludeStereotype	Optional. Exclude system properties with these stereotypes or whose system type block has these stereotypes
systemExcludeTypeName	Optional. Exclude systems whose type block has these names.
systemIncludeName	Optional. Include system whose role has these names.
systemIncludeStereotype	Optional. Include system properties with these stereotypes or whose type has these stereotypes.
systemIncludeTypeName	Optional. Include systems whose type block has these names.
topAssociationType	Optional. Specifies what kind of association to follow for hierarchical properties. 1 for composite, 2 for shared, 0 for both (default is 0).
topExcludeName	Optional. Exclude properties with these names.
topExcludeStereotype	Optional. Exclude properties with these stereotypes or whose type has these stereotypes.
topExcludeTypeName	Optional. Exclude properties whose type are these type names.
topIncludeName	Optional. Include properties with these names.
topIncludeStereotype	Optional. Include properties with these stereotypes or whose type has these stereotypes.
topIncludeTypeName	Optional. Include properties whose type are these type names.
topOrder	Optional. Give a list of property names your want to appear in order. The following rule would be applied: for each property hierarchy level: leaf properties will always appear first, user order will then be applied, the rest is alphabetical. If this is empty and topIncludeName is not empty, the order from topIncludeName will be used for ordering instead.

Properties Table By Attributes Example

Figure 3.25. Example Properties Table

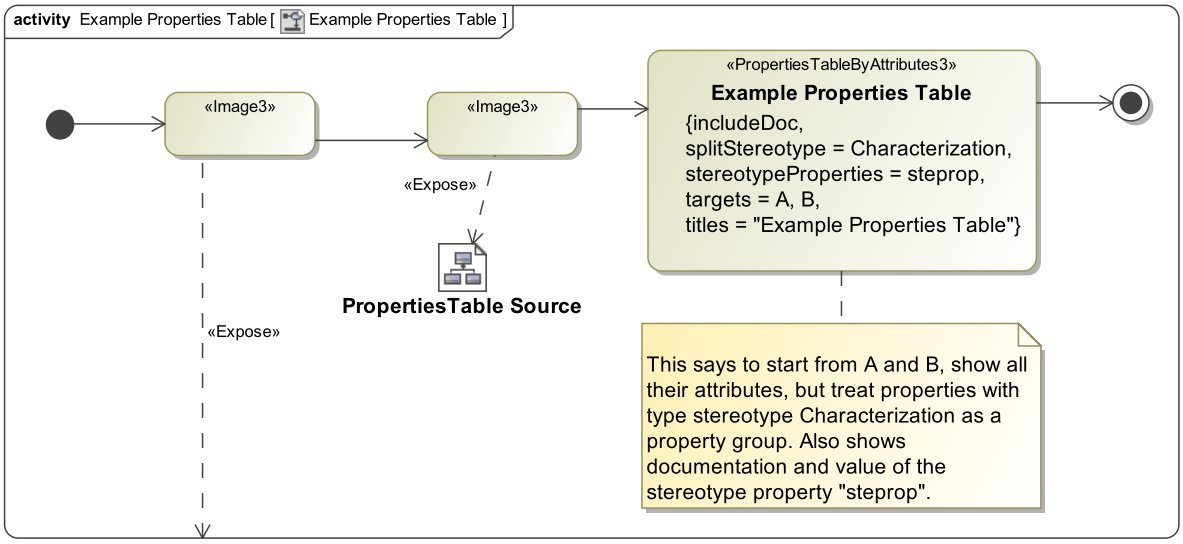


Figure 3.26. PropertiesTable Source

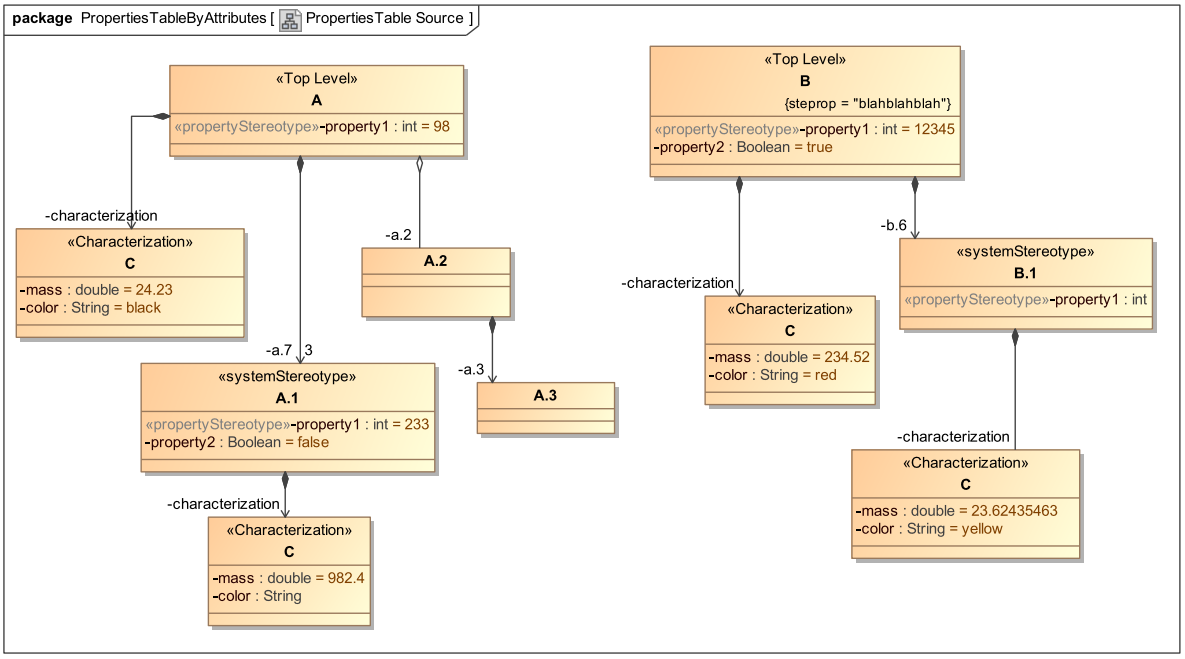


Table 3.11. Example Properties Table

	Description	property1	property2	characterization				steprop
				color	mass	blah		
						bored	sleep	
A	Top level class	98	n/a	black	24.23	n/a	n/a	
a.2	This documentation	n/a	200	n/a	n/a	n/a	n/a	

	Description	property1	property2	characterization				steprop
				color	mass	blah		
						bored	sleep	
	is from the type block of the property shown.							
a.3		n/a	n/a	n/a	n/a	n/a	n/a	
a.7		233	false		982.4	n/a	n/a	
B	Another top level class	12345	true	red	234.52	true	false	blahblahblah
b.6			n/a	yellow	23.62435463	n/a	n/a	

Making User Scripts (Very Advanced)

If none of the current premade templates suit you, you can write your own template using Jython, Groovy, or QVT scripts. When DocGen gets to your script, it'll pass it the targets and you can use the Magicdraw API or metamodel to do whatever you like with it and return the Docbook elements you want in the document. The most relevant elements (like paragraph, list, table) are instanced as Java objects. If you want to use other Docbook elements, you can also return your block of text as is and DocGen will inject it into the document verbatim.

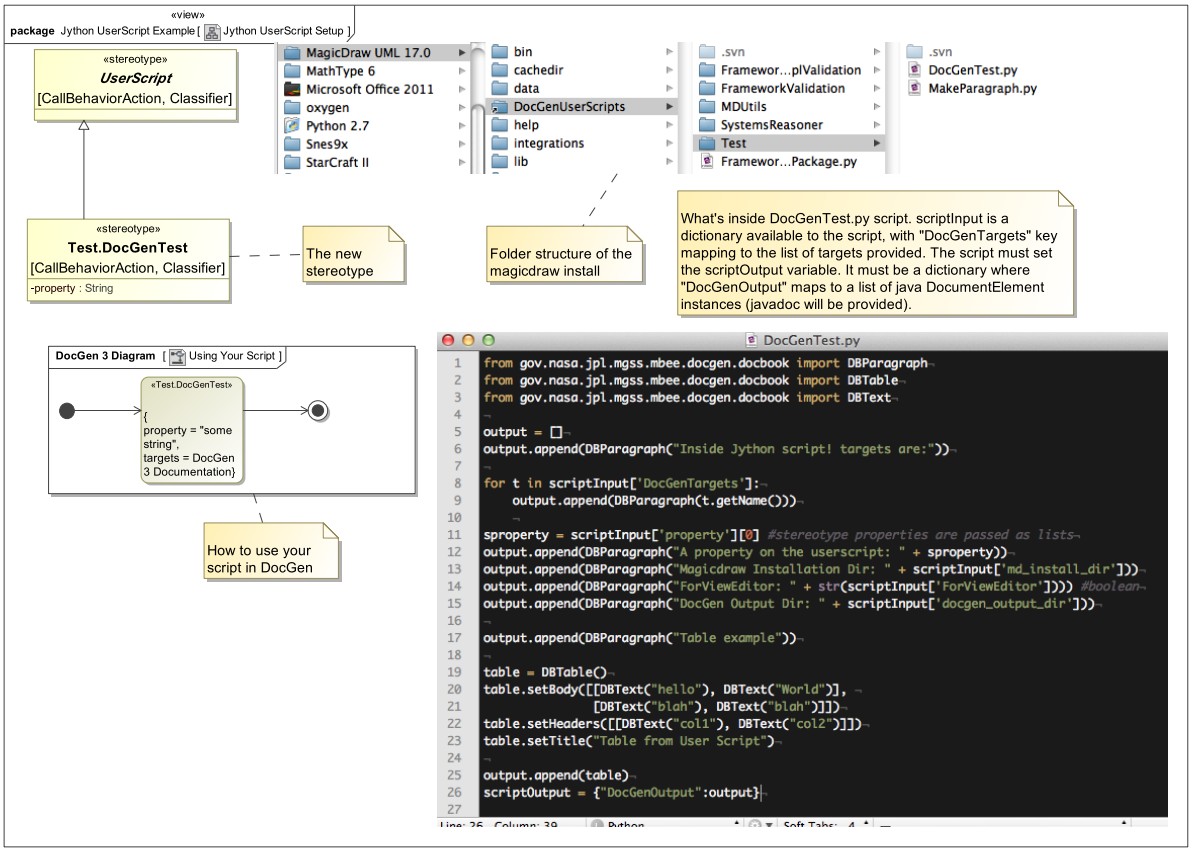
To make your own script, you must make a new stereotype that specializes the UserScript stereotype from the document profile. Look in your Magicdraw install directory and make a folder called DocGenUserScripts if it's not already there. The naming of your new stereotype need to match the folder structure of where you will put your script. For example, if your script will be "[md install dir]/DocGenUserScripts/Test/blah.py", the name of your stereotype will be "Test.blah".

If you want to use your script on DocWeb, you must contact the server admin to upload the script first. The javadoc is here: http://docgen.jpl.nasa.gov/opsrev/downloads/docgen/javadoc/

Look for classes under gov.nasa.jpl.mgss.mbee.docgen.docbook (the package structure may change later)

Jython UserScript Example

Figure 3.27. Jython UserScript Setup



Inside Jython script! targets are:

DocGen 3 Documentation

A property on the userscript: some string

Magicdraw Installation Dir: /Users/dlam/Downloads/MagicDraw UML1702/

ForViewEditor: False

DocGen Output Dir: /Users/dlam/Documents/docgen

Table example

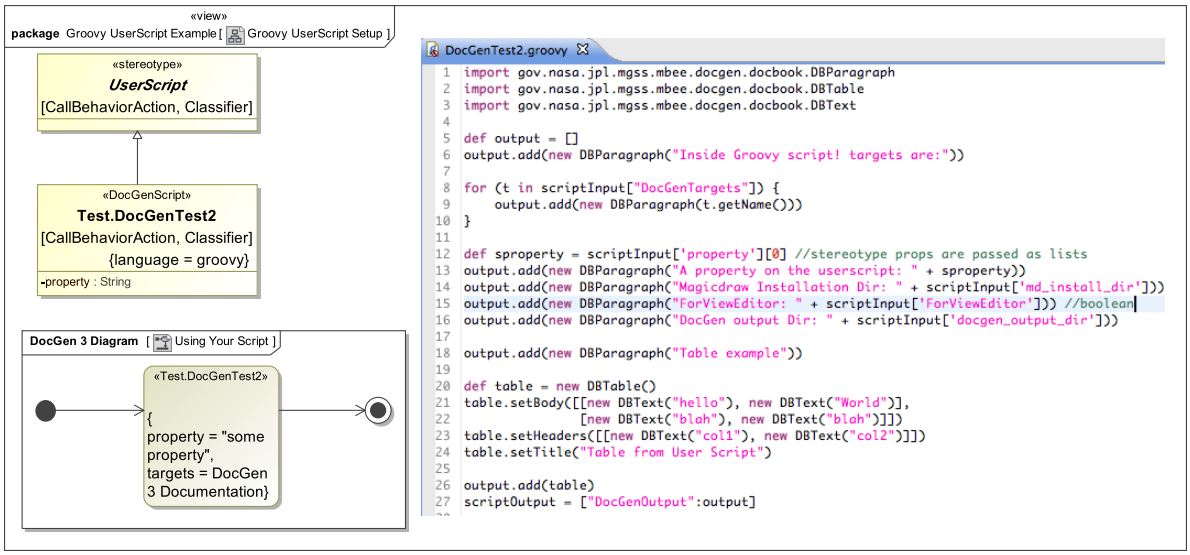
Table 3.12. Table from User Script

col1	col2
hello	World
blah	blah

Groovy UserScript Example

For non-jython scripts, you'll need to stereotype your user script stereotype itself with <<DocGenScript>> and set the language tag.

Figure 3.28. Groovy UserScript Setup



Inside Groovy script! targets are:

DocGen 3 Documentation

A property on the userscript: some property

Magicdraw Installation Dir: /Users/dlam/Downloads/MagicDraw UML1702/

ForViewEditor: false

DocGen output Dir: /Users/dlam/Documents/docgen

Table example

Table 3.13. Table from User Script

col1	col2
hello	World
blah	blah

QVT UserScript Example

For QVT, the setup is the same except you'll need both the .qvto and compiled .qvtox file in the right directory. Stereotype tag values and other predefined values are passed as configuration properties. The input elements is of type uml metamodel, "selectedElements", and the model itself is passed as "currentModel". Define an output that's of dgview metamodel called "docgenOutput".

You'll need to get the dgview ecore file at http://docgen.jpl.nasa.gov/opsrev/downloads/docgen/dgview.ecore and register it in Eclipse to compile your QVT scripts. Save the ecore file somewhere in the workspace, then right click on the project where you write your QVT userscript and set the metamodel mappings as shown.

Figure 3.29. QVT UserScript Setup

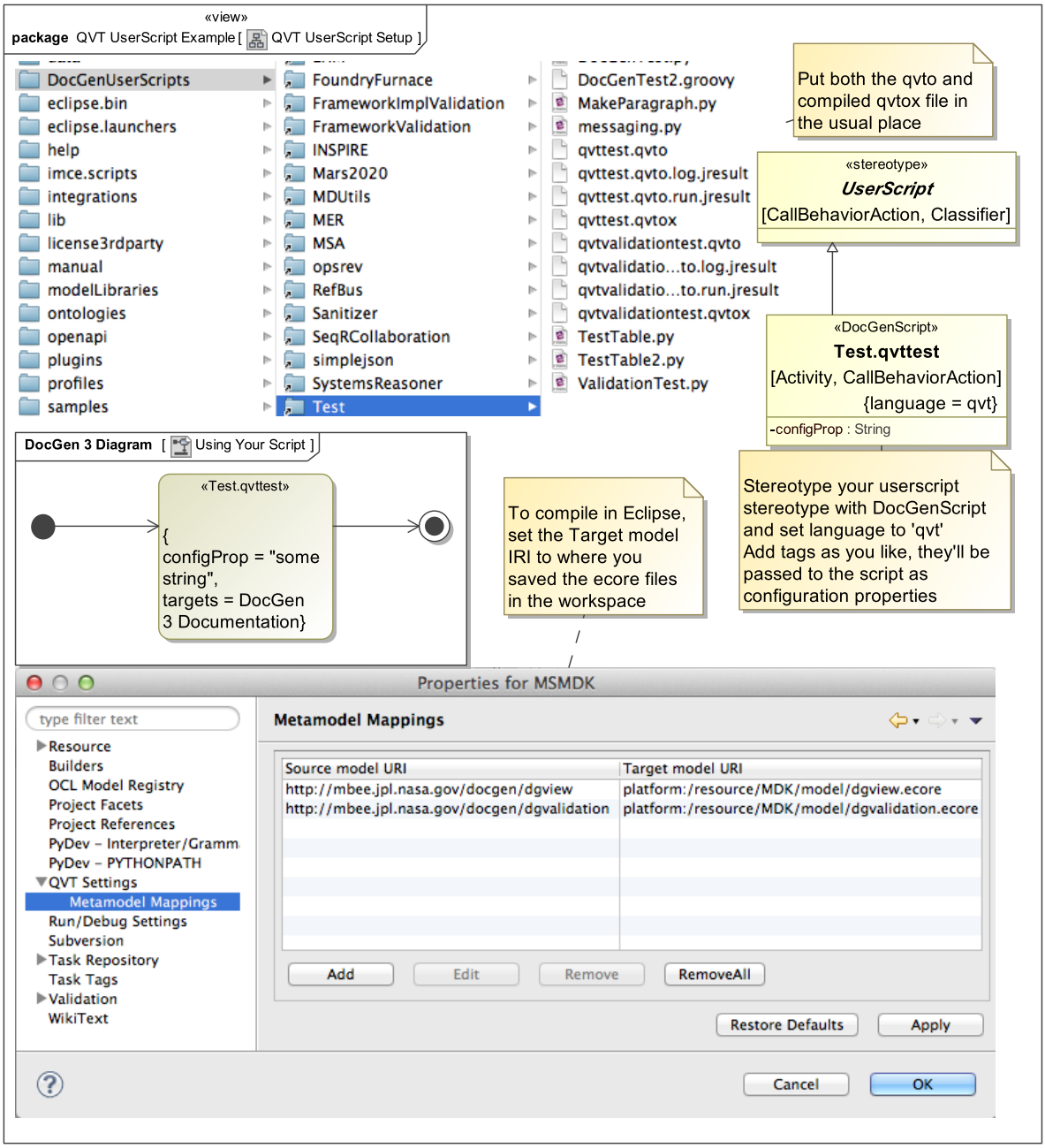
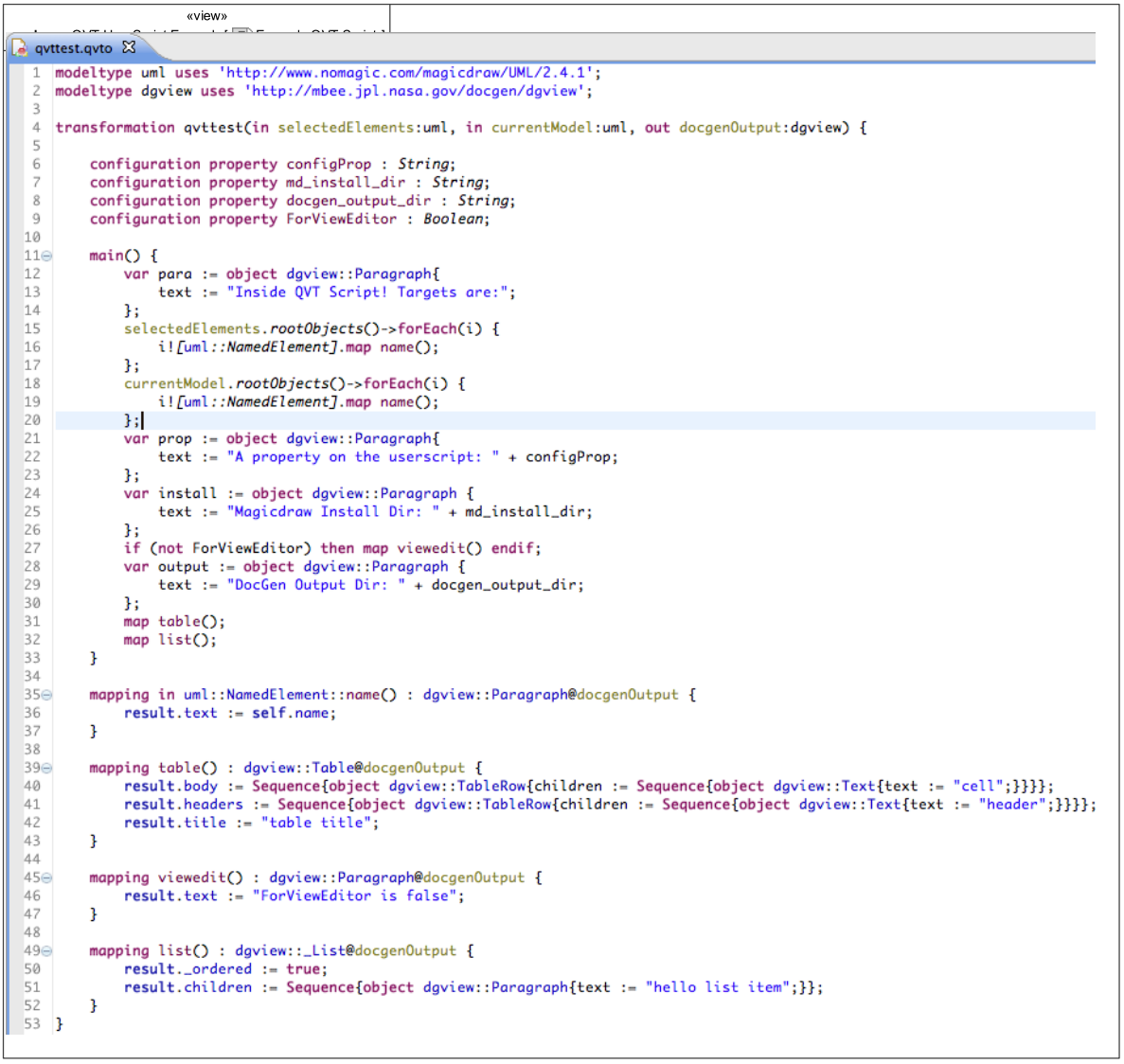


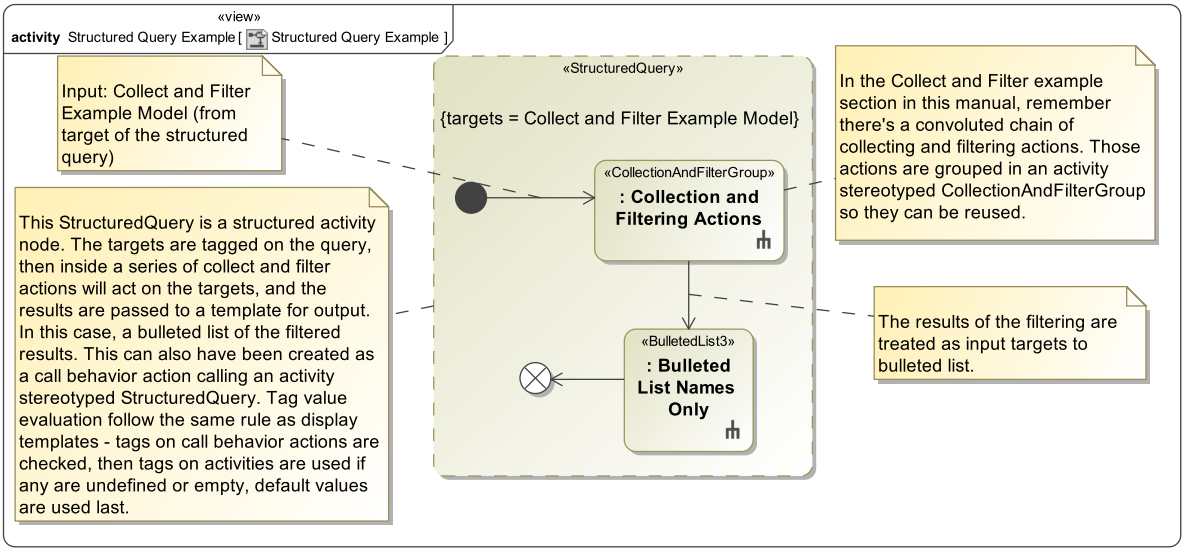
Figure 3.30. Example QVT Script



Structured Queries and Viewpoint Behaviors

StructuredQueries serve as a partition of the document model. StructuredQueries can be defined as an activity and instanced using an action, or can be a structured activity node in an existing activity. You've seen an example of structured query already, in the collect and filter example.

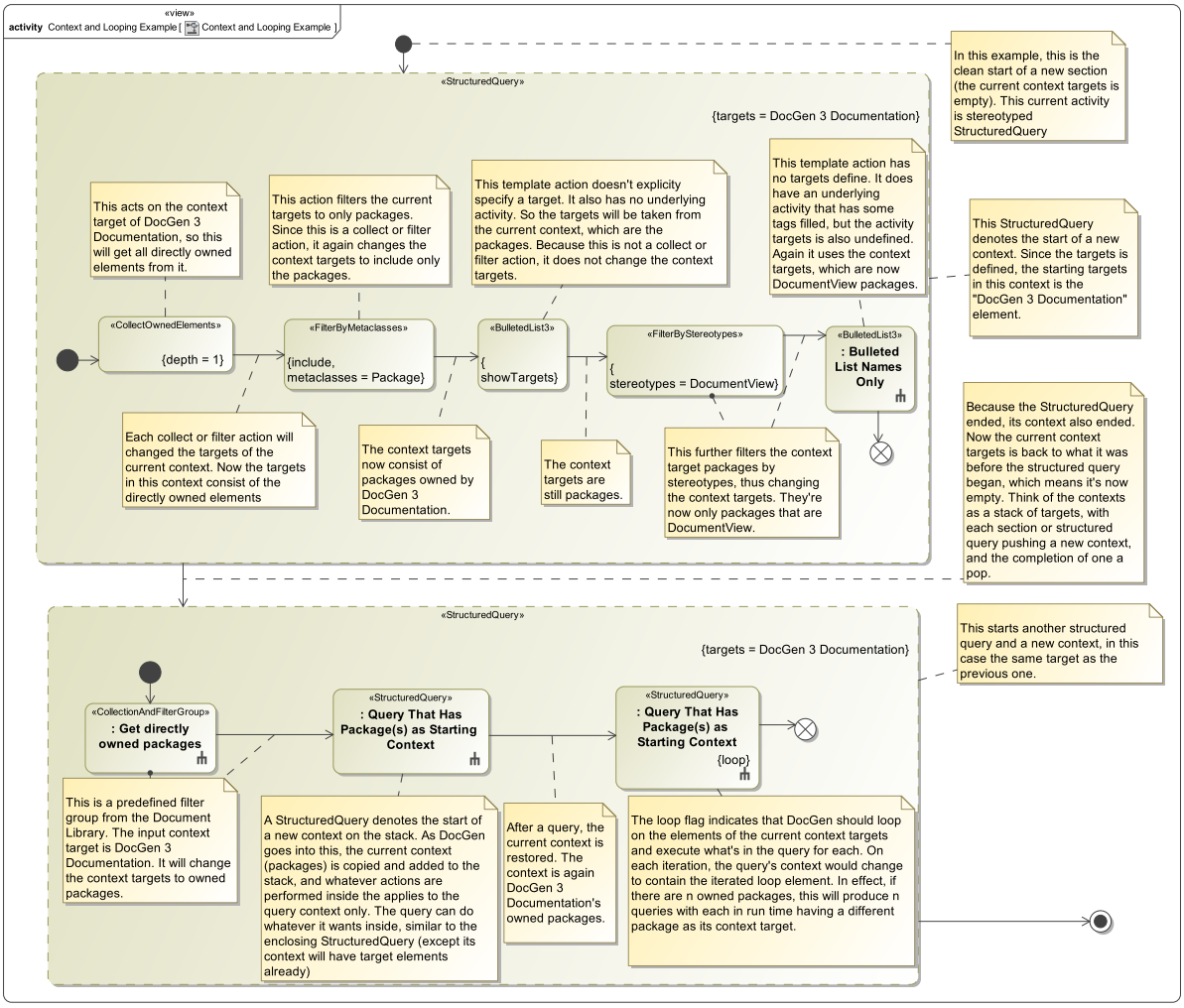
Figure 3.31. Structured Query Example



Context Targets Stacks and Looping

To really understand what targets you're giving to your filters and templates, you need to understand how DocGen parses the activity diagrams and what's happening to the "targets" in a context. What I call a context is basically what the targets are in a particular activity or structured query during run time. This is important because instead of having to explicitly define your targets for each action, the actions can act on the targets of the context and change them for the next action. This is what makes viewpoints possible. This concept is illustrated in the diagrams below.

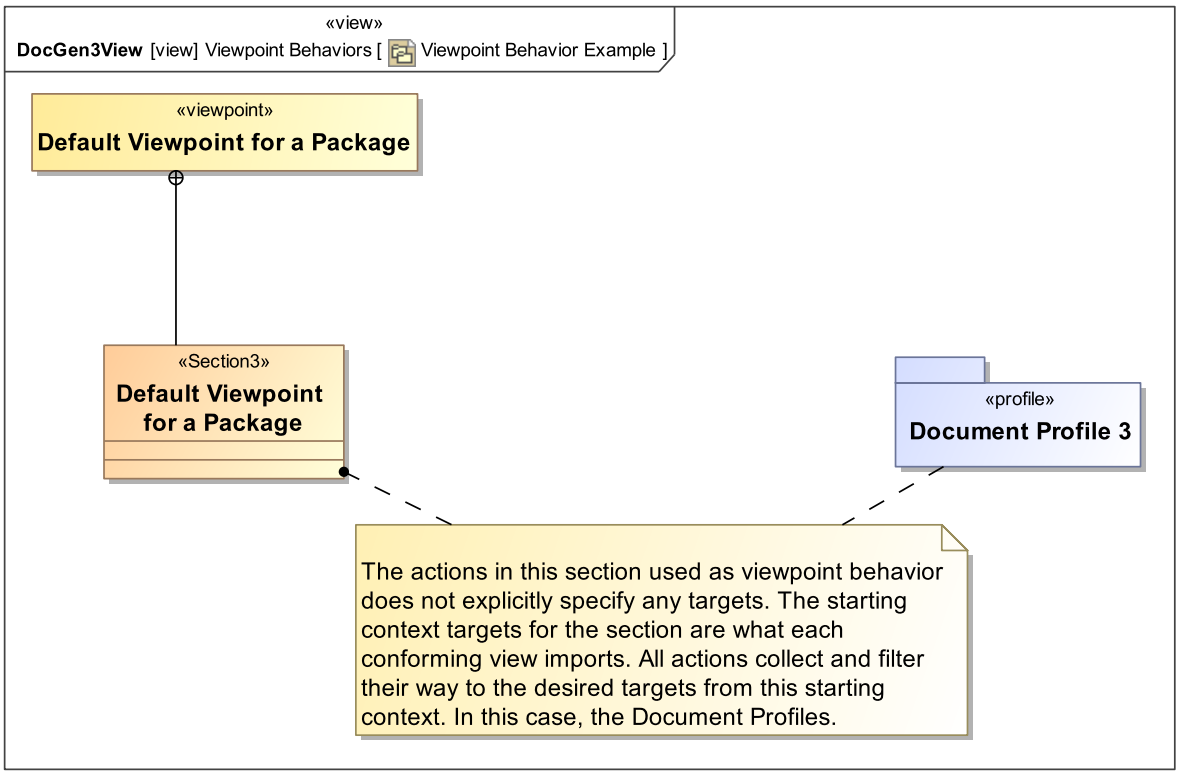
Figure 3.32. Context and Looping Example



Viewpoint Behaviors

Now that you know about contexts, it's easy to explain how to make viewpoint behaviors. A viewpoint behavior is simply an activity that's like a structured query. It'll be in a section because that's what each view implies, and each view conforms to a viewpoint. The starting context of the viewpoint behavior would be whatever the conforming view imports (using Element Import or Package Import).

Figure 3.33. Viewpoint Behavior Example

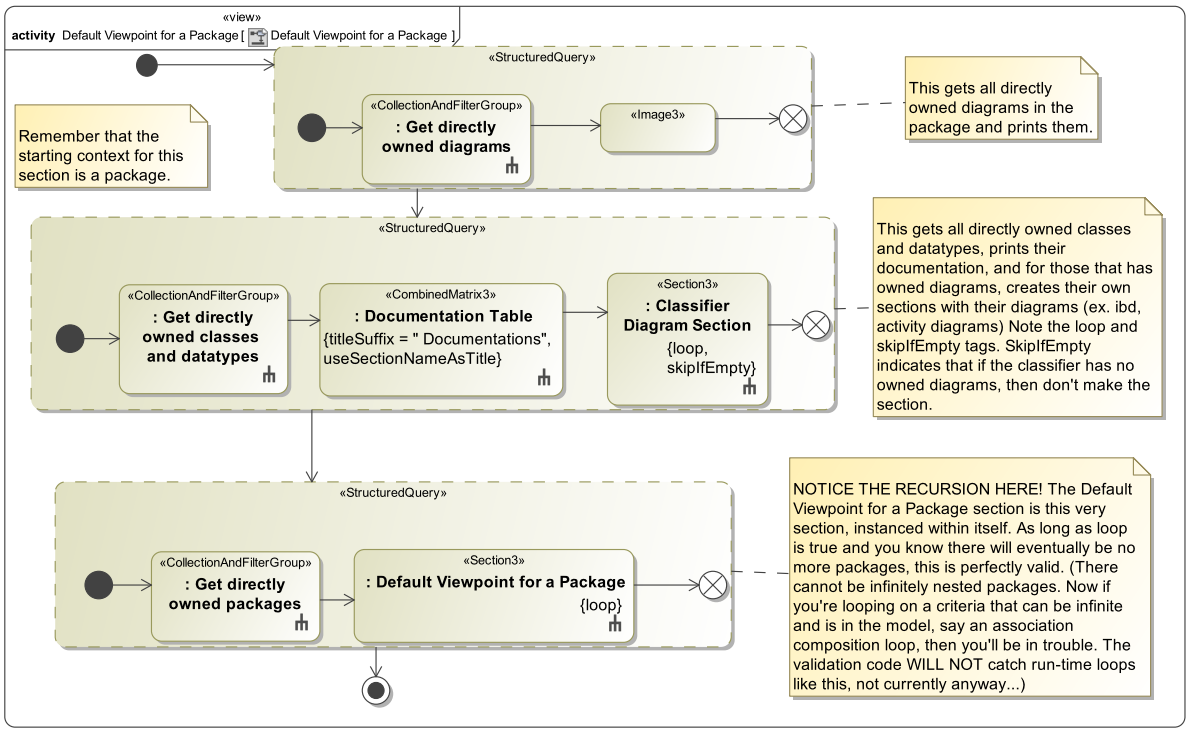


Using Recursion

A more advanced and useful way of taking advantage of contexts is using recursion to auto create sections or batches of queries that follows some kind of model pattern. For example, creating a section structure that mimics the model package structure, or generalization tree structure. If there is a recurring pattern in your model that can be traversed using the collect and filter actions, and each successive filter will eventually result in empty targets, then you can use recursion. (DON'T try this on an infinite loop in the model.) Here's an example on how it's done. Below's the activity for the Default Package Viewpoint in the Document Library.

NOTE Using this method to create Sections (Views) in the document is bad practice, though sometimes useful. Sections created this way will not show up in View Editor!

Figure 3.34. Default Viewpoint for a Package



Validate, Preview, and Generate Document

The validation, preview, and generation features are all available via the context menu under DocGen submenu when right-clicking on a document. (This means something stereotyped by Document3 or DocumentView stereotypes)

- Validate DocGen 3 Document will catch mismatched stereotypes, duplicate first/next stereotypes, explicit loops, and various other errors. The result will be outputted in the log (if you don't see the on screen log, go to Window and show Messages Window)
- Preview DocGen 3 Document will show a 2 column table depicting the outline of your document. It'll show sections and the templates inside, and any targets and relevant paragraphs. If you use recursion or loops in your document, they will be expanded as they're resolved. (The preview does not take into account "skipIfEmpty" flags on sections)
- Generate DocGen 3 Document will parse the document model and output to a docbook xml file. It'll prompt you to choose a file to output to. Wait until it finishes and you can open the file with any xml viewer/transformer.

On Document3 and DocumentView stereotypes, there are tags related to the whole document you can fill out. The table below shows what they are.

Table 3.14. Document Options

Name	Description
acknowledgements	
Approver	The Approver(s) of the document. You may have as many as you want. The format is: Firstname,Lastname,jobtitle,organization,section. Example: Peter J.,Di Pasquale,Ground Data Systems Engineer,JPL,393b
Author	The Author(s) of the document. You may have as many as you want. The format is: Firstname,Lastname,jobtitle,organization,section. Example: Peter J.,Di Pasquale,Ground Data Systems Engineer,JPL,393b

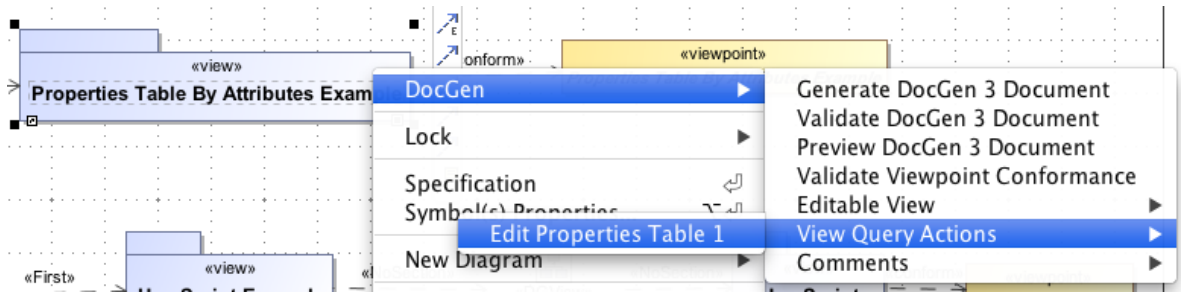
Name	Description
chunkFirstSections	Optional stylesheet parameter. Whether to chunk first sections. If not, this means in html, each section or chapter page will include the first section. Default is true. (Only matters to docweb)
chunkSectionDepth	Optional stylesheet parameter. This is for html chunking - what's the max depth of a section that'll be put on a different page. Default is 20. (Only matters to docweb)
Collaborator Email	You may add as many collaborator emails as you like by pressing the + button and adding one email per plus.
Concurrence	The concurrence section of the document. You may have as many as you want. The format is: Firstname,Lastname,jobtitle,organization,section. Example: Peter J.,Di Pasquale,Ground Data Systems Engineer,JPL,393b
coverImage	Optional. The Diagram to display on the cover page, right after title and authors (if there's any)
Document Acronym	The abbreviated title of the document.
Docushare Link	
footer	Optional footer string on each page of the document. (for docweb) Do not put docbook tags in here! This should be a simple string with no markup.
Footer Legal Notice	Legal notice for the document footer. Default value is to display that paper documents may not be current.
Formal Project Title	The title of the project.
genNewImages	This property allows you to specify whether you will generate new versions of images associated with the document. If you are generating frequently and not changing the images, this is useful because generating images slows the generation process significantly when there are a lot of images. If an image does not exist yet, it will be generated regardless of whether this property is checked.
header	Optional header string on each page of the document. (for docweb) Do not put docbook tags in here! This should be a simple string with no markup.
index	Check this box to generate an index. To register things that should go in the index you must place a tag at the point in the text where the reader should be referred if they were to look it up. Use the following syntax: `<indexterm><primary>some term</primary></indexterm>`
InstLogo	
InstLogoSize	
Insttxt1	
Insttxt2	
JPL Document ID	The JPL ID Number of the Document.
JPL Version	The JPL Version of the Document. Enter as a string.
legalNotice	Optional. A string to display after a coverImage if there's any.
Logo Alignment	Supported Alignments are: "left", "right", "center".
Logo Location	This is the file location of your logo. It can be a local file, or a file accessible via the Internet.
Logo Size	
Project Acronym	

Name	Description
Remove Blank Pages	Set to one for blank pages, set to zero for none. Note: Currently not functional.
Revision History	The history of the revision. You may have as many as you want. Must follow this format: Version|DateEdited|EditorFirstName|EditorLastName|Changes. Example: Draft|4/12/13|John Q|Smith|Document Created
subfooter	Optional. Goes above footer. Do not put docbook tags in here! This should be a simple string with no markup.
subheader	Optional. Goes underneath header. Do not put docbook tags in here! This should be a simple string with no markup.
subtitle	This is the subtitle of the document. It usually appears under the title in italics, or after a colon. The formatting is up to you.
Title Page Legal Notice	JPL Standard Legal Notice. Default value is to display not cleared for export control.
tocSectionDepth	Optional stylesheet parameter. How many nested sections to display in table of contents for html. Default is 20. (Only matters to docweb)
UseDefaultStylesheet	

View Context Actions

If you have any kind of userscript/actions in a viewpoint that can result in running a script/validation or editing a table, you can access the action via the context menu from the conforming view. Under the View Query Actions submenu, it will show all possible actions you can take from a particular view - it will evaluate the view dynamically so any actions will be given the right targets.

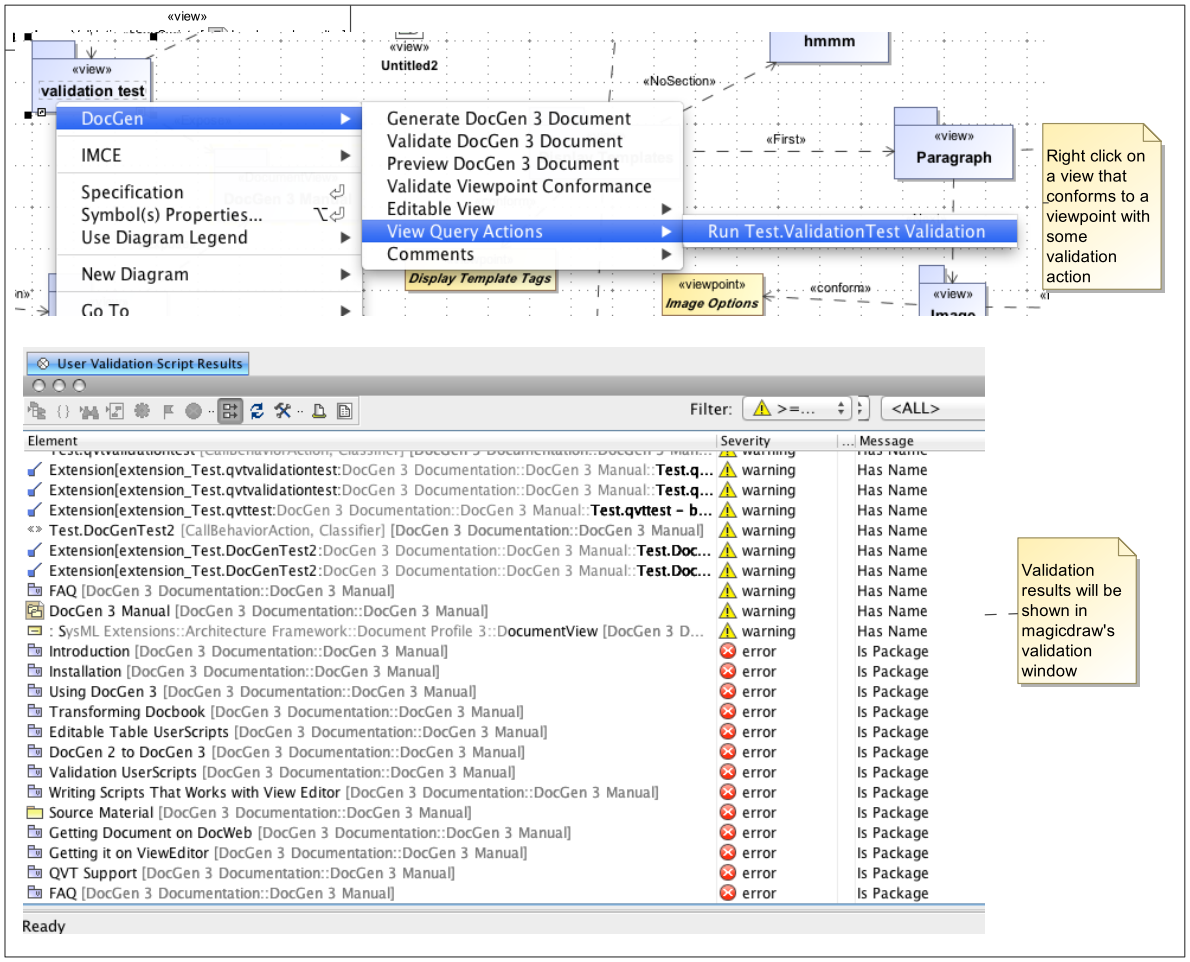
Figure 3.35. Untitled2



4. Validation UserScripts

Validation scripts can be used to output a common table layout for validation suite, rules, and violations. A validation suite consists of one or more rules, and each rule can have one or more violations. A view that conforms to a viewpoint with some validation action can also be run inside Magicdraw that's tied into the Magicdraw validation window.

Figure 4.1. Invoke and results

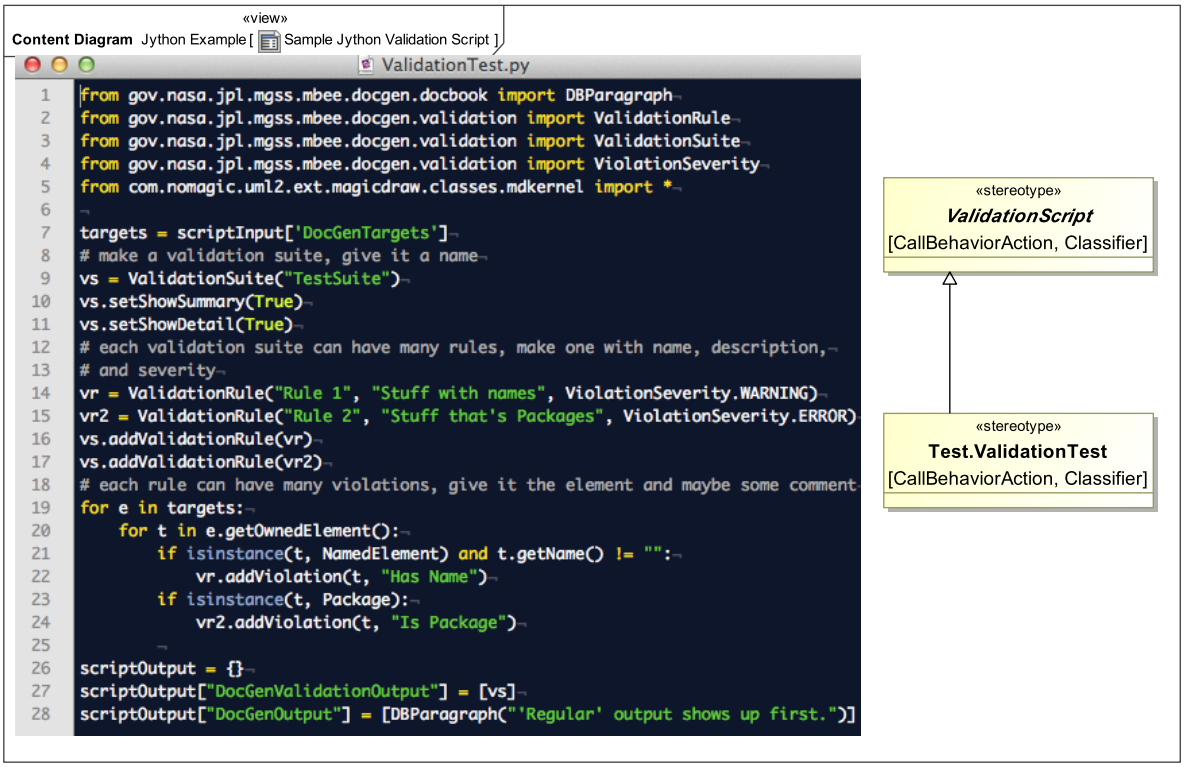


Jython Example

Jython and Groovy examples are similar, so here only the Jython example is shown.

The return result should be a list of ValidationSuites, with key "DocGenValidationOutput". This can be in addition to regular DocGen outputs to the key "DocGenOutput".

Figure 4.2. Sample Jython Validation Script



Like UserScript or EditableTables, make your own stereotype that corresponds to the path under DocGenUserScripts directory, then specialize ValidationScript under the Document Profile 3.

'Regular' output shows up first.

Table 4.1. TestSuite Summary

Validation Rule	Description	Severity	Violations Count
Rule 1	Stuff with names	WARNING	23
Rule 2	Stuff that's Packages	ERROR	14

Table 4.2. TestSuite Detail

Validation Rule	Element	Description
Rule 1	DocGen 3 Documentation::DocGen 3 Manual::Introduction	Has Name
Rule 1	DocGen 3 Documentation::DocGen 3 Manual::Installation	Has Name
Rule 1	DocGen 3 Documentation::DocGen 3 Manual::Using DocGen 3	Has Name
Rule 1	DocGen 3 Documentation::DocGen 3 Manual::Transforming Docbook	Has Name
Rule 1	DocGen 3 Documentation::DocGen 3 Manual::Editable Table UserScripts	Has Name
Rule 1	DocGen 3 Documentation::DocGen 3 Manual::DocGen 2 to DocGen 3	Has Name

Validation Rule	Element	Description
Rule 1	DocGen 3 Documentation::DocGen 3 Manual::Test.TestTable	Has Name
Rule 1	DocGen 3 Documentation::DocGen 3 Manual::Test.DocGenTest	Has Name
Rule 1	DocGen 3 Documentation::DocGen 3 Manual::Test.TestTable2	Has Name
Rule 1	DocGen 3 Documentation::DocGen 3 Manual::Validation UserScripts	Has Name
Rule 1	DocGen 3 Documentation::DocGen 3 Manual::Test.ValidationTest	Has Name
Rule 1	DocGen 3 Documentation::DocGen 3 Manual::Test.CF	Has Name
Rule 1	DocGen 3 Documentation::DocGen 3 Manual::Source Material	Has Name
Rule 1	DocGen 3 Documentation::DocGen 3 Manual::Getting Document on DocWeb	Has Name
Rule 1	DocGen 3 Documentation::DocGen 3 Manual::Getting Document on ViewEditor	Has Name
Rule 1	DocGen 3 Documentation::DocGen 3 Manual::Test.qvttest	Has Name
Rule 1	DocGen 3 Documentation::DocGen 3 Manual::Test.qvtvalidationtest	Has Name
Rule 1	DocGen 3 Documentation::DocGen 3 Manual::Test.DocGenTest2	Has Name
Rule 1	DocGen 3 Documentation::DocGen 3 Manual::FAQ	Has Name
Rule 1	DocGen 3 Documentation::DocGen 3 Manual::MagicDraw API Primer	Has Name
Rule 1	DocGen 3 Documentation::DocGen 3 Manual::View Class Hierarchy	Has Name
Rule 1	DocGen 3 Documentation::DocGen 3 Manual::OCL Expression Primer	Has Name
Rule 1	DocGen 3 Documentation::DocGen 3 Manual::DocGen 3 Manual	Has Name
Rule 2	DocGen 3 Documentation::DocGen 3 Manual::Introduction	Is Package
Rule 2	DocGen 3 Documentation::DocGen 3 Manual::Installation	Is Package
Rule 2	DocGen 3 Documentation::DocGen 3 Manual::Using DocGen 3	Is Package
Rule 2	DocGen 3 Documentation::DocGen 3 Manual::Transforming Docbook	Is Package
Rule 2	DocGen 3 Documentation::DocGen 3 Manual::Editable Table UserScripts	Is Package

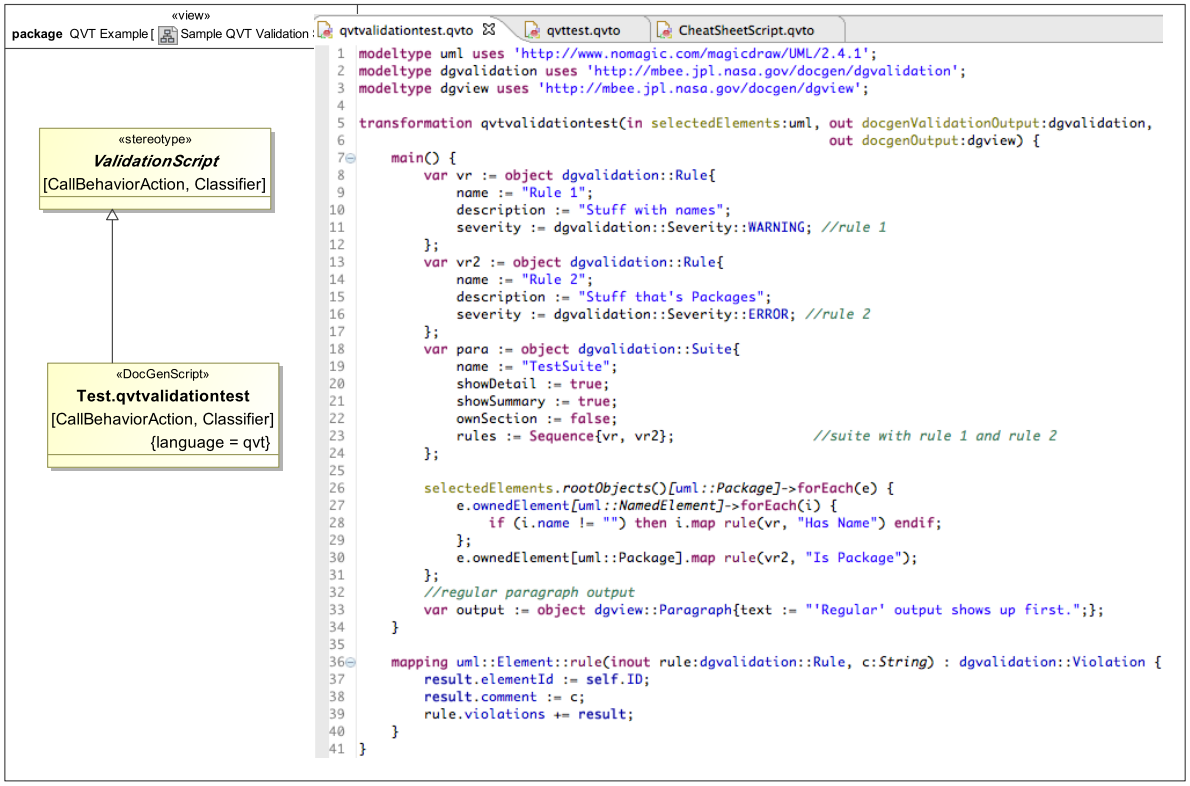
Validation Rule	Element	Description
Rule 2	DocGen 3 Documentation::DocGen 3 Manual::DocGen 2 to DocGen 3	Is Package
Rule 2	DocGen 3 Documentation::DocGen 3 Manual::Validation UserScripts	Is Package
Rule 2	DocGen 3 Documentation::DocGen 3 Manual::Source Material	Is Package
Rule 2	DocGen 3 Documentation::DocGen 3 Manual::Getting Document on DocWeb	Is Package
Rule 2	DocGen 3 Documentation::DocGen 3 Manual::Getting Document on ViewEditor	Is Package
Rule 2	DocGen 3 Documentation::DocGen 3 Manual::FAQ	Is Package
Rule 2	DocGen 3 Documentation::DocGen 3 Manual::MagicDraw API Primer	Is Package
Rule 2	DocGen 3 Documentation::DocGen 3 Manual::View Class Hierarchy	Is Package
Rule 2	DocGen 3 Documentation::DocGen 3 Manual::OCL Expression Primer	Is Package

QVT Example

As with Jython or Groovy scripts, the keyword for outputting validation model is "docgenValidationOutput", and can be in addition to the regular "docgenOutput".

The ecore model for dgvalidation is at http://docgen.jpl.nasa.gov/opsrev/downloads/docgen/dgvalidation.ecore

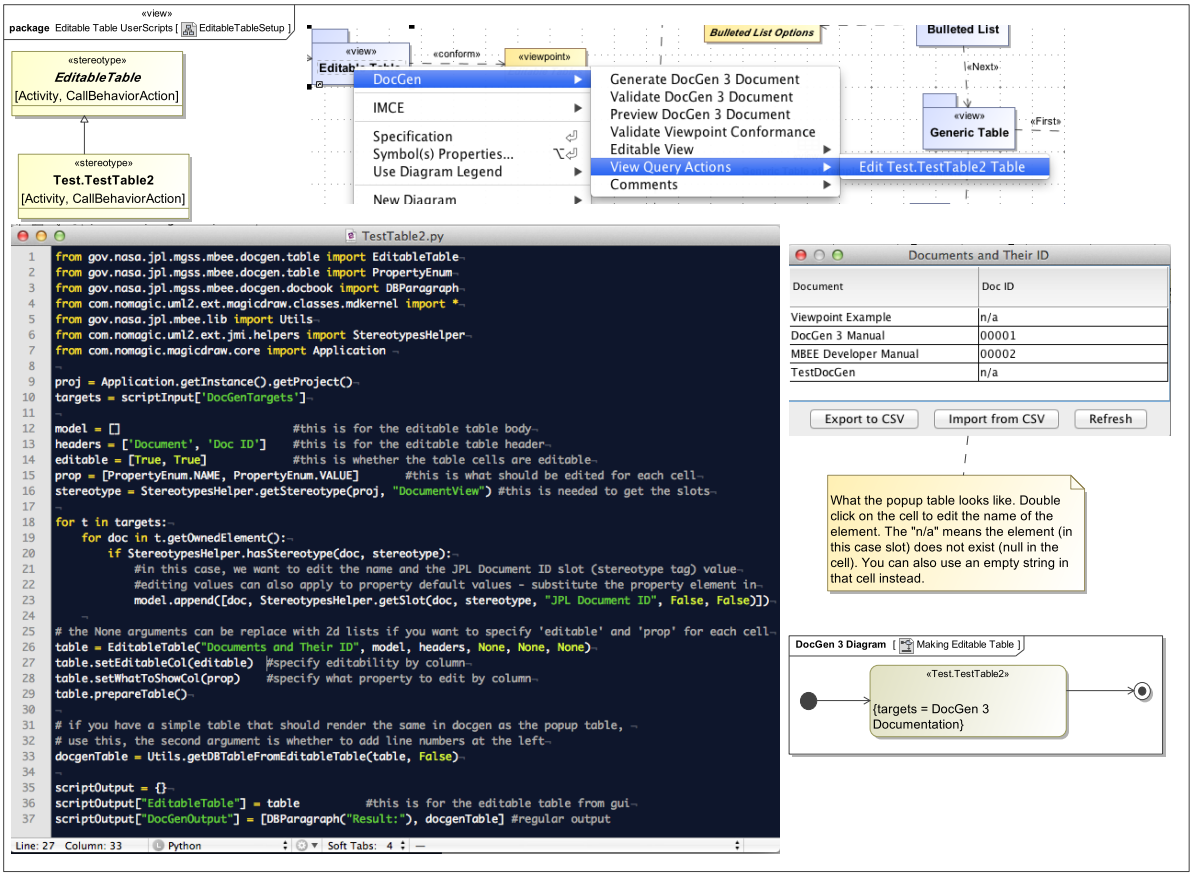
Figure 4.3. Sample QVT Validation Script



5. Editable Table UserScripts

Similar to UserScripts, DocGen 3 now has a mechanism for you to define a table of model elements with certain editable attributes. If you make a Jython or Groovy userscript that extends the EditableTable stereotype in Document Profile 3, you can invoke an Edit Table option when you right click on actions with that user stereotype. Here's an example.

Figure 5.1. EditableTableSetup



Result:

Table 5.1. Documents and Their ID

Document	Doc ID
DocGen 3 Manual	00001
MBEE Developer Manual	00002
TestVEUI	n/a

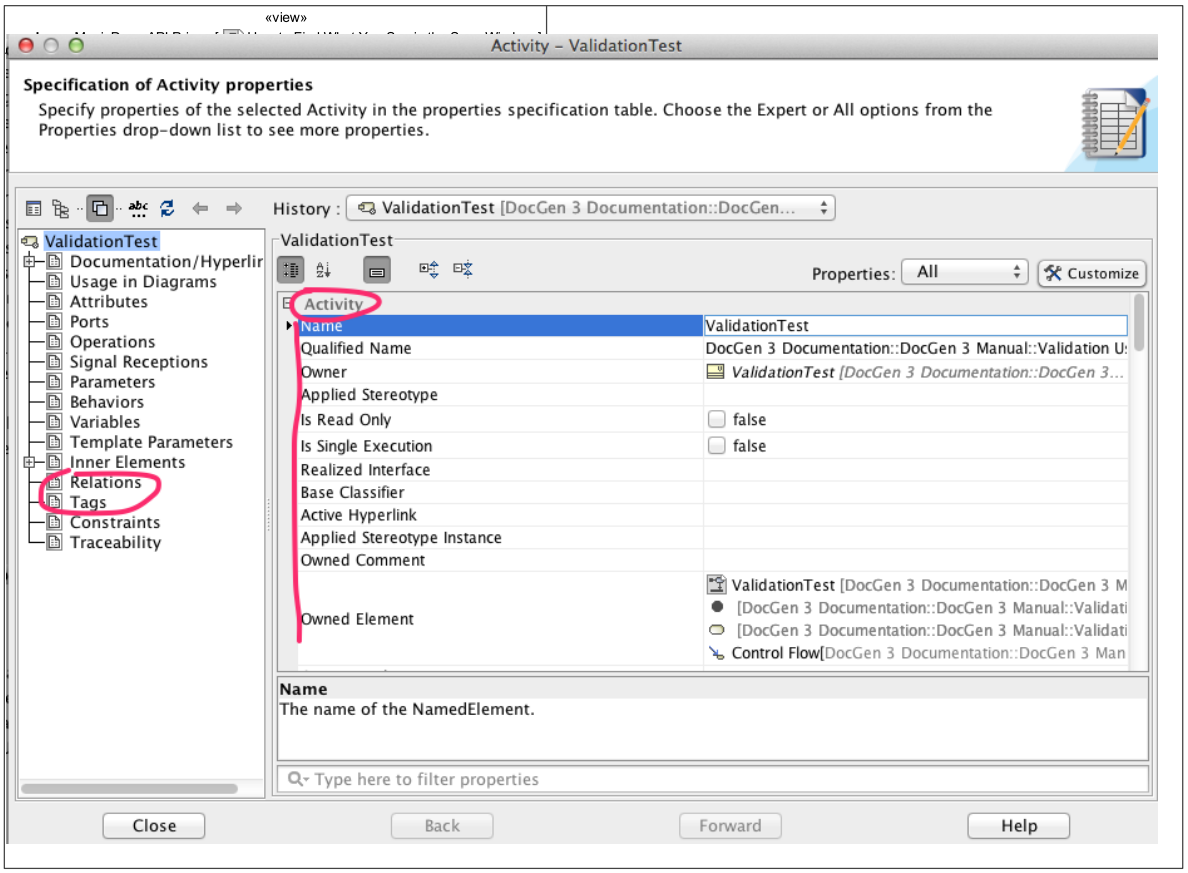
6. MagicDraw API Primer

The MagicDraw Java API's javadoc is in your [md.install.dir]/openapi/docs

Below are useful helper classes in the API:

- ModelHelper
  - Used to get documentation of elements and other things
- ModelElementsManager
  - Used to move/delete elements
- ElementsFactory
  - Used to create new elements
  - get instance by using project.getElementsFactory()
- Project
  - Used to get the elements factory and other things
  - get instance by using Application.getInstance().getProject()
- Application
  - Used to get GUILog and current project
- GUILog
  - Used for logging to gui log
  - get instance by using Application.getInstance().getGUILog()
- StereotypesHelper
  - Used to get/set stereotypes and tag values

Figure 6.1. How to Find What You See in the Spec Window



The "Activity" circled usually tells you the name of the corresponding Java interface, in this case this Activity model element corresponds to an instance of com.nomagic.uml2.ext.magicdraw.activities.mdfundamentalactivities.Activity

There is a huge inheritance hierarchy of uml metaclass interfaces in the javadoc. The things you see in the spec window usually corresponds to some getter/setter of that interface or its superinterfaces. For example there's:

- get/setOwner, getOwnedElement - from Element
- get/setName, getQualifiedName - from NamedElement
- etc.

All elements can get directed relationships or any subclass (like dependencies) using the following:

- get_directedRelationshipOfSource - get outgoing relationships
- get_directedRelationshipOfTarget - get incoming relationships
- get_commentOfAnnotatedElement - get any comments attached

To get things in the Tags pane, use StereotypesHelper

- get/setStereotypePropertyValue
- addStereotype
- getStereotype
- hasStereotypeOrDerived
- etc.

Common error(s):

- ConcurrentModificationException: this means you're adding/deleting elements from a collection while iterating through it. What's most likely happening is you're doing some kind of loop over Element.getOwnedElement() or similar functions directly and inside the loop you're doing a setOwner or addElement that'll result in the Owned Element list changing. To prevent this just make a copy of the collection you're iterating through and loop through the copy instead.

7. OCL Expression Primer

8. Transforming Docbook

Docbook is just a way to mark the structure of a document. To render it as a more traditional document, DocBook XML files must be run through some kind of processor. In the workflow I will be describing, that processing is XSL-FO. What happens is that Oxygen uses an XSLT Processor (like Saxon or XSLTproc) to transform the Docbook XML to Formatted Object (FO). The "rules" of the transformation exist as XSL stylesheets, which are provided with Oxygen. These stylesheets tell the processor how to render various tags and elements it finds in a DocBook XML file. For example, section numbering, font, etc. are defined by the stylesheets. FO is an intermediary step that you never see. It is essentially your document but rendered into sequential "blocks." A FO processor (Oxygen uses "Fop," an Apache product) to transform the serialized blocks of content into a PDF. HTML output is also an option.

Oxygen gives you a somewhat convenient interface to modify the parameters in the stylesheets. Oxygen has a concept of "transformation scenarios," which are linked processing chains which result in different output (like HTML vs. PDF). Oxygen allows you to clone and modify the parameters and customize the output.

Using Oxygen:

1. Get oxygen from the sscae-help site (where you got MD from: type "sscae-help" in your browser).
2. Generate your report, and open the resulting xml in Oxygen.
3. Press the little button at the top that looks like a play button with a wrench on it.
4. Pick DocBook PDF.
5. Pick transform now

That's the basics of the transformation. Here are some notes:

- There are not an infinite number of oxygen licenses! If you aren't using oxygen, please close it and free the license.
- If you are using eclipse with an oxygen plugin, you will be using a license.
- To change how the document looks, go to step 4 above, and then pick "edit." Pick the parameters button.
- For information on the parameters and how to use DocBook xsl, Look at Sagehill's "Docbook XSL: The Complete Guide." Link here: http://www.sagehill.net/docbookxsl/
- DocGen is currently using DocBook 5
- Click on the author mode tab at the bottom of the screen for a preview of your document. You can also write xml in this mode.
- There should ideally be a little green box at the top right of your document before you transform it. If it's not green you can click where it's red to get information about what is wrong.
- You don't HAVE to use Oxygen, any xslt processor will work and fop can produce pdf (this is the setup that docweb uses), the stylesheets are also on the sagehill site.

Useful Parameters

Below are some often used xsl parameters when transforming to html or pdf. These parameters are set for you in DocWeb (see also attributes of the DocumentView stereotype for 3 parameters you can set for DocWeb to pick up). If you're transforming the document yourself in Oxygen, you can set these parameters when you click on the "wrench" button (transformation scenario), and clicking on the Parameters button.

Useful Parameters

Table 8.1. Useful Docbook XSL Parameters

Name	Description
chunkFirstSections	Optional stylesheet parameter. Whether to chunk first sections. If not, this means in html, each section or chapter page will include the first section. Default is true. (Only matters to docweb)
chunkSectionDepth	Optional stylesheet parameter. This is for html chunking - what's the max depth of a section that'll be put on a different page. Default is 20. (Only matters to docweb)
tocSectionDepth	Optional stylesheet parameter. How many nested sections to display in table of contents for html. Default is 20. (Only matters to docweb)

Making HTML Output with Frames

If you want to manually transform the html to have frames for the table of contents, you can do so with some stylesheet parameter tweaks. The parameters to note are:

- chunk.tocs.and.lots
- chunk.tocs.and.lots.has.title
- generate.toc

1. Set chunk.tocs.and.lots to 1. This will generate the table of contents as a separate html file instead of on the title page.
2. Set chunk.tocs.and.lots.has.title to 0. Normally, the separate table of contents would have the document title. If you don't want that since the title will be visible on the actual title page, set this to 0 to make it look nicer.
3. Now that you'll have the table of contents visible always in a frame, you probably wouldn't want redundant table of contents to be generated in each chapter (which is the default). Edit generate.toc. This parameter is a key value pair separated by whitespace. There should already be default values there. Look for the entry "chapter toc,title" and replace "toc,title" with "nop". This will suppress generation of table of contents for each chapter.

You'll also need to make some stylesheet changes that tells the links which target frame to open links in. We'll be calling our content frame "body." In the html/chunk_custom.xsl file, add the following to the stylesheet:

```
<xsl:param name="target.window" select="'body'"/>
<xsl:template name="user.head.content">
<base target="{$target.window}"/>
</xsl:template>
```

Of course, this also applies to all the content pages, but that's ok.

Now when you generate html from Oxygen, there should be an index.html that has the title page, and a bk01-toc.html that's the table of contents. Make another frames.html file thats the table of contents file on the left frame and the index page on the right and you're done!

```
<html>
<head><title>Document Title</title></head>
<frameset cols="30%,*" frameborder="1" framespacing="0" border="1">
<frame src="bk01-toc.html" name="list">
<frame src="index.html" name="body">
</frameset>
```

```
</html>
```

Using DocWeb Stylesheet for Local Generation

Below is a link to a stylesheet customization that is currently used on DocWeb. The output will look similar to an official JPL document. To create a compatible document, make sure the "use default stylesheet" option on DocumentView is FALSE and fill in all desired information. This stylesheet is for PDF. Thanks to Charles Galey for the excellent customization!

Latest DocWeb PDF stylesheet [http://docgen.jpl.nasa.gov/opsrev/downloads/docgen/mgss4.xsl]

To use this with Oxygen, replace the contents of [oxygen install]/frameworks/docbook/xsl/fo/docbook_custom.xsl with the contents of mgss4.xsl.

If you want larger headers, you can use the old stylesheet which includes parameters for jpl.header, jpl.subheader, jpl.footer, and jpl.subfooter. The link below is the PDF stylesheet prior to May 2013.

Old DocWeb PDF Stylesheet [http://docgen.jpl.nasa.gov/opsrev/downloads/docgen/mgss.xsl]

9. Getting Document on DocWeb

To get your document on DocWeb, prepare the following info and contact Doris Lam, Peter Di Pasquale, or Cin-Young Lee.

- The name of your document (name of the DocumentView element)
- The name of the teamwork project where the document is.
- The teamwork server
- Project name (the tab you want it under on docweb)
- A category name (category for the document on the left menu)
- Any local custom profiles or modules, or user scripts - they need to be uploaded to the server where DocWeb runs. (if your things are distributed within the SSCAE Magicdraw prebuilts, you don't need to give us anything)

10. Getting Document on ViewEditor

View editor allows certain properties of document elements to be edited on the web in a sandbox. A Magicdraw user is still responsible for importing or exporting the information to view editor. You can use view editor to edit names and documentation or any element, or the default value of value properties, as shown in a view in the document.

Web interface: https://docgen.jpl.nasa.gov:8443/editor/ui/

URL to put in MD prompt: http://docgen.jpl.nasa.gov:8080/editor

The above url are for MagicDraw 1702SP3-2 or above release by SSCAE (July 18, 2013).

For those still using an older build, use the following url:

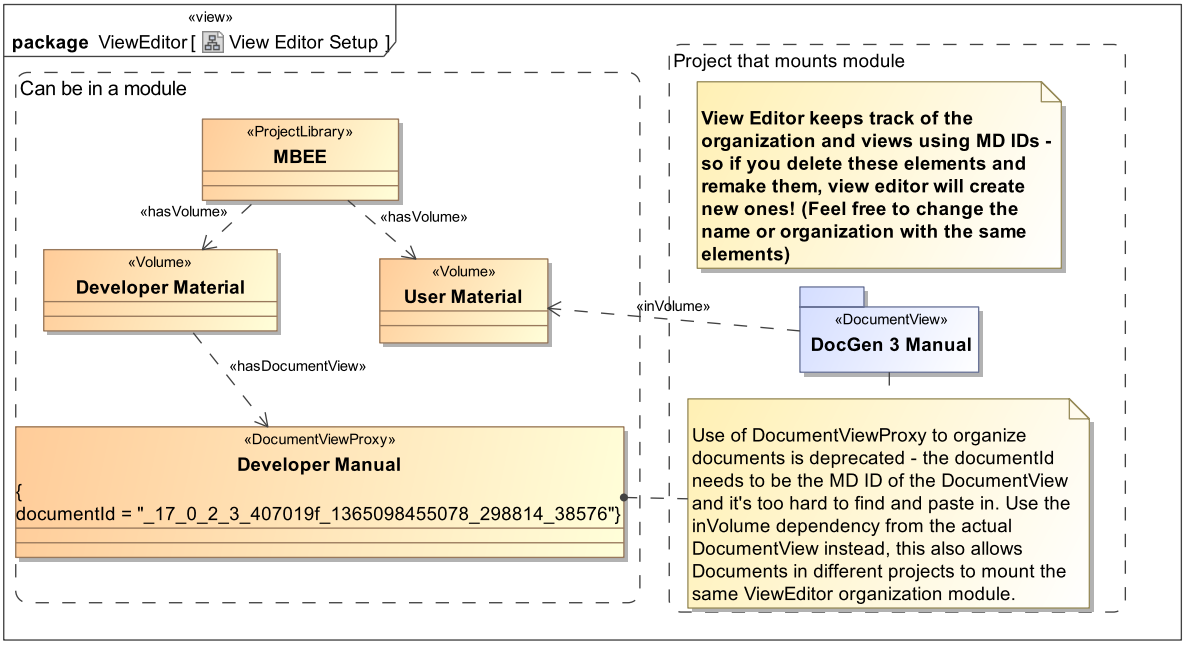
Web interface: https://mbee.jpl.nasa.gov:8443/editor/ui/

URL for MD prompt: http://mbee.jpl.nasa.gov/editor

(Europa have its own separate instance, do not use either of the above)

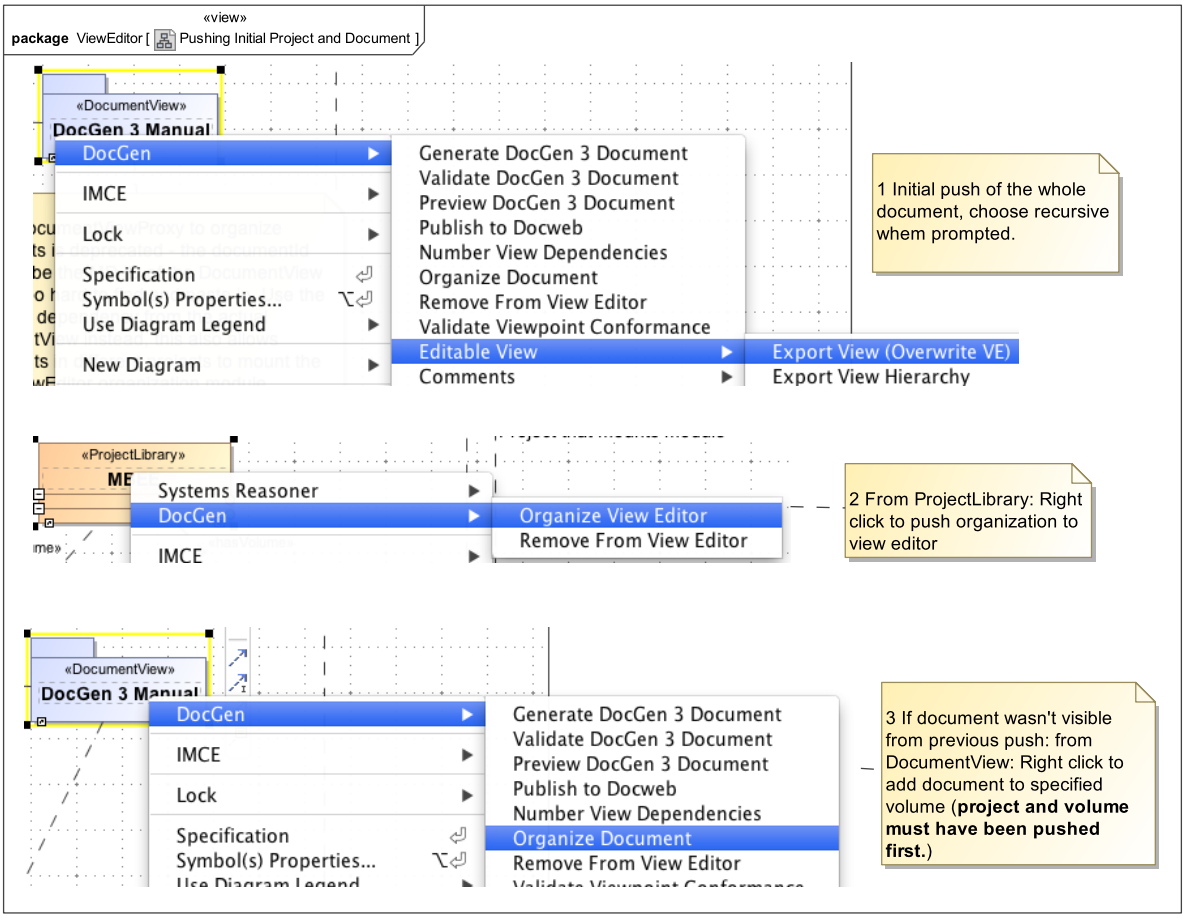
View Editor Setup

Figure 10.1. View Editor Setup



Pushing Initial Project and Document

Figure 10.2. Pushing Initial Project and Document



Editable View Menu Actions

The following explains the options in the EditableView sub-menu, all view import/export/sync actions will show changed or applicable elements in the Magicdraw validation window (for example, elements changed on view editor, elements not locked, elements changed in model, etc)

- Changing project/volume/document organization: just re-organize project or document, right click on each respectively
- Exporting View: can be recursive (include all children) or single view, overwrites what's on view editor
- Importing View: can be recursive or single view, overwrites what's in Magicdraw (if editable)
- Exporting view hierarchy: updates the view tree on view editor only - useful if you want to change the order of views but don't want to import/export yet.
- Synchronize View (merge conflicting): can be recursive or single view, combines an export and import. On export, conflicts between model and view editor is marked with MERGED keyword and contents combined.
- Validate Sync: dry run version of Synchronize View - will indicate what elements have conflicting content.
- Import Comments: can be recursive or single, imports new or edited comments on views from view editor
- Export Comments: can be recursive or single, exports new or edited comments on views to view editor

- View Comments: brings up a dialog showing all comments on that view (that has been imported)

Writing Scripts That Works with View Editor

If you want certain outputs in your userscripts to be editable on the view editor, you'll need to add in element and property information. Currently, the view editor supports documentation, name, number or string slot value or property default values.

The DBText and DBParagraph constructors can be supplied with a magicdraw element and an enum. (See the javadoc for details). Instead of creating the DB*** elements with just text, pass in the element where the text comes from, and which thing to edit. For example, if you had this before:

DBParagraph("some documentation")

Do this instead:

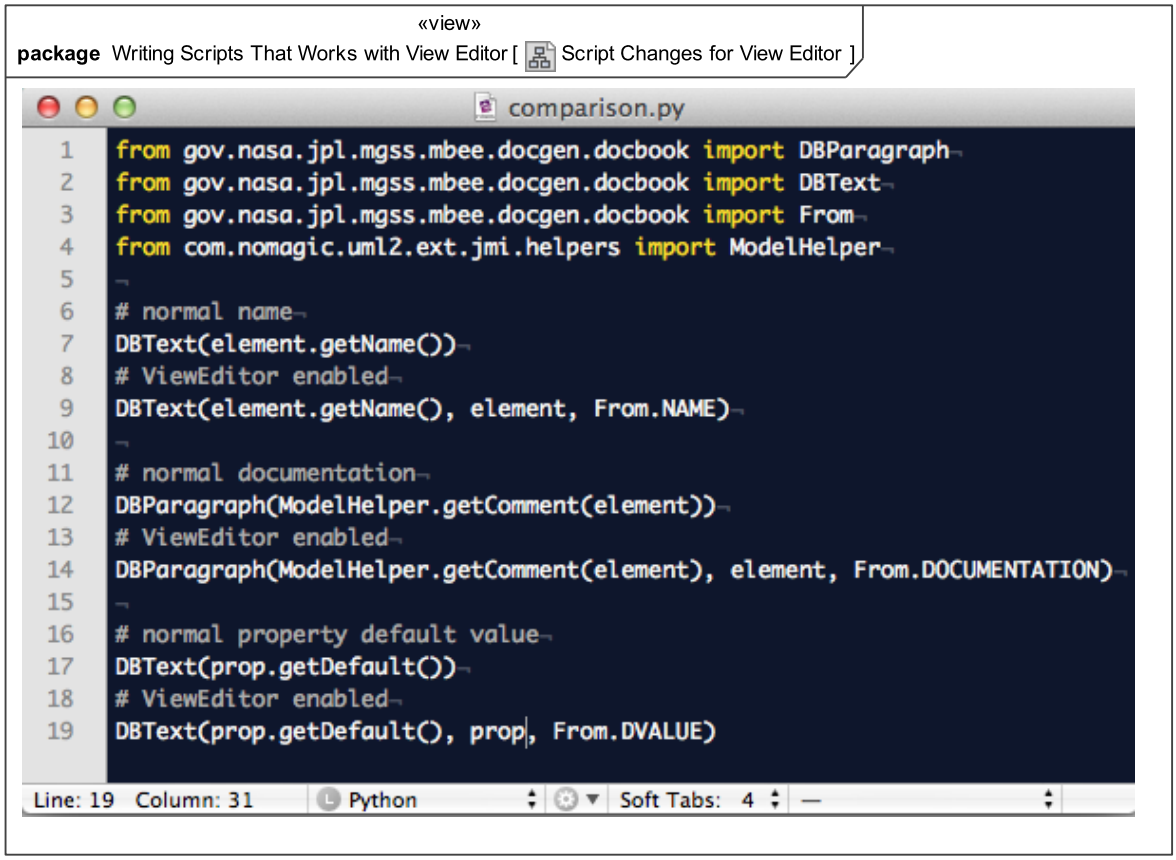
DBParagraph("some documentation", magicdrawElement, From.DOCUMENTATION)

This will tell DocGen that the text comes from the documentation of the passed in element, and if exporting to view editor, that documentation will be editable.

If you already use the EditableTable class and the Utils functions to get back a DBTable, it will be done for you (for anything in the table that's editable).

For property default values, the view editor will display/edit the actual value in the model, while DocWeb or non view editor outputs will use the string text supplied. Therefore, it's possible to have different display of values between view editor and regular output. This is useful if you want, say, a different precision in your reports than the view editor.

Figure 10.3. Script Changes for View Editor



11. View Class Hierarchy

As an alternative to the current view structure modeling pattern (using view Packages, First, Next, noSection stereotypes), DocGen supports the pattern that's been added to the SysML RTF, using a view (class) hierarchy. Products akin to documents can specify their children views and what to exclude from the hierarchy. This pattern's benefit over the old pattern is that the same views can be easily incorporated into different documents (but they're still subject to the defined view hierarchy).

DocWeb will work with this new pattern, but only the Alfresco based ViewEditor is supported. Do not export documents made using this pattern to the non-Alfresco ViewEditor.

Also do not mix and match the 2 patterns!

12. FAQ

- Can I make a table with A, B, C as columns?
  - Take a look at TableStructure, if it doesn't suit your needs, you'll need a userscript
- But what's the PropertiesTable?
  - That's very specific to showing a BSTed system decomposition model with properties, if that's what you need, then great! There's a bunch of options, so experiment a bit and ask if you're stuck.
- How do I make those MEL and PEL tables that Europa have?
  - Ask Chris Delp/David Coren/people on Europa modeling team for the modeling pattern
- So how do I make userscripts/validation scripts/editable tables?
  - Read the following sections: Making User Scripts, Validation Scripts, Editable Table, MagicDraw API Primer
- I'm sold on QVT! How do I start?
  1. Get the Yoxos Eclipse installation (SSCAE Level 4)
     - Alternatively, use this [http://docgen.jpl.nasa.gov/opsrev/downloads/docgen/yoxos-lite.yoxos] lighter weight version. This uses the EE distro as a base and includes all IMCE addons and plugins for MD development, QVT, metamodels. You'll have to install subclipse and/or egit yourself.
  2. Get the dgview [http://docgen.jpl.nasa.gov/opsrev/downloads/docgen/dgview.ecore] and/or dgvalidation [http://docgen.jpl.nasa.gov/opsrev/downloads/docgen/dgvalidation.ecore] metamodels
  3. Read the following sections: QVT UserScript Example, QVT Validation Example
  4. Look at the sample cheatsheet QVT script Alek Kerzhner made on https://sscae-cm.jpl.nasa.gov/svn/mbse-cop/trunk/examples/qvto/gov.nasa.jpl.magicdraw.qvto.example - checkout into Eclipse workspace
  5. attempt to write your script
  6. Ask Alek, Bjorn Cole, Marc Sarrel, Nick Rouquette, or on the mea list if you have questions
  7. Good luck!
- How do I get my document up on DocWeb or ViewEditor?
  - Read the following sections: DocWeb, View Editor
- I'm new to DocGen, how do I start?
  - Go through this section, it's really not that much to read!
- I found a non-critical bug, I wish it does A, why doesn't it do B, can you change it to C?
  - Create a Jira here [https://ehm.jpl.nasa.gov/jira/browse/MDEV]
- I'm totally lost!
  - Have you tried reading the manual, at least the simple examples and tried generating it?