MDK Manual

Jet Propulsion Laboratory
California Institute of Technology

Table of Contents

List of Figures

List of Tables

Section 1. How Do I Install Magic Draw with MDK?

Mac Users -

Optional: This set of instructions is a recommended best practice only, location of MD install does not impact custom icon for Mac Users

1. Open the Finder and Click on Applications in the left hand side
   <img></img>
2. Click on the new folder icon in the upper left corner of the window and name the new folder "MagicDraw"
   <img></img>

   <img></img>
3.
4. Download Latest MD Release from Europa MDev Site on Alfresco:Link
5. Once signed in, navigate to the site "Engineering Modeling System"
6. Copy the downloaded .zip to \Applications\MagicDraw\ and double click to extract it
7. Open the newly extracted Folder and locate "MagicDraw UML"
8. Select this icon and drag it down to your OSX Dock
9. Click the Icon in the Dock to start MagicDraw

Windows Users -

Method 1: Using Fixed Install Directory and Packaged Shortcut (Intermediate)

1. Open Windows Explorer and Create (if it doesnt exist) a folder called "MagicDraw" (Case sensitive) in your C:\ directory.
   <img></img>
   <img></img>
2. Download Latest MD Release from Europa MDev Site on Alfresco:Link
3. Once signed in, navigate to the site "Engineering Modeling System"
4. Extract the .zip to C:\MagicDraw\
5. Navigate your Explorer window to C:\MagicDraw\%mdRelease%\bin
6. Right click on the "Magic Draw UML" shortcut and select Send To...->Desktop (create shortcut)
   <img></img>
7. Click on the Link on your Desktop to start MagicDraw!

Method 2: Creating Your Own Shortcut (Advanced)

1. Download Latest MD Release from Europa MDev Site on Alfresco: Link
2. Once signed in, navigate to the site "Engineering Modeling System"
3. Extract MagicDraw to the location of your choice (Recommended C:\MagicDraw\) and open %mdInstallDir %\bin
4. Right click on mduml.exe and select Send To...->Desktop (create shortcut)
5. Navigate to your desktop and right click on mdul - Shortcut and click Rename, enter in "MagicDraw UML" and hit enter
6. Now right click on "MagicDraw UML" and select Properties
   <img></img>
7. In the Properties window click "Change Icon..."
   <img></img>
8. The Change Icon dialog will open, select Browse... and Navigate to \%mdInstallDir%\bin\Europa_ %ReleaseName%_Logo_256.ico and click Open
   <img></img>
9. Select the Europa Icon that appears and Click OK

10.Hit Apply and OK
11.Double-click on the shortcut to open MagicDraw!

Section 2. How Do I Create Documents and Views?

Documents are a key part of systems engineering, one of the key pieces to widespread adoption of SysML at JPL was when DocGen enabled Model Based Systems Engineering (MBSE) practitioners to produce documents from their models. In order to create these documents the view and viewpoint pattern was introduced, this pattern has since been adopted by OMG and was incorporated into SysML 1.4. A document follows a Class hierarchy where each class is a document section. The root element is the document itself which contains all the information about the document including authors, and other formatting metadata. This data can be manipulated by DocBook to produce professional documents that mirror hand created documents used in standard JPL practice. Manupulation and input of this metadata is discussed in
<a>["How Do I Use the Docgen Stylesheet?":name:_17_0_2_3_8660276_1394734844442_201184_71850]

2.1. How Do I Create a Document?

2.2. How Do I Order and Create View Hierarchy's?

Figure 2.1. Multiple Views Use Case

Each view is represented by a section in a document. For example, this section/ view that is currently being displayed is ["How Do I Connect Multiple Views as One section?":name:_17_0_2_3_e9f034d_1384381686222_52312_66992] . Notice in the diagram below, this section of the document has multiple views connected to it these associations serve to establish view hierarchy. Essentially the hierarchy is established by adding additional associations "underneath" other views to create a tree.

Sections can be ordered by rearranging the attributes within the Specification window. For example, see the sub-sections for Section 2 are shown in the Attributes tab below,

This order is incorrect, it would be better to have ["How Do I Create Non-standard Views?":name:_17_0_2_3_8660276_1395106664281_814376_67950] before ["How Do I Input Text Into Documents?":name:_17_0_2_3_e9f034d_1384384755210_265104_68051. This can be rectified by selecting ["How Do I Input Text Into Documents?":name:_17_0_2_3_e9f034d_1384384755210_265104_68051]in the attributes section and clicking "Down'

The sections will now be rendered in the proper order. The diagram below shows how the view hierarchy is represented in the View Diagram.

Figure 2.2. How Do I Create Views and Documents?

2.3. How Do I Create Non-standard Views?

Figure 2.3. How Do I Create Views and Documents?

2.3.1. How Do I Connect Multiple Views as One section?

Figure 2.4. No Section Use Cases

By default, each <<view>> denotes a new section in the document. Use the "No Sections" property in the <<product>> or <<document>> element to specify <view>'s that are to not be made into sections. The "No Sections" property with a view selected is shown in the diagram below. This is the specification window for the Docgen Manual <<document>>.

This is the <
<view> <view>> to be rendered as not a new section. </view>

2.3.2. How Do I Create Diagram Only Views?

Figure 2.5. How Do I Create Diagram Only Views?

The right hand purple box of the view diagram above acts as the view to be rendered and is shown in the document as a subsection of the "How do I Order and Create Multiple Views" section.

2.4. How Do I Input Text Into Documents?

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 <u>underline</u> or hyperlinked text, email, and $_{\text{subscripts}}$ and $^{\text{superscripts}}$.

```
Other preformmatted text like code and xml (the < chars in xml still have to be escaped in the html source).
<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 2.1. 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)

You can access html on the magicdraw side by going to the specification window and documentation and selecting html. You can also directly insert html on view editor.
<img></img>

Section 3. Viewpoints

3.1. What is a Viewpoint?

A viewpoint can be thought of as a template for a view and the user is responsible for piecing together this template in order to attain the desired result. Viewpoints allow for the inclusion of images, text, tables, and other document elements and they also extend existing SysML and are being formally introduced in SysML 1.4, as part of Onto-Behavior.

Two main elements make up a viewpoint:

1. Element stereotyped <<viewpoint>>

2. An associated activity known as a viewpoint method

The two use cases below will be addressed next.

Figure 3.1. Use Cases

Later, advanced operations for sorting, filtering and collecting elements with a method will be discussed.

3.2. Linking a Viewpoint to a View

Viewpoints are added to a view via the <<conform>> generalization. Each view can be linked to only one viewpoint using conform. The proper modelling practice for views is that they should always conform to a viewpoint to convey information. If the view does not conform to a viewpoint the validation window will show a warning each time a document is generated/posted online.

Figure 3.2. Linking a viewpoint to a view

3.3. Creating a Viewpoint diagram

The most efficient way to develop a viewpoint method is by creating a viewpoint method diagram. In this section the creation of a viewpoint method diagram is discussed. In subsequent sections the various methods are discussed along with examples on how they might be used in documenting a model.

Figure 3.3. Viewpoint Creation View Diagram

Viewpoint methods are created by right clicking on an already created viewpoint element and selecting New Diagram > MDK > 'Viewpoint Method Diagram'

Once created the method will be automatically named the same as the viewpoint. In general it is good modelling practice to ensure that the method is named after the viewpoint. Each viewpoint should only have **one** viewpoint method, multiple methods will result in an error.

A viewpoint method consists of multiple actions or 'methods' that determine how information is displayed within the view. What each method displays is based on the elements <<exposed>> by the view. These elements can

be operated upon via Collect, Filter, and Sorting operations. Once the desired set of content is obtained, the information can be displayed in tables, paragraphs, images, lists etc. Below is an example viewpoint method that collects the elements owned by the exposed package (see the view diagram above) and then displays the collected elements using a <<Structured Query>> which contains an additional method that was created specifically for this document to display Use Case, View Diagrams and Viewpoint Method Diagrams (This additional method is symbolized by the pitch fork symbol in the <<StructuredQuery>> element).

Figure 3.4. Your Viewpoint

3.4. How Do I Collect/Sort/Filter Elements Exposed to a Viewpoint?

Once exposed to a view, elements can can be operated on by viewpoint methods. There are three types of viewpoint operators, Collect, Sort and Filter. These operations can be used to expand or narrow the collection of elements that are used to display information in ViewEditor.

For example a common use case would be when a group of elements contained within a package need to be exposed to a view. Instead of individually exposing each element, a user can employ CollectOwnedElements within a viewpoint and then expose only the package. The collection method will look at the package and return all the elements that it owns. If the user only wants Blocks to be a part of the output a FilterByStereotypes can be added specifying the <<Block>> stereotype to focus the operation. Finally if the blocks should appear in a certain order, a sort operation can rearrange the elements.

Figure 3.5. Operations Use Cases

3.4.1. How Do I Collect?

Collection is a class of viewpoint operations that changes the items exposed to the view.

Most of the examples use the following "CollectModel" to demonstrate the various collection methods.

Figure 3.6. CollectModel

The above diagram is the example model that is used for this specific example.

3.4.1.1. CollectOwnedElements

The most commonly used collect method is CollectOwnedElements, which, can take an exposed element and collect all things owned by it.

Note: CollectOwnedElements is only needed if the desired elements are owned by the element exposed to the view. It is **not needed if the element is directly exposed to the view**.

Figure 3.7. CollectOwnedElements

This is the view diagram used to demonstrate CollectOwnedElements. This example uses the CollectModel described here . Note that the package and activity diagrams exposed are a part of the documentation of this example and are filtered out for the purposes of demonstrating the CollectOwnedElements method.

Figure 3.8. CollectOwnedElements

This is the viewpoint method used to demonstrate the use of CollectOwnedElements.

Note: The FilterByMetaclasses method is used to remove the diagrams used to document this example and is not necessarily needed for the average use of CollectOwnedElements.

User Editable Tags:

Depth-> Allows user to select the recursive depth of collection

Below is a table showing all the elements that were collected from the Package "CollectModel" using CollectOwnedElements

Table 3.1.

Element Name	Owner	Stereotypes
Curly	CollectModel	ThirdStooge
	Curly	
Slot		
	Brother = Shemp	
Larry	CollectModel	
Curly	Larry	PartProperty
	Curly	
	Larry	
	Larry	
Moe	CollectModel	Block
	Moe	
nyuh nyuh nyuh	Moe	
	CollectModel	
	CollectModel	
	CollectModel	
	CollectModel	accountableFor
ThirdStooge	CollectModel	
base_Element	ThirdStooge	
Brother	ThirdStooge	
	CollectModel	
extension_unnamed1		
	extension_unnamed1	
	extension_unnamed1	
Shemp	CollectModel	
Generalization	Shemp	

Element Name	Owner	Stereotypes
quote	CollectModel	
John	CollectModel	ThirdStooge
	John	

3.4.1.2. CollectThingsOnDiagram

CollectThingsOnDiagram will collect all the elements depicted on a diagram. To demonstrate this the bdd of the CollectModel was used again.

Figure 3.9. CollectModel

The above diagram is the example model that is used for this specific example.

Figure 3.10. CollectThingsOnDiagram

This is the view diagram created for this demonstration. Note that the CollectModel bdd is exposed, the other diagrams are used as documentation of this example and are filtered out prior to the collection like the previous example.

Figure 3.11. CollectThingsOnDiagram

This is the viewpoint method used to demonstrate CollectThingsOnDiagram.

Note: The FilterByDiagramType method is used to remove the diagrams used to document this example. While it is not necessarily needed for the average use of CollectThingsOnDiagram, it can be used to ensure that only elements on certain diagram types are collected.

User Editable Tags:
Depth-> Allows user to select the recursive depth of collection

Below is a table showing the elements collected from the CollectModel bdd.

Table 3.2.

Element Name	Owner	Stereotypes
	CollectModel	
	CollectModel	
Curly	Larry	PartProperty
nyuh nyuh nyuh	Moe	
Curly	CollectModel	ThirdStooge
Larry	CollectModel	
CollectModel	Viewpoint Operations	DiagramInfo
	CollectModel	
	Larry	
	Larry	

Element Name	Owner	Stereotypes
Moe	CollectModel	Block
	CollectModel	accountableFor
CollectModel	Viewpoint Operations	
quote	CollectModel	

3.4.1.3. CollectOwners

CollectOwners allows the user to expose an element and collect its owning elements.

Figure 3.12. CollectOwners

This is the view constructed to demonstrate the CollectOwners viewpoint method. To demonstrate this method the component 'Larry' from the CollectModel example is exposed.

Figure 3.13. CollectOwners

The viewpoint above is used to demonstrate CollectOwners.

Note: The FilterByMetaclasses method is used to remove the diagrams used to document this example and is not necessarily needed for the average use of CollectOwners.

User Editable Tags:
Depth-> Allows user to select the recursive depth of collection.

Below is a table showing the elements collected using CollectOwners on the component 'Larry':

Table 3.3.

Element Name	Owner	Stereotypes
CollectModel	Viewpoint Operations	
Viewpoint Operations	Unit Testing	
Unit Testing	Data	
Data		SSCAEProjectModel ModelManagementSystem

3.4.1.4. CollectByStereotypeProperties

CollectByStereotypeProperties is a single method used to collect certain properties of a stereotype. This is set by selecting the property from the inspection window. To demonstrate this collection, a stereotype ThirdStooge was created with a tag property called "Brother". The end goal of the collection will be the specific instance of the property when the stereotype is attached to the Curly class element.

Figure 3.14. CollectByStereotypeProperties

This is the view used to demonstrate the CollectByStereotypeProperties viewpoint method. For this test, the class 'Curly' from CollectModel above, was given a stereotype <<ThirdStooge>>. ThirdStooge has a property "Brother" which will be the target of the collection. For reference the ThirdStooge stereotype is shown on the lower right in diagram above.

Figure 3.15. CollectByStereotypeProperties

Note: The FilterByMetaclasses method is used to remove the diagrams used to document this example and is not necessarily needed for the average use of CollectByStereotypeProperties

User Editable Tags:
Stereotype Properties-> Allows user to select the property element to collect from the exposed elements. This can be done by navigating to the StereotypePropertiesChoosable within the CollectByStereotypeProperties specification window.
<img></img>
Selecting [...] will open a select window allowing the user to navigate to the stereotype element and select the desired property (don't forget to add the selected element by using the '+' between the two windows).
<img></img>
Below is a table showing the elements collected when CollectByStereotypeProperties is applied to an element containing the <<ThirdStooge>> stereotype.

Table 3.4.

name
Shemp

3.4.1.5. CollectByDirectedRelationshipMetaclasses

The CollectByDirectedRelationshipMetaclasses viewpoint method will take elements and collect them based on the relationships that connect them together. For this example, the CollectModel package is used for collecting all elements in the model and grabbing those that are related by the specific relationship set in the method.

Figure 3.16. CollectByDirectedRelationshipMetaclasses

This is the view used to demonstrate the method, the element exposed for this example is the package CollectModel described in above sections.

Figure 3.17. CollectByDirectedRelationshipMetaclasses

This viewpoint is designed to demonstrate CollectByDirectedRelationshipMetaclasses. Often the user will want specify a package rather than the relationship itself, as in this example. To handle this a CollectOwnedElements method should be used to expose the model elements and relationships to the view.

User Editable Tags:
Metaclasses-> Allows user to select the desired type of relationships to collect. This can be done by navigating to the MetaclassChoosable within the CollectByDirectedRelationshipMetaclasses specification window (shown below).
<img></img>
Selecting [...] will open the element selector, allowing the selection of the desired metaclass. **Make sure the metaclass option is selected in the lower left corner** or else metaclasses will not show in the search.
<img></img>
Below is a table showing the results from using CollectByDirectedRelationshipMetaclasses on the CollectModel package.

Table 3.5.

Element Name	Owner	Stereotypes
Curly	CollectModel	ThirdStooge

Element Name	Owner	Stereotypes
Larry	CollectModel	

3.4.1.6. CollectByDirectedRelationshipStereotypes

The CollectByDirectedRelationshipStereotypes viewpoint method will take elements and collect them based on the relationships that connect them together. For this example, the CollectModel package is used by collecting all elements in the model and grabbing those that are related by a dependency stereotyped by <<accountableFor>>.

Figure 3.18. CollectByDirectedRelationshipSterotypes

This is the view used to demonstrate the method, the element exposed for this example is the package housing the collected model.

Figure 3.19. CollectByDirectedRelationshipStereotypes

This viewpoint is designed to demonstrate CollectByDirectedRelationshipMetaclasses. Often the user will want specify a package rather than the relationship itself, as in this example. To handle this a CollectOwnedElements method should be used to expose the model elements and relationships to the view.

User Editable Tags:
Stereotypes-> Allows user to select the desired stereotype of relationships to collect based on. This can be done by navigating to the StereotypesChoosable within the CollectByDirectedRelationshipStereotypes specification window.
<img></img>

Selecting the [...] will open an element selection window allowing the user to pick the desired stereotype.

Below is the result from using CollectByDirectedRelationshipStereotypes on CollectModel.

Table 3.6.

Element Name	Owner	Stereotypes
Larry	CollectModel	

3.4.1.7. CollectByAssociation

Figure 3.20. CollectByAssociation

Figure 3.21. CollectByAssociation

Table 3.7.

Element Name	Owner	Stereotypes
Curly	CollectModel	ThirdStooge

3.4.1.8. CollectTypes

Figure 3.22. CollectTypes

Figure 3.23. CollectTypes

Table 3.8.

Element Name	Owner	Stereotypes
Curly	CollectModel	ThirdStooge

3.4.1.9. CollectClassifierAttributes

Figure 3.24. CollectClassifierAttributes

Figure 3.25. CollectClassifierAttributes

Table 3.9.

Element Name	Owner	Stereotypes
nyuh nyuh nyuh	Moe	

3.4.1.10. CollectByExpression

Figure 3.26. CollectByAssociation

Figure 3.27. CollectByExpression (Basic)

Table 3.10.

Element Name	Owner	Stereotypes
Curly	Larry	PartProperty
	Larry	
	Larry	

3.4.2. How Do I Sort?

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

Figure 3.28. Sort Use cases

Figure 3.29. Example model

This is the example that will be used throughout the table explanation.

The table below shows an unsorted table of information that will later be used to sort the elements in a specific manner. How to create this table is covered in a different section .

Table 3.11.

Name of Element	Mass Values	Part ID	Color
Bolt	1.0	2	black
Car	211.0	3	red yellow
Screws	2.0	4	silver
Steering Wheel	30.0	5	red
Tire	40.0	1	black

3.4.2.1. Sort By Attribute

This first example shows how a list can be alphabetically sorted either in normal order or reverse. The result is the following bulleted lists, one in alphabetical and one in reverse alphabetical.

Figure 3.30. Sorting example

This diagram shows the method used.

Sort by name attribute

- Bolt
- Car
- Screws
- Steering Wheel
- Tire

Sort by name attribute reversed

- Tire
- Steering Wheel
- Screws
- Car
- Bolt

3.4.2.2. Sort by Property

Here, the sort feature used is showing how sort can be applied to value properties. Specifically, id and mass are numerical values. When using the 'SortByProperty' action, these values are ordered from smallest number to largest number.

Figure 3.31. Sort by Property

Sort by "mass" property

Table 3.12.

Name of Element	Mass Values	Part ID	Color
Bolt	1.0	2	black
Screws	2.0	4	silver
Steering Wheel	30.0	5	red
Tire	40.0	1	black
Car	211.0	3	red yellow

3.4.2.3. Sort By Expression

Using an OCL expression can also result in doing a numbered or alphabetical sort, however, using OCL gives the capability to sort on many different properties that aren't reachable via the typical attribute and property sorts.

Figure 3.32. Sorting example

This diagram shows the method used.

Sort by name expression

- Bolt
- Car
- Screws
- Steering Wheel
- Tire

3.4.3. How Do I Filter

Once collections have been specified, filters can be used to further refine the elements that will be a part of the view. Each type of filter has an include tag that can be set to true or false. If the filter is for inclusion (include is 'true', which is the default) the system will only allow elements that meet the filter criteria to pass. If include is false the filter will exclude items that meet the filter criteria and allow all other elements to pass.

3.4.3.1. FilterByDiagramType

Figure 3.33. FilterByDiagramType

Figure 3.34. FilterByDiagram

This method is intended to filter out all exposed elements except for activity diagrams and display the diagram particulars in a table. This works well for all standard MagicDraw diagram types. It does not currently support custom diagrams such as viewpoint method and view diagrams. A work around for these diagrams can be found in the 'Document Method' viewpoint method shown below, which, is also utilized throughout this document.

Figure 3.35. Document Method

3.4.3.2. FilterByNames

Figure 3.36. FilterByNames

Figure 3.37. FilterByNames

Regex Tutorial - Link

Table 3.13.

Element Name	Owner	Stereotypes
Curly	CollectModel	ThirdStooge
Larry	CollectModel	
Curly	Larry	PartProperty

3.4.3.3. FilterByMetaclasses

Figure 3.38. FilterByMetaclasses

Figure 3.39. FilterByMetaclasses

Table 3.14.

Element Name	Owner	Stereotypes
Larry	CollectModel	

3.4.3.4. FilterByStereotypes

Figure 3.40. FilterByStereotypes

Figure 3.41. FilterByStereotypes

Table 3.15.

Element Name	Owner	Stereotypes
Curly	CollectModel	ThirdStooge
John	CollectModel	ThirdStooge

3.4.3.5. FilterByExpression

Filter by expression is a versatile viewpoint method that allows the user to filter items using an OCL expression. For more information on OCL and other query langauges please refer to this section

Figure 3.42. FilterByExpression

As usual, when a package is exposed the first action of the viewpoint is to <<CollectOwnedElements>> which will further expose the collect model. The proper use of the FilterByExpression block requires an OCL string that is a complete Boolean expression. In this example, elements of Type 'Property' are selected using: "oclIsTypeOf(Property) = true". The result statement is required for the filter to operate properly.

Figure 3.43. FilterByExpression

As depicted in the viewpoint, the result should be a table that prints all the properties that are owned by the Collect and Filter Model:

Table 3.16.

Element Name	Owner	Stereotypes
Curly	Larry	PartProperty
	Larry	
nyuh nyuh nyuh	Moe	
base_Element	ThirdStooge	
Brother	ThirdStooge	

3.5. How Do I Display Images & Paragraphs In A Viewpoint?

The simplest viewpoint method is Paragraph, which allows text to be typed in the "body" tag of the viewpoint method. If "body" is empty, it prints out the documentation of the targets. For any templates that involve 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 this link... HTML

Figure 3.44. Paragraph

Below is the activity diagram for this whole section.

Figure 3.45. 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

3.6. How Do I Make a Bulleted List?

A more complex viewpoint method allows the creation of bulleted lists based on model elements. 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. The table below shows the tag options that are unique to bulleted lists.

Table 3.17.

Name	Description
orderedList	Optional. Check this to make the list numbered. Default is just bullets.
showTargets	Optional. If showDoc or stereotypeProperties is filled in, this controls whether the owning element name will be shown as the high level bullet. default is false.
showStereotypePropertyNames	Optional. Prints out the stereotype property name before listing its values. default is false.

Figure 3.46. Monster Things

The above diagram is the example material used to create the bulleted lists.

Figure 3.47. Bulleted List

Below are the results for the above viewpoint method...multiple different bullet lists that are editable.

- 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

3.7. How Do I Make Tables?

Table creation is quite possibly the most interesting part of complex document development. Tables can be used to display a type of analysis, or a collect and filter of a model, in a more visually appealing format. The most direct way to build tables in docgen is to use the <<TableStructure>> stereotype. This stereotype indicates the start of a table for a viewpoint method. In other words, <<TableStructure>> will "table-ize" the info from a given viewpoint. Inside a table, a user can specify what each column in the table will represent. (Currently, DocGen does not allow users to specify what each row will encompass). The following views will show several different columns a user can choose to display information in a table. All examples will use the model shown in the next diagram.

Figure 3.48. Table Use Cases

Figure 3.49. Example model

This is the example that will be used throughout the table explanation.

The table below shows an unsorted table of information that will later be used to sort the elements in a specific manner. How to create this table is covered in a different section .

3.7.1. How Do I Get Names of Elements to Show up in a Table?

This viewpoint method shows a table that displays the name of the elements in the example above. In order to display the name of the elements, the stereotype <<TableAttributeColumn>> was selected. Also, in this method are the typical collects and filters needed to grab the relevant information from the model. For more information on collecting, filtering, and sorting read the Operate on Elements in a Viewpoint section.

Figure 3.50. Name of Elements

The column name ("Name of Element") shown below in the table, is the name of the action that is stereotyped <<TableAttributeColumn>> , as seen above. **Note: the users is required to fill out the specification window to indicate the type of attribute to be shown in the table. In the example above, desiredAttribute = Name. This can be done by going to the specification window for the activity stereotyped <<TableAttributeColumn>> and scrolling to the bottom.
<img></img>
The results of the viewpoint method showing the name of elements in a table is shown below: (queried from Car Example Model)

Table 3.18.

Name of Element
Bolt
Car
Screws
Steering Wheel
Tire
Part1

3.7.2. How Do I Get Element Documentation Shown in a Table?

Again, using the activity stereotyped <<TableAttributeColumn>>, "Documentation" can be selected as a desiredAttributes. Choosing this option will display the element documentation in the table.

Figure 3.51. Documentation of Elements

This table shows the name of elements and their documentation as a result of the viewpoint method above.(queried from Car Example Model)

Table 3.19.

Name of Element	Documentation of Element
Bolt	Animated Dog.
Car	Will be self driving in 2020.
Screws	Inclined plan wrapped around pointed cylinder.
Steering Wheel	Spinning thing for people.
Tire	Spinning thing.
Part1	This is a part.

3.7.3. How Do I Get Values Shown in a Table?

Using the same activity stereotyped <<TableAttributeColumn>>, choose the "Value" desired attribute to show the property default values.

Figure 3.52. Property of Elements

Note: some properties don't actually have values defined or names. (queried from Car Example Model)

Table 3.20. Name of Elements Table

Name of Property	Property Values of Element
mass	1.0
unnamed1	
mass	211.0
mass	2.0
mass	30.0
mass	40.0
base_Element	
id	
colors	black
extension_Part	

3.7.4. How Do I Get Property Values and Tag Values Shown in a Table? (Includes StereoType Tag Values)

Properties and tagged values are commonly displayed in tables. In order to display property type information, the activity stereotyped <<TablePropertyColumn>> is preferred. In the specification window for this activity, a "desiredProperty" needs to be selected by the user.

Figure 3.53. Property Values of Elements

img_1392068625518 The result of the viewpoint method showing the name and property value columns (mass, id, colors) is shown below. Note that each case is depicted in this table, the mass is just a property that is defined on each element while each element is also stereotyped <<Part1>> and the tag values for color and ID are filled out and displayed in the table below. (queried from Car Example Model)

Table 3.21.

Name of Element	Mass Values	Part ID	Color
Bolt	1.0	2	black
Car	211.0	3	red yellow
Screws	2.0	4	silver
Steering Wheel	30.0	5	red
Tire	40.0	1	black
Part1			black

3.7.5. How Do I Make a Validation/Constraint Table?

Figure 3.54. Viewpoint Constraint Use Cases

In the viewpoint below the main focus is the action stereotyped <<ViewpointConstraint>>. A viewpoint constraint element allows a user to specify an OCL expression that evaluates as true false. You can think of the ocl expression itself as an analysis or a validation/verification check. In order to input the ocl expression the specification window is used and the tag "Expression" is filled out as shown below:

The viewpoint method below shows how to connect the <<viewpointconstraint>> element into the method to be evaluated. Note that the action itself just passes the collection of elements on to the next action. (Noted with pass through)

Figure 3.55. Viewpoint Constraint

img_1392068187105 The viewpoint constraint is evaluated and creates two validation tables shown below. The first table shows a summary of the rules and how many violations their are for that given rule. The second table gives a more detailed look a the elements that are violating a specific rule. The element column lists the full qualified name of any element that breaks the validation rule and the description column list the magicdraw element id for the violating element and the specific ocl rule that is being broken. The final table in this section shows the element that the viewpoint constraint evaluated using the <<TableStructure>> context.

Table 3.22. List Specified Elements Conformance to Relationship Summary

Validation Rule	Description	Severity	Violations Count
List Specified Elements Conformance to Relationship	Viewpoint Constraint	WARNING	1

Table 3.23. List Specified Elements Conformance to Relationship Detail

Validation Rule	Element	Description
List Specified Elements Conformance to Relationship	EMS & Architecture Training Material::MDK Manual::Resources::Docgen::Docgen Example Viewpoints::Viewpoint Constraint::Viewpoint Constraint::::List Specified Elements Conformance to Relationship	constraint List Specified Elements Conformance to Relationship[_17_0_2_3_e9f034d_1385071806409_76 with expression, "r('Conform')->size() = 1 " is violated for Test View with no Conform Relationship[_17_0_2_3_e9f034d_1385072333098_34

Table 3.24.

Name of Element for test
Test View with no Conform Relationship

3.7.6. How Do I Make Custom Content in a Column (OCL Table Expression)

Figure 3.56. Table Expression Use Cases

Built into docgen and specifically the rapid table framework there are many different options for querying the model. Not every single possibility in the metamodel is covered by the docgen activities and built in collects, filters, etc. , as a result there is the idea of a <<TableExpressionColumn>>. The table expression column gives more flexibility for querying data out of the model and displaying it in a table.

Figure 3.57. CollectModel

The above diagram is the example model that is used for this specific example.

Figure 3.58. Table Expression

The viewpoint method shows using a <<TableExpressionColumn>> with the 'expression' tag containing an OCL expression to retrieve the qualified Name of an element. The results are shown in the table below. *Note that OCL is it's own language and expressions only work with OCL as the language. In relation to the Docgen User Manual there is a OCL/Analysis User Manual (need Link) In order to figure out all of the different queries you can make for a given element refer to the MagicDraw UML METAMODEL UserGuide. The user guide can be found in the following location: (user custom install directory for magicdraw)/manual/MagicDraw UML MetaModel UserGuide.pdf
<img></img>

Table 3.25.

Full Name of Element
Unit Testing::Viewpoint Operations::CollectModel::Larry

3.8. Advanced Topics

3.8.1. Nested Behaviors

3.8.1.1. Empty Nested Behavior

Figure 3.59. Empty Nested Behavior

Figure 3.60. Viewpoint No Behavior

Figure 3.61. Thing that is supposed to have behavior

Figure 3.62. Thing that is supposed to have behavior collect and filter group

Table 3.26.

Name	Owned Elements	Collect and Filter Group Test
Example	Example	Example
Some Block	Some Block	Some Block
Result 2	Result 2	Result 2
Result	Result	Result
PayloadCamera	PayloadCamera	PayloadCamera
Specific Owner	Specific Owner	Specific Owner
Specific	Specific	Specific
Specific Owner 2	Specific Owner 2	Specific Owner 2
Specific 2	Specific 2	Specific 2
CandF	CandF	CandF
Transclusion	Transclusion	Transclusion

Section 4. How Do I Publish my Model and Document Online (MMS and View Editor)?

4.1. How Do I Upload My Model?

New for Release 0.1.0 Bender is the ["Model Management System (MMS)":name:_17_0_2_3_8660276_1391030682972_633268_64200] which allows the user to export the entire document to the Alfresco server. Uploading the Model to Alfresco is the first step in getting a model ready for View Editor. For a video walk-through of this feature click here.

First, the project will need to be configured for the MMS. To do this the "Data" model at the top of the containment tree will need to be stereotyped <<ModelManagementSystem>>.

To select the server and site to upload the model information to, the user will need to edit two tagged values within "Data". These values are, "site" and "url"

In general all production models should point the url tag to "https://europaems/alfresco/service". Any test models should be sent to "https://sheldon/alfresco/service. **All other url's are deprecated as of 0.1.0 Bender.**

NOTE: When using a release candidate all transactions are forced to the Europa Site on Sheldon, this allows users to test with production models without updating the MMS tagged values.

In order to initialize the model first the user will need to log in to the MMS,

This will open the MMS client login dialog,

Once logged in right click on 'Data' and select 'MMS -> Validate Model':

??
<img></img>

If the project is not already on the Alfresco Server the validation report should return with "The thing you're trying to validate or get wasn't found on the server, see validation window". :

Click 'OK' and check the validation window:

To send the model to the server right click on the rule violation in the validation screen and select "Initialize Project and Model". The system will offer the option to upload the items in the background *(recommended)*:

WARNING: If you click 'NO' MagicDraw will remain unresponsive during export (up to 2+ hours).

A quick rule of thumb is that 1000 model elements will take approximately 1 min. Note that once uploaded it may take an additional 30 min or more for the server to index all the newly updated elements. Once all elements are uploaded the user can validate the contents of the model against the server by right clicking on any element and clicking MMS->Validate Model.

This will recursively validate all model elements owned by the selected element against their counterparts on the Alfresco server. Validating the entire model will take approximately 10ms per element (approx. 5min for 30k elements) Once completed, any discrepancies or changes between MagicDraw and the server will be reported for each element in the validation window

NOTE: If you do not have permission to add models or edit that specific model on the server you will get the following error.

If you feel this server response is incorrect please contact your task manager or the MDev Team.

4.2. How Do I Compare Differences Between My Model and MMS?

4.3. How Do I Publish my Document to View Editor?

You can see a video about how to connect to View Editor here:
<a>Link
(NEEDS UPDATING)

View Editor allows certain properties of document elements to be edited on the web in a sandbox. MagicDraw users are still responsible for importing or exporting the information to View Editor. You can use View Editor to edit names and documentation of any element, create text cross-referencing between elements and views and much more.

NOTE: the process for uploading documents to the View Editor has changed since MDev release 0.1.0 (Bender) all methods used to upload to View Editor previously are deprecated.

Before uploading a document to View Editor first ensure that the document model is up to date in the Alfresco database. To validate the model against the Alfresco database, connect to Alfresco by selecting MMS->Login from the menu bar.

Once logged in, right-click on the package containing the document model and select MMS->Validate Model.

Make any changes or updates as desired from the Validation window that appears. Next, click on the root Class of the document (the model element with <<Document>>) and right-click selecting MMS->Validate View.

If the document is not already on the server the validation window will return with "[EXIST] This View doesn't exist on view editor yet"

To transform the view and upload the information to the view editor, right click on the rule violation in the validation window and select 'Export view hierarchy'. This will recursively upload the document to the server. In versions before 0.1.0 the document would be located in the location specified manually by a tree of library components. Now the document is automatically placed on the view editor based on its location in the model's Package hierarchy.

OLD:

NEW:

Once a user makes changes to an element in the view editor the MagicDraw user can import or reject the changes by running a view validation again. The validation window message field will explain the differences between web and model as well as offer right click options to import the server value (accept) or export the current model value (reject). This can be done on an element by element basis, or in bulk by selecting multiple entries at once.

NOTE: If you do not have permission to add documents or edit that specific model on the server you will get the following error.

If you feel this server response is incorrect please contact your task manager or the MDev Team.

4.4. How Do I Compare Differences Between My Document Model and View Editor?

Once you have an existing structure on View Editor the view hierarchy can be changed by validating the <<Document>> level view hierarchy. For example adding a new view:

Right Click on the <<Document>> Class and select "MMS->Validate View Hierarchy"

if the view was not originally exported to the server the system will return,

This indicates that the element linked to the view hierarchy does not exist on the database, click "OK" and the system will return the following in the validation window:

First, right click on the "[EXIST}" error and select "Export view" (This will also add the element and its relation to the model database).

Then, right click on "[Hierarchy]" and select Export View Hierarchy.

The new view and order should be now on the server. To move a view or reorder the view hierarchy, again start by "MMS->Validate View Hierarchy" from the <<Document>> Class. In this case 'Test View' was moved from the ["Model Management System (MMS)":name:_17_0_2_3_8660276_1391030682972_633268_64200] view to the ["Release Highlights":name:_17_0_2_3_8660276_1389735498069_824045_64152]view. The resulting validation results are,

The first error is due to the fact that the ["Release Highlights":name:_17_0_2_3_8660276_1389735498069_824045_64152]view exposes ["Test View":name:_17_0_2_3_8660276_1391118569544_803397_64283] 's new parent ["New Feature Descriptions":name:_17_0_2_3_8660276_1389735543642_93601_64264]. Unlike the previous operation these items can be remedied in any order by right clicking and executing the fix option.

<img></img>

Section 5. How Do I Publish Ofline and Read-Only Documents (DocGen and DocWeb)?

5.1. What is Docgen?

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 document is an example 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.

5.2. How Do I Install Docgen?

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.

IMPORTANT : 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:

a. If you want to be on the bleeding edge of development and don't mind if things are subject to change you can download the preview/release versions here: LINK (update)

1. Get the latest SSCAE MagicDraw prebuilt distribution. **The SSCAE download is preferred** unless you're told otherwise by an MDK developer or someone else who knows exactly what they are doing.
2. If you have the SSCAE prebuilt, there's no need to do the following steps. If you have a 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 select the SysML Extensions.mdxml file (this should be in the md.install/profiles/MDK directory)
3. Now you can use the stereotypes from Document Profile, inside DocGen>MDK EMP Client

5.3. How Do I Use the Docgen Stylesheet?

5.4. How Do I Publish my Document to DocWeb?

5.5. How Do I Generate My Document Locally?

Section 6. How Do I Use the OCL Rules Engine?

6.1. How Do I Get Started Using OCL?

6.1.1. What Is OCL and How Do I Use It?

OCL Primer By Bradly Clement

The
<a>Object Constraint Language (OCL)
is a text language (a subset of
<a>QVT
) that can be used to customize views and viewpoints in various ways that otherwise may require writing external code in Jython, QVT, Java, etc. You can use OCL to specify how to collect, filter, sort, constrain, and present model data.

OCL syntax can be tricky since it accesses model information through the UML metamodel (see the UML metamodel manual in your Magicdraw installation directory, under manual). Here we try to give some tips on how to write OCL expressions to more easily customize views without having to write external scripts in Jython, Java, QVT, etc.

OCL Resources

For help with OCL, you can try the
<a>OCL Cheat Sheet
or
<a>OCL Helper
or search the web for example OCL. Some example will also be given in the next two sections below.

NoMagic's UML metamodel specifies what objects can be referenced in an OCL expression for the different UML types. A pdf manual of the metamodel can be found in your MagicDraw installation in the manual folder. If you use Eclipse, try opening the Metamodel Explorer view from the menu: Window -> Show View -> Other. There may be many metamodels, but look for the com.nomagic.uml2 metamodel. There is a button on the view where you can search for a type, but be careful since there may be more than one UML metamodel. You want MagicDraw's UML metamodel. There is also a button for showing inherited members that can be referenced in OCL.

Viewpoint Elements

<<**CollectByExpression**>> Collect elements using an OCL expression. This works like other collect stereotypes.

<<**FilterByExpression**>> Filter a collection using an OCL expression. This works like other filter stereotypes.

<<**SortByExpression**>> Sort a collection using an OCL expression. This works like other sort stereotypes.

<<**Constraint**>> You can add this stereotype to a comment or an action in an activity diagram to evaluate an OCL expression on the action's results. The expression should return true or false. If false, the constraint is violated, and the violation will be added to the validation results panel. A constraint comment can be anchored to multiple actions to apply to all of the actions' results separately.

<<**TableExpressionColumn**>> Apply an OCL expression to the target elements. This can be used to chain operations on elements and relatonships that would be otherwise be difficult or impossible.

<<**CustomTable**>> A CustomTable allows columns to all be specified as OCL expressions in one viewpoint element. Target elements each have a row in the table. Title, headings, and captions are specified as with other tables.

6.1.2. How Do I Test OCL Queries (OCL Query Tool)?

Test out your OCL expressions in MagicDraw by

1. selecting some model elements in a diagram or the containment tree,
2. finding the MDK menu, and
3. selecting "Run OCL Query."
4. You may need to resize the popup window to see the entry fields.
5. Enter an expression in OCL, hit OK, and see the result (or error).

There are two fields, an entry field where queries can be inputted and a result field that publishes the results of the query. The popup has a drop-down that allows you to re-evaluate the past 20 expressions. Potential "command completion" choices are listed in the messages window.

NOTE All other windows will be non-interactive until the window is closed.

6.1.3. What are the OCL Black Box Expressions?

Some functions were added to OCL as shorthand for some common operations.

- m('text') gets the member whose name or type is 'text'

- **m(), member(), or members()** returns the owned elements

- m('likes').oclAsType(Dependency).target

- r('likes') gets all of the relationships of name or type 'likes'
- r('likes').oclAsType(Dependency).target

- t('atypical') returns the type, atypical, for input with a type f that name

- **r(), relationship(), or relationships()** returns all relationships owned or for which the element is a target
- **n(), name(), or names()** returns the name
- **t(), type(), or types()** returns the type. t() and types() return all types (Stereotypes, metaclass, and Java classes and interfaces).
- **s(), stereotype(), or stereotypes()** returns the Stereotypes. This is basically short for appliedStereotypeInstance.classifier. s() and stereotypes() should return all stereotypes, and stereotype() should just return one.
- **e(), evaluate(), or eval()** evaluates an ocl expression retrieved from an element
- **value()** returns the value of a property or slot
- **owners()** returns owner and owner's owners, recursively
- **log()** prints to Message Window in choice of color
- **run(View/Viewpoint)** runs DocGen on a single View or Viewpoint with specified inputs and get the result

The following Model is used to demonstrate the various capabilities of the black box expressions.

Figure 6.1. Black Box Model

6.1.3.1. Relationships "r()"

- r('likes') gets all of the relationships of name or type 'likes'
- r('likes').oclAsType(Dependency).target

- **r(), relationship(), or relationships()** returns all relationships owned or for which the element is a target

Table 6.1.

Name of element	Likes Relationships	Likes Target
@justinbieber	likes	@ladygaga

6.1.3.2. Names "n()"

- **n(), name(), or names()** returns the name

For example "n()" on @justinbieber returns:

Table 6.2.

Name of Element
@justinbieber

6.1.3.3. Stereotypes "s()"

- **s(), stereotype(), or stereotypes()** returns the Stereotypes. This is basically short for appliedStereotypeInstance.classifier. s() and stereotypes() should return all stereotypes, and stereotype() should just return one.

For example @justinbieber is stereotyped <<Twitter Feed>>

Table 6.3.

Name of element	Stereotype of Element
@justinbieber	Twitter Feed

6.1.3.4. Members "m()"

- m('text') gets the member whose name or type is 'text'
- **m(), member(), or members()** returns the owned elements
- m('likes').oclAsType(Dependency).target

Table 6.4.

Name of element	Attributes	Attribute Default Value
@justinbieber	hairstyle	beiberhair

6.1.3.5. Types "t()"

Table 6.5.

Name of element	Type of Element
@justinbieber	interface com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class class com.nomagic.uml2.ext.magicdraw.compositestructures.mdports.impl.Enc class com.nomagic.uml2.ext.magicdraw.classes.mdkernel.impl.ClassImpl Twitter Feed

6.1.3.6. Validation

Table 6.6.

Name of element	Owned elements
@justinbieber	hairstyle

6.1.3.7. Evaluate "eval()"

Evaluates an ocl expression retrieved from an element

This is the constrained element with the constraint specification:
<img></img>

This is eval() taking in the constrained element and evaluation the constraint:
<img></img>

This is the output of the complete method:

Table 6.7.

Component Name	CBE Mass	Rollup Number
Propulsion Subsystem	30	30
Hydrazine Tank	10	
Thruster	20	

6.1.3.8. owners()

Returns owner and owner's owners, recursively
<img></img>
<img></img>

Table 6.8.

Owners
Black Box Model
Black Box Examples
OCL Constraint and Expression Testing
Unit Testing
Data

6.1.3.9. log()

6.1.3.10. run(View/Viewpoint)

6.1.3.11. value()

6.1.4. OCL Examples

6.2. How Do I Create OCL Rules?

6.2.1. How Do I Create Rules on Specific Model Elements?

Figure 6.2. How Do I Create Rules on Specific Model Elements?

6.2.2. How Do I Create Rules Within Viewpoints?

6.2.3. How Do I Validate OCL Rules in my Model?

There are two methods for validating OCL queries in a model, via a right click and via the MD Validation Window. Creation of validation rules is discussed here .

To validate OCL rules using the right click menu, select the package containing the elements to be validated and select MDK -> Validate Constraints

6.3. How Do I Use OCL Expressions in a Viewpoint?

6.3.1. Using Collect/Filter/Sort by Expression

6.3.1.1. CollectByExpression

Figure 6.3. CollectByAssociation

Figure 6.4. CollectByExpression (Basic)

Table 6.9.

Element Name	Owner	Stereotypes
Curly	Larry	PartProperty
	Larry	
	Larry	

6.3.1.2. FilterByExpression

Filter by expression is a versatile viewpoint method that allows the user to filter items using an OCL expression. For more information on OCL and other query langauges please refer to this section

Figure 6.5. FilterByExpression

As usual, when a package is exposed the first action of the viewpoint is to <<CollectOwnedElements>> which will further expose the collect model. The proper use of the FilterByExpression block requires an OCL string that is a complete Boolean expression. In this example, elements of Type 'Property' are selected using: "oclIsTypeOf(Property) = true". The result statement is required for the filter to operate properly.

Figure 6.6. FilterByExpression

As depicted in the viewpoint, the result should be a table that prints all the properties that are owned by the Collect and Filter Model:

Table 6.10.

Element Name	Owner	Stereotypes
Curly	Larry	PartProperty
	Larry	
nyuh nyuh nyuh	Moe	
base_Element	ThirdStooge	
Brother	ThirdStooge	

6.3.1.3. Sort By Expression

Using an OCL expression can also result in doing a numbered or alphabetical sort, however, using OCL gives the capability to sort on many different properties that aren't reachable via the typical attribute and property sorts.

Figure 6.7. Sorting example

This diagram shows the method used.

Sort by name expression

- Bolt
- Car
- Screws
- Steering Wheel
- Tire

6.3.2. How Do I Make Custom Content in a Column (OCL Table Expression)

Figure 6.8. Table Expression Use Cases

Built into docgen and specifically the rapid table framework there are many different options for querying the model. Not every single possibility in the metamodel is covered by the docgen activities and built in collects,

filters, etc. , as a result there is the idea of a <<TableExpressionColumn>>. The table expression column gives more flexibility for querying data out of the model and displaying it in a table.

Figure 6.9. CollectModel

The above diagram is the example model that is used for this specific example.

Figure 6.10. Table Expression

The viewpoint method shows using a <<TableExpressionColumn>> with the 'expression' tag containing an OCL expression to retrieve the qualified Name of an element. The results are shown in the table below. *Note that OCL is it's own language and expressions only work with OCL as the language. In relation to the Docgen User Manual there is a OCL/Analysis User Manual (need Link) In order to figure out all of the different queries you can make for a given element refer to the MagicDraw UML METAMODEL UserGuide. The user guide can be found in the following location: (user custom install directory for magicdraw)/manual/MagicDraw UML MetaModel UserGuide.pdf
<img></img>

Table 6.11.

Full Name of Element
Unit Testing::Viewpoint Operations::CollectModel::Larry

6.3.3. Advanced Topics

6.3.3.1. Use of the Iterate Flag

Figure 6.11. Use of the Iterate Flag

Figure 6.12. Viewpoint Pattern Summary

Figure 6.13. Test View Collection

Table 6.12.

Package Name	Documentation	Old Count	Count Test
Viewpoint Pattern Summary			0

6.4. How Do I Create Viewpoint Expressions?

Figure 6.14. How Do I Create Viewpoint Expressions?

Figure 6.15. Viewpoint Constraint

Table 6.13. List Specified Elements Conformance to Relationship Summary

Validation Rule	Description	Severity	Violations Count
List Specified Elements Conformance to Relationship	Viewpoint Constraint	WARNING	1

Table 6.14. List Specified Elements Conformance to Relationship Detail

Validation Rule	Element	Description
List Specified Elements Conformance to Relationship	EMS & Architecture Training Material::MDK Manual::Resources::Docgen::Docgen Example Viewpoints::Viewpoint Constraint::Viewpoint Constraint::::List Specified Elements Conformance to Relationship	constraint List Specified Elements Conformance to Relationship[_17_0_2_3_e9f034d_1385071806409_76 with expression, "r('Conform')->size() = 1 " is violated for Test View with no Conform Relationship[_17_0_2_3_e9f034d_1385072333098_34

Table 6.15.

Name of Element for test
Requirement Name!!

6.5. How Do I Create Expression Libraries?

6.6. How Do I Use RegEx In my Queries?

http://www.vogella.com/tutorials/JavaRegularExpressions/article.html http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html

6.7. How Do I Create Transclusions with OCL Queries?

Figure 6.16. How Do I Create Transclusions with OCL Queries?

Figure 6.17. Transclusion Test

Table 6.16.

Name	Documentation
Transcluded Element	This is the documentation of the Transcluded Element

Table 6.17.

Name (from constructed Transclusion)	Documentation (from constructed Transclusion)

Section 7. How Do I Perform Black Magic?

7.1. 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 7.1. Editable Table UserScript Use Cases

In the diagram below you see the required stereotype specialization needed to make an editable table work correctly. You also see the method for running the view that uses the View Interaction menu option. When you run this the custom userscript code name is shown. This allows a user in magicdraw to view a table that is editable. Finally, the code for the table is shown.

Figure 7.2. EditableTableSetup

When using a userscript in a viewpoint you create and action and stereotype it the name of the code. Also there is a strict naming convention for the stereotype you create so that the code can be found. Here the naming convention is that the first name is the name of the package that is found in the Docgen UserScripts Folder and then the rest of the convention is the userscript specific name. Shown is the packing structure found in the magicdraw install. Notice that there is a package called test and a script file called TestTable2.py
<img></img>
img_1392067997148

Figure 7.3. Making Editable Table

The table below shows the result of the user script code. Note that both the columns are editable as specified in the userscript.

Result:

Table 7.1. Documents and Their ID

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

7.2. How Do I Make 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 7.4. Validation Userscripts

Right clicking on a view that has a validation userscript inside allows for the View Query Actions menu item to be activated. When you select this menu item the name of code relevant to that viewpoint will appear and the user can execute a validation rule check. Below also shows the Validation Script Results. Noting any elements that are warnings and errors. The validation rules themselves as well as the severity of validation rules can be set in the actual code.
<img></img>
<img></img>

7.2.1. How Do I Make a Jython Validation Script?

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 7.5. Jython UserScripts

Figure 7.6. Sample Jython Validation Script

Like UserScripts or EditableTables, a stereotype can be created that corresponds to the path under DocGenUserScripts directory, then specialize ValidationScript under the Document Profile 3. (refer to Editable Table UserScripts)

Figure 7.7. Validation UserScripts

'Regular' output shows up first.

Table 7.2. TestSuite Summary

Validation Rule	Description	Severity	Violations Count
Rule 1	Stuff with names	WARNING	25
Rule 2	Stuff that's Packages	ERROR	14

Table 7.3. TestSuite Detail

Validation Rule	Element	Description
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::DocGen 3 Manual	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Introduction	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3	Has Name

Validation Rule	Element	Description
	Documentation::DocGen 3 Manual::Installation	
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Using DocGen 3	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Transforming Docbook	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Editable Table UserScripts	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::DocGen 2 to DocGen 3	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Test.TestTable	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Test.DocGenTest	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Test.TestTable2	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Validation UserScripts	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Test.ValidationTest	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Test.CF	Has Name

Validation Rule	Element	Description
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Source Material	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Getting Document on DocWeb	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Getting Document on ViewEditor	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Test.qvttest	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Test.qvtvalidationtest	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Test.DocGenTest2	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::FAQ	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::MagicDraw API Primer	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::View Class Hierarchy	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::OCL Expression Primer	Has Name

Validation Rule	Element	Description
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::gov.nasa.jpl.test.JavaExtensionTest	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Java Extensions	Has Name
Rule 2	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Introduction	Is Package
Rule 2	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Installation	Is Package
Rule 2	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Using DocGen 3	Is Package
Rule 2	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Transforming Docbook	Is Package
Rule 2	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Editable Table UserScripts	Is Package
Rule 2	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::DocGen 2 to DocGen 3	Is Package
Rule 2	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Validation UserScripts	Is Package
Rule 2	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Source Material	Is Package
Rule 2	EMS & Architecture Training Material::zzOld::Docgen	Is Package

Validation Rule	Element	Description
	Manual::DocGen 3 Documentation::DocGen 3 Manual::Getting Document on DocWeb	
Rule 2	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Getting Document on ViewEditor	Is Package
Rule 2	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::FAQ	Is Package
Rule 2	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::MagicDraw API Primer	Is Package
Rule 2	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::View Class Hierarchy	Is Package
Rule 2	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::OCL Expression Primer	Is Package

7.2.2. How Do I Make a QVT validation Script?

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 7.8. Sample QVT Validation Script

Figure 7.9. Test QVT Userscript

'Regular' output shows up first.

Table 7.4. TestSuite Summary

Validation Rule	Description	Severity	Violations Count
Rule 1	Stuff with names	WARNING	25

Validation Rule	Description	Severity	Violations Count
Rule 2	Stuff that's Packages	ERROR	14

Table 7.5. TestSuite Detail

Validation Rule	Element	Description
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Test.ValidationTest	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Test.DocGenTest	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::OCL Expression Primer	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Editable Table UserScripts	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Transforming Docbook	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::MagicDraw API Primer	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::DocGen 3 Manual	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Test.qvtvalidationtest	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Getting Document on DocWeb	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen	Has Name

Validation Rule	Element	Description
	Manual::DocGen 3 Documentation::DocGen 3 Manual::Source Material	
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Getting Document on ViewEditor	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Using DocGen 3	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::DocGen 2 to DocGen 3	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Test.DocGenTest2	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::gov.nasa.jpl.test.JavaExtensionTest	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Introduction	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Test.CF	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Test.TestTable	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Java Extensions	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3	Has Name

Validation Rule	Element	Description
	Documentation::DocGen 3 Manual::View Class Hierarchy	
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Test.TestTable2	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Installation	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Validation UserScripts	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Test.qvttest	Has Name
Rule 1	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::FAQ	Has Name
Rule 2	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Getting Document on DocWeb	Is Package
Rule 2	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Source Material	Is Package
Rule 2	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Getting Document on ViewEditor	Is Package
Rule 2	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::OCL Expression Primer	Is Package
Rule 2	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3	Is Package

Validation Rule	Element	Description
	Documentation::DocGen 3 Manual::Editable Table UserScripts	
Rule 2	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Using DocGen 3	Is Package
Rule 2	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::DocGen 2 to DocGen 3	Is Package
Rule 2	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Transforming Docbook	Is Package
Rule 2	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Introduction	Is Package
Rule 2	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::MagicDraw API Primer	Is Package
Rule 2	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::View Class Hierarchy	Is Package
Rule 2	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Installation	Is Package
Rule 2	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::Validation UserScripts	Is Package
Rule 2	EMS & Architecture Training Material::zzOld::Docgen Manual::DocGen 3 Documentation::DocGen 3 Manual::FAQ	Is Package