MAGICDRAW REPORT WIZARD

version 17.0.1

user guide

No Magic, Inc.

2011

CONTENTS

CONTENTS

CONTENTS

1. MagicDraw Report Wizard Overview

Report Wizard is a report engine for MagicDraw version 14.0 and greater. It is designed to solve several problems of our legacy report engines (XSL/XSLT and JPython).

The Report Wizard report engine is built on top of Velocity Engine (Open Source Templating engine) and is integrated with the MagicDraw application. To make the best of Report Wizard, you need to understand the Report Wizard UI, the Velocity Template Language (VTL), the application's Open API, and the helper modules.

Report Wizard supports text-based templates to generate reports. Each report file format depends on the type of the templates. The type of template files that Report Wizard supports includes plain text, RTF, HTML, Office Open XML (ISO/IEC 29500:2008), OpenDocument format (ISO/IEC 26300), and XML template (DocBook or FO).

All commercial MagicDraw editions will have full use of all features within Report Wizard. The MagicDraw Community edition allows you to generate output with watermarks.

Before generating reports, you need to open Report Wizard.

To open **Report Wizard**:

- On the **Tools** menu, click **Report Wizard**. The **Report Wizard** dialog will appear.

1.1 Report Wizard Dialog

The **Report Wizard** dialog (Figure 1) consists of 2 panes: (1.1.1) Control buttons and (1.1.2) Content Management pane.

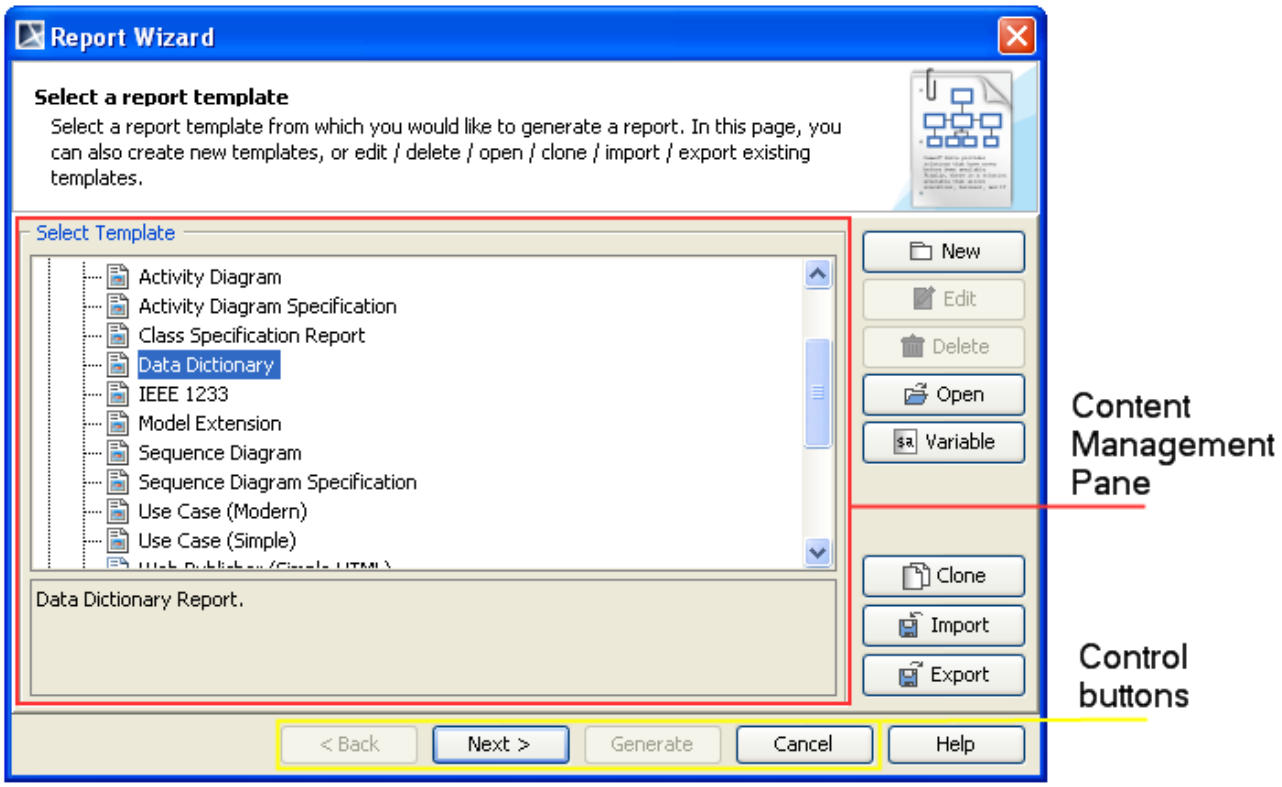


Figure 1 -- Report Wizard Dialog

1.1.1 Control Buttons

There are 4 control buttons:

(i) The **Back** button is used for proceeding to the previous content management pane.
(ii) The **Next** button is used for proceeding to the next content management pane.
(iii) The **Generate** button is used for generating a report.
(iv) The **Cancel** button is used for cancelling the report generation process.

1.1.2 Content Management Pane

This pane is used for managing the template content and includes the following panes:

(1.1.2.1) Template Management pane
(1.1.2.2) Report Data Management pane
(1.1.2.3) Select Element Scope pane
(1.1.2.4) Generate Output pane

Click the **Back** or **Next** button to go to a specific pane.

1.1.2.1 Template Management Pane

This pane consists of the **Select Template** pane (Figure 2).

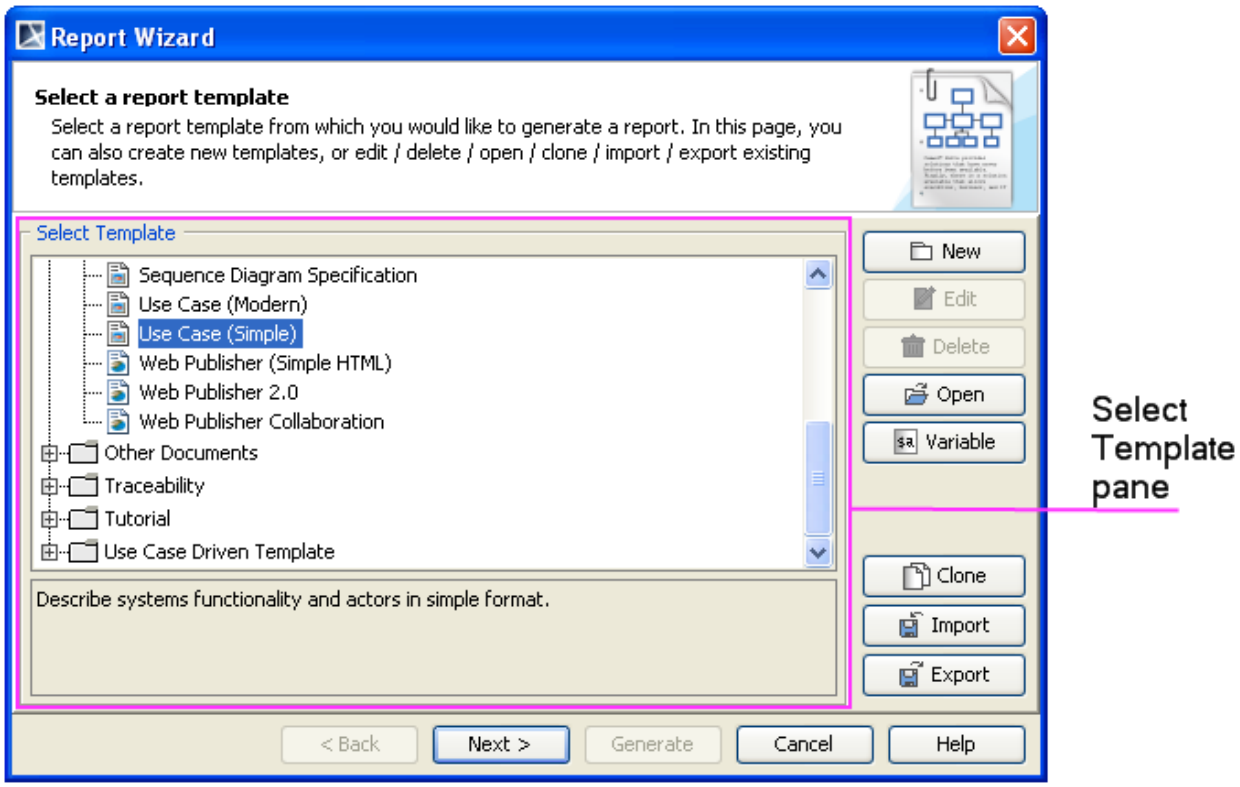


Figure 2 -- Select Template Pane

Report Wizard provides predefined templates such as *Use Case*, *Model Extension, Data Dictionary, IEEE 1233, Class Specification Diagram, Business Process Diagram, and Web Publisher* templates. Choose the relevant template to manage or generate a report.

To select a template:

1. On the **Tools** menu, click **Report Wizard**. The **Report Wizard** dialog will open.
2. In the **Select Template** pane (Figure 2), select a template. A template description will be displayed in the lower part of the **Select Template** pane.

You can manage the template from the **Template Management** pane or click the **Next** button to go to the next step for generating the report.

This pane contains 8 buttons: (a) **New,** (b) **Edit,** (c) **Delete,** (d) **Open,** (e) **Variable,** (f) **Clone,** (g) **Import,** and (h) **Export**.

(a) New button

Click the **New** button (Figure 2) to create a new template.

To create a new template:

1. In the **Report Wizard** dialog, click the **New** button. The **New Template** dialog will open (Figure 3).

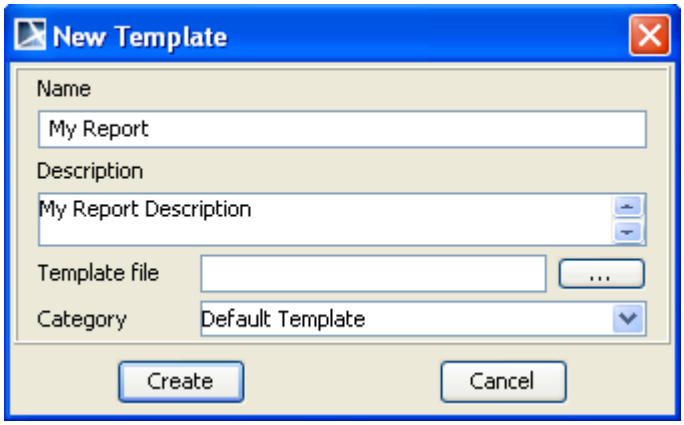


Figure 3 -- New Template Dialog

2. Enter the template name, description, and location of the new template in the **New Template** dialog.

Table 1 -- New Template Dialog Fields and Buttons

Field Name	Description	Default Value	Possible Value	Required
Name	Enter a new template name.	Blank	Text	Yes
Description	Enter a template description.	Blank	Text	No
Template file	Select an RTF template.	Blank	Text	Yes
Create	Create a new template under the Template tree.	Disable	Disable/ Enable	-
Cancel	Close the dialog.	Enable	-	-
Category	Choose the existing category or enter a new category name.	-	Text	No

3. Click the "**...**" button. The **Select Location** dialog will appear (Figure 4).

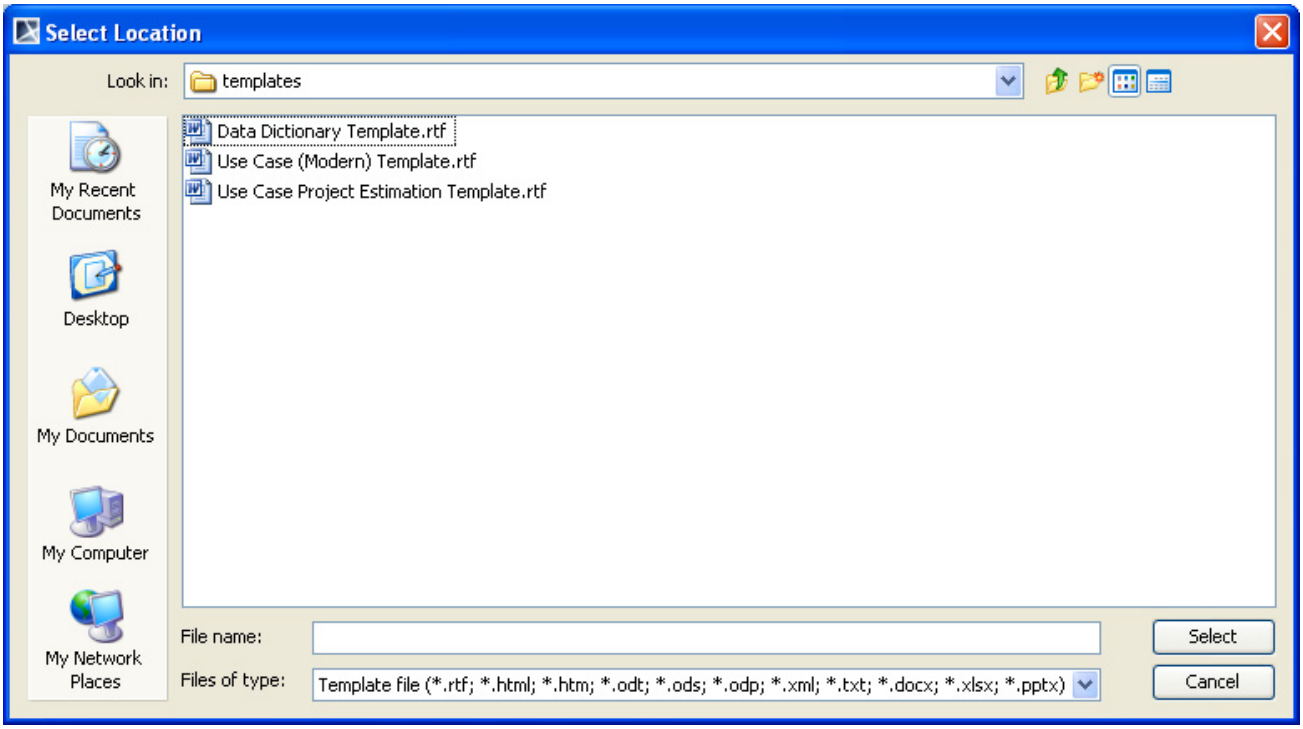


Figure 4 -- Selecting Template File in the Select Location Dialog

4. Select the template file location and type. Enter the filename and click **Select**.

(b) **Edit** button

Click the **Edit** button (Figure 2) to edit a template.

To edit a template:

1. In the **Report Wizard** dialog, select a template and click the **Edit** button. The **Edit Template** dialog will appear (Figure 5).

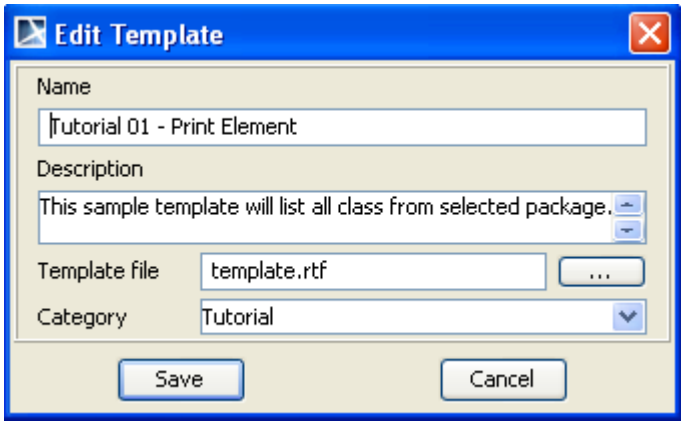


Figure 5 -- Edit Template Dialog

2. Edit the template name and description and locate the template file's location.

Table 2 -- Edit Template Dialog Fields and Buttons

Field Name	Description	Default Value	Possible Value	Required
Name	Edit the template name.	Existing name	Text	Yes
Description	Edit the template description.	Existing description	Text	No
Template file	Change the RTF template.	Existing template file	Text	Yes
Save	Save the edited template under the Template Tree.	Enable	Enable/ Disable	-
Cancel	Close the dialog.	Enable	Enable/ Disable	-
Category	Choose an existing category or enter a new category name.	Existing category	Text	No

(c) **Delete** button

Click the **Delete** button (Figure 2) to delete a template.

To delete a template:

1. In the **Report Wizard** dialog, select a template and click the **Delete** button. The **Confirm delete** dialog will appear (Figure 6).

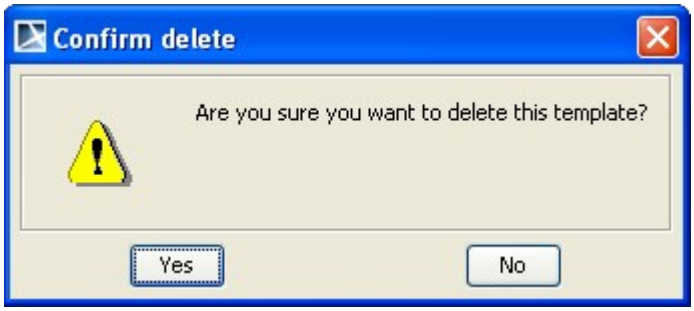


Figure 6 -- Confirm Delete Dialog

2. Click either **Yes** to delete the selected template from the template list or **No** to cancel the operation.

(d) **Open** button

Click the **Open** button (Figure 2) to open a template file in the default editor.

To open a template field in the default editor:

- In the **Report Wizard** dialog, select a template and click the **Open** button. The template file will open in the default editor.

(e) **Variable** button

Click the **Variable** button (Figure 2) in the **Report Wizard** dialog. The Template Variable dialog (Figure 7) will appear, allowing you to create a new template variable, modify or delete an existing template variable, using the **Template Variables** dialog (Figure 7).

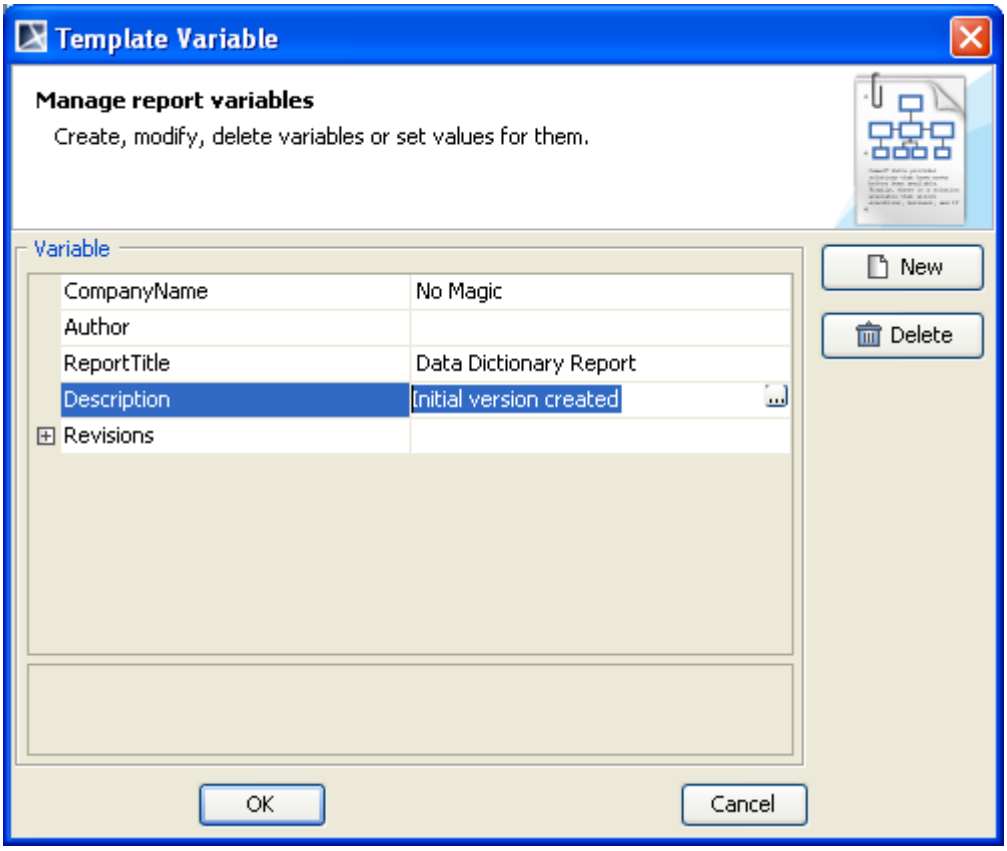


Figure 7 -- Template Variable Dialog

Table 3 -- Variable Dialog Fields and Buttons

Field Name	Function	Default Value	Possible Value	Required
Name	To enter the variable name.	Blank	Text	Yes
Value	To enter the variable description.	Blank	Text	No
New	To create a new template description.	Enable	Enable/ Disable	-
Cancel	To close the dialog.	Enable	Enable/ Disable	-

The **Template Variable** dialog (Figure 7) has the (i) **Variable** pane and (ii) **Control** buttons.

(i) **Variable Pane**

This pane consists of a report description table and the **Variable Value** text box. The first column of the table is the variable name, and the second column is the variable value. You can use the **Variable Value** text box or the second column of the table to view the value of a selected variable. To edit the value, please do so in the second column of the table.

(ii) **Control Buttons**

There are two control buttons: **OK** and **Cancel**.

To create a new variable:

1. In the **Template Variable** dialog, click the **New** button. The **New Variable** dialog will appear (Figure 8).

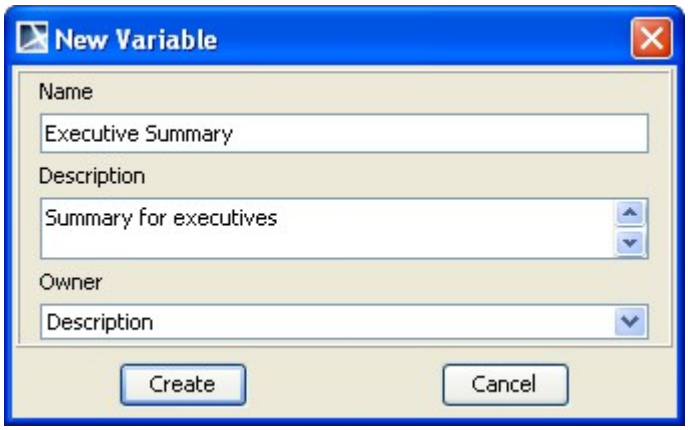


Figure 8 -- New Variable Dialog

2. Enter the variable name and value. Next, select an owner for this new variable from the Owner drop-down list and then click **Create**. You will see the newly-created variable's name and value in the **Variable** pane.
3. In the **Template Variables** dialog, click **OK**.

You can modify a variable value in either the (i) **Variable** pane or (ii) **Variable Value** pane.

(i) To modify a value in the **Variable** pane:

- Click a variable name column in the **Variable** pane and modify the variable value in the table (Figure 9).

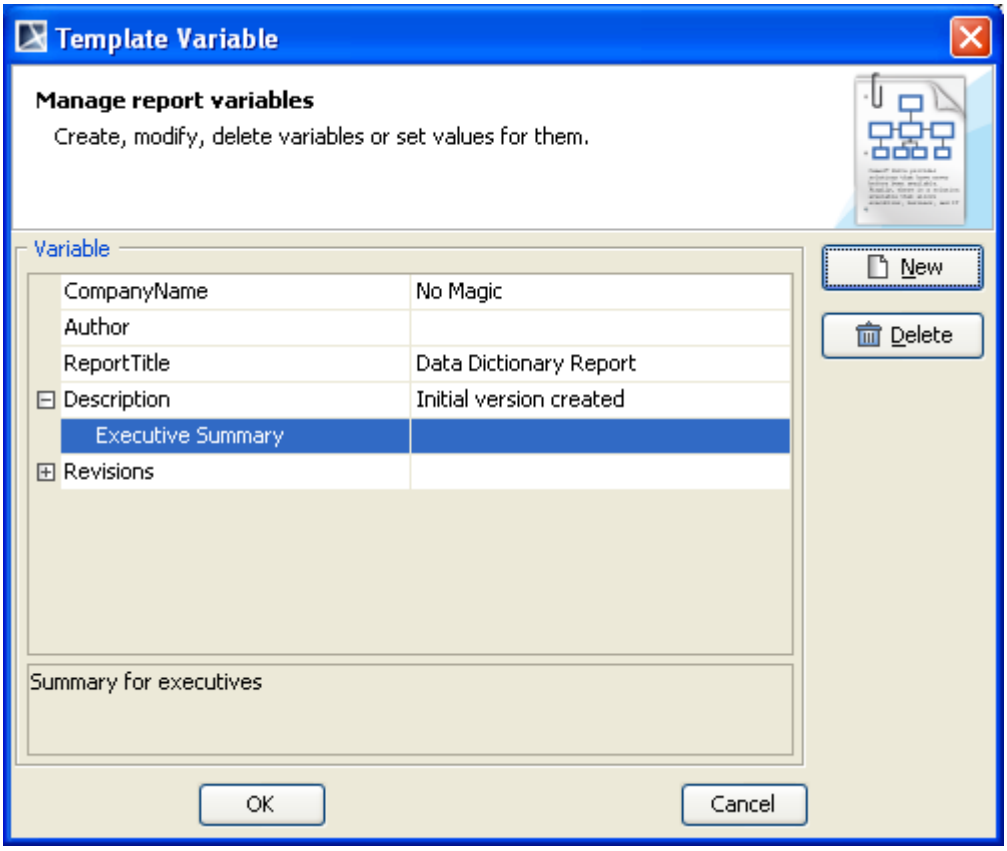


Figure 9 -- Modifying a Variable Value in the Variable Pane

(ii) To modify a variable value in the **Variable Value** pane:

1. Click a variable value column in the table in the **Variable** pane and click the "**...**" button (Figure 10). The **Variable Value** pane will appear.
2. Modify the variable value and click **OK** (Figure 11).

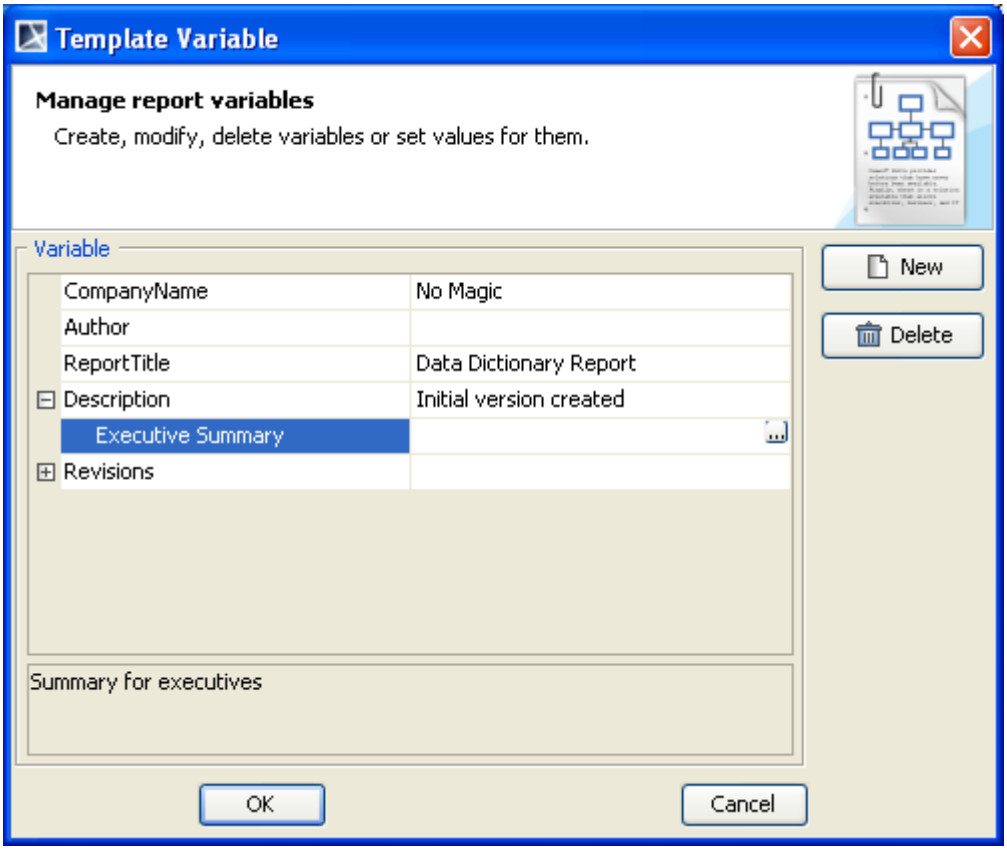


Figure 10 -- Modifying a Variable Value with the "..." Button

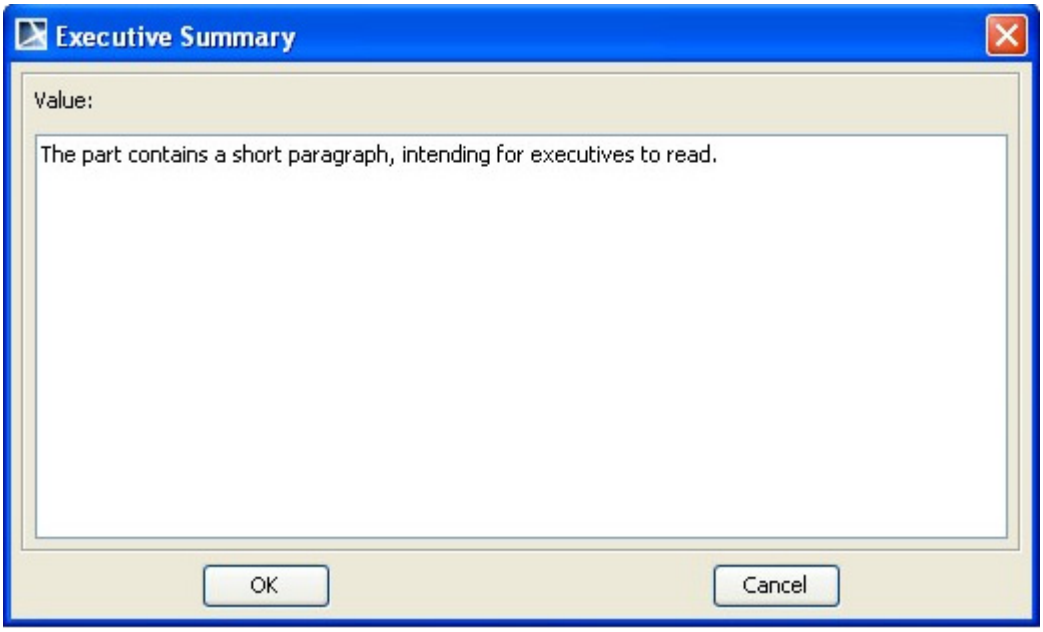


Figure 11 -- Variable Value Pane

3. When you finish modifying variables and their values in the **Template Variable** dialog, click either **OK** to confirm the changes or **Cancel** to discard them.

To delete a variable:

1. Select a variable in the table and click the **Delete** button. The **Question** dialog will appear for confirming the deletion of the variable.
2. Click **Yes** to delete the selected variable (Figure 12).

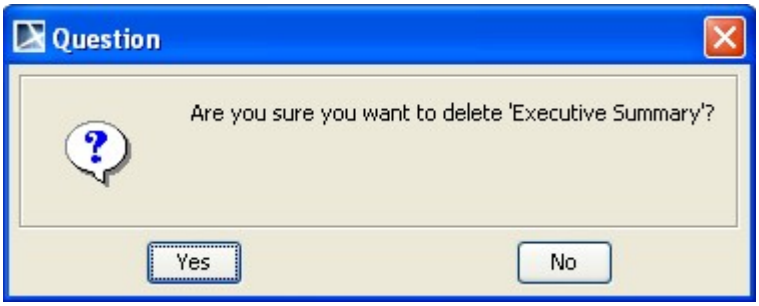


Figure 12 -- Question Dialog To Confirm Deletion

(f) **Clone** button

Click the **Clone** button (Figure 2) to clone a template.

To clone a template:

1. In the **Report Wizard** dialog, select a template and click the **Clone** button. The **Clone Template** dialog will appear (Figure 13).

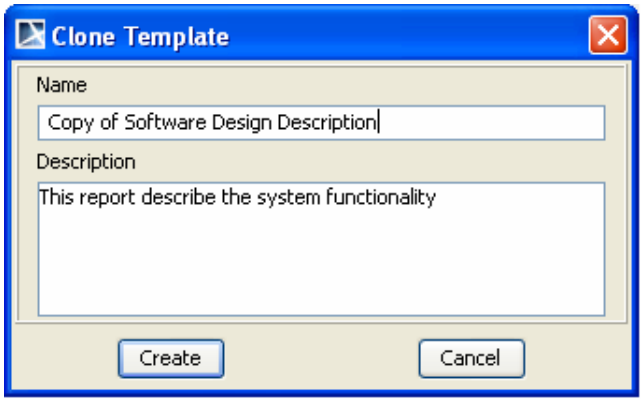


Figure 13 -- Clone Template Dialog

2. Enter the name and description. The name of the cloned template should begin with **Copy of** (the name).
3. Click the **Create** button to clone the template.

Table 4 -- Clone Template Dialog Fields and Buttons

Field Name	Description	Default Value	Possible Value	Required
Name	Enter a new template name.	Copy of the selected template name	Text	Yes
Description	Enter the template description.	Existing description	Text	No
Create	Create a template.	Enable	Enable/ Disable	No
Cancel	Close the dialog.	Enable	Enable/ Disable	No

(g) **Import** button

Click the **Import** button (Figure 2) to import a template.

To import a template:

1. In the **Report Wizard** dialog, click the **Import** button. The **Select Location** dialog will appear (Figure 14).

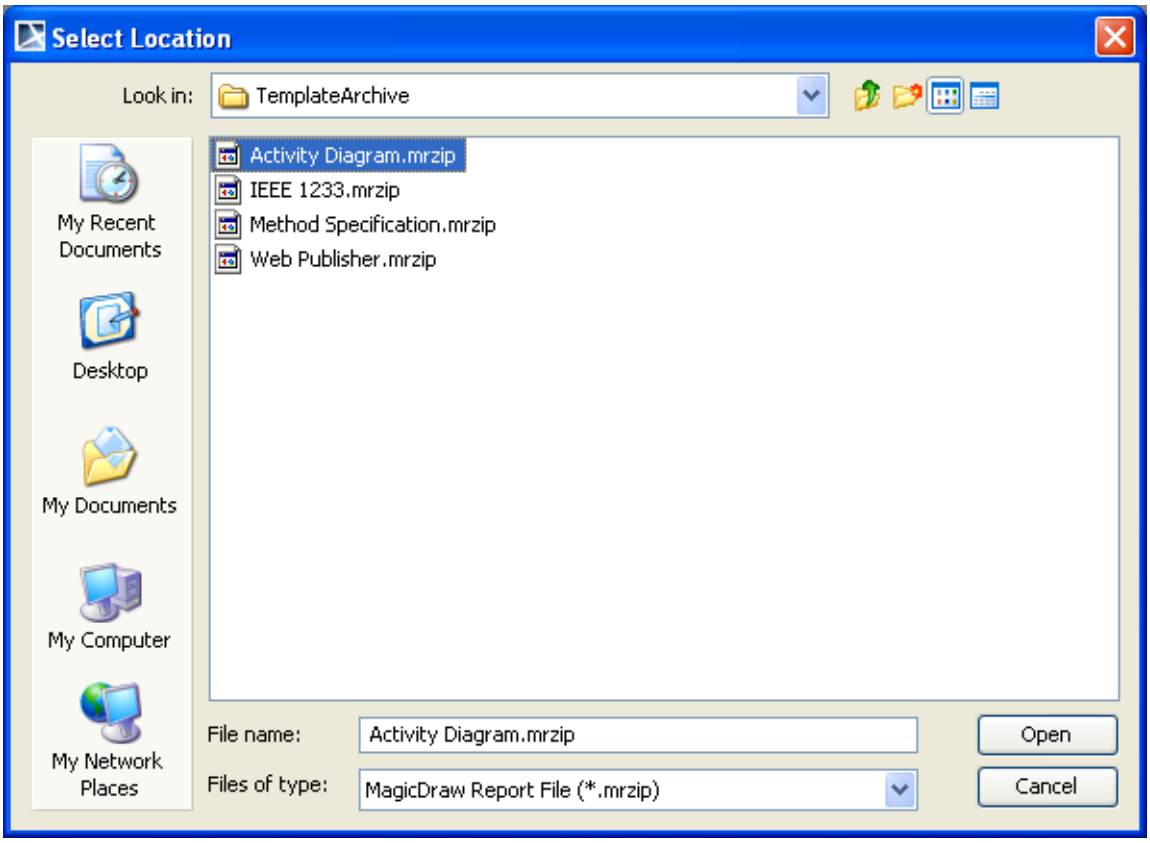


Figure 14 -- Select Location Dialog

2. Select a Report Wizard template with the filename extension ***.mrzip** and click **Open**. The following **Message** dialog will open (Figure 15).

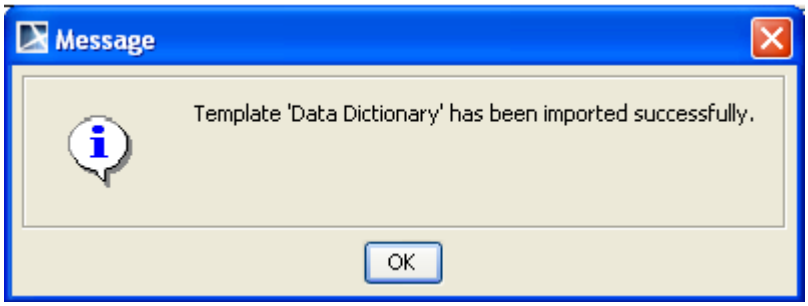


Figure 15 -- Message Dialog of Successful Import

(h) **Export** button

Click the **Export** button (Figure 2) to export a template.

To export a template:

1. In the **Report Wizard** dialog, select a template and click the **Export** button. The **Select Location** dialog will open (Figure 16).

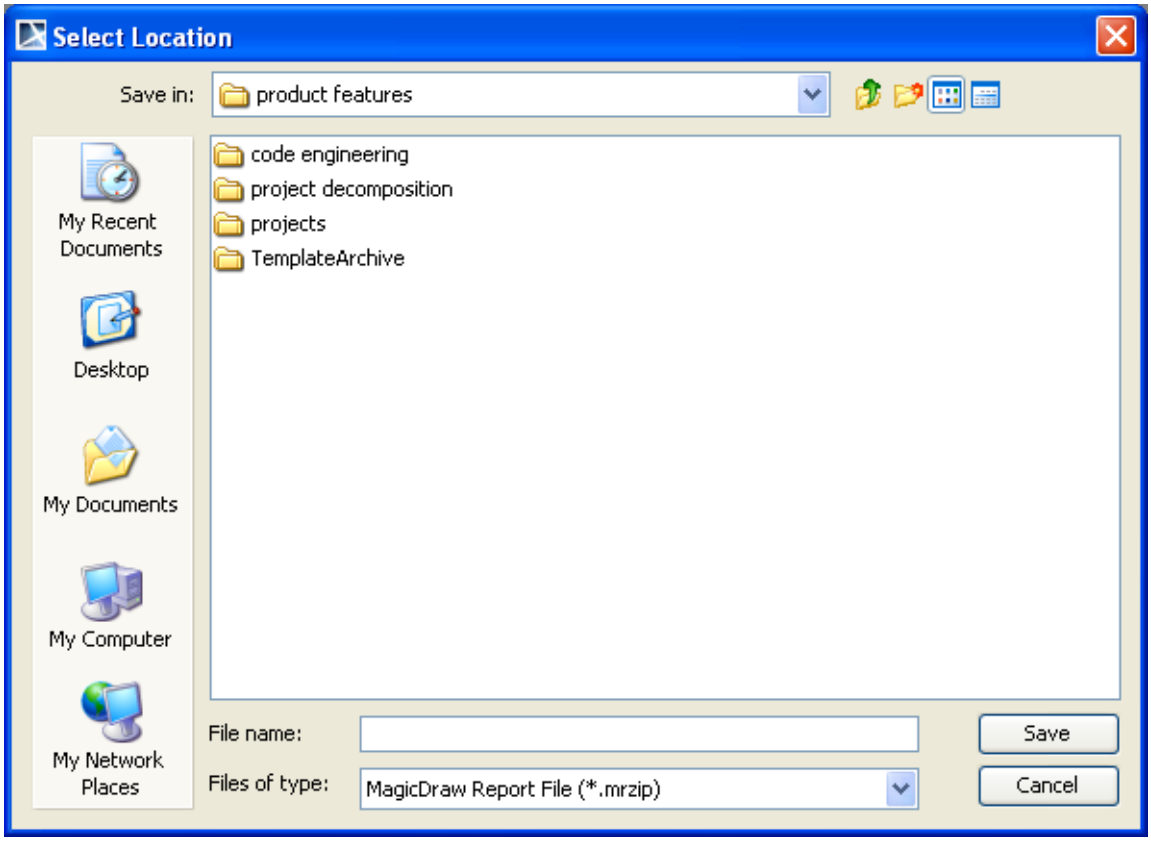


Figure 16 -- Select Location Dialog

2. Select the template destination where to export, type the filename, and click **Save**.
3. The following **Message** dialog will open (Figure 17). Click **OK**.

Figure 17 -- Message Dialog of Successful Export

1.1.2.2 Report Data Management Pane

Report Wizard allows you to (i) create a report data and (ii) organize its variables. You can now create a report data as an element in the containment tree inside a MagicDraw project with the use of profiles, allowing you to commit the report data to Teamwork Server and share it with other users.

You can also create child variables under any variables. This will help you organize information into groups, keep revision history, etc.

(i) Creating a Report Data

A report data is a collection of variables.

(ii) Organizing Its Variables

Variables are created because some information is not “naturally” contained in a model such as the company's name, author's name, revisions, etc. Although you can put all the information in a model's specification (Document/Hyperlinks in the **Specification** dialog), it is very hard and tedious to get data from the model's specification, as this results in more lines of codes and scripts in the report templates. Therefore, instead of writing lines of codes, you can simply create a variable, give it a name, and call it directly from the template. For example, you can create a variable called “Author” in Report Wizard and write `$Author` in the template. The report generated will then show the value inside “Author.” In short, a variable is used to represent data which you would like it to be in the report generated, not in the model.

Figure 18 below shows the **Select Report Data** pane in the **Report Wizard** dialog.

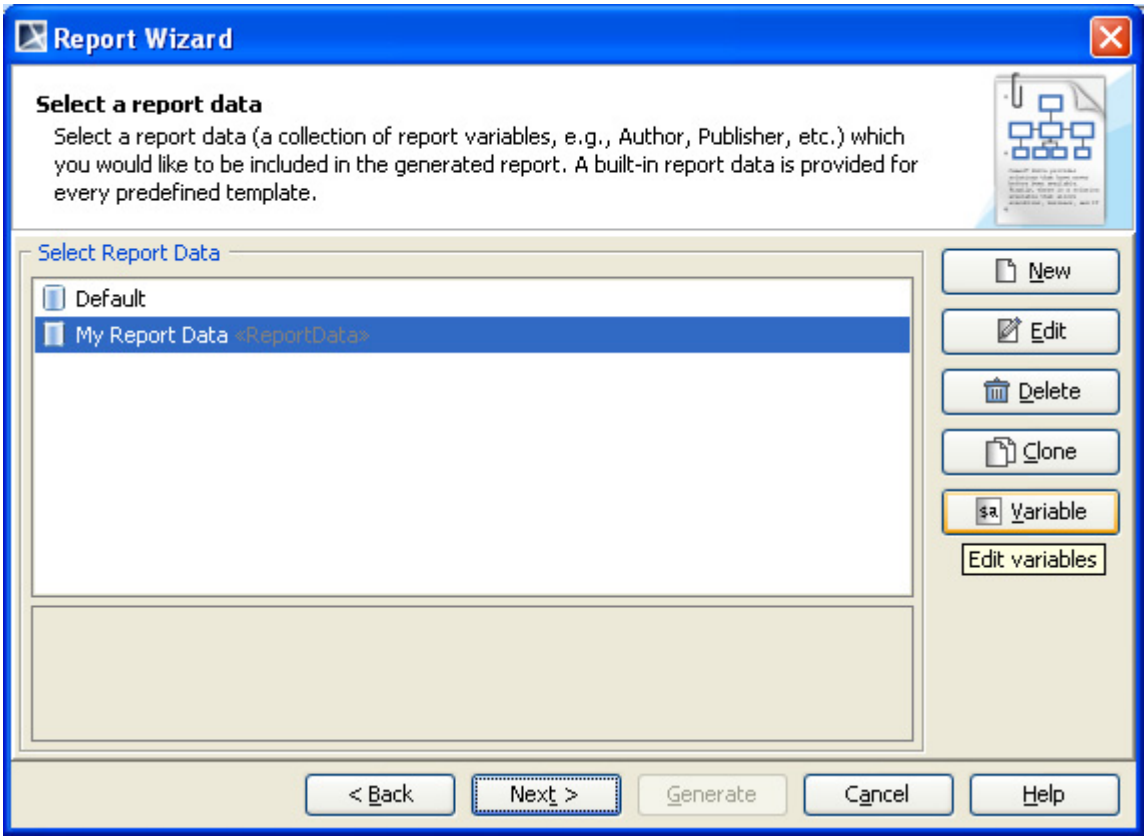


Figure 18 -- Report Data Management Pane

Use the **Report Data Management** pane to select and organize a report data and its variables. Report Wizard provides a built-in report data for every predefined template. A detailed description of the report data will be displayed in the lower part of the **Select Report Data** pane.

You can:

- create, edit, delete, or clone a report data
- use the **Variable** button to organize variables in a report data.

The **Select Report Data** pane (Figure 18) has (a) **New**, (b) **Edit**, (c) **Delete**, (d) **Clone**, and (e) **Variable** buttons.

(a) **New** button

Click the **New** button to create a new report data (Figure 18).

To create a new report:

1. Click the **New** button. The **New Report Data** dialog will appear (Figure 19).

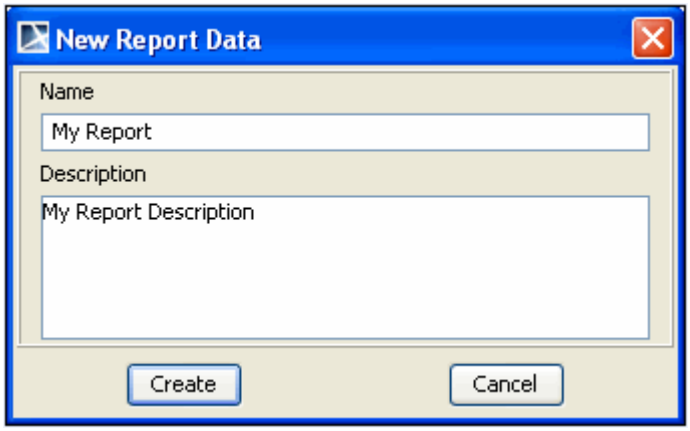


Figure 19 -- New Report Data Dialog

2. Enter the new report's name and description, and then click **Create**.

Table 5 -- New Report Data Fields and Buttons

Field Name	Description	Default Value	Possible Value	Required
Name	Enter the report's name.	Blank	Text	Yes
Description	Enter the report's description.	Blank	Text	No
Create	Create the report.	Disable	Enable/Disable	-
Cancel	Close the dialog.	Enable	-	-

(b) Edit button

Click the **Edit** button to edit a report data (Figure 18).

To edit a report data:

1. Click the **Edit** button. The **Edit Report Data** dialog will appear (Figure 20).

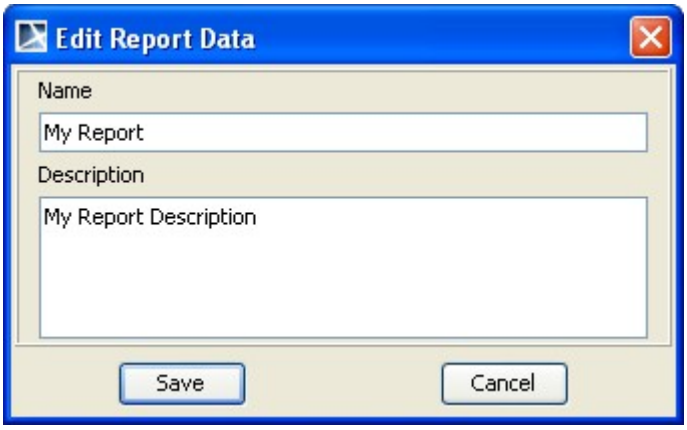


Figure 20 -- Edit Report Data Dialog

2. Edit the report's name and description, and then click **Save**.

Table 6 -- Edit Report Data Dialog Fields and Buttons

Field Name	Description	Default Value	Possible Value	Required
Name	Edit the report's name.	Existing Name	Text	Yes
Description	Edit the report's description.	Existing Description	Text	No
Save	Save the report.	Enable	Enable/Disable	-
Cancel	Close the dialog.	Enable	Enable/Disable	-

(c) **Delete** button

Click the **Delete** button to delete a report data (Figure 18).

To delete a report:

1. Click the **Delete** button. The **Confirm delete** dialog will appear (Figure 21).

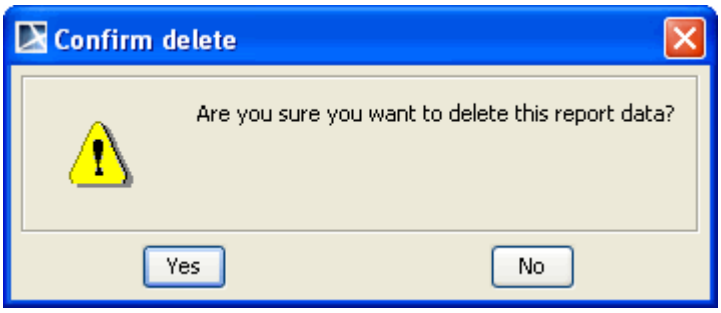


Figure 21 -- Confirm Delete Dialog

2. Click either **Yes** to delete the selected data or **No** to cancel the operation.

(d) **Clone** button

Click the **Clone** button to clone a report data (Figure 18).

To clone a report:

1. Click the **Clone** button. The **Clone Report** dialog will appear (Figure 22).

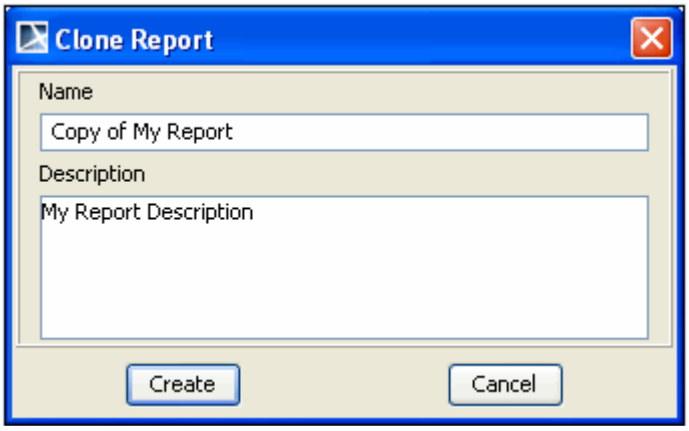


Figure 22 -- Clone Report Dialog

2. Enter the name and description of the report. The name should begin with **Copy of** (cloned report name).
3. Click the **Create** button. The cloned report data will then be created (Figure 23).

Table 7 -- Clone Report Dialog Fields and Buttons

Field Name	Description	Default Value	Possible Value	Required
Name	Enter the report name.	Copy of the selected report name	Text	Yes
Description	Enter the report description.	Existing Description	Text	No
Create	Create the clone report.	Enable	Enable/ Disable	-
Cancel	Close the dialog.	Enable	Enable/ Disable	-

NOTE	Whenever a clone report data is created, a copy of the report data (variable) will also be created.

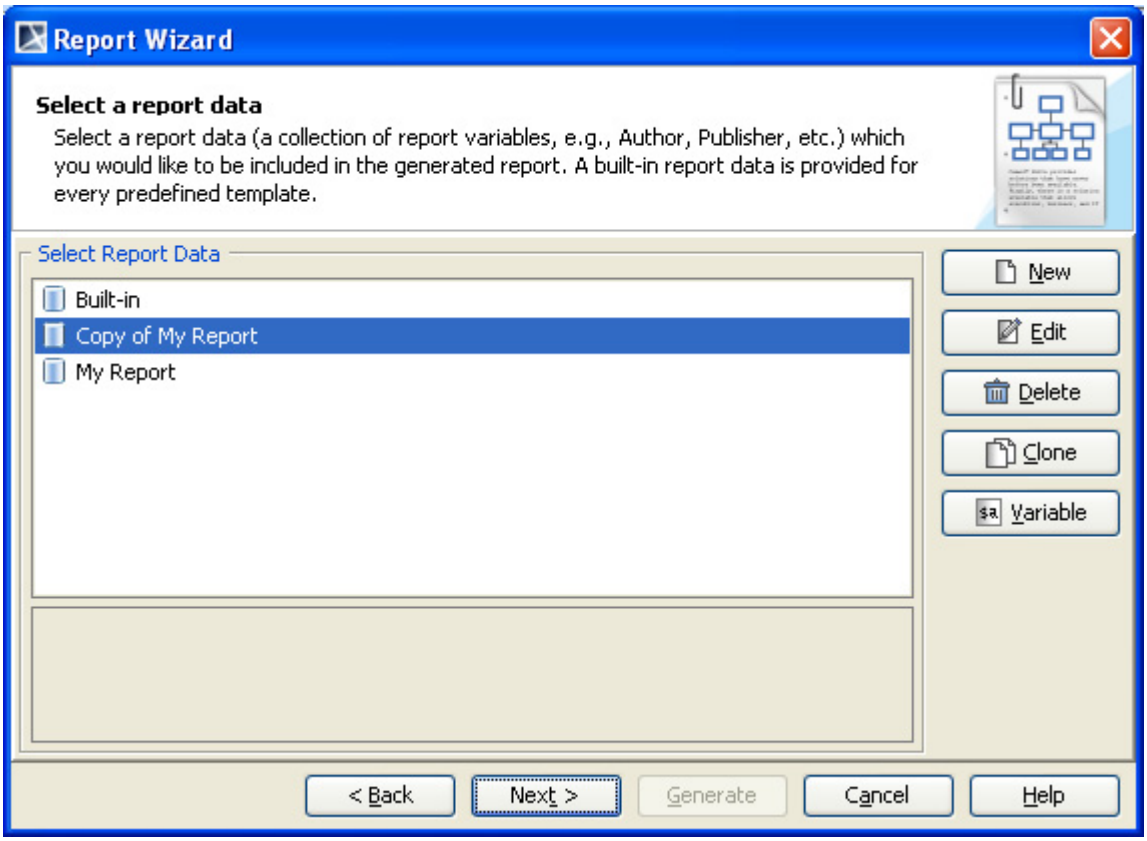


Figure 23 -- Copy of My Report Created in Report Data Management Pane

(e) **Variable** button

Click the **Variable** button to create a new variable (Figure 18).

To create a new variable:

1. Click the **Variable** button. The **New Variable** dialog will then open.
2. Type the variable name and value, and then click **Create**.

Creating Report Data and Its Variables

You can create a report data either in (i) Report Wizard where all data will be saved into an XML file or (ii) a MagicDraw project, in which case the report data will be saved alongside with the project itself. Saving a report data in a project will enable you to commit the report to Teamwork Server.

(i) Creating a Report Data in Report Wizard

To create a report data in Report Wizard:

1. Open the **Select Report Data** pane.
2. Click the **New** button (Figure 24).

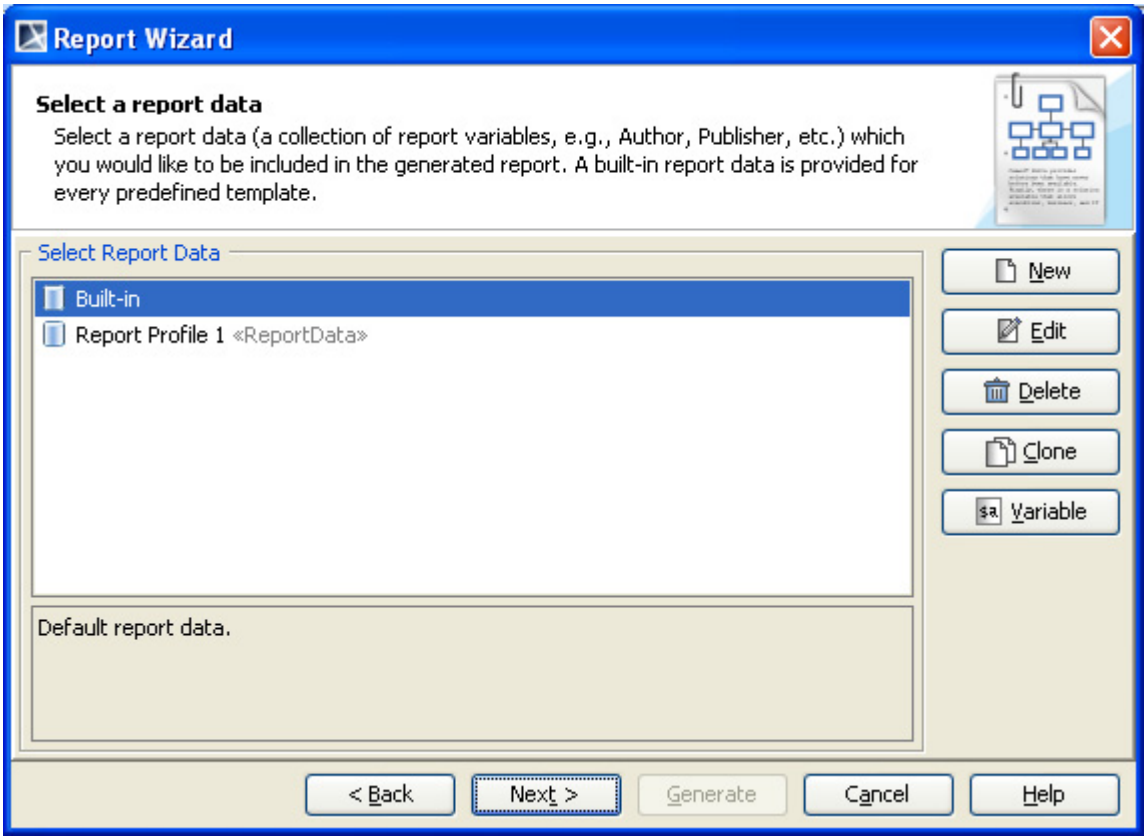


Figure 24 -- Creating Report Data in Report Wizard

3. Enter the new report's name and description in the **New Report Data** dialog (Figure 25).

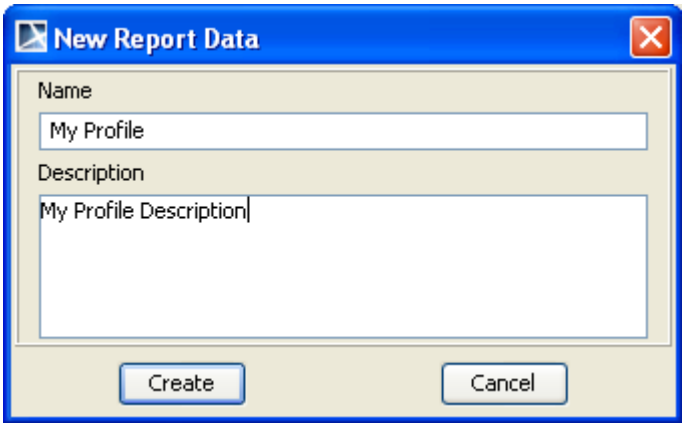


Figure 25 -- Entering Report's Name and Description

4. Click **Create**.

(ii) Creating a Report Data in a MagicDraw Project

Before creating a report data in a MagicDraw project, you need to use a report profile, `Report Profile.mdzip`, which is located in the `<install.root>\profiles` folder.

To use a report profile (Report Profile.mdzip):

1. Click **File** > **Use Module...** on the MagicDraw main menu (Figure 26). The **Use Module** dialog will open (Figure 27).

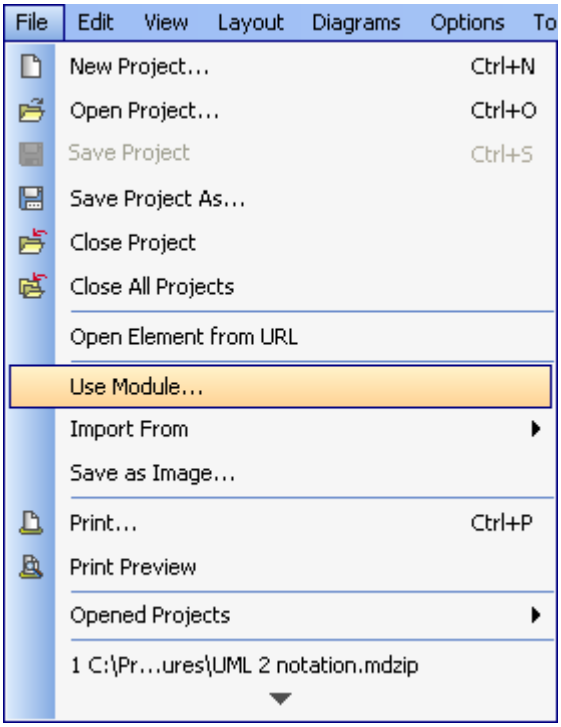


Figure 26 -- Use Module Menu

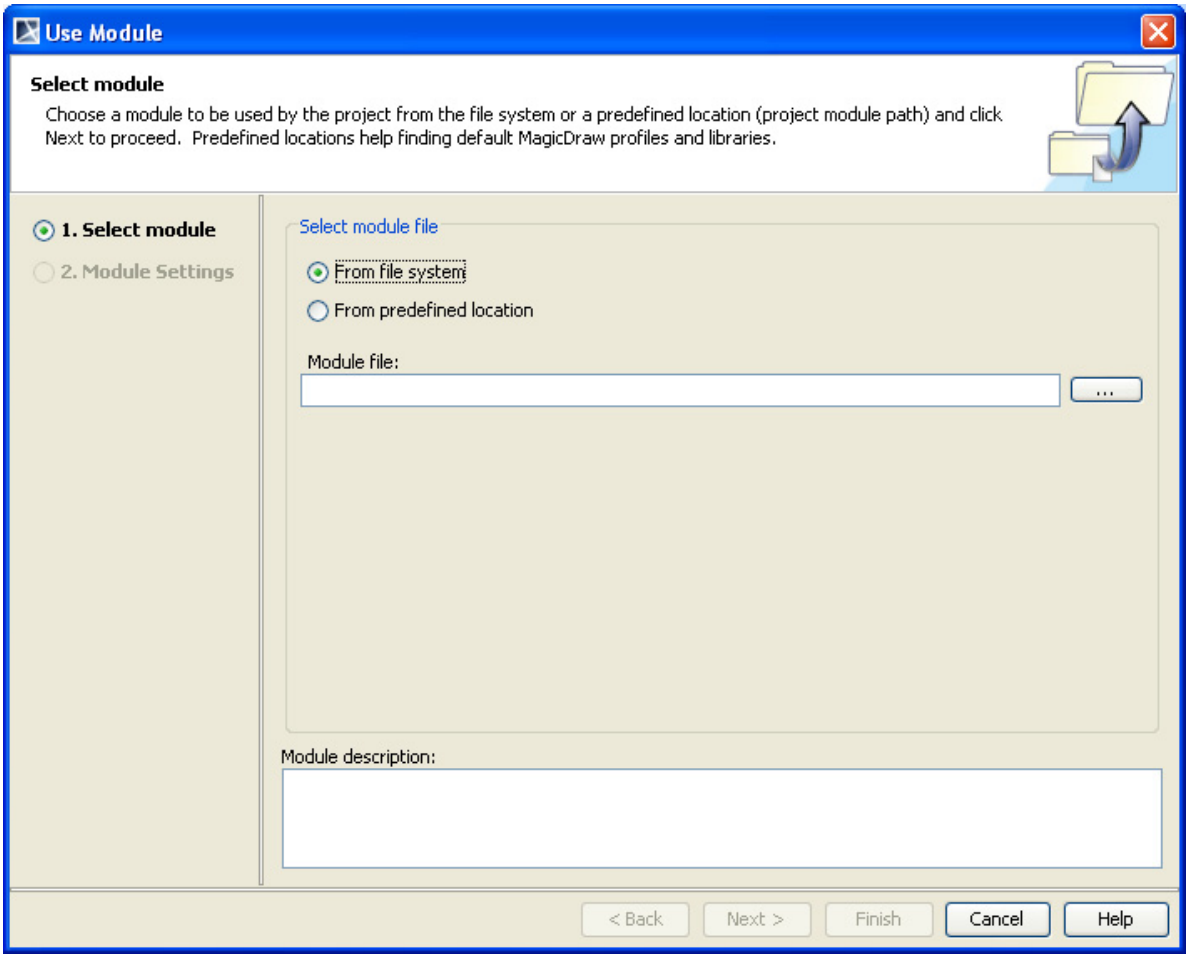


Figure 27 -- Use Module Dialog

2. Click the **From predefined location** button in the **Select module file** pane (Figure 28).

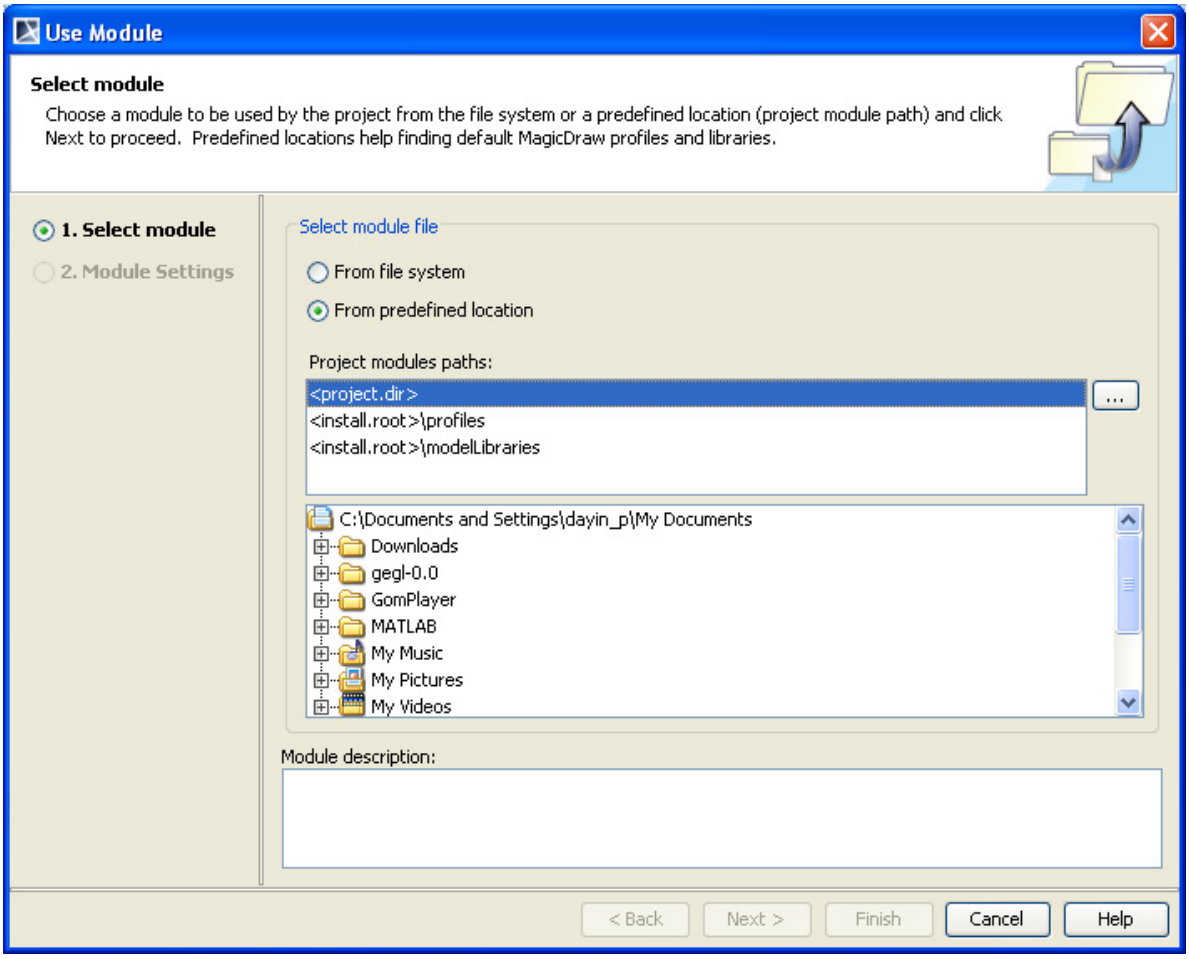


Figure 28 -- Selecting Module File Pane

3. Select **<install.root>\profiles** as the project module path (Figure 29).

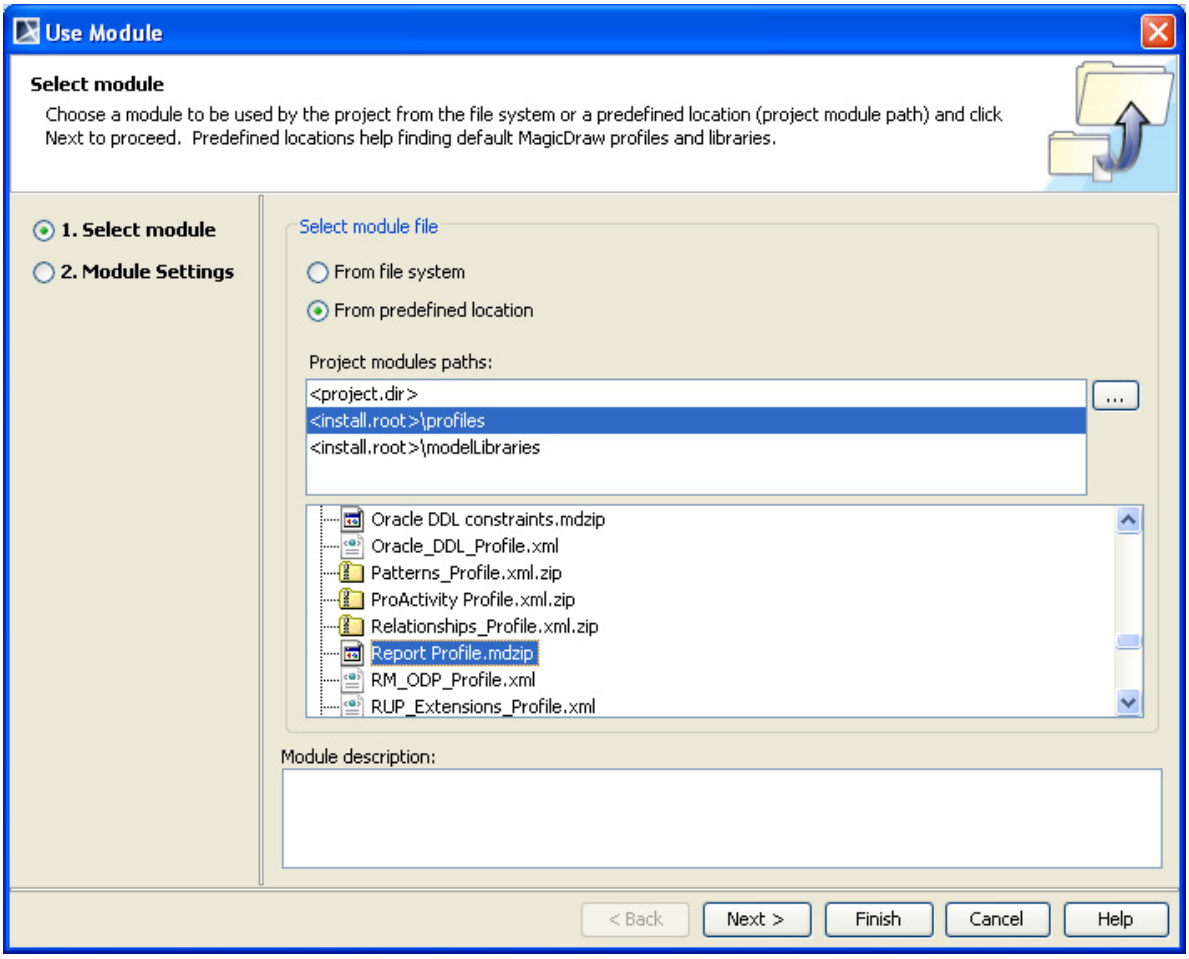


Figure 29 -- Selecting Project Module Path

4. Select **Report Profile.mdzip** and click **Finish**. The “Report Profile” profile will open in the Containment tree as a read-only profile. You can now use it in your project.

To create a report data in a MagicDraw project:

1. Use **Report Profile.mdzip** in your MagicDraw project (see "To use a report profile (Report Profile.mdzip):", on page 31).
2. Right-click a data model in the Containment tree and select **New Element** > **Report Profile** > **Report Data** from the shortcut menu (Figure 30).

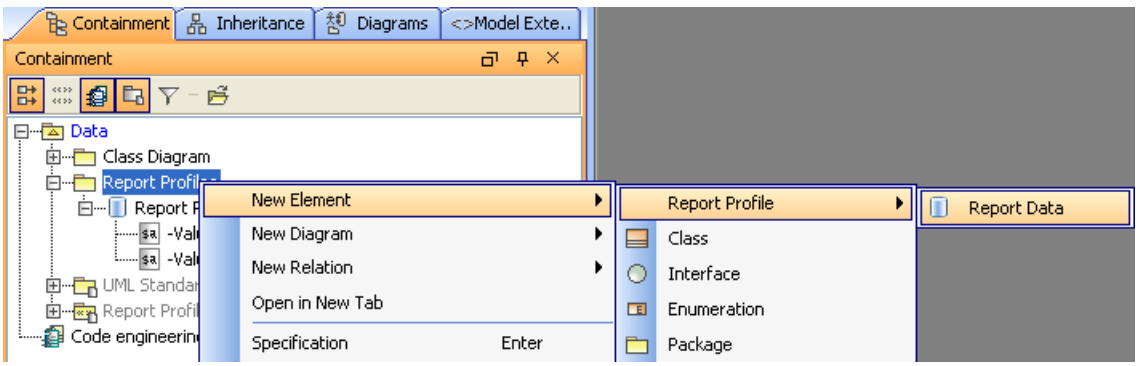


Figure 30 -- Creating Report Data in MagicDraw Project

3. Type the name of the report data element in the Containment tree.

NOTE	You can right-click a data model, package, or profile in the Containment tree to create a report data.

Table 8 -- Tag Values of Report Data

Tag Values	Function
autoImageSize	To change the image size.
imageFormat	To select an image format, either JPEG or PNG.
emptyText	To store a string value that will be replaced with an empty variable.
data	Contain elements that will be published by Report Wizard.
template	To determine which template to use a particular report data.
generateRecursively	To determine whether or not a report will be generated recursively.

Creating Variables for Report Data

Variables contain information that you want to store in a project, such as names and dates.

To create a variable for a report data:

- Right-click a report data in the Containment tree and select **New Element** > **Variable** (Figure 31). The variable will appear inside the report data.

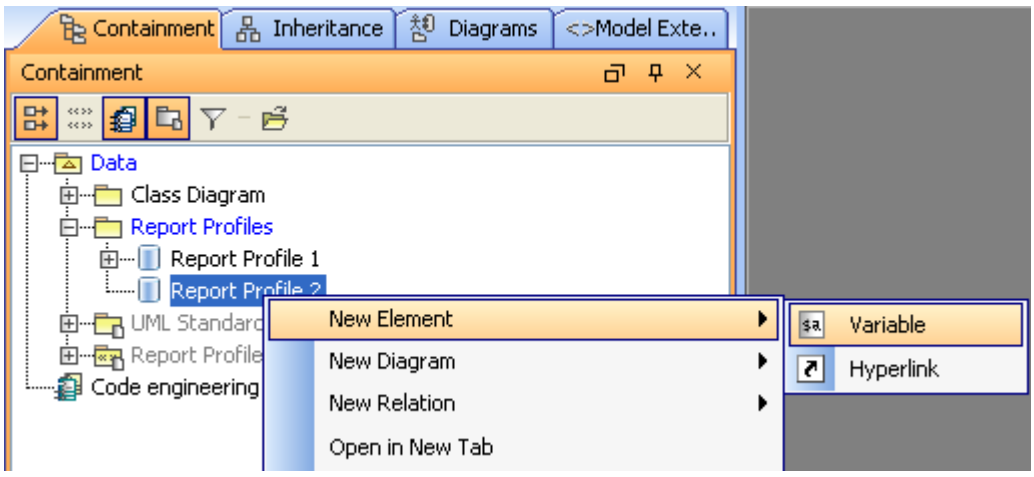


Figure 31 -- Creating Variable for Report Data

Creating, Editing, and Deleting Variables

You can create, edit, and delete variables through the **Report Variable** dialog in Report Wizard.

To open the **Report Variable** dialog:

1. Click **Tools** > **Report Wizard...** on the MagicDraw main menu.
2. Select a report template and click **Next** (Figure 32).

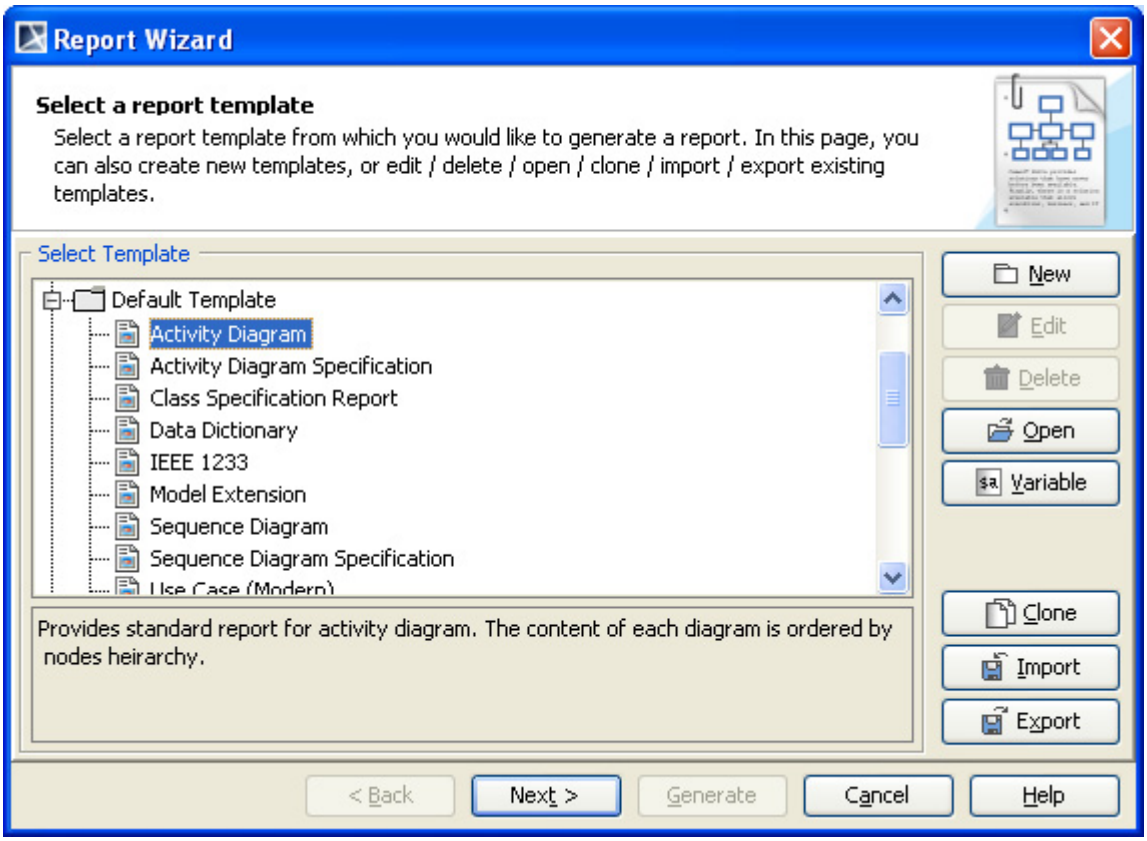


Figure 32 -- Opening Report Wizard Dialog

3. Either create (with the **New** button) or select a report data and click the **Variable** button (Figure 33). The **Report Variable** dialog will open (Figure 34)

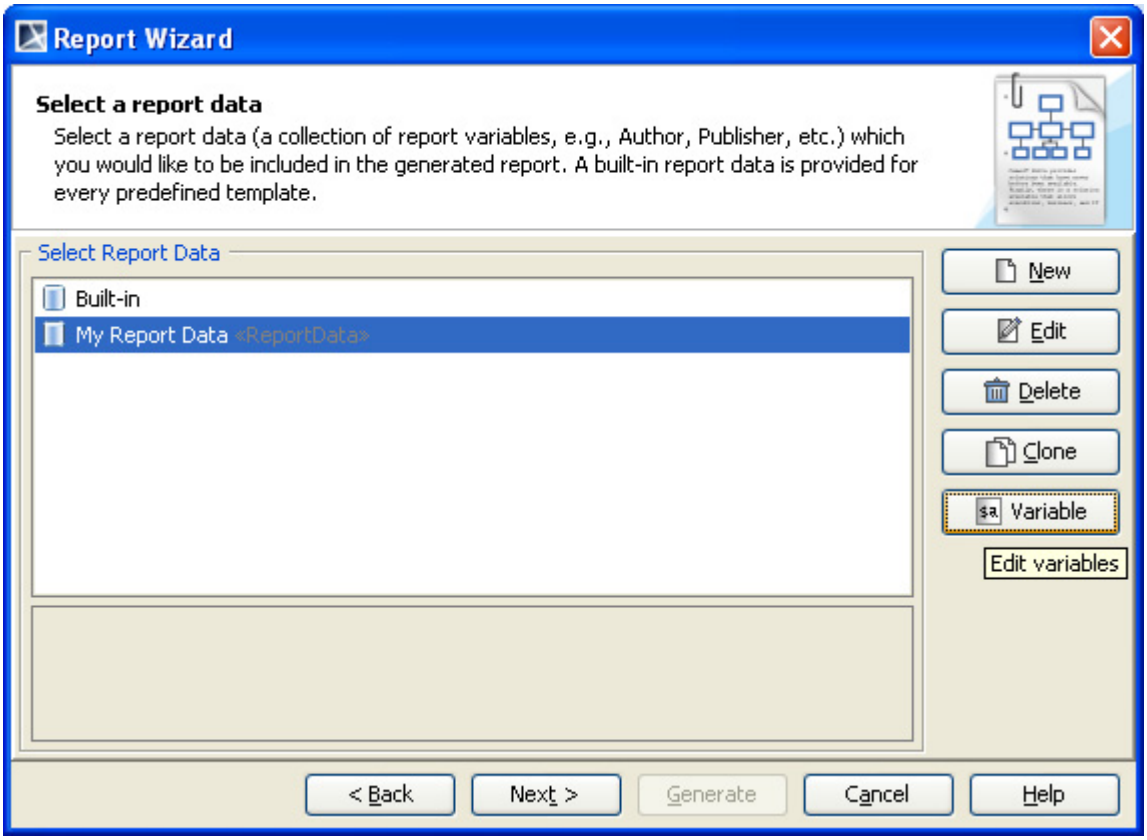


Figure 33 -- Opening Variable Dialog

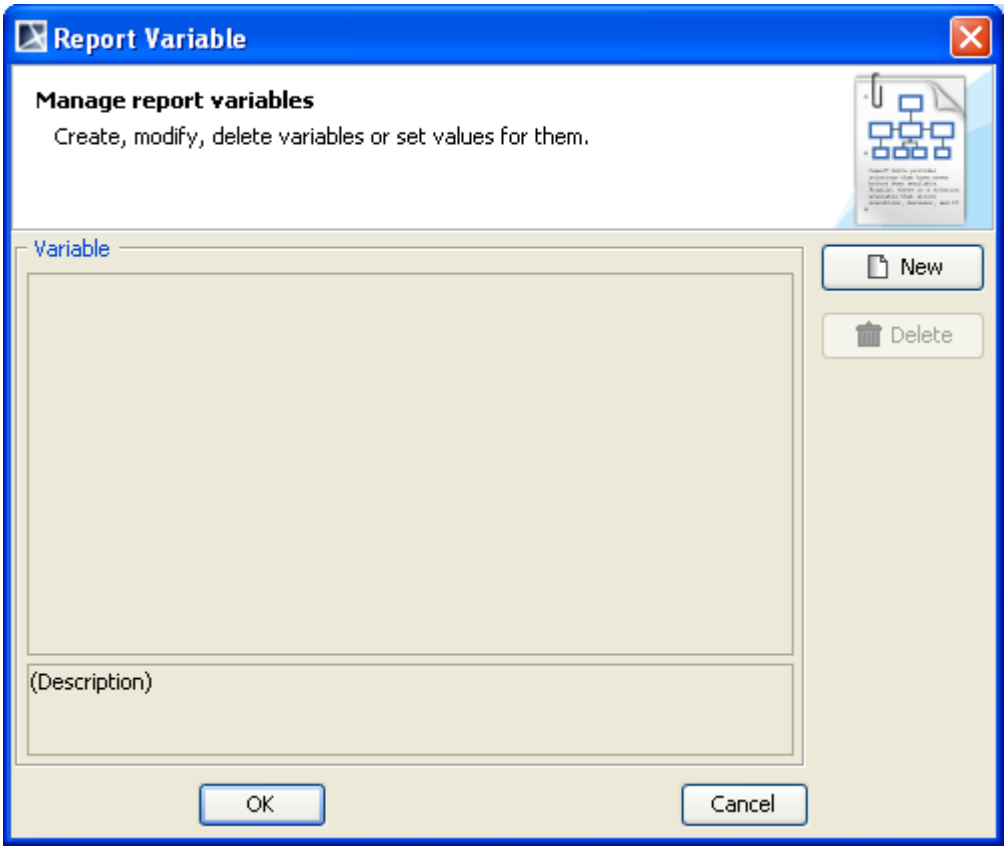


Figure 34 -- Report Variable Dialog

To create a variable in Report Wizard:

1. Open the **Report Variable** dialog (see "To open the Report Variable dialog:", on page 36 above).
2. Click **New**. The **New Variable** dialog will open. Fill in the variable name and description (Figure 35). You can create either (i) a parent variable or (ii) a child variable in the **Owner** box.

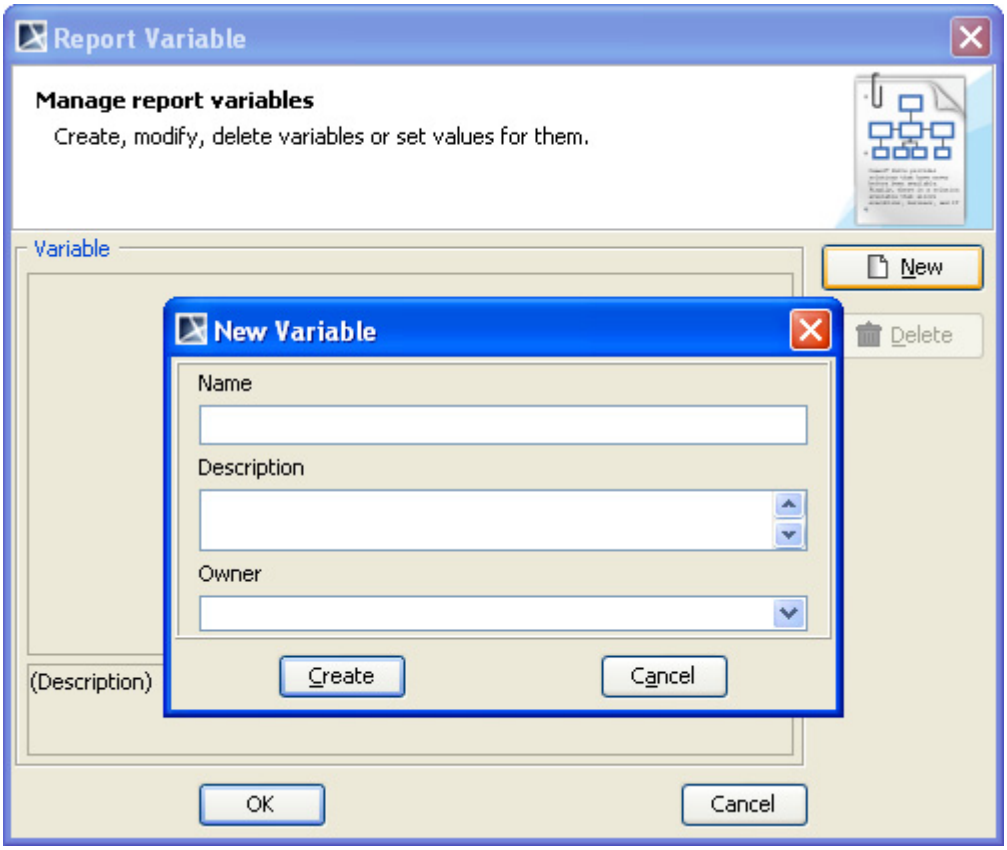


Figure 35 -- New Variable Dialog

(i) To create a parent variable, type the variable name in the **Name** box, enter the description, select an empty value in the **Owner** box, and then click **Create** (Figure 36).

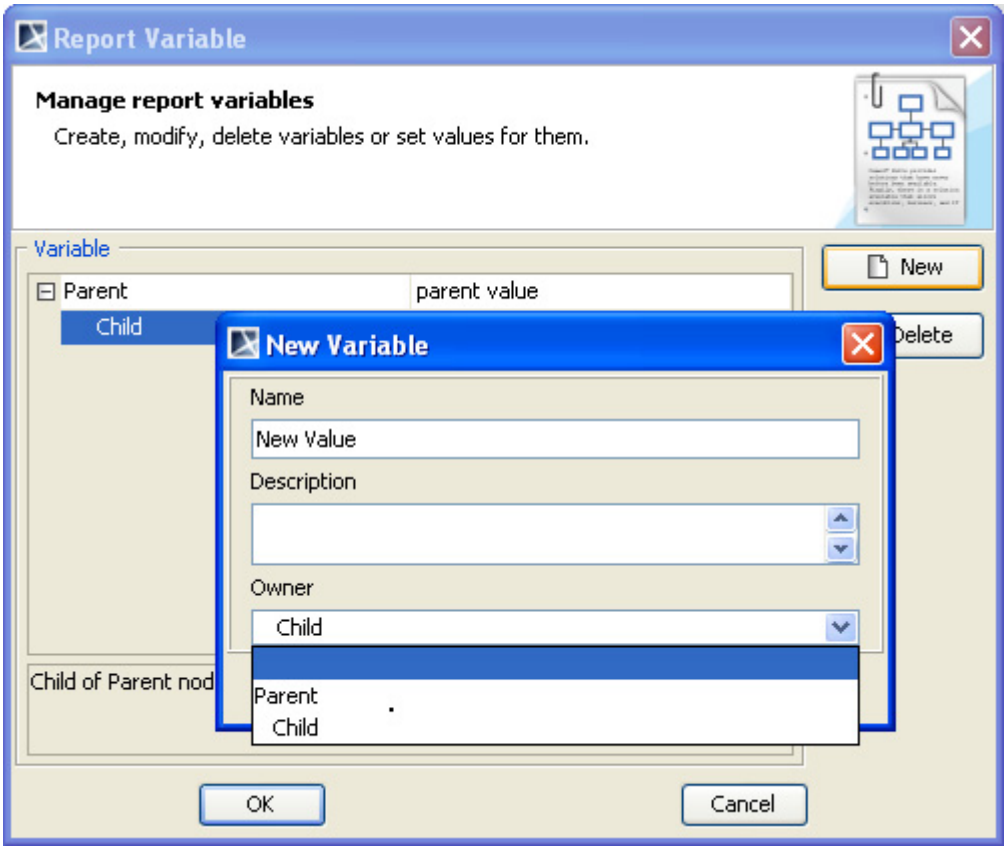


Figure 36 -- Creating a Parent Variable

(ii) To create a child variable, type the variable name in the **Name** box, enter the description, select **Parent** in the **Owner** box, and then click **Create** (Figure 37).

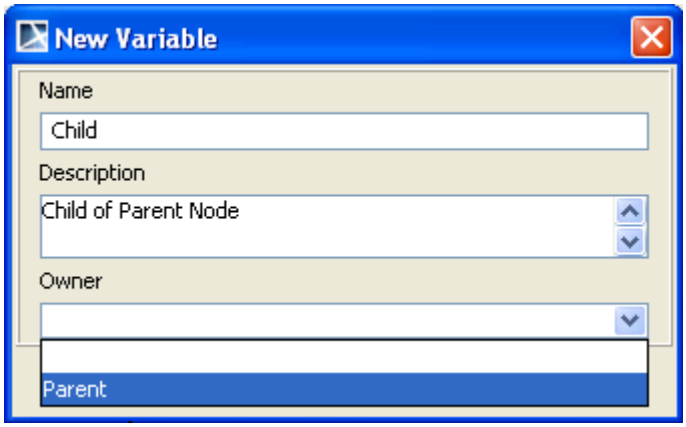


Figure 37 -- Creating a Child Variable

3. The variable will appear in the table in the **Report Variable** dialog.
4. Click **OK** to save the variables in the report data.

To edit a variable in Report Wizard:

1. Open the **Report Variable** dialog (see "To open the Report Variable dialog:", on page 36 above).

2. Double-click the column next to the variable name column and click the button (Figure 38). A dialog will open for you to edit the variable value.

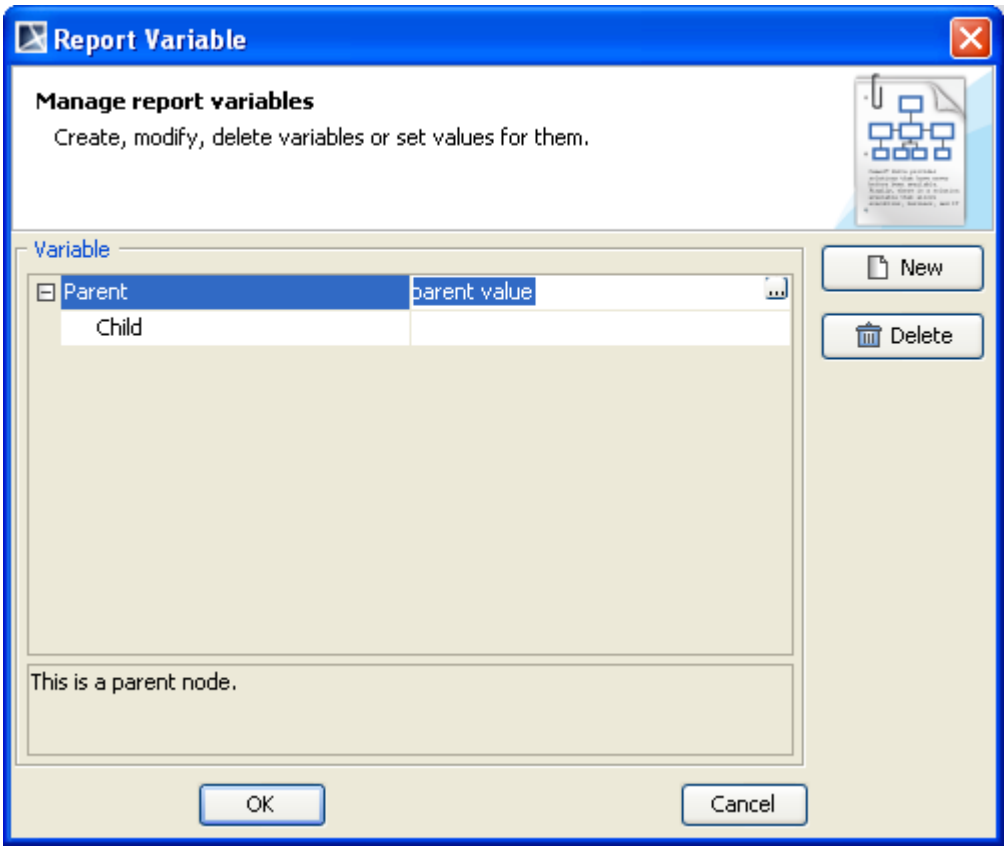


Figure 38 -- Editing Variables

3. Click **OK**. The new variable value will appear in the column next to the variable name column.

To delete a variable in Report Wizard:

1. Open the **Report Variable** dialog.
2. Click a variable in the table, and then click **Delete**. A dialog will open prompting you to click either **Yes** or **No** (Figure 39).

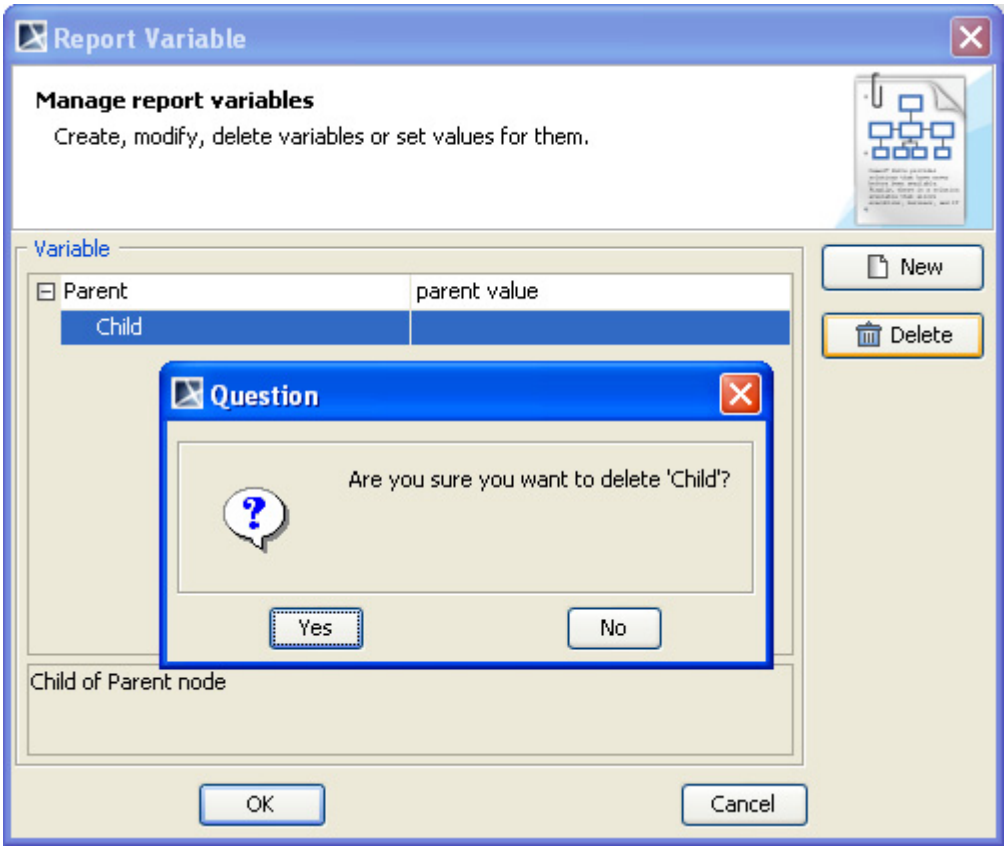


Figure 39 -- Deleting Variable

3. Click **Yes**, and the variable will then be deleted.

Including Variables in Templates

When you include the variables you have created in a template, the generated report will print each variable value. This section will use the following Report Data as an example (Figure 40).

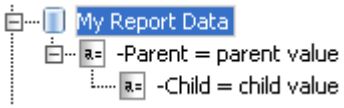


Figure 40 -- Sample of Report Data

To get the value of a top-level variable:

1. Open a blank document in Microsoft Word.
2. Type: **$Parent** (Figure 41).

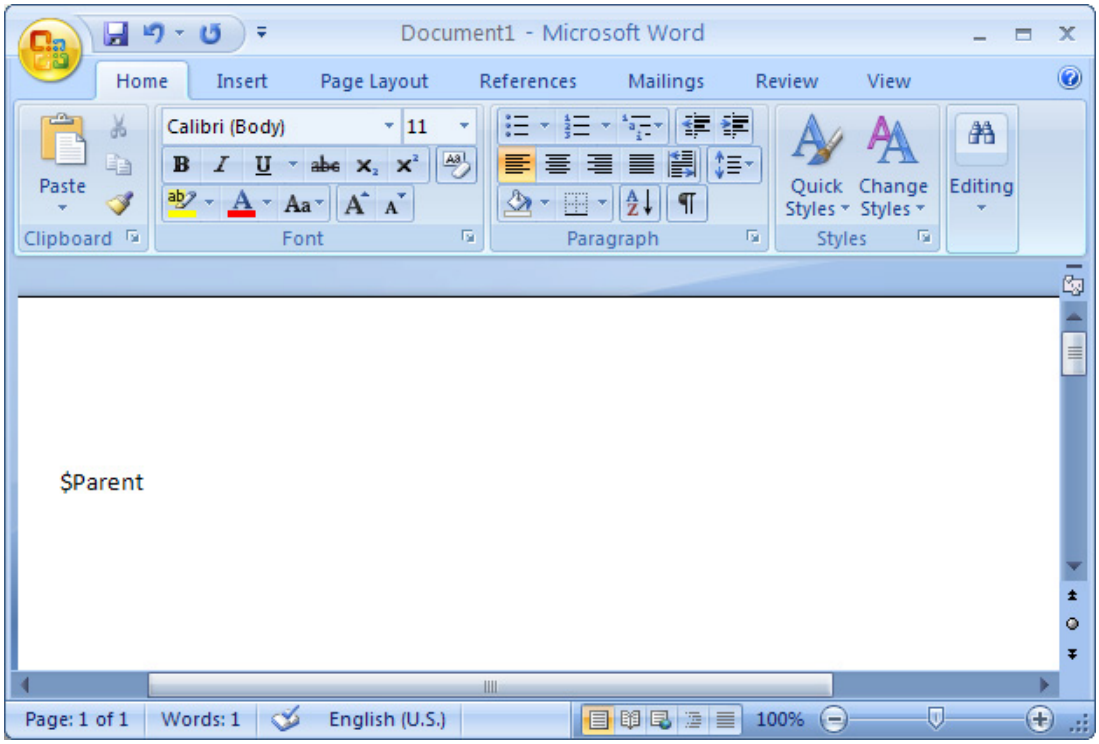


Figure 41 -- Referencing to Parent Variable in the Template

3. Save the file as “sampletemplate.rtf”. Choose **Rich Text Format** (*.rtf) as the file type (Figure 42).

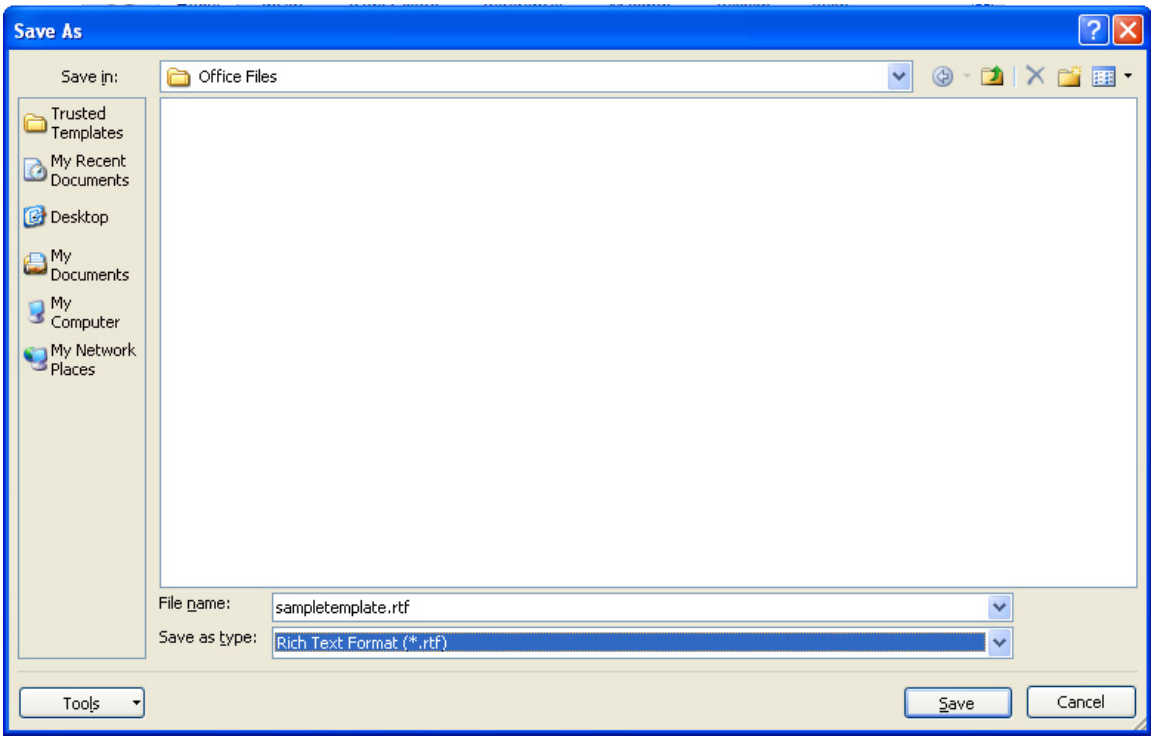


Figure 42 -- Saving Template as .rtf File in Microsoft Word

4. Open the **Report Wizard** dialog and create a new template with the **New** button. The **New Template** dialog will open.

5. Type the name and description of the new template. Click the "**...**" button to select "sampletemplate.rtf" as the template file.
6. Click **Create** > **Next**. The **Select Report Data** pane will open.
7. Select **My Report Data** and click **Generate** to generate a report.
8. The output of the generated report will be as shown in Figure 43:

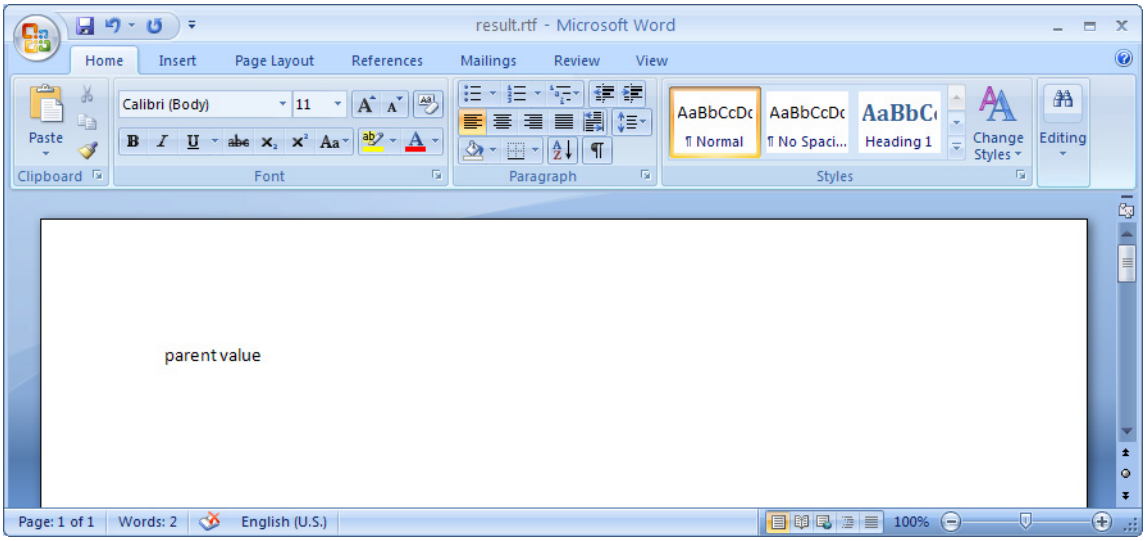


Figure 43 -- Output of $parent after Report Generation

To get the value of the child of a variable:

1. Open a blank document in Microsoft Word.
2. Type any of the following to print a child variable (Figure 44):

   (i) **$Parent.get(1)**: to get a child value by referencing its name (in this case "Child")

   (ii) **$Parent.get("Child")**: to get a child value by name comparison.

   (iii) **$Parent.Child**: to get a child value by index.

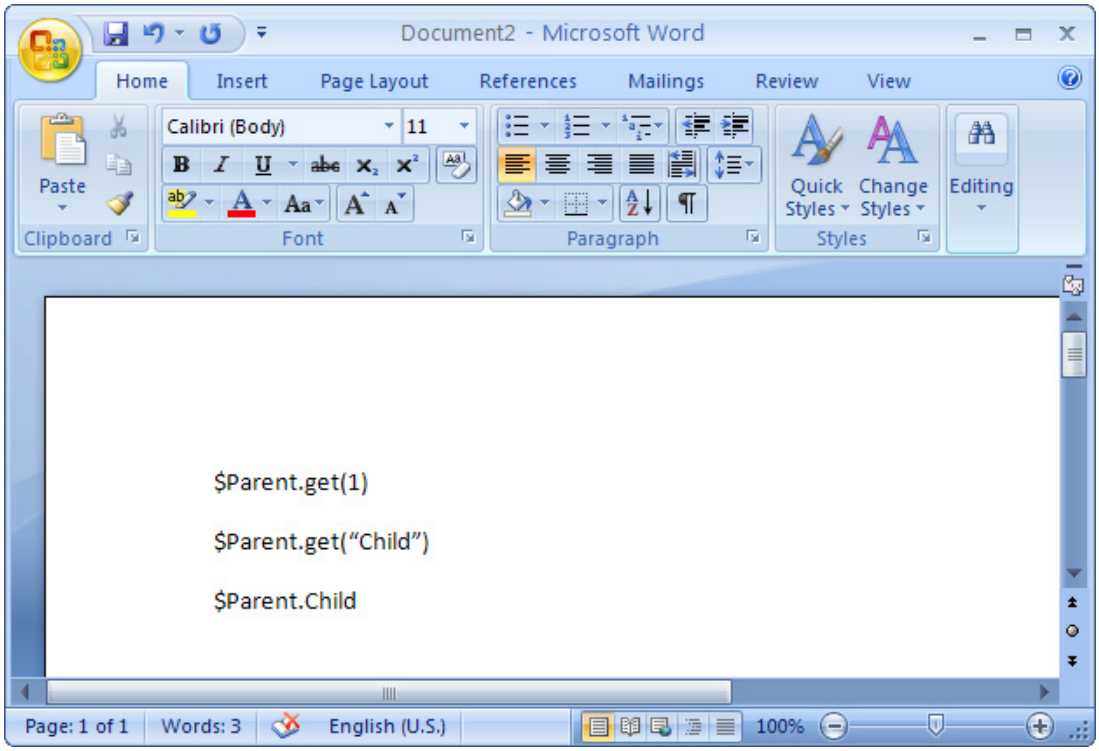


Figure 44 -- Referencing to Child Variable in the Template

3. Save the file as "sampletemplate.rtf". Choose **Rich Text Format** (*.rtf) as the file type.
4. Open the **Report Wizard** dialog and create a new template by clicking the **New** button. The **New Template** dialog will open.
5. Type the name and description of the new template. Click the "**...**" button to select "sampletemplate.rtf" as the template file.
6. Click **Create** > **Next**. The **Select Report Data** pane will open.
7. Select **My Report Data** and click **Generate** to generate a report.
8. The output of the generated report will be as shown in Figure 45:

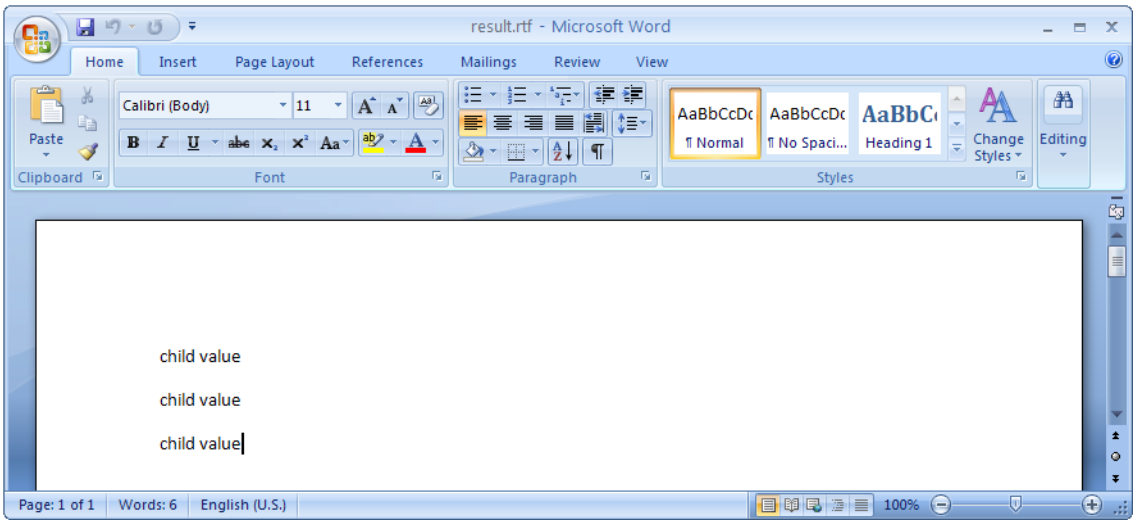


Figure 45 -- Output of $Parent.get(1), $Parent.get("Child"), and $Parent.Child

1.1.2.3 Select Element Scope Pane

The **Select Element Scope** pane (Figure 46) allows you to select the scope of MagicDraw data to generate reports.

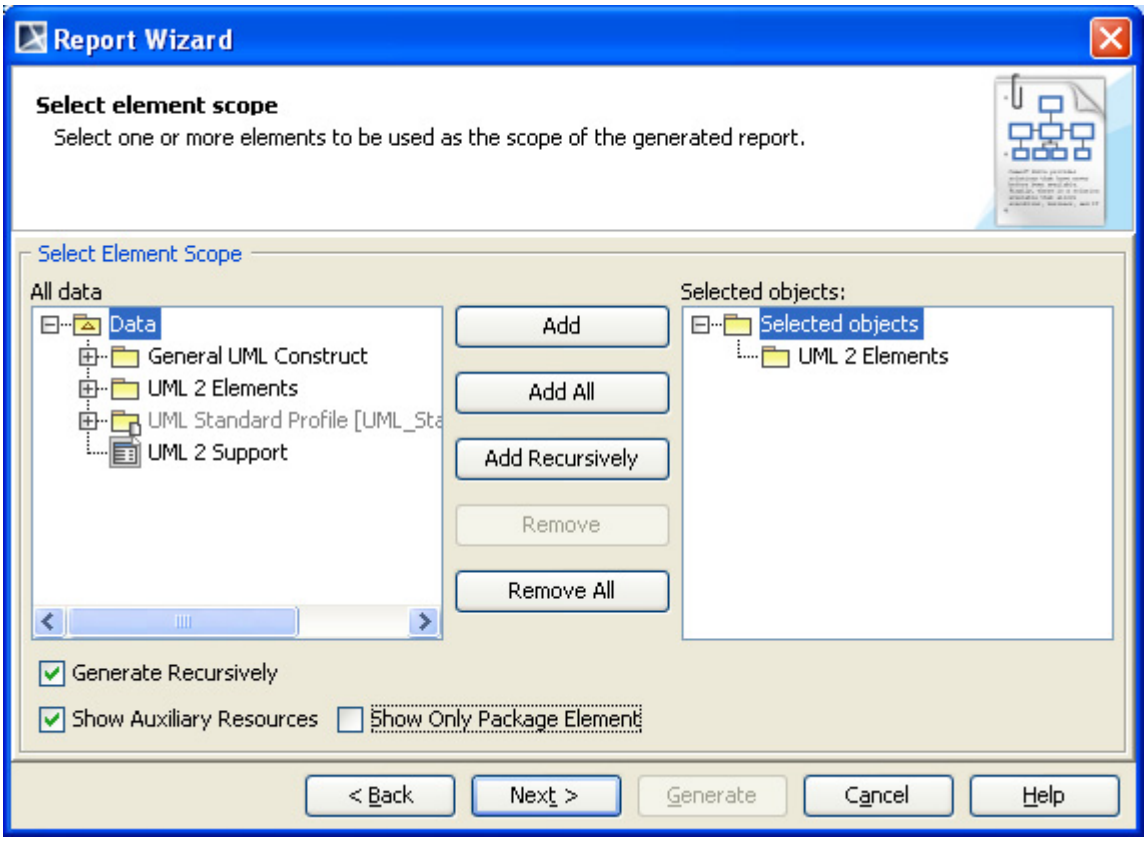


Figure 46 -- Select Element Scope Pane

The table below describes the detail of each component in the **Select Element Scope** pane.

Table 9 -- Components in the Select Element Scope Pane

Component Name	Action
All data tree	Select the desired packages from the **All data** tree then add them to the **Selected objects** tree.
Selected objects tree	Select packages and click the **Add**, **Add All**, or **Add Recursively** button. The selected packages will be added to the **Selected objects** tree.
Add button	Select packages and click **Add**. The selected packages will be added to the **Selected objects** tree.
Add All button	Select packages and click **Add All**. The selected packages and the elements directly owned by those packages will be added to the **Selected objects** tree.
Add Recursive button	Select packages and click **Add Recursively**. The selected packages and its recursive packages will be added to the **Selected objects** tree.
Remove button	Select packages and click **Remove**. The selected packages will be removed from the **Selected objects** tree.
Remove All button	Click **Remove All** and all packages in the **Selected objects** tree will be removed.

You can perform the following operations in the **Select Element Scope** pane:

(i) Add packages into the selected object tree.

(ii) Remove a selected package from the selected object tree.

(iii) Select or clear the Generate Recursively option.

(iv) Show auxiliary resources.

(v) Show only package elements.

(i) To add packages:

- In the **Select Element Scope** pane (Figure 46), select the packages from the **All data** tree and click **Add**, **Add All**, or **Add Recursively** to add them into the **Selected objects** tree.

(ii) To remove packages:

- In the **Select Element Scope** pane (Figure 46), select the packages from the **Selected objects** tree and click **Remove** or **Remove All** to remove them from the **Selected objects** tree.

(iii) To select or clear the **Generate Recursively** option:

- Select or clear the **Generate Recursively** check box in Figure 46.

(iv) To show auxiliary resources:

- Select or clear the **Auxiliary Resources** check box in Figure 46.

(v) To show only package elements:

- Select or clear the **Show Only Package Element** check box in Figure 46.

NOTE	Figure 46 shows the UML 2 Elements package and the **Generate Recursively** check box were selected. It means that the UML 2 Elements package and its subpackages will be generated in the report.

1.1.2.4 Generate Output Pane

The **Generate Output** pane in Report Wizard allows you to configure report files, for example, to select the report files' output location and image format and to display the report in the viewer (Figure 47).

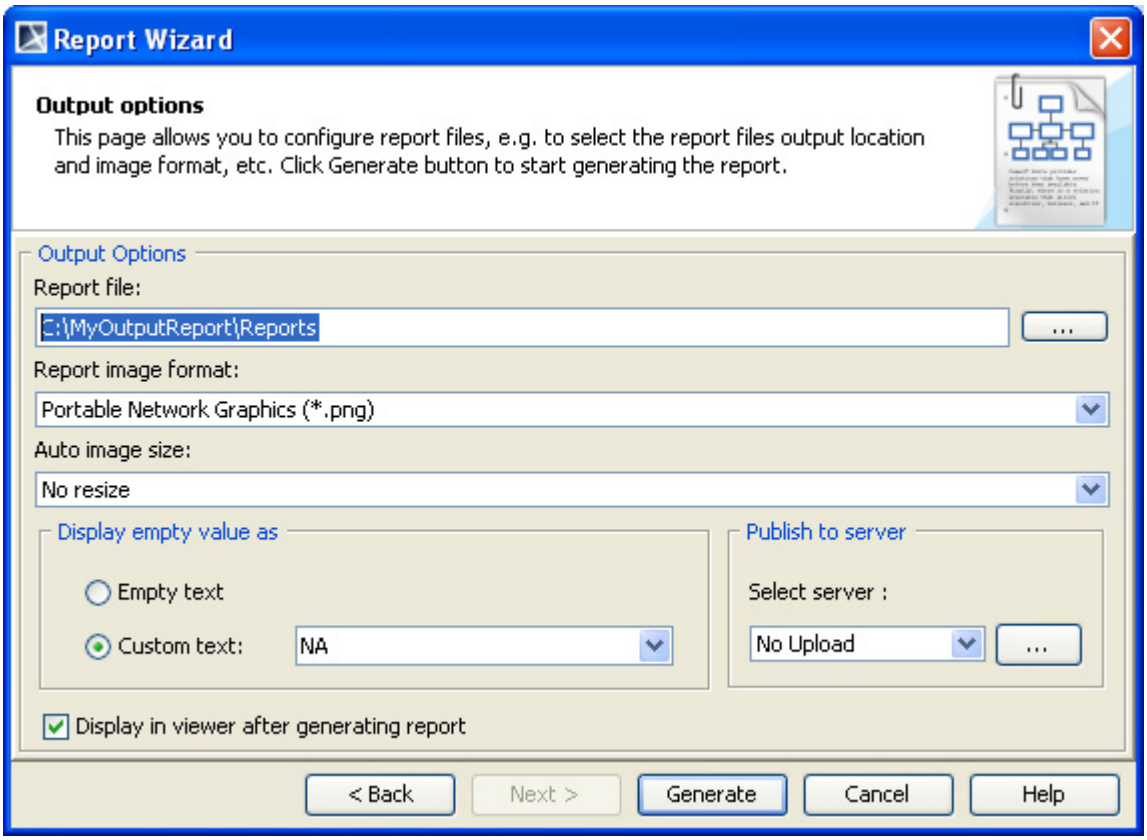


Figure 47 -- Generate Output Pane

Table 10 -- Components in Generate Output Pane

Component Name	Function
Report File	To show the report file's location and name. The default report location will be **\data\template_folder\reports**. The default report name will be the same as the report name defined by the user.
... button	To open the **Select Location** dialog in order to locate the report file.
Report Image Format	To select an image format for your report: JPG, PNG, SVG, EMF, or WMF. **Note:** ● Use *.JPG and *.PNG for any template format. ● Use *.SVG for text and HTML templates. ● Use *.EMF and *.WMF for text and Microsoft Office templates (RTF, DOCX, XLSX, and PPTX).
Auto image size	To change the size and orientation of an image before inserting it into a document. There are 4 options available: ● **No Resize**: Image will not be resized or rotated. ● **Fit image to paper (large only)**: Large image will be fitted within the paper size. ● **Fit and rotate (clockwise) image to paper (large only)**: Large image will be fitted within the paper size and rotated clockwise. ● **Fit and rotate (counter-clockwise) image to paper (large only)**: Large image will be fitted within the paper size and rotated counter-clockwise.
Display empty value as	● **Empty text**: To leave an empty value of the report output blank. ● **Custom text as**: To enter a custom value for an empty value. The default value is **NA**.
Display in viewer after generating report	To display a complete report in the viewer. Otherwise, the report will be created and kept in the selected location.
Generate	To generate a report.
Cancel	To cancel the report generation process and close the Report Wizard dialog.
Help	To provide the Help content

1.2 MRZIP File Automatic Deployment

An MRZIP file is a report template package file. You can place an MRZIP file into your template folder "<user folder>/data/reports", and MagicDraw will automatically deploy the template into the Report Wizard dialog.

Figure 48, 49, and 50 below will show you how Report Wizard can automatically deploy the MRZIP file.

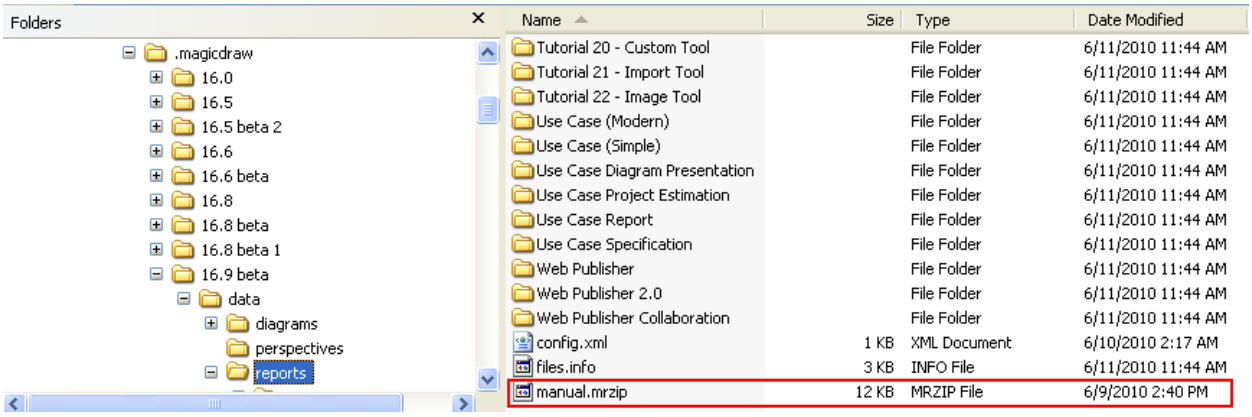


Figure 48 -- Copying MRZIP File into a Template Folder

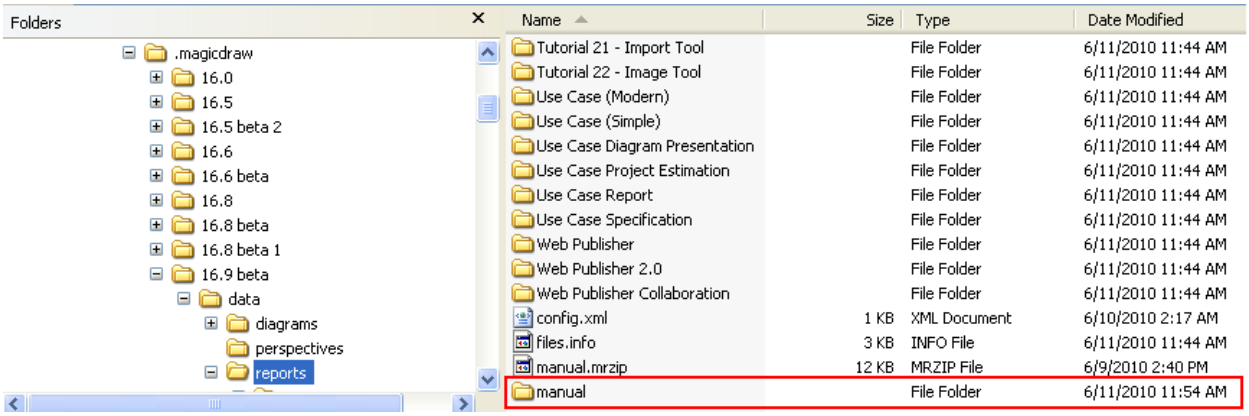


Figure 49 -- Automatic Deployment of MRZIP File

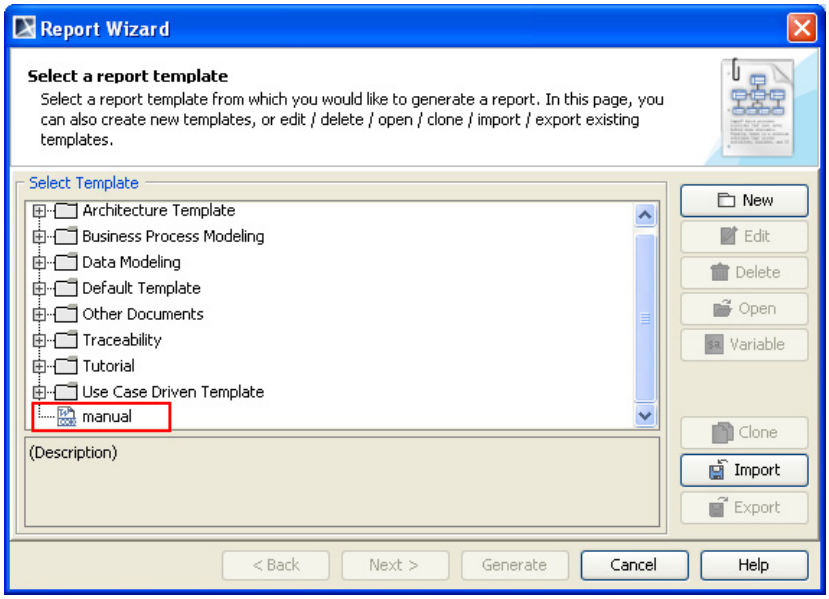


Figure 50 -- Report Template Installed in the Report Wizard Dialog

2. Report Wizard Template Language

Report Wizard is built on Velocity Engine. Knowledge of the Velocity Template Language and the Report Wizard Custom Language used within the Report Wizard template is necessary for understanding, editing, or creating templates.

2.1 Velocity Template Language

The user guide for Velocity (VTL) can be found at: **http://velocity.apache.org/engine/devel/user-guide.html**.

NOTE	• The formal reference such as `${hello}` is not supported in RTF templates. • A macro cannot be used as macro parameter in any RTF report template. This is a limitation of VTL itself: Velocity treats a macro like copy-and-paste content defined in the macro (between #macro and #end) at the inserted position. • In an RTF report template, the style and formatting defined on a directive will have no effect on the report output. In addition, the paragraph symbol at the end of line will also be removed.

2.2 Report Wizard Custom Language

2.2.1 #forrow Directive

The Velocity Template Language does not support loops inside a table structure. However, the Report Wizard engine introduces a new custom syntax that allows looping inside the table structure in order to clone the table rows.

The syntax: `#forrow <query data> #endrow`

For example:

Name	Owner
#forrow ($uc in $UseCase) $uc.name	$uc.owner.humanName #endrow

The output will be:

Name	Owner
UC1	Package Requirement
UC2	Package Requirement

2.2.2 #forpage Directive

The `#forpage` directive is used to provide a loop over the codes within a page. This directive provides implementation like `#forrow`, but it creates a loop over a page instead of a row.

Example:

```
#forpage($uc in $UseCase)
$uc.name
#endpage
```

The report will contain a UseCase name for each document page.

2.2.2.1 OpenDocument Specific Usage

When this directive appears in the OpenDocument Presentation template, it will create a loop over all directives that are present on the current page. All directives on this page will be included inside `#forpage` (Figure 52).

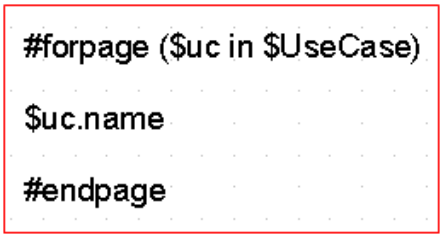


Figure 51 -- OpenDocument Presentation #forpage Template

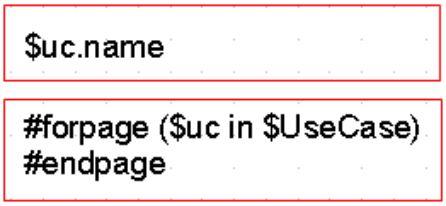


Figure 52 -- OpenDocument Presentation #forpage Template - UseCase

Figure 51 and 52 are sample templates that generate the same output. For more information, see the Open-Document Presentation template section.

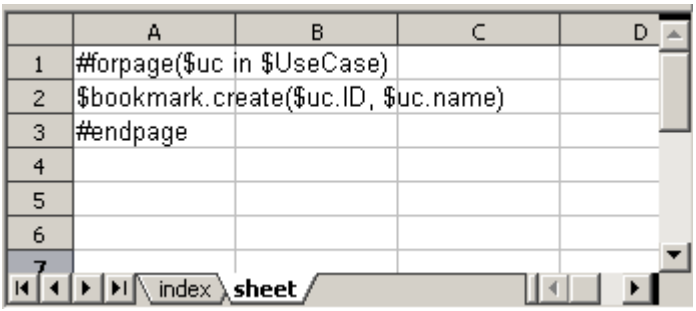


Figure 53 -- OpenDocument Spreadsheet for #forpage Template

Figure 53 displays an example of the `#forpage` directive inside an ODS file. When this directive is used in an ODS template, it will create a sheet for codes within the template sheet.

Figure 54 shows the output produced from the template.

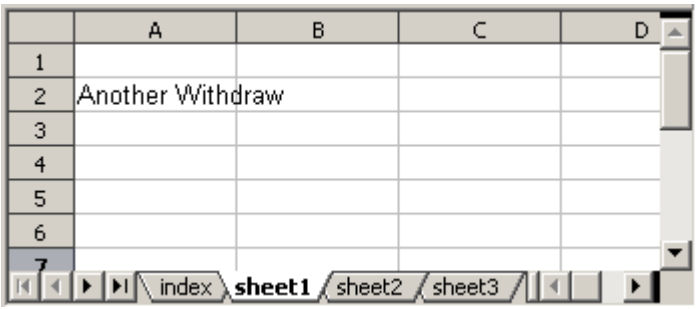


Figure 54 -- OpenDocument Spreadsheet Output for #forpage Template

2.2.3 #forcol Directives

This directive is designed only for the Spreadsheet templates, which are ODS and XLSX. This directive provides looping over the column.

	A	B	C
1	#forcol($uc in $UseCase)	$uc.name	#endcol
2			

Figure 55 -- OpenDocument Spreadsheet #forcol Template

Based on the sample in Figure 55, the engine will generate a report with different columns for each Use Case name. The output from this sample will be as shown in Figure 56:

	A	B	C	D
1	Use Case A	Use Case B	Use Case C	Use Case D
2				

Figure 56 -- OpenDocument Spreadsheet #forcol Output

You can combine both `#forrow` and `#forcol` and produce a more complex output report (Figure 57).

	A	B	C
1	#forrow($p in $Package)	#forcol($e in $p.ownedElement)$e.name#endcol	#endrow

Figure 57 -- OpenDocument Spreadsheet #forrow and #forcol Templates

Figure 58 is the output generated from the Magic Library sample project.

	A	B	C	D	E	F
1	Domain Analysis	Diagram Loan Collaboration	Diagram Loa	Diagram Obj	Dependency	Dependency
2	conceptual perspective					
3	Implementation	Node Remote Web Client	Node Remot	Artifact Interr	Artifact Magi	Artifact Magi
4	High Level Domain Analysis	Diagram conceptualPerspective1	Diagram con	Class Penalt	Association I	Association I
5	collaboration	Collaboration Loan Library Item				
6	Analysis and Design	Diagram Analysis and Design	Package Hig	Package Dor	Package Dor	Package Tech
7	object diagram	Instance Specification	Instance Spe	Instance Spe	Instance Spe	Instance Spe
8	Technical Design	Diagram domain.user	Diagram dom	Diagram dom	Diagram Mag	Package Magi
9	MagicLibrary	Instance Specification	Class Library	Class Books	Class VideoP	Class AudioP
10	Domain Design	Diagram Package Dependencies	Diagram Imp	Dependency	Dependency	Package Imple
11	composite structure	Collaboration Loan	Class Partici	Collaboration	ReadingItem	Loan
12						

Figure 58 -- OpenDocument Spreadsheet #forrow and #forcol Output Report

Since the `#forrow` syntax is similar to the `#foreach` syntax, the `#foreach` syntax can then be used in `#forrow`.

More samples about the usage of `#forrow` and `#forcol` can be found in the **Report Wizard** dialog, “Other Documents” templates.

2.2.4 #includeSection Directive

The original Velocity Engine provides two include directives: (i) `#include` and (ii) `#parse`.

(i) `#include` allows you to import another template. The contents of the file will not be rendered through the template engine.

(ii) `#parse` allows you to import another template. The contents of the file will be rendered through the template engine. However, the file being included will be inserted with all contents.

Report Wizard introduces a statement, which allows a template to include any section of a document from another template. This statement requires the template to define the beginning and the end of the section.

The logical concept of the `#includeSection` and `#parse` directives is similar. Both directives allow a template to include another template and render it through the template engine. However, `#includeSection` can be used to specify only the section that you would like to include.

To declare a section:

```
#sectionBegin (sectionName)
...
#sectionEnd
```

To include a section:

```
#includeSection (templateFilename, sectionName)
```

2.2.5 #include, #parse, and #includeSection: A Comparison

The `#include` and `#parse` directives are built-in directives provided by Velocity. The `#includeSection` directive is a custom directive implemented by MagicDraw. Table 11 on page 54 below shows the differences among these three directives.

Table 11 -- Directives Differences

	#include	**#parse**	**#includeSection**
Execution Time	Executed at runtime.	Executed as a separate template at runtime.	Executed at translation time.
Variable Scope	Variables declared in the parent template are not accessible in the included page.	Variables declared in the parent template can be accessed in the included page.	Variables declared in the parent template can be accessed in the included page.
Rendering	The Include template is not rendered through the template engine.	The Include template is rendered through the template engine as a separate process.	The Include template is rendered through the template engine as a single process.
Include only required section	No	No	Yes
Size of parent template	The size of the parent template remains unchanged.	The size of the parent template remains unchanged.	The size of the parent template is increased by the included section.

	#include	#parse	#includeSection
Processing overhead	The #include directive increases the processing overhead with the necessity of an additional call to the template engine.	The #parse directive increases the processing overhead with the necessity of an additional call to the template engine.	The #includeSection directive does not increase the processing overhead.
Support RTF template	No	No	Yes
Support ODF template	No	No	Yes
Support DOCX template	No	No	Yes
Support XLSX template	No	No	No
Support PPTX template	No	No	No

3. Template Variables

The variables imported to the template are collected by the type of the element selected in the package scope. Use the fourth step of the **Report Wizard** dialog to select the package scope (see subsection 1.1.2.3 above).

In this example, we take a class diagram with the class name Customer (Figure 59) to print it in a report.

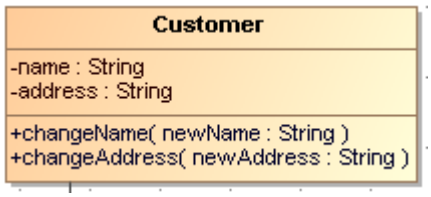


Figure 59 -- Class Diagram: Customer

To print the attribute of the Customer class in a report:

- Right-click the Customer class diagram and open the Specification dialog. The element type will appear on the dialog title, and the attribute name will appear on the right-hand side of the dialog box (Figure 60).

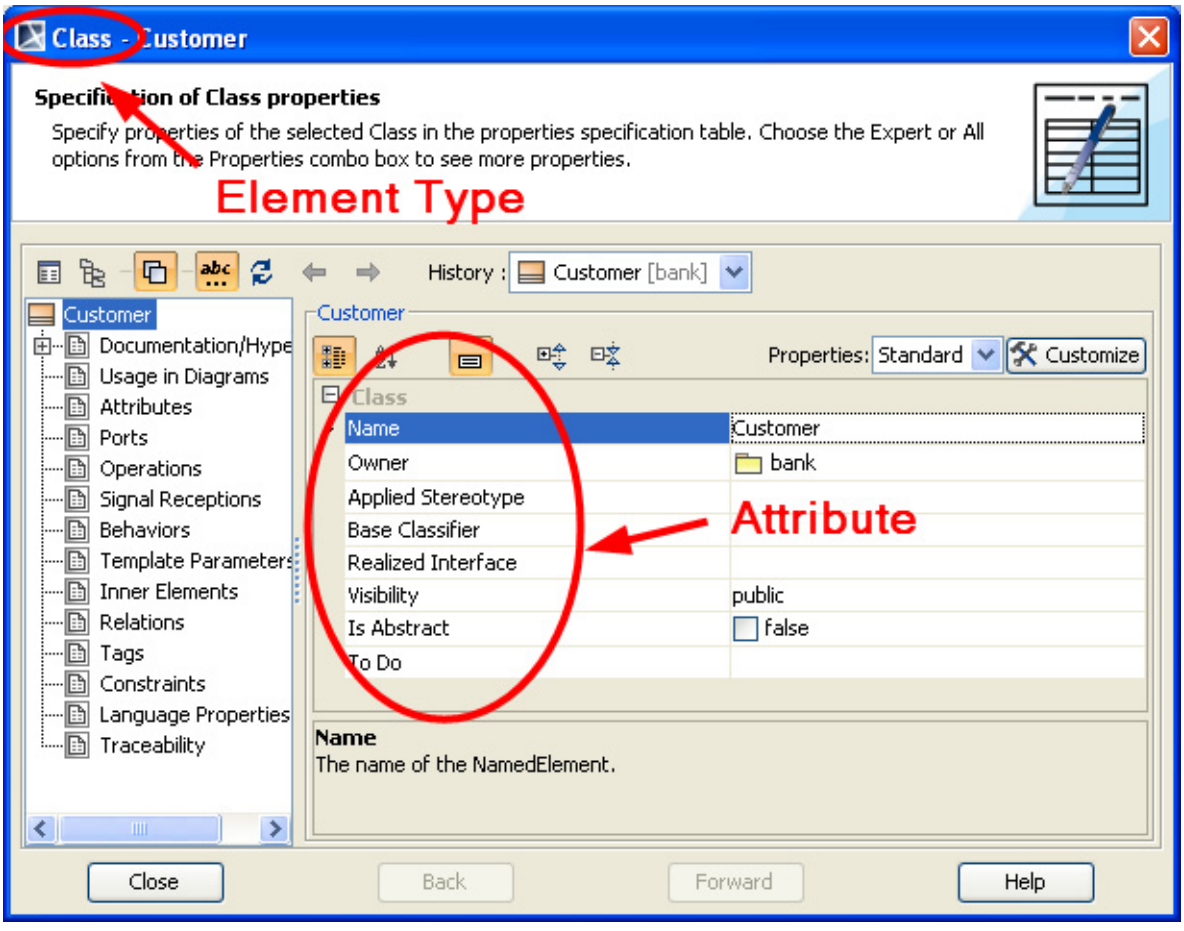


Figure 60 -- Specification Dialog

With the following VTL code, you can print other attributes of a Class element:

```
#foreach ($class in $Class)
   Name: $class.name
   Owner:  $class.owner.name
   Visibility: $class.visibility
   Is Abstract: $class.isAbstract
#end
```

The output will be as follows:

```
Name: Customer
Owner:  com
Visibility: public
Is Abstract: false
```

There are additional properties, which are not part of the UML specifications, but retrievable by Report Wizard (Table 12).

Table 12 -- Additional Properties Retrievable by Report Wizard

Element Owner	Property Name	Function
Diagram	image	To output the diagram image.
Diagram	diagramType	The diagram type.
Element	image	To output the element image or to print an empty text if the element does not refer to any diagram.
Element	elementType	To return the name of the type (metaclass / stereotype) of an element, in lowercase, without space format. For example, • return "usecase" for a Use Case, • return "class" for a Class, • return "callbehavioraction" for a Call Behavior Action, • return "flowport" for a Flow Port (a stereotype defined in SysML).
Element	humanName	Texts representing an element object. In general, a human name consists of a string concatenation between the human element type and element name.
Element	humanType	Text representing the element type.
Element	appliedStereotype	The applied stereotype.
Element	activeHyperlink	The active hyperlink.
Element	hyperlinks	All hyperlinks attached to this element.
Element	toDo	The element to do text.
Element	elementID	The element ID.
Element	documentation	The element comment/documentation.
Element	presentationElement	The OpenAPI presentation element.
Element	typeModifier	The element type modifier.
Element	tags	A list of element tags.

Element Owner	Property Name	Function
Element	text	Return a text representation of the element. In general, the 'text' property returns a string which "textually represents" this element. The string may be different, depending on symbol properties and environment options. This property returns the result from OpenAPI RepresentationTextCreator.getRepresentedText(Element element)
Classifier	baseClassifier	The base classifier element.
BehavioredClassifier	realizedInterface	The Classifier Realized Interface.
Package	appliedProfile	A list of profiles applied to the package.
Stereotype	metaClass	The Meta Class for the stereotype.
DurationConstraint	min	The minimum value.
DurationConstraint	max	The maximum value.
TimeConstraint	min	The minimum value.
TimeConstraint	max	The maximum value.
MultiplicityElement	multiplicity	The multiplicity value.
Property	navigable	The navigable value.
Property	ownedBy	The property owner.
Lifeline	type	The lifeline type.
Message	event	The message event.
Message	operation	The message operation.
Message	signal	The message signal.
Message	number	The message number.
Trigger	eventType	To trigger the event type.
RequirementsUseCase	number	The use case number.

Some predefined variables, which are not part of the UML specifications, are usable in templates:

- $elements: Contains a list of all the elements selected from the element scope.
- $packageScope: Contains a selected package from the element scope.
- $empty: Contains a String for empty value, which is specified from the **Output Options** pane.

NOTE	humanName and humanType are locale specific text. If you plan to generate a report in different languages, try to avoid humanType and humanName. Instead, use elementType for non locale specific report.

4. Helper Modules

For more details on element types, see **OpenAPI**.

4.1 $report

This module is a utility module enabling a template to get MagicDraw data.

$report.containsStereotype(element, stereotypeName)

Returns true if the element contains a stereotype for the specified stereotype name.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	The element to be checked.
	stereotypeName	java.lang.String	The stereotype name to be tested.
Return	-	boolean	True, if the element contains a stereotype for the specified stereotype name.

$report.createValueSpecificationText(specification)

Creates texts representing the ValueSpecification element.

	Name	Type	Description
Parameter(s)	specification	com.nomagic.uml2.ext.magicdraw.classes.mdkernel .ValueSpecification	A given ValueSpecification.
Return	-	java.lang.String	A String value for ValueSpecification.

$report.filterDiagram(diagramList, digramTypes)

Returns a collection of diagrams from the diagram list filtered by types.

	Name	Type	Description
Parameter(s)	diagramList	java.util.Collection	A collection of diagrams.
	digramTypes	java.util.Collection	A collection of diagram types used as filters.
Return	-	java.util.Collection	A collection of filtered diagrams.

Example:

```
#foreach($diagram in
$report.filterDiagram($Diagram,["Class
Diagram","Communication Diagram"]))
$diagram.name : $diagram.diagramType
#end
```

$report.filterElement(elementList, humanTypes)

Returns a collection from the element list filtered by types.

	Name	Type	Description
Parameter(s)	elementList	java.util.Collection	A collection of elements.
	humanTypes	java.util.Collection	A collection of human types used as filters.
Return	-	java.util.Collection	A collection of filtered elements.

Example:

```
#foreach($element in $report.filterElement($elements,
["Use Case", "Actor"]))
$element.name : $element.humanType
#end
```

$report.filter(elementList, propertyName, propertyValue)

Returns a collection of elements filtered by a specified property name.

	Name	Type	Description
Parameter(s)	elementList	java.util.Collection	A collection of elements.
	propertyName	java.lang.String	A property name.
	propertyValue	java.util.Collection	A collections of property names used as filters.
Return		java.util.Collection	A collection of filtered elements.

For example:

```
#foreach ($e in $report.filter($elements, "name",
["foo", "bar"]))
   $e.name
#end
```

$report.findElementInCollection(elementList, name)

Finds an element in the collection by name.

	Name	Type	Description
Parameter(s)	elementList	java.util.Collection	A collection of elements.
	name	java.lang.String	An element name.
Return	-	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element instance. If it cannot find the element, it will return a null value.

$report.findRelationship(modelPackage)

Searches and returns a collection of relationship elements inside a package.

	Name	Type	Description
Parameter(s)	modelPackage	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Package	A package element.
Return	-	java.util.Collection	A collection of relationships in a package.

$report.findRelationship(modelPackage, recursive)

Searches and returns a collection of relationship elements inside the package.

	Name	Type	Description
Parameter(s)	modelPackage	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Package	A package element.
	recursive	boolean	If true, performs recursively.
Return	-	java.util.Collection	A collection of relationships in a package.

$report.getAppliedStereotypeByName(element, stereotypeName)

Returns the stereotype assigned to the element.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
	stereotypeName	java.lang.String	A stereotype name.
Return	-	com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype	An assigned stereotype with a specified name.

$report.getBaseClassAssociations(classifier)

Gets associations of the classifier.

	Name	Type	Description
Parameter(s)	classifier	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Classifier	A classifier.
Return	-	java.util.Collection	A collection of associations.

$report.getBaseClassInheritableAttributes(classifier)

Gets inheritable attributes of the classifier.

	Name	Type	Description
Parameter(s)	classifier	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Classifier	A classifier.
Return	-	java.util.Collection	A collection of attributes.

$report.getBaseClassInheritableOperations(classifier)

Gets inheritable operations of the classifier.

	Name	Type	Description
Parameter(s)	classifier	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Classifier	A classifier.
Return	-	java.util.Collection	A collection of operations.

$report.getBaseClassPorts(classifier)

Gets ports of the classifier.

	Name	Type	Description
Parameter(s)	classifier	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Classifier	A classifier.
Return	-	java.util.Collection	A collection of ports.

$report.getBaseRealizedInterfaces(behavioredClassifier)

Gets realized interfaces of the behaviored classifier.

	Name	Type	Description
Parameter(s)	behavioredClassifier	com.nomagic.uml2.ext.magicdraw.commonbehaviors.mdbasicbehaviors.BehavioredClassifier	A BehavioredClassifier.
Return	-	java.util.Collection	A collection of RealizedInterfaces.

$report.getBaseRelations(classifier)

Gets relations of the classifier.

	Name	Type	Description
Parameter(s)	classifier	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Classifier	A classifier.
Return	-	java.util.Collection	A collection of relations.

$report.getBaseClassifiers(child)

Returns based elements of the classifier.

	Name	Type	Description
Parameter(s)	child	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Classifier	A child class.
Return	-	java.util.Collection	A collection of base elements (classifiers).

$report.getClientElement(element)

Returns the client of the relationship.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	A relationship model element.
Return	-	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	The relationship's client.

$report.getComment(element)

Returns the documentation of the given element.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
Return	-	java.lang.String	The documentation of an element.

$report.getDerivedClassifiers(parent)

Returns derived elements of the classifier.

	Name	Type	Description
Parameter(s)	parent	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Classifier	A parent class.
Return	-	java.util.Collection	A collection of derived elements (classifiers).

$report.getDiagramElements(diagram)

Gets elements from a diagram.

	Name	Type	Description
Parameter(s)	diagram	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram	A diagram instance.
Return	-	java.util.Collection	A collection of elements in a diagram.

$report.getDiagramType(diagram)

Returns a diagram type.

	Name	Type	Description
Parameter(s)	diagram	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram	A diagram instance.
Return	-	java.lang.String	A diagram type.

$report.getDSLProperty(element, propertyName)

Returns the DSL Property.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
	propertyName	java.lang.String	A property name.
Return	-	java.lang.Object	A property value.

$report.getElementComment(element)

Returns a comment attached to an element. If no comment is attached, it will return a null value.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
Return	-	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Comment	A comment instance of the element.

$report.getElementName(element)

Gets an element name. If the name is empty, it will return "<unnamed>". This method performs the following procedures:

1. If the element is generalized from NamedElement, it will return `$element.name`
2. If the element is generalized from Slot, it will return `$element.definingFeature.name`
3. If the element is a UML element, it will return `$element.humanName`
4. Else return `$element.toString()`

	Name	Type	Description
Parameter(s)	element	java.lang.Object	An element.
Return	-	java.lang.String	An element name.

$report.getIconFor(element)

Returns an image icon for an element.

	Name	Type	Description
Parameter(s)	element	com.nomagic.magicdraw.uml.BaseElement	A MagicDraw element.
Return	-	com.nomagic.magicreport.Image	An image object for the element's icon.

$report.getIconFor(type)

Returns an image icon for an element.

	Name	Type	Description
Parameter(s)	type	java.lang.String	The type name.
Return	-	com.nomagic.magicreport.Image	An image object for the element's icon.

$report.getIncludeUseCase(useCase)

Returns included elements of a UseCase.

	Name	Type	Description
Parameter(s)	useCase	com.nomagic.uml2.ext.magicdraw.mdusecases.UseCase	A UseCase instance.
Return	-	java.util.Collection	A collection of included UseCases.

$report.getInnerElement(element)

Returns a collection of inner elements.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
Return	-	java.util.Collection	A collection of inner elements.

$report.getInteractionMessageType(message)

Returns an interaction message type.

	Name	Type	Description
Parameter(s)	message	com.nomagic.uml2.ext.magicdraw.interactions.mdbasicinteractions.Message	A message instance.
Return	-	java.lang.String	A message type.

$report.getMetaClass(stereotype)

Returns a stereotype meta class.

	Name	Type	Description
Parameter(s)	stereotype	com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype	A stereotype instance.
Return	-	java.util.Collection	A collection of Meta Classes.

$report.getPresentationDiagramElements(diagram)

Returns presentation elements in a diagram.

	Name	Type	Description
Parameter(s)	diagram	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram	A diagram instance.
Return	-	java.util.Collection	A collection of elements.

$report.getPresentationElementBounds(diagram, element)

Returns bounds of an element in the form of Polygon objects. The bounds specify the component coordinates to its diagram.

	Name	Type	Description
Parameter(s)	diagram	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram	A target diagram.
	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	A presentation element on the given diagram.
Return	-	java.util.List	Returns the bound of an element as a Polygon. If the element is not found in the diagram, it will return null.

$report.getPresentationElementRectangle(diagram, element)

Returns bounds of an element in the form of a Rectangle object. The bounds specify the component coordinates to its diagram.

	Name	Type	Description
Parameter(s)	diagram	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram	A target diagram.
	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	A presentation element on the given diagram.
Return	-	java.util.List	Returns the bound of an element as a Rectangle. If the element is not found in the diagram, it will return null.

$report.getQualifiedName(namedElement, separator)

Gets a qualified name. Qualified is a name that allows the NamedElement to be identified within a hierarchy of nested Namespaces. It is constructed from the names of the containing namespaces starting at the root of the hierarchy and ending with the name of the NamedElement itself.

	Name	Type	Description
Parameter(s)	namedElement	NamedElement	A NamedElement.
	separator	java.lang.String	Separator symbol. If the value is null or an empty string, the '::' will be used.

	Name	Type	Description
Return	-	java.lang.String	A qualified name.

$report.getPackageQualifiedName(namedElement, separator)

Gets a qualified name by considering only Packages and the given element. Models and Profiles will not be included on the qualified name.

For example, given the element hierarchy:

```
Design : Model -> com : Package -> nomagic : Package -> ui -> Package -> BaseDialog
: Class
```

and the template code:

```
$report.getPackageQualifiedName($class, ".")
```

If `$class` is “BaseDialog” element, the result from the above template code will be:

```
com.nomagic.ui.BaseDialog
```

	Name	Type	Description
Parameter(s)	namedElement	NamedElement	A NamedElement.
	separator	java.lang.String	Separator symbol. If the value is null or an empty string, the '::' will be used.
Return	-	java.lang.String	A qualified name.

$report.getRecievingOperationalNode(element)

Gets the needline association ends.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
Return	-	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	A needline association instance.

$report.getRelationship(element)

Returns an element’s relationship.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
Return	-	java.util.Collection	A collection of relationships.

$report.getRelationship(element, recursive)

Returns a collection of element relationships.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.element	An element.
	recursive	boolean	If true, performs recursively.
Return	-	java.util.Collection	A collection of relationships.

$report.getRelativeActor(element)

Returns an element's relative actor.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
Return	-	java.util.Collection	A collection of actors.

$report.getSendingOperationalNode(element)

Returns the needline association ends.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
Return	-	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	A needline association instance.

$report.getStereotypeProperty(element, stereotypeName, propertyName)

Gets a stereotype property.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
	stereotypeName	java.lang.String	A stereotype name.
	propertyName	java.lang.String	A property name.
Return	-	java.lang.Object	A property value.

$report.getStereotypePropertyString(element, stereotypeName, propertyName)

Returns a stereotype property as String value.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element instance.
	stereotypeName	java.lang.String	A stereotype name.
	propertyName	java.lang.String	A property name.
Return	-	java.lang.String	A property value.

$report.getStereotypes(element)

Returns all stereotypes applied to an element. This method is replaced by `$element.appliedStereotype`.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
Return	-	java.util.List	A list of stereotype instances.

$report.getSupplierElement(element)

Returns the supplier of a relationship.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	A relationship model element.
Return	-	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	The relationship’s supplier.

$report.getUsageElements(usagesMap, element)

Returns the usage of a specified element.

	Name	Type	Description
Parameter(s)	usagesMap	java.util.Map	The Usage Map of MD.
	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
Return	-	java.util.Collection	A collection of element usages.

$report.getUsages(selectedObjects)

Returns the Usage Map of MD. Uses the getUsageElements method to return the usage of a specified element.

	Name	Type	Description
Parameter(s)	selectedObjects	java.lang.Object	An element.

	Name	Type	Description
Return	-	java.util.Map	A Map instance of element usages.

$report.hasStereotype(element)

Checks if an element has stereotypes.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element to check.
Return	-	boolean	True if it has a stereotype.

$report.containsStereotype(element, stereotypeName)

Returns true if the element contains a stereotype for the specified stereotype name.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element to check.
	stereotypeName	java.lang.String	A stereotype name to be tested.
Return	-	boolean	True if the element contains a stereotype for the specified stereotype name.

$report.isDerivedClassifier(parent, child)

Checks if a child is derived from the parent by generalization.

	Name	Type	Description
Parameter(s)	parent	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Classifier	A parent.
	child	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Classifier	A possible parent.
Return	-	boolean	True if it is derived from the parent by generalization.

$report.isNamedElement(element)

Returns whether an element is a NamedElement.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element to test.
Return	-	boolean	True if the given element is a NamedElement; otherwise, false.

$report.isNull(obj)

Tests and returns true if an object is null.

	Name	Type	Description
Parameter(s)	obj	java.lang.Object	An object being tested.
Return	-	boolean	True if the object is null.

$report.isRelationship(element)

Tests and returns true if an element is a relationship.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element being tested.
Return	-	boolean	True if the object is a relationship.

$report.serialize(hyperlink)

Converts **com.nomagic.magicdraw.hyperlinks.Hyperlink** to **com.nomagic.magicdraw.plugins.impl.magicreport.helper.Hyperlink**. Report Wizard needs the *com.nomagic.magicdraw.plugins.impl.magicreport.helper.Hyperlink* class wrapper to return *com.nomagic.magicdraw.hyperlinks.Hyperlink* data.

	Name	Type	Description
Parameter(s)	hyperlink	com.nomagic.magicdraw.hyperlinks.Hyperlink	A MagicDraw hyperlink.
Return	-	com.nomagic.magicdraw.plugins.impl.magicreport.helper.Hyperlink	A Report Wizard hyperlink instance.

$report.getUsedBy(element)

Returns a list of element(s) used by this element (except diagrams).

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element you like to find its usage.
Return	-	java.util.Collection	A collection of elements used by an input element.

$report.hasProperty(element, propertyName)

Returns true when a property with a given name is specified in this element, otherwise returns false.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
	propertyName	java.lang.String	A property name.

	Name	Type	Description
Return	-	java.lang.Boolean	Returns true when a property with a given name is specified in this element, otherwise returns false.

Example:

```
#foreach($element in $elements)
$report.hasProperty($element, "data")
#end
```

Figure 61 -- Sample of $report.hasProperty(element, propertyName)

- $element is an element.
- “data” is the name of a property.

$report.findElementByName(source, regex)

Searches and returns elements from names by a regular expression.

	Name	Type	Description
Parameter(s)	source	java.util.Collection	A collection of elements.
	regex	java.lang.String	A regular expression with which the name is to be matched.
Return	-	java.util.Collection	A collection of matched elements.

Example:

```
#foreach($ele in $report.findElementByName($elements, "[A]+.*"))
$ele.name
#end
```

Figure 62 -- Sample of $report.findElementByName(source, regex)

- $elements is a collection of elements to be found.
- “[A]+.*” is a regular expression to find which name is to be matched. In this example, the following names are matched names: Auxiliary, AppServer, Alternative Fragment, etc.

$report.getPresentationElements(diagram)

Gets the presentation element in a diagram. This method is equivalent to $report.getPresentationDiagramElements(diagram).

	Name	Type	Description
Parameter(s)	diagram	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Diagram	A diagram instance.
Return	-	java.util.Collection	A collection of elements.

Example:

```
#foreach($diagram in $Diagram)
#foreach($d in $report.getPresentationElements($diagram))
$d.name
#end
#end
```

Figure 63 -- Sample of $report.getPresentationElements(diagram)

- $diagram is a diagram instance.

$report.getUsageRepresentationText(baseElement, bool)

Formats the usage subject. The output string is the same as the Result column of Used by table in Magic Draw.

	Name	Type	Description
Parameter(s)	baseElement	com.nomagic.magicdraw.uml.BaseElement	A model element to be formatted.
	bool	java.lang.Boolean	True if a full path is used, otherwise false.
Return	-	java.lang.String	A formatted element.

Example:

```
#foreach($baseElement in $elements)
$report.getUsageRepresentationText($baseElement, false)
#end
```

Figure 64 -- Sample of $report.getUsageRepresentationText(baseElement, bool)

- $baseElement is a model element to be formatted.
- False if a full path is not used. For this example, a full path is not used.

$report.getUseCaseNumber(element)

Returns the use case number of the element.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	A model element.
Return	-	java.lang.String	A use case number

Example:

```
#foreach ($uc in $RequirementsUseCase)
$report.getUseCaseNumber($uc) $uc.name
#end
```

Figure 65 -- Sample of $report.getUseCaseNumber(element)

4.2 $project

This module is a project reference enabling a template to return the project information.

$project.getName()

Returns a project name.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	java.lang.String	A project name.

$project.getTitle()

Gets a project title.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	java.lang.String	A project title.

$project.getFileName()

Gets a project filename.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	java.lang.String	A project filename.

$project.getExtension()

Gets a project filename extension.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	java.lang.String	A project filename extension.

$project.getDirectory()

Gets a project directory name.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	java.lang.String	A project directory name.

$project.getVersionList()

Returns a list of version information from an open Teamwork Server project.

	Name	Type	Description
Return	-	java.util.List<Version>	A list of com.nomagic.teamwork.common.projects.Version.

NOTE	Version information consists of the following attributes:
	• comment: a version committed comment.
	• date: a committed date
	• dateAsString: a committed date as text
	• number: a committed version
	• numberAsString: a committed version as text
	• user: a committer's name
	
	Sample of code:
	`Current version : $project.version`
	`All version:`
	`----------`
	`#foreach ($version in $project.versionList)`
	`Date : $version.date`
	`Number : $version.number`
	`Number as String : $version.numberAsString`
	`User : $version.user`
	`Comment : $version.comment`
	`----------`
	`#end`

$project.getType()

Returns a file type. A file type is one of the following values:

- 0 – UNDEF
- 2 – XML_NATIVE
- 3 – UNSIYS_XMI

	Name	Type	Description
Return	-	int	The value of a file type.

$project.getDiagrams()

Returns all existing diagrams stored in a particular project.

	Name	Type	Description
Return	-	java.util.Collection	A collection of diagram instances.

$project.getDiagrams(type)

Returns existing diagrams of a given type stored in a particular project.

	Name	Type	Description
Parameter(s)	type	java.lang.String	A diagram type.
Return	-	java.util.Collection	A collection of diagram instances.

For example:

```
#set($classDiagram = $project.getDiagrams("Class Diagram"))
#foreach($cl in $classDiagram)
$cl.name
#end
```

Figure 66 -- Sample of $project.getDiagrams(type)

- “Class Diagram” is a diagram type

$project.getPresentationDiagrams()

Returns all existing presentation diagrams stored in a particular project.

	Name	Type	Description
Return	-	java.util.Collection	A collection of diagram views.

$project.getPresentationDiagrams(type)

Returns all existing presentation diagrams of a given type stored in a particular project.

	Name	Type	Description
Parameter(s)	type	java.lang.String	A diagram type.
Return	-	java.util.Collection	A collection of diagram views.

Example:

```
#set($classDiagram = $project.getPresentationDiagrams("Class Diagram"))
#foreach($cl in $classDiagram)
$cl.name
#end
```

Figure 67 -- Sample of $project.getPresentationDiagrams(type)

- “Class Diagram” is a diagram type.

$project.isRemote()

Returns the remote or non-remote state of a project.

	Name	Type	Description
Return	-	java.lang.Boolean	Returns true if a project is a remote project, otherwise false.

$project.isDirty()

Returns true if that particular project was modified after it had been saved or loaded.

	Name	Type	Description
Return	-	java.lang.Boolean	Returns true if a project was modified after it had been saved/loaded, otherwise false.

$project.getElementByID(id)

Returns an element with a given ID.

	Name	Type	Description
Parameter(s)	ID	java.lang.String	An element ID.
Return	-	com.nomagic.magicdraw.uml.BaseElement	An element with a given ID or null if the element with such ID is not registered in the project.

For example:

```
#set($ele = $project.getElementByID("_9_0_62a020a_1105704887361_983947_8206"))
$ele.name
$ele.humanType
```

Figure 68 -- Sample of $project.getElementByID(id)

- “_9_0_62a020a_1105704887361_983947_8206” is an element ID.

$project.getAllElementId()

Returns a collection of all element IDs in a project.

	Name	Type	Description
Return	-	java.util.Collection	A collection of all element IDs in a project.

$project.getXmiVersion()

Returns the XMI version of a project.

	Name	Type	Description
Return	-	int	An XMI version.

$project.getVersion()

Returns a project version number.

	Name	Type	Description
Return	-	long	A project version number.

$project.getModel()

Returns a model (The root container of all model structures).

	Name	Type	Description
Return	-	com.nomagic.uml2.ext.magicdraw.auxiliarycon-structs.mdmodels.Model	A model.

$project.getModuleList()

Returns a list of ModuleDescriptors from an open Teamwork Server project.

	Name	Type	Description
Return	-	java.util.Collection	A list of com.nomagic.magic-draw.core.modules.Module-Descriptor.

$project.getSharedModule(module)

Returns a list of shared modules from a specified module.

	Name	Type	Description
Parameter(s)	module	com.nomagic.magic-draw.core.mod-ules.ModuleDescriptor	A module.
Return	-	java.util.Collection	A list of com.nomagic.magic-draw.core.modules.Module-Descriptor.

For example:

```
#foreach ($module in $project.getModuleList())
name             : $module.representationString
description      : $module.description
version          : $module.version
required version : $module.requiredVersion
shared module    :
#foreach ($child in $project.getSharedModule($module))
   - $child.representationString
#end
========================
#end
```

Figure 69 -- Sample of $project.getSharedModule(module)

- "$module" is a module from Teamwork Server.

4.3 $iterator

This module is used with the #foreach loops. It wraps a list to let you specify a condition to terminate the loop and reuse the same list in a different loop. The following example shows how to use `$Iterator`.

```
#set ($list = [1, 2, 3, 5, 8, 13])
#set ($numbers = $iterator.wrap($list))
#foreach ($item in $numbers)
#if ($item < 8) $numbers.more()#end
#end
$numbers.more()

Output
------
1 2 3 5
8
```

$iterator.wrap(list)

Wraps a list with the tool.

	Name	Type	Description
Parameter(s)	list	java.util.List	An array or list instance.
Return	-	IteratorTool	Returns the IteratorTool instance.

$<IteratorTool instance>.hasMore()

Checks if the iteration has more elements.

	Name	Type	Description
Return	-	boolean	Returns true if there are more elements in the wrapped list.

$<IteratorTool instance>.more()

Asks for the next element in the list.

	Name	Type	Description
Return	-	java.lang.Object	An element instance.

$<IteratorTool instance>.remove()

Removes the current element from the list.

$<IteratorTool instance>.reset()

Resets the wrapper so that it will start over at the beginning of the list.

$<IteratorTool instance>.stop()

Puts a condition to break out of the loop.

$<IteratorTool instance>.toString()

Returns an object as a string.

4.4 $list

This module is the list module used to work with lists and arrays in templates. It provides a method to get and set specified elements. It also provides methods to perform the following actions for a list or an array:

- Checks if it is empty.
- Checks if it contains certain elements.

Example:

```
$primes                          -> new int[] {2, 3, 5, 7}
 $list.size($primes)             -> 4
 $list.get($primes, 2)           -> 5
 $list.set($primes, 2, 1)        -> (primes [2] becomes 1)
 $list.get($primes, 2)           -> 1
 $list.isEmpty($primes)          -> false
 $list.contains($primes, 7)      -> true
```

$list.contains(list, element)

Checks if a list or array contains certain elements.

	Name	Type	Description
Parameter(s)	list	-	A list or array instance.
	element	-	An element instance.
Return	-	boolean	Returns true if a list or array contains certain elements. Otherwise, returns false.

$list.get(list, index)

Returns a specified element of a list or array.

	Name	Type	Description
Parameter(s)	list	-	A list or array instance.
	index	-	An index of an object in a list.
Return	-	object	Returns an object in a list.

$list.isArray(object)

Checks if an object is an array.

	Name	Type	Description
Parameter(s)	object	-	An object
Return	-	boolean	Returns true if the object is an array. Otherwise, returns false.

$list.isEmpty(list)

Checks if a list or array is empty.

	Name	Type	Description
Parameter(s)	list	-	A list of objects
Return	-	boolean	Returns true if the list or array is empty. Otherwise, returns false.

$list.isList(object)

Checks if an object is a list.

	Name	Type	Description
Parameter(s)	object	-	An object
Return	-	boolean	Returns true, if the object is a List. Otherwise, return false.

$list.set(list, index, value)

Sets a specified element of the List/array.

	Name	Type	Description
Parameter(s)	list	-	A list of objects.
	index	-	An index of an object in the list.
	value	-	An object that will be set in the list.
Return	-	object	An object at the previously-specified position with the set value.

$list.size(list)

Returns the size of a List or array.

	Name	Type	Description
Parameter(s)	list	-	A list of objects.
Return	-	Integer	Returns the size of the list as an Integer instance.

4.5 $bookmark

This module contains utility functions for bookmarking. The functions of this module are accessible from templates via `$bookmark`.

NOTE	In RTF reports, the default style of bookmarks depends on the RTF editor used. For example, Microsoft Word 2003 displays hyperlinks in blue.

$bookmark.openURL(url, content)

Creates a hyperlink to open a URL. For example, `$bookmark.openURL`("http://www.nomagica-sia.com","NoMagic Asia").

	Name	Type	Description
Parameter(s)	url	java.lang.String	A URL text.
	content	java.lang.Object	The text content.
Return	-	com.nomagic.magicreport.Link	The text content with a hyperlink in the RTF format.

$bookmark.openURL(url, content)

Creates a hyperlink to open a URL. For example, `$bookmark.openURL`($url, "NoMagic Asia").

	Name	Type	Description
Parameter(s)	url	java.net.URL	A URL instance.
	content	java.lang.Object	The text content.
Return	-	com.nomagic.magicreport.Link	The text content with a hyperlink in the RTF format.

$bookmark.openURL(url, content)

Creates a hyperlink to open a URL. For example, `$bookmark.openURL`($url, "NoMagic Asia").

	Name	Type	Description
Parameter(s)	uri	java.net.URI0	A URI instance.
	content	java.lang.Object	The text content.
Return	-	com.nomagic.magicreport.Link	The content with a hyperlink in the RTF format.

$bookmark.open(content)

Creates a hyperlink for a bookmark. The bookmark ID will be automatically generated from the content.

	Name	Type	Description
Parameter(s)	content	java.lang.Object	The text content.
Return	-	com.nomagic.magicreport.Link	The text content with a hyperlink in the RTF format.

$bookmark.open(bookmarkId, content)

Creates a hyperlink for a bookmark. The bookmark ID value must match the parameter bookmark ID that passes into the link.

	Name	Type	Description
Parameter(s)	bookmarkId	java.lang.String	The bookmark ID.
	content	java.lang.Object	The text content.
Return	-	com.nomagic.magicreport.Link	The text content with a hyperlink in the RTF format.

$bookmark.create(bookmarkObject)

Creates a bookmark. The bookmark ID will be automatically generated from the bookmark object.

	Name	Type	Description
Parameter(s)	bookmarkObject	java.lang.Object	A bookmark object or content.
Return	-	com.nomagic.magicreport.Bookmark	The content with a bookmark in the RTF format.

$bookmark.create(bookmarkId, bookmarkObject)

Creates a bookmark. The default element type for this function is "label".

	Name	Type	Description
Parameter(s)	bookmarkId	java.lang.String	The bookmark ID.
	bookmarkObject	java.lang.Object	A bookmark object or content.
Return	-	com.nomagic.magicreport.Bookmark	The content with a bookmark in the RTF format.

$bookmark.create(bookmarkId, bookmarkObject, elementType)

Creates a bookmark with a specified element type.

	Name	Type	Description
Parameter(s)	bookmarkId	java.lang.String	The bookmark ID.
	bookmarkObject	java.lang.Object	A bookmark object or content.
	elementType	java.lang.String	An element type name, for example, html tag.
Return	-	com.nomagic.magicreport.Bookmark	The content with a bookmark in the RTF format.

$bookmark.getBookmarkId(id)

Returns a bookmark ID from the given string.

	Name	Type	Description
Parameter(s)	id	java.lang.String	An original string value.
Return	-	com.nomagic.magicrep-ort.Bookmark	The bookmark ID.

4.6 $sorter

This module is used to sort a collection of report templates.

$sorter.sort(Collection, fieldName)

The sort function for report templates. The context name of this class is "sorter". Use $sorter to access public functions of this class through templates.

	Name	Type	Description
Parameter(s)	collection	java.util.Collection	A collection to be sorted
	fieldname	java.lang.String	A fieldName to be sorted and the sort direction.
Return	-	java.util.Collection	A sorted collection.

For example:

```
#foreach ($rel in $sorter.sort($package, "name"))
$rel.name
#end
```

- `$package` is the collection to be sorted.
- "name:desc" is separated by ":" in two parts:
  - The first part is to identify fieldName to be sorted.
  - The second part is the option to identify the sorting direction. Sometimes, the direction is not identified. It is ascending by default.

$sorter.sort(Collection)

This is the sort function for report templates. The context name of this class is "sorter". Use $sorter to access public functions of this class through the templates.

For example:

```
#foreach ($rel in $sorter.sort($package))
$rel.name
#end
```

$package is a collection to sort

	Name	Type	Description
Parameter(s)	collection	java.util.Collection	A collection to be sorted.
Return	-	java.util.Collection	A sorted collection.

$sorter.sortByFirstNumber(Collection, fieldName)

The **sortByFirstNumber** function is for report templates. The context name of this class is "sorter". Use $sorter to access public functions of this class through templates.

	Name	Type	Description
Parameter(s)	collection	java.util.Collection	A collection to be sorted.
	fieldName	java.lang.String	A fieldName to be sorted and the sorting direction.
Return	-	java.util.Collection	A sorted collection.

For example:

```
#foreach ($rel in $sorter.sortByFirstNumber($package,
"name:desc"))
$rel.name
#end
```

- `$package` is a collection to be sorted by **FirstNumber**.
- "name:desc" is separated by ":" in two parts:
  - The first part is to identify fieldName to be sorted.
  - The second part is the option to identify the sorting direction. Sometimes, the direction is not identified. It is ascending by default.

$sorter.sortByFirstNumber(Collection)

The **sortByFirstNumber** function for report templates. The context name of this class is "sorter". Use $sorter to access public functions of this class through templates.

	Name	Type	Description
Parameter(s)	collection	java.util.Collection	A collection to be sorted.
Return	-	java.util.Collection	A sorted collection.

For example:

```
#foreach ($rel in $sorter.sortByFirstNumber($package))
$rel.name
#end
```

- `$package` is a collection to be sorted by **FirstNumber**.

$sorter.sortByLocale(Collection, String)

The sorting function for report templates. The context name of this class is "sorter". Use $sorter to access public functions of this class through templates. To sort the given collection by a particular language, identify the country code to specify the language.

	Name	Type	Description
Parameter(s)	collection	java.util.Collection	A collection to be sorted.
	countryCode	java.lang.String	The country code to specify a language to perform sorting.
Return	-	java.util.Collection	A sorted collection.

Example:

```
#foreach ($p in $sorter.sortByLocale($package, "DE"))
$p.name
#end
```

- `$package` is the collection to be sorted.
- "DE" is the country code for GERMANY (ISO country code).

NOTE	This method performs the sorting by the “name” attribute of each element by default.

$sorter.sortByLocale(Collection, String, String)

This is the sorting function for report templates. The context name of this class is "sorter". Use $sorter to access public functions of this class through templates. To sort the given collection by a particular language, specify the language by identifying the country code and field name.

	Name	Type	Description
Parameter(s)	collection	java.util.Collection	A collection to be sorted.
	fieldName	java.lang.String	A fieldName to be sorted.
	countryCode	java.lang.String	The country code to specify a language to perform sorting.
Return	-	java.util.Collection	A sorted collection.

For example:

```
#foreach ($p in $sorter.sortByLocale($package, "name",
“DE”))
$p.name
#end
```

- `$package` is the collection to be sorted.
- “name” is the field name to be sorted.
- "DE" is the country code for GERMANY (ISO country code).

$sorter.humanSort(collection, fieldName)

This is a special sorting function in a human-like order, which splits text to be sorted into numeric and non-numeric chunks then sorts so that the numeric chunks are treated as numbers. This makes "foo10" sorted after "foo2".

	Name	Type	Description
Parameter(s)	collection	java.util.Collection	A collection to be sorted.
	fieldName	java.lang.String	A fieldName to be sorted and the sorting direction.
Return		java.util.Collection	A sorted collection.

For example:

```
#foreach ($rel in $sorter.humanSort($package, "name:desc"))
$rel.name
#end
```

Figure 70 -- $sorter.humanSort(collection, fieldName)

- $package is the collection to be sorted.
- "name:desc" is separated by ":" in two parts:
  - The first part is to identify fieldName to be sorted.
  - The second part is the option to identify the sorting direction. Sometimes, the direction is not identified. It is ascending by default.

$sorter.humanSort(collection)

A special sorting function in a human-like order, which splits text to be sorted into numeric and non-numeric chunks then sorts so that the numeric chunks are treated as numbers. This makes "foo10" sorted after "foo2".

	Name	Type	Description
Parameter(s)	Collection	java.util.Collection	A collection to be sorted.
Return	-	java.util.Collection	A sorted collection.

For example:

```
#foreach ($rel in $sorter.humanSort($package))
$rel.name
#end
```

Figure 71 -- Sample of $sorter.humanSort(collection)

- $package is the collection to be sorted.

4.7 $template

This module is the tool used for getting a template's information.

$template.getName()

Returns the name of a template.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	java.lang.String	Returns a template name.

$template.getResourcesLocation()

Returns the folder location of the resource file.

	Name	Type	Description
Parameter(s)	-	-	-

	Name	Type	Description
Return	-	java.lang.String	The location of a resource file folder.

$template.getTemplateFile()

Returns a template filename.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	java.lang.String	Returns a template filename.

$template.getTemplateLocation()

Returns the folder location of a template file.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	java.lang.String	The location of a template file folder.

$template.getOutputFile()

Returns the output filename.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	java.lang.String	Returns the output filename.

$template.getOutputFileNoExt()

Returns the output name.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	java.lang.String	The output name without a file-name extension.

$template.getOutputLocation()

Returns the folder location of the output file.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	java.lang.String	Returns the location of an output file folder.

4.8 $file

This module allows generating an output report file in a template file.

$file.silentCreate(template)

A shortcut to create a file, the output filename of which is the template name, not an import context object. For example, `$file.silentCreate`('overview.html').

	Name	Type	Description
Parameter(s)	template	java.lang.String	The input template name.
Return	-	void	-

$file.silentCreate(template, importObject)

A shortcut to create a file, the output filename of which is the template name. For example, `$file.silentCreate`('overview.html','').

	Name	Type	Description
Parameter(s)	template	java.lang.String	The input template name.
	importObject	java.lang.Object	An object reference, which will be $importer in the template file. You can use the $importer variable in the template file to get data.
Return	-	void	-

$file.silentCreate(template, outputFileName, importObject)

Generates a report output from a given template name. For example, `$file.silentCreate`('overview.html','overview.html','').

	Name	Type	Description
Parameter(s)	template	java.lang.String	The input template name.
	outputFileName	java.lang.String	The output filename.
	importObject	java.lang.Object	An object reference, which will be $importer in the template file. You can use the $importer variable in the template file to get data.
Return	-	void	-

$file.silentCreate(templateType, template, outputname, importObject)

Generates a report output from a given template name. For example, `$file.silentCreate`('html','overview','overview','').

	Name	Type	Description
Parameter(s)	templateType	java.lang.String	A template type such as rtf, html, or htm.
	template	java.lang.String	The input pathname string without a filename extension.
	outputFileName	java.lang.String	The output filename without a filename extension.
	importObject	java.lang.Object	An object reference, which will be $importer in the template file. You can use the $importer variable in the template file to get data.
Return	-	void	-

$file.create(template)

A shortcut to create a file, the output filename of which is the template name, not an import context object. For example, `$file.create`('overview.html').

	Name	Type	Description
Parameter(s)	template	java.lang.String	The input template name.
Return	-	java.lang.String	The output pathname. It will return an empty string if there is an error.

$file.create(template, importObject)

A shortcut to create a file, the output filename of which is the template name. For example, `$file.create`('overview.html','')

	Name	Type	Description
Parameter(s)	template	java.lang.String	The input template name.
	importObject	java.lang.Object	The object reference, which will be importer in the template file. You can use the $importer variable in the template file to get data.
Return	-	java.lang.String	The output pathname. It will return an empty string if there is an error.

$file.create(template, outputFileName, importObject)

Generates a report output from a given template name. For example, `$file.create`('overview.html','overview.html','').

	Name	Type	Description
Parameter(s)	template	java.lang.String	The input template name.
	outputFileName	java.lang.String	The output filename.
	importObject	java.lang.Object	An object reference, which will be $importer in the template file. You can use the $importer variable in the template file to get data.
Return	-	java.lang.String	The output pathname. It will return an empty string if there is an error.

$file.create(templateType, template, outputname, importObject)

Generates a report output from a given template name. For example: `$file.create`('html','overview','overview','').

	Name	Type	Description
Parameter(s)	templateType	java.lang.String	A template type such rtf, html, htm.
	template	java.lang.String	The input pathname string without a filename extension.
	outputname	java.lang.String	The output filename without a filename extension.
	importObject	java.lang.Object	An object reference, which will be $importer in the template file. You can use the $importer variable in the template file to get data.
Return	-	java.lang.String	The output pathname. It will return an empty string if there is an error.

$file.copy(inputFilename)

Copies an input file to an output file using the same name in the binary format. For example: $file.copy('icon.gif').

	Name	Type	Description
Parameter(s)	inputFilename	java.lang.String	The input filename
Return	-	java.lang.String	The output pathname. It will return an empty string if there is an error.

$file.copy(inputFilename, outputFilename)

Copies an input file to an output file in the binary format. For example: $file.copy('icon.gif','icon.gif').

	Name	Type	Description
Parameter(s)	inputFilename	java.lang.String	The input filename.
	outputFilename:	java.lang.String	The output filename.
Return	-	java.lang.String	The output pathname. It will return an empty string if there is an error.

$file.exists(pathname)

Tests whether a file denoted by a specific pathname exists. By default, the current directory refers to the template location.

For example:

```
$file.exists("$template.resourcesLocation/myimage.png")
$file.exists("C:/myfolder/myimage.png")
$file.exists("mytemplate.txt")
```

	Name	Type	Description
Parameter(s)	pathname	java.lang.String,	A pathname string.
Return	-	java.lang.String	True if and only if a file or directory denoted by a specific pathname exists; otherwise, false.

$file.computeName(directory, name)

Creates a pathname string from a given directory and name.

	Name	Type	Description
Parameter(s)	directory	java.lang.String	A parent directory of a file.
	name	java.lang.String	A filename (excludes the filename extension).
Return	-	java.lang.String	A pathname from the given directory and name.

Example:

```
 $file.computeName('actors', $ac.ID, 'html')
output: 'actor/_123456789.html'
```

$file.computeName(directory, name, fileType)

Creates a pathname string from a given directory, name, and file type.

	Name	Type	Description
Parameter(s)	directory	java.lang.String	A parent directory of a file.
	name	java.lang.String	A filename (excluding the extension).
	fileType	java.lang.String	A file type, such as rtf, txt, and html.
Return	-	java.lang.String	A pathname from the given directory, name, and file type.

4.9 $array

Use $array to create an Array or HashSet instance.

$array.createArray()

Creates an empty ArrayList.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	java.util.List	An instance of ArrayList.

$array.createArray(collection)

Constructs an ArrayList containing elements of the specified collection, in the order they are returned by the collection's iterator. Returns a zero size ArrayList if the given collection is null.

	Name	Type	Description
Parameter(s)	collection	java.util.Collection	A collection of instances whose elements are to be placed into a list.

	Name	Type	Description
Return	-	java.util.List	An ArrayList containing elements of the specified collection.

$array.subList(list, size)

Creates an ArrayList of a portion of the list in the given size.

	Name	Type	Description
Parameter(s)	list	java.util.List	An original list.
	size	int	A view size of the given list.
Return	-	java.util.List	An ArrayList of a specified view size within the given list.

$array.addCollection(parent, child)

Adds a child list into a parent collection. It helps the template to handle if the child is null.

	Name	Type	Description
Parameter(s)	parent	java.util.Collection	A parent collection.
	child	java.util.Collection	A child collection.
Return	-	void	-

$array.createHashSet()

Creates a HashSet instance.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	java.util.Set	An instance of HashSet.

4.10 $group

$group.create()

Creates a new instance of a group tool.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	GroupTool	An instance of a group tool.

$group.init()

Initializes a group tool.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	void	-

$group.groupNames()

Returns a set of group names.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	java.util.Set	A set of group names.

$group.contains(groupName)

Tests whether a group name is contained in a set of group names.

	Name	Type	Description
Parameter(s)	groupName	java.lang.String	A group name.
Return	-	-	True if the group name is contained in a set of group names, otherwise false.

$group.put(groupName, object)

Adds an object into a group.

	Name	Type	Description
Parameter(s)	groupName	java.lang.String	A group name.
	object	java.lang.Object	An object being added.
Return	-	void	-

$group.get(groupName)

Returns a list of group objects.

	Name	Type	Description
Parameter(s)	groupName	java.lang.String	A group name.
Return	-	java.util.List	A list of group objects.

$group.remove(groupName)

Removes a group from the specified group name.

	Name	Type	Description
Parameter(s)	groupName	java.lang.String	A group name.
Return	-	java.util.List	A list of group objects previously associated with a specified group name, or null if there is no group corresponding to the key.

$group.removeAll()

Removes all groups.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	void	-

$group.clear()

Removes all mappings.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	void	-

4.11 $map

$map.createHashMap()

Creates a HashMap instance.

	Name	Type	Description
Parameter(s)	-	-	-
Return	-	java.util.HashMap	An instance of HashMap.

4.12 $date

A tool for accessing and formatting the “current” date.

$date

Displays the current date and time, for example, Oct 19, 2003 and 9:54:50 PM respectively.

$date.long

Displays the current date and time in long format, for example, October 19, 2003 and 9:54:50 PM. respectively.

$date.full_date

Displays the current date in full format, for example, Sunday, October 19, 2003.

$date.get('format')

Displays the current date in a specific format, for example, $date.get('yyyy-M-d H:m:s') is displayed as 2003-10-19 21:54:50.

Table 13 -- Date and Time Format

Letter	Date or Time Component	Example
G	Era designator	AD
y	Year 1996	96
M	Month in year	July; Jul; 07
w	Week in year	27
W	Week in month	2
D	Day in year	189
d	Day in month	10
F	Day of week in month	2
E	Day in week	Tuesday; Tue
a	Am/pm marker	PM
H	Hour in day (0-23)	0
k	Hour in day (1-24)	24
K	Hour in am/pm (0-11)	0
h	Hour in am/pm (1-12)	12
m	Minute in hour	30
s	Second in minute	55
S	Millisecond	978
z	Time zone	Pacific Standard Time; PST; GMT-08:00
Z	Time zone	Time zone -0800

Year

- If the number of pattern letters is 4 or more, a calendar specific long form is used.
- Otherwise, a calendar specific short or abbreviated form is used.

Date and Time Pattern	Result
yy	10
yyyy	2010

Month

- If the number of pattern letters is less than 3, the number form is used.
- If the number of pattern letters is 3, a short or abbreviated form is used.
- If the number of pattern letters is 4 or more, the full form is used.

Date and Time Pattern	Result
M	4
MM	04
MMM	Apr
MMMM	April

Day in week

- If the number of pattern letters is 4 or more, the full form is used.

- Otherwise, a calendar specific short or abbreviated form is used

Date and Time Pattern	Result
E	Mon
EEEE	Monday

4.13 $profiling

A tool for accessing MagicDraw meta-model information.

$profiling.getGeneralizationName(modelName)

Returns a generalization model of modelName.

	Name	Type	Description
Parameter(s)	modelName	java.lang.String	A meta-model name.
Return	-	java.util.Collection<String>	A list of generalization model names.

$profiling.getDeclaringElementName (modelName, propertyName)

Retrieves the meta-model name which is the declared property name.

	Name	Type	Description
Parameter(s)	modelName	java.lang.String	A meta-model name.
	propertyName	java.lang.String	A property name.
Return	-	java.lang.String	A meta-model name.

$profiling.getPropertyTypeName (modelName, propertyName)

Retrieves property types from meta-model names.

	Name	Type	Description
Parameter(s)	modelName	java.lang.String	A meta-model name.
	propertyName	java.lang.String	A property name.
Return	-	java.lang.String	A property type name.

$profiling.getPropertyTypeName (element, propertyName)

Retrieves property types from an element.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
	propertyName	java.lang.String	A property name.
Return	-	java.lang.String	A property type name.

$profiling.getElementProperties(modelName)

Retrieves a collection of element property names from meta-model names.

	Name	Type	Description
Parameter(s)	modelName	java.lang.String	A meta-model name.
Return	-	java.util.Collection<String>	A collection of element property names.

$profiling.getElementProperties(element)

Retrieves a collection of element property names from elements.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
Return	-	java.util.Collection<String>	A collection of element property names.

$profiling.getElementProperty(element, propertyName)

Retrieves property values of specified elements and property names.

	Name	Type	Description
Parameter(s)	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element.
	propertyName	java.lang.String	A property name.
Return	-	java.lang.Object	A property object.

$profiling. getHumanPropertyName(element, propertyName)

Returns text representing a property name.

	Name	Type	Description
Parameter(s)	propertyName	java.lang.String	A property name.
	element	com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element	An element
Return	-	java.lang.String	Text representing a property name.

4.14 $image

$image is an image tool that provides a rich set of image manipulation methods that enable you to transform images during report generation. Images can be scaled, rotated, and resized.

Syntax

These manipulation methods can be broadly divided into five categories:

Multiple image transformations will have cumulative effects. Desired sizes can be specified either as pixel counts or as measurements in inches, centimeters, or millimeters. While the parameters for resizing an image with a specific measurement (in, cm, mm, pt, or px) must be quoted, all other size parameters do not need to be quoted.

The **keepRatio** parameter takes either a true or false value; true will resize the image and keep the original image height/width ratio; false will only resize one axis, as set by the method name, resulting in a stretched image.

4.14.1 Scaling

There are two scaling methods that can scale an image using a given factor (for example, 1.5). Method (i) provides height and width parameters (**scaleWidth** and **scaleHeight**), while Method (ii) provides a single parameter that will scale both axes equally (**scaleFactor**). These three parameters must be positive real numbers.

(i) $image.scale(image, scaleWidth, scaleHeight)

Returns an image icon for an element.

	Name	Type	Description
Parameter(s)	image	com.nomagic.magicreport.Image	An image object for the element icon.
	scaleWidth	java.lang.Double	The width parameter of the image.
	scaleHeight	java.lang.Double	The height parameter of the image.

	Name	Type	Description
Return	-	com.nomagic.magicreport.Image	The resized image object for the element icon.

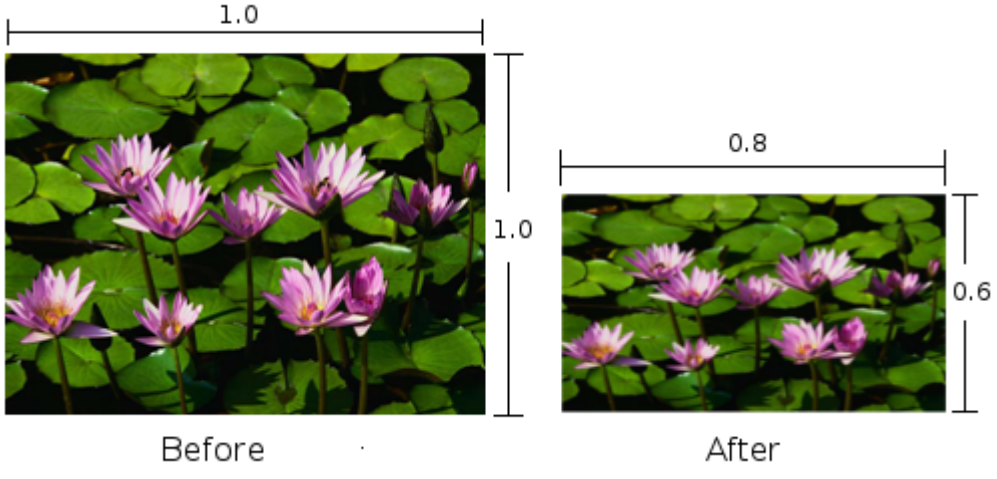
(ii) $image.scale(image, scaleFactor)

Returns an image icon for an element.

	Name	Type	Description
Parameter(s)	image	com.nomagic.magicreport.Image	An image object for the element icon.
	scaleFactor	java.lang.Double	The scaling parameter of the image.
Return	-	com.nomagic.magicreport.Image	The resized image object for the element icon.

Use **$image.scale($diagram.image, 0.5)**, for example, to scale down the image to half the original size. The following photos show the result.

Use **$image.scale($diagram.image, 0.8, 0.6)**to scale the image's width down to 80% and height to 60%. The following photos show the result.



Scaling Quality

Because the image tool provides several methods for scaling images, the quality of the scaled image depends on the used algorithm. To set the image scaling quality, **$image.setScalingQuality()** code must be inserted before scaling images. For example:

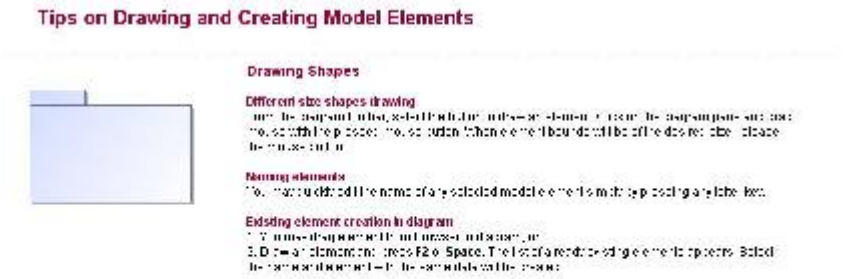
```
$image.setScalingQuality(5)
$image.scale(0.5)
```

The above code will set the image scaling quality to ‘highest’. The possible values are from 1 to 5, where 5 is ‘highest’ and 1 is ‘lowest’.

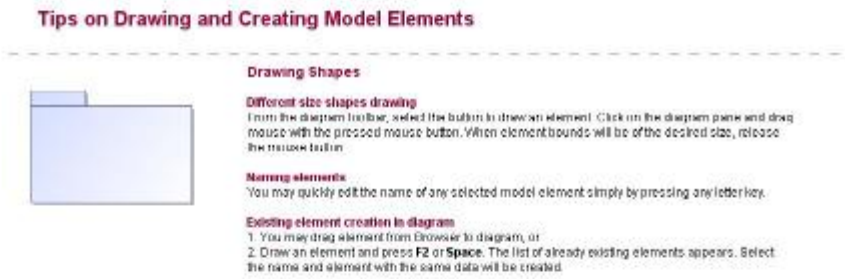
Value	Description
1	lowest' – Use one-step nearest neighbor interpolation.
2	'low' – Use one-step bi-cubic interpolation.
3	'medium – Use multi-step bi-linear interpolation.
4	'high' – Use area average image scaling algorithm.
5	'highest' – Use lossless transformation when the template is ODF, DOCX or RTF; otherwise, area average image scaling algorithm will be used instead.

Sample pictures from different image scaling qualities with their sizes scaled down to 50% are indicated below (ranging from the value 1 ‘lowest’ to 5 ‘highest’):

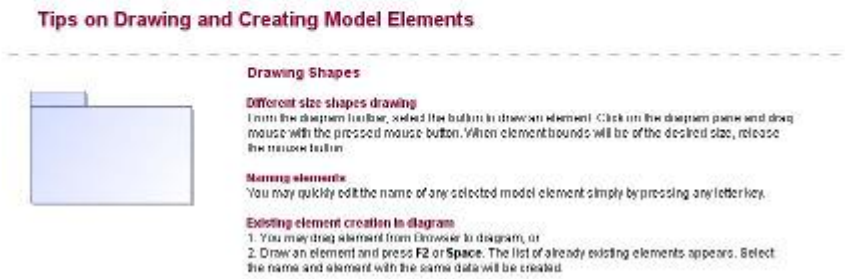
$image.setScalingQuality(1)



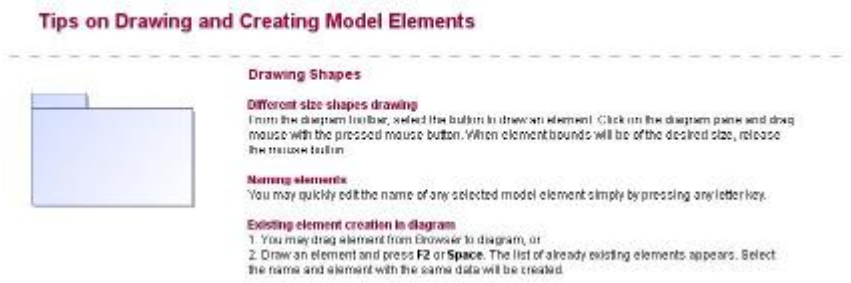
$image.setScalingQuality(2)



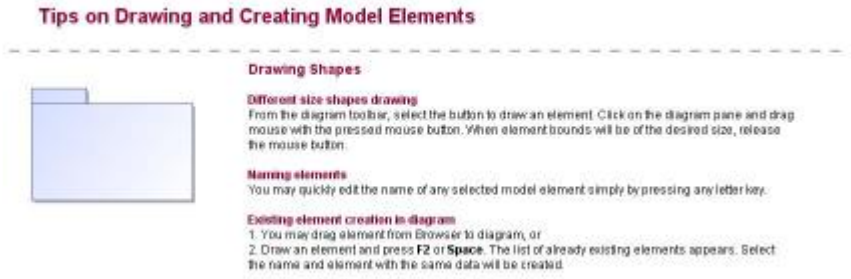
$image.setScalingQuality(3)



$image.setScalingQuality(4)



$image.setScalingQuality(5)



4.14.2 Rotating

There are two rotating methods that can rotate a given image by 90 degrees: (i) **rotateRight** for clockwise rotation and (ii) **rotateLeft** for counterclockwise rotation.

(i) $image.rotateRight(image)

Returns an image icon for an element.

	Name	Type	Description
Parameter(s)	image	com.nomagic.magicreport.Image	An image object for the element icon.
Return	-	com.nomagic.magicreport.Image	The rotated image object for the element icon.

(ii) $image.rotateLeft(image)

Returns an image icon for an element.

	Name	Type	Description
Parameter(s)	image	com.nomagic.magicreport.Image	An image object for the element icon.
Return	-	com.nomagic.magicreport.Image	The rotated image object for the element icon.

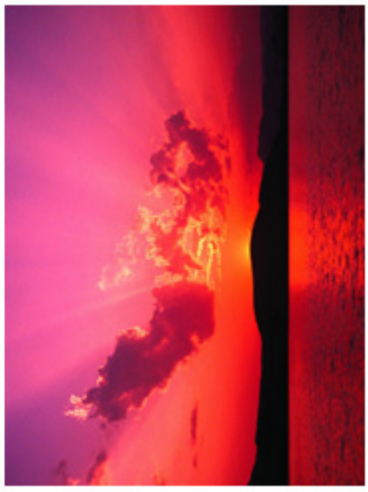
Use **$image.rotateRight($diagram.image)**, for example, to rotate the image 90 degrees clockwise. The following photos show the result.

Before

After

Use **$image.rotateLeft($diagram.image)**, for example, to rotate the image 90 degrees counterclockwise. The following photos show the result.

Before



After

NOTE	RTF Image rotation, with the scaling quality set to 'highest,' is not supported in any RTF document when opened with OpenOffice.org.

4.14.3 Fixed-Pixels Resizing

There are five methods to specify the resulting dimensions for an image to be resized in pixel size (the size parameters must be positive numbers or -1 number). If the value is -1, it will resize an image to the document paper dimensions. For example:

```
$image.setWidth($diagram.image, -1)
```

As shown by the example, the image will be resized to a paper width while maintaining the aspect ratio. The value -1 applies only to certain template types such as RTF, ODT, and DOCX.

If the value is -2, it will resize an image to document paper bounds if and only if image bounds are larger than paper bounds. For example:

```
$image.setWidth($diagram.image, -2)
```

Using the value -2 also maintains the image aspect ratio. However, it can only be applied to certain template types such as RTF, ODT, and DOCX.

(i) $image.setSize(image, sizeWidth, sizeHeight)

Returns an image icon for an element. This method is used to resize the image to an exact size; the width and height in pixels.

	Name	Type	Description
Parameter(s)	image	com.nomagic.magicreport.Image	An image object for the element icon.
	sizeWidth	java.lang.Integer	The width of the image in pixels.
	sizeHeight	java.lang.Integer	The height of the image in pixels.
Return	-	com.nomagic.magicreport.Image	The resized image object for the element icon.

(ii) $image.setHeight(image, size)

Returns an image icon for an element. This method is used to resize the image to a specific height (in pixels), while maintaining the image aspect ratio.

	Name	Type	Description
Parameter(s)	image	com.nomagic.magicreport.Image	An image object for the element icon.
	size	java.lang.Integer	The height of the image in pixels.
Return	-	com.nomagic.magicreport.Image	The resized image object for the element icon.

(iii) $image.setHeight(image, size, keepRatio)

Returns an image icon for an element. This method is used to resize the image to a specific height (in pixels) and to specify whether the image aspect ratio is to be maintained or not (depending on the **keepRatio** parameter).

	Name	Type	Description
Parameter(s)	image	com.nomagic.magicreport.Image	An image object for the element icon.
	size	java.lang.Integer	The height of the image in pixels.
	keepRatio	java.lang.Boolean	A flag to maintain the image aspect ratio

	Name	Type	Description
Return	-	com.nomagic.magicreport.Image	The resized image object for the element icon.

(iv) $image.setWidth(image, size)

Returns an image icon for an element. This method is used to resize the image to a specific width (in pixels), while maintaining the image aspect ratio.

	Name	Type	Description
Parameter(s)	image	com.nomagic.magicreport.Image	An image object for the element icon.
	size	java.lang.Integer	The width of the image in pixels.
Return	-	com.nomagic.magicreport.Image	The resized image object for the element icon.

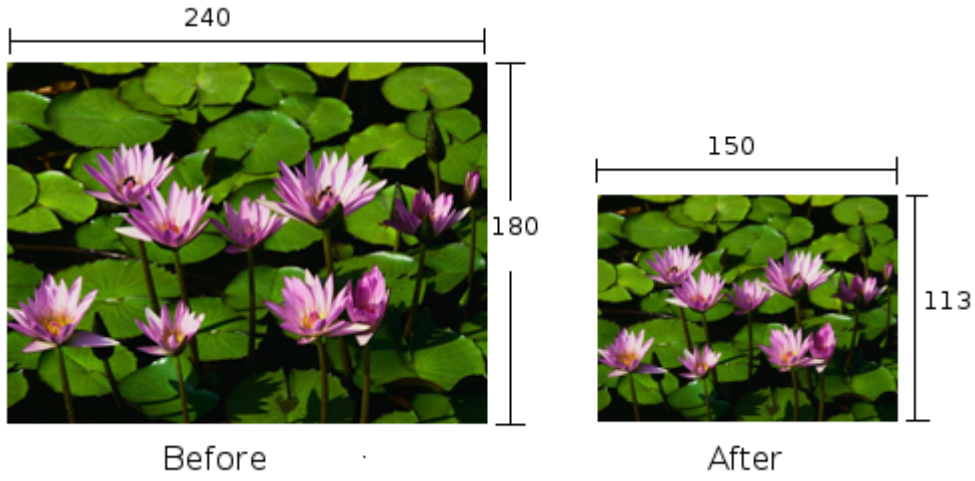
(v) $image.setWidth(image, size, keepRatio)

Returns an image icon for an element. This method is used to resize the image to a specific width (in pixels) and to specify whether the image aspect ratio is to be maintained or not (depending on the **keepRatio** parameter).

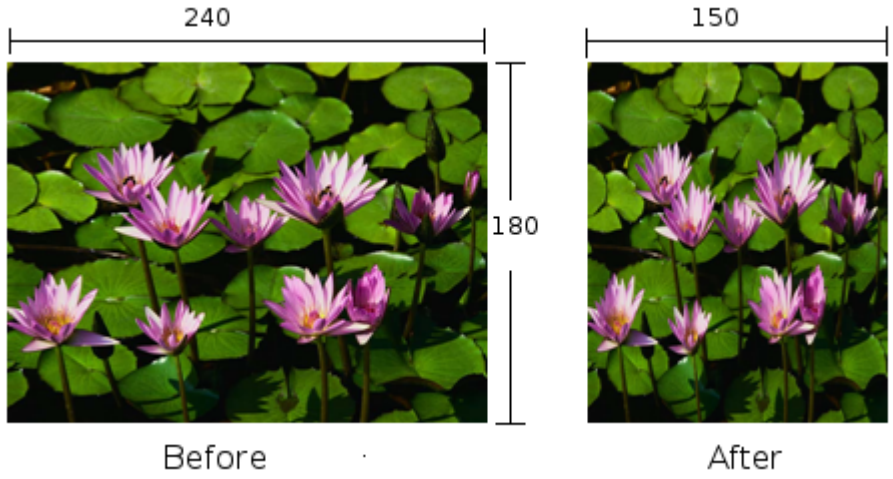
	Name	Type	Description
Parameter(s)	image	com.nomagic.magicreport.Image	An image object for the element icon.
	size	java.lang.Integer	The width of the image in pixels.
	keepRatio	java.lang.Boolean	A flag to maintain the image aspect ratio
Return	-	com.nomagic.magicreport.Image	The resized image object for the element icon.

Use **$image.setSize($diagram.image, 200, 200)**, for example, to resize the image's width and height to 200 pixels. The following photos show the result.

Use either **$image.setWidth($diagram.image, 150)** or **$image.setWidth($diagram.image, 150, true)**, for example, to resize the image's width to 150 pixels and maintain the image aspect ratio. The following photos show the result.



Use **$image.setWidth($diagram.image, 150, false)**, for example, to resize the image's width to 150 pixels and ignore the image aspect ratio. The following photos show the result.



4.14.4 Fixed-Measurement Resizing

There are six methods to specify the resulting dimensions in terms of measurements to resize an image, for example, in inch (in), centimeter (cm), millimeter (mm), point (pt), or pixel (px)], and specifying the dot-per-inch (DPI) value. Every measurement parameter must be quoted.

(i) $image.setSize(image, measureWidth, measureHeight)

Returns an image icon for an element. This method is used to resize the image to an exact size; the width and height in terms of measurements.

	Name	Type	Description
Parameter(s)	image	com.nomagic.magicreport.Image	An image object for the element icon.

	Name	Type	Description
Parameter(s)	measureWidth	java.lang.String	The width of the image in a term of measurement.
	measure-Height	java.lang.String	The height of the image in a term of measurement.
Return	-	com.nomagic.magicreport.Image	The resized image object for the element icon.

(ii) $image.setHeight(image, measureSize)

Returns an image icon for an element. This method is used to resize the image to a specific height in terms of measurement while maintaining the image aspect ratio.

	Name	Type	Description
Parameter(s)	image	com.nomagic.magicreport.Image	An image object for the element icon.
	measureSize	java.lang.String	The height of the image in a term of measurement.
Return	-	com.nomagic.magicreport.Image	The resized image object for the element icon.

(iii) $image.setHeight(image, measureSize, keepRatio)

Returns an image icon for an element. This method is used to resize the image to a specific height (a term of measurement), and to specify whether the image aspect ratio is to be maintained or not (depending on the **keepRatio** parameter).

	Name	Type	Description
Parameter(s)	image	com.nomagic.magicreport.Image	An image object for the element icon.
	measureSize	java.lang.String	The height of the image in a term of measurement.
	keepRatio	java.lang.Boolean	A flag to maintain the image aspect ratio
Return	-	com.nomagic.magicreport.Image	The resized image object for the element icon.

(iv) $image.setWidth(image, measureSize)

Returns an image icon for an element. This method is used to resize the image to a specific width in a term of measurement, while maintaining the image aspect ratio.

	Name	Type	Description
Parameter(s)	image	com.nomagic.magicreport.Image	An image object for the element icon.
	measureSize	java.lang.String	The width of the image in a term of measurement.
Return	-	com.nomagic.magicreport.Image	The resized image object for the element icon.

(v) $image.setWidth(image, measureSize, keepRatio)

Returns an image icon for an element. This method is used to resize the image to a specific width (a term of measurement), and to specify whether the image aspect ratio is to be maintained (depending on the **keepRatio** parameter).

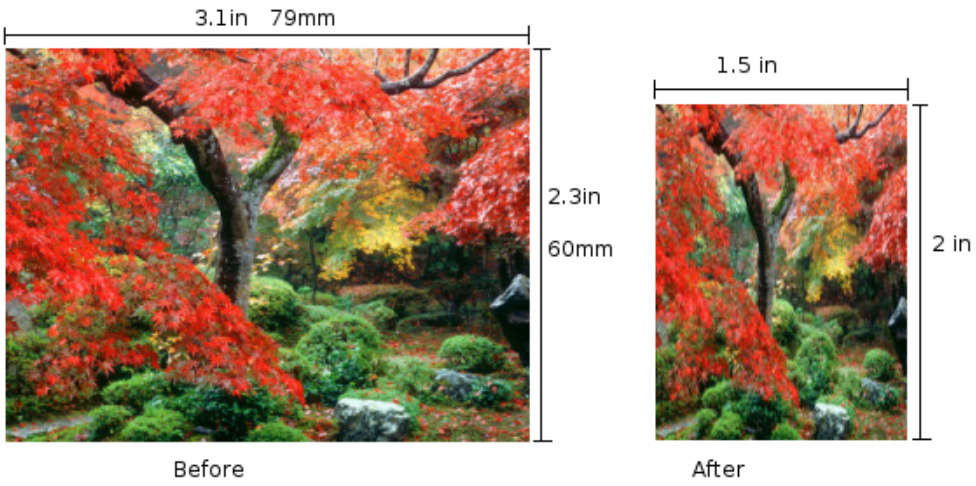
	Name	Type	Description
Parameter(s)	image	com.nomagic.magicreport.Image	An image object for the element icon.
	measureSize	java.lang.String	The width of the image in a term of measurement.
	keepRatio	java.lang.Boolean	A flag to maintain the image aspect ratio.
Return	-	com.nomagic.magicreport.Image	The resized image object for the element icon.

(vi) $image.setDPI(dotsPerInches)

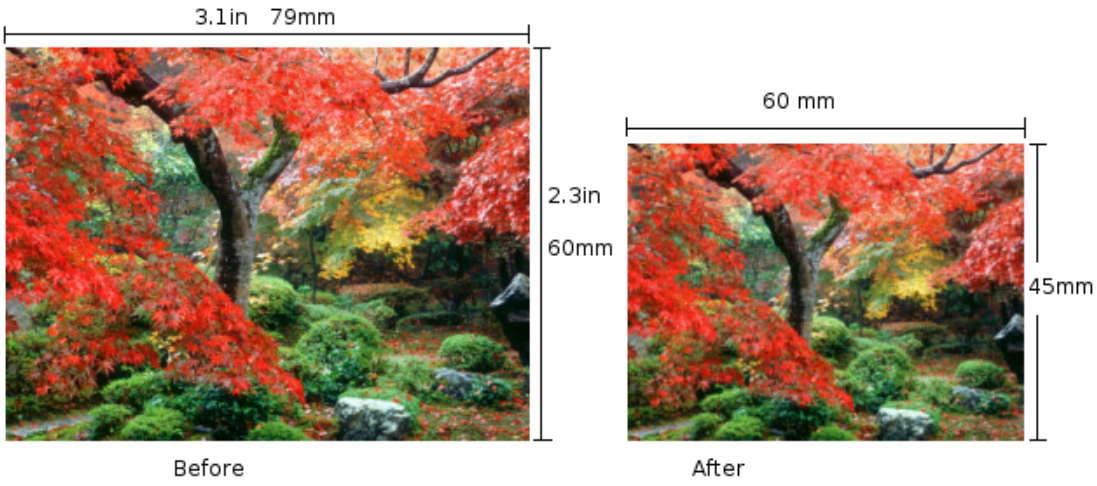
This method is used to set the resolution (dpi) of images. The default value is 96 dpi. The dpi value will only affect the methods that take inches, centimeters, and millimeters as their parameters.

	Name	Type	Description
Parameter(s)	dotsPerInches	java.lang.Integer	The resolution (dpi) of images
Return	-	-	-

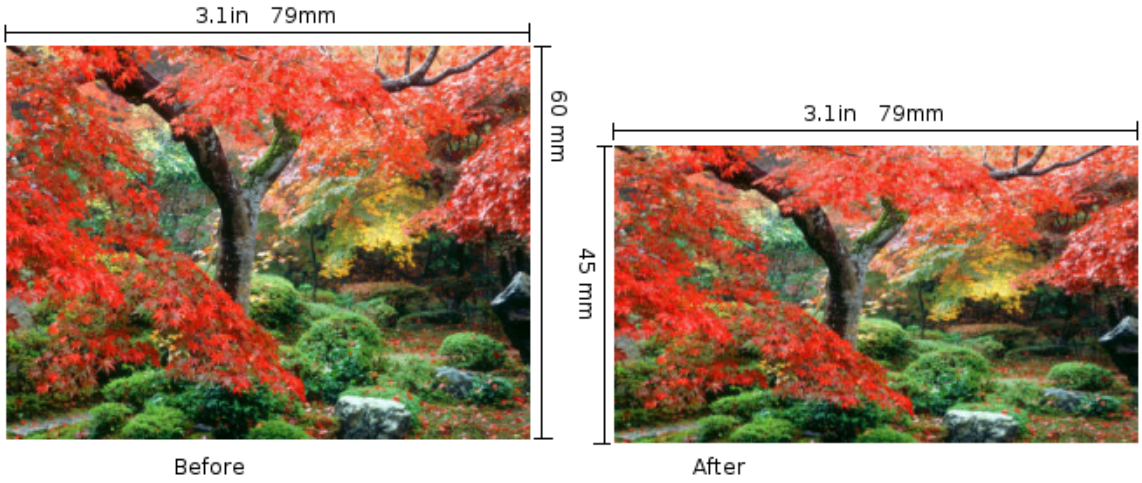
Use **$image.setSize($diagram.image, '1.5in', '2in')**, for example, to resize the image's width to 1.5 inches and height to 2 inches. The following photos show the result.



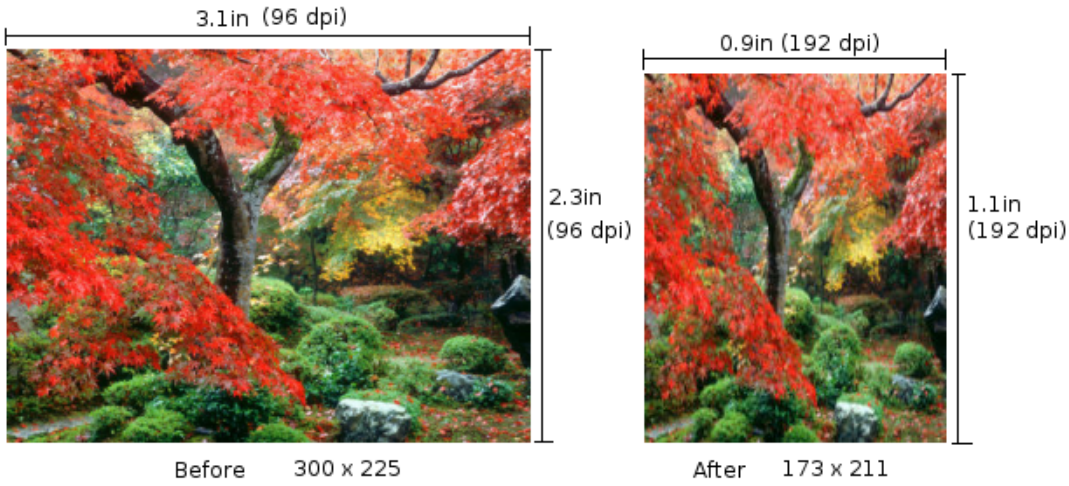
Use either **$image.setHeight($diagram.image, '45mm')** or **$image.setHeight($diagram.image, '45mm', true)**, for example, to resize the image's height to 45 millimeters and maintain the image aspect ratio. The following photos show the result.



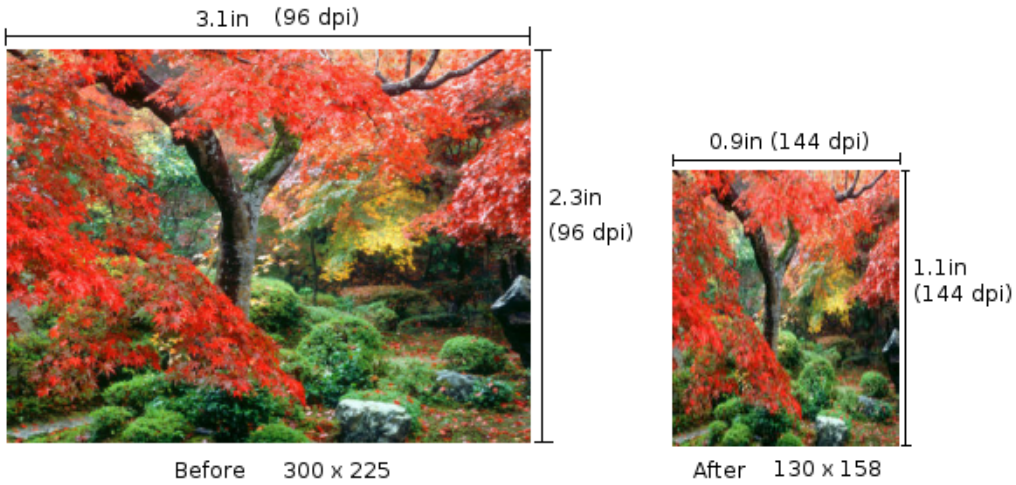
Use **$image.setHeight($diagram.image, '45mm', false)**, for example, to resize the image's height to 45 millimeters and ignore the image aspect ratio. The following photos show the result.



Use **$image.setDPI(192)** and then **$image.setSize($diagram.image, '0.9in', '1.1in')**, for example, to set the resolution (dpi) of the image to 192 dots per inch, and then resize the image width to 0.9 inches and height to 1.1 inches. The resulting image size will become 173x211 pixels. The following photos show the result.



Instead of using **$image.setDPI(192)**, if you set the resolution (dpi) to 144 using **$image.setDPI(144)**, the resulting image size will become 130x158 pixels. The following photos show the result.



4.14.5 Include Image

You can include images from an external source in a report. The supported image types are BMP, JPG, WBMP, GIF, and PNG.

(i) $image.include(location)

Includes an external image from a local file or URL in a report.

	Name	Type	Description
Parameter(s)	location	java.lang.String	An image's location. The location format can be either a URL or a local file.
Return	-	com.nomagic.magicreport.Image	An image included in a report.

For example:

```
$image.include("c:/my document/logo.gif")
$image.include("http:/www.magicdraw.com/images/
product_boxes/MD.gif")
```

5. Report Wizard Template Editor

Template Editor is a Microsoft Word Add-In. It provides a list of fields that can be used in a Report Wizard template.

5.1 Installation

To install Report Wizard Template Editor in Microsoft Word:

1. Download and install Microsoft XML Services (MSXML) from the following URL (You can skip this step if the latest version of MSXML is already installed in your system):

   http://www.microsoft.com/downloads/details.aspx?FamilyID=d21c292c-368b-4ce1-9dab-3e9827b70604&DisplayLang=en

2. Copy all files and subdirectories from the **`<install.dir>\plugins\com.nomagic.magicdraw.reportwizard\vba`** folder to **`%APPDATA%\Microsoft\Word\STARTUP`** (%APPDATA% is the location where user data is stored). For example,
   - On Windows 2000 and Windows XP: **`C:\Documents and Settings\User Name\Application Data\Microsoft\Word\STARTUP`**
   - On Windows Vista: **`C:\Users\UserName\AppData\Roaming\Microsoft\Word\STARTUP`**
3. Restart Microsoft Word.

5.2 Opening Template Editor

After installation has been completed, the Template Editor menu will appear on the Microsoft Word menu bar (Figure 72).

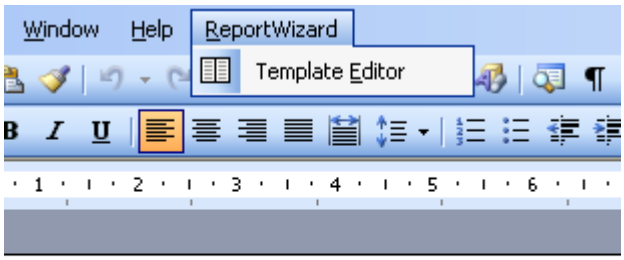


Figure 72 -- Microsoft Word 2003 Template Editor Menu

To open Template Editor:

- On the Microsoft Word 2000 – 2003 menu, click **ReportWizard** > **Template Editor** (Figure 72) or
- On the Microsoft Word 2007 menu, click **Add-Ins** > **ReportWizard** > **Template Editor** (Figure 73).

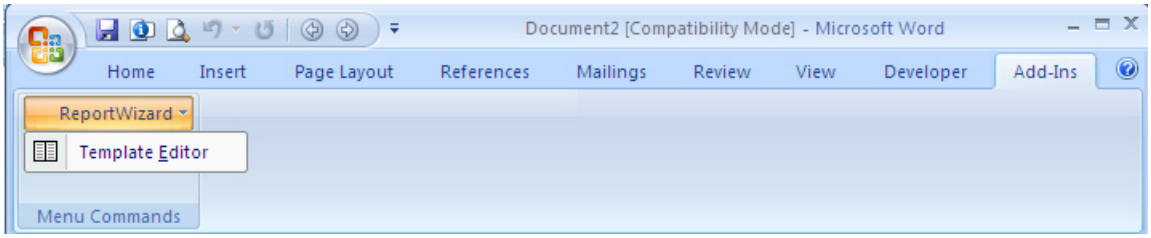


Figure 73 -- Microsoft Word 2007 Template Editor Menu

NOTE	The macro enabled option in Microsoft Word is required to open Template Editor.

When Template Editor is open, the Report Wizard Template Editor Dialog will show up (Figure 74).

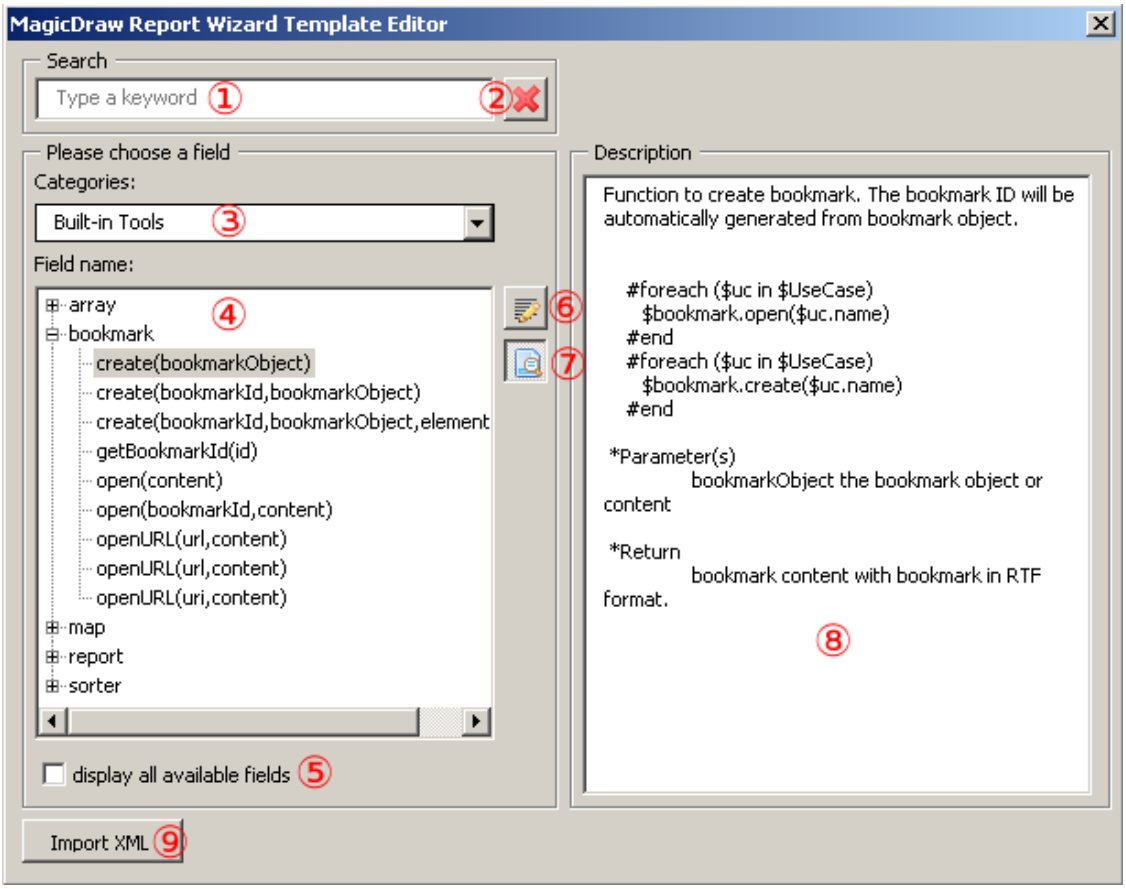


Figure 74 -- Report Wizard Template Editor Dialog

Table 14 -- Report Wizard Template Editor

Name	Function
Search box	To filter a list of fields. Only fields that contain a keyword of search as part of their names can be shown in the *(4) List of fields.*
Clear search results	To clear the current search result.
Categories combo box	To select categories of fields. Fields are shown in *(4) List of fields* according to their categories.
List of fields	To show a list of fields. Double-click a field name to insert the code.
Display all available fields check box	To show all fields, otherwise it will show only commonly used fields.
Insert button	To insert a code for a selected field into the document.
Show/Hide description button	To show or hide *(8) description pane*.
Description pane	To show the description of a selected field.
Import button	To import data file.

5.3 Data File

Template Editor allows you to create a new data file and import it to the **Report Wizard Template Editor** dialog. Click the **Import** button to import the data file.

Data file is written in the XML format.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="data">
   <xs:complexType>
    <xs:sequence>
     <xs:element minOccurs="0" maxOccurs="1" name="elements">
      <xs:complexType>
       <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="unbounded" name="element">
         <xs:complexType>
          <xs:all>
           <xs:element minOccurs="1" maxOccurs="1" name="name" type="xs:string" />
           <xs:element minOccurs="0" maxOccurs="1" name="description" type="xs:string" />
           <xs:element minOccurs="0" maxOccurs="1" name="members">
            <xs:complexType>
             <xs:sequence>
             <xs:element minOccurs="0" maxOccurs="unbounded" name="function">
              <xs:complexType>
               <xs:sequence>
                <xs:element minOccurs="1" maxOccurs="1" name="name" type="xs:string" />
                <xs:element minOccurs="0" maxOccurs="1" name="description" type="xs:string" />
                <xs:element minOccurs="0" maxOccurs="unbounded" name="param">
                 <xs:complexType>
                  <xs:all>
                   <xs:element minOccurs="1" maxOccurs="1" name="name" type="xs:string" />
                   <xs:element minOccurs="0" maxOccurs="1" name="description" type="xs:string" />
                   <xs:element minOccurs="0" maxOccurs="1" name="type" nillable="true" type="xs:string" />
                   <xs:element minOccurs="0" maxOccurs="1" name="direction">
                    <xs:simpleType>
                     <xs:restriction base="xs:string">
                      <xs:enumeration value="in" />
                      <xs:enumeration value="out" />
                     </xs:restriction>
                    </xs:simpleType>
                   </xs:element>
                  </xs:all>
                 </xs:complexType>
                </xs:element>
               </xs:sequence>
              </xs:complexType>
             </xs:element>
             <xs:element minOccurs="0" maxOccurs="unbounded" name="attribute">
              <xs:complexType>
               <xs:all>
                <xs:element minOccurs="1" maxOccurs="1" name="name" type="xs:string" />
                <xs:element minOccurs="0" maxOccurs="1" name="description" type="xs:string" />
                <xs:element minOccurs="0" maxOccurs="1" name="type" nillable="true" type="xs:string" />
               </xs:all>
              </xs:complexType>
             </xs:element>
            </xs:sequence>
           </xs:complexType>
          </xs:element>
          </xs:all>
          <xs:attribute name="often" type="xs:boolean" />
         </xs:complexType>
        </xs:element>
       </xs:sequence>
      </xs:complexType>
     </xs:element>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" />
   </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 75 -- XML Data File Schema

Table 15 -- XML Data File Schema Elements and Their Descriptions

Element	Attribute / Element	Description
data		The root element.
	name : string	Data name. This value will be used as a category name *(3) Category.*
	elements : complexType	Group of <element> element.
elements		Group of <element> element.
element		Element display on *(4) List of fields.*
	name : string	Element name.
	description : string	Element description.
	often : boolean	If an attribute is true, the element will be managed as a suggested element.
	members : complexType	Group of <function> or <attribute>
function		The function member.
	name : string	Function name.
	description : string	Function description.
	param : complexType	Group of function parameters.
param		Function parameter.
	name : string	Parameter name.
	description : string	Parameter description.
	type : string	Parameter type.
	direction : simpleType	A direction of parameter. in - indicates this parameter is an input out - indicates this parameter is an output
attribute		The attribute member.
	name : string	Attribute name.
	description : string	Attribute description.
	type : string	Attribute type.

```
<?xml version="1.0" encoding="UTF-8"?>
<data name="Built-in Tools" xmlns:xsi="http://www.w3.org/2001/XMLSchemainstance"
xsi:noNamespaceSchemaLocation="fields.xsd">
  <elements>
    <element often="true">
      <name>array</name>
      <description>Use for creating the Array or HashSet instance.</description>
      <members>
        <function>
          <name>subList</name>
          <description>Create ArrayList of the portion of list in given size.</
description>
          <param>
            <name>list</name>
            <direction>in</direction>
            <type>List</type>
            <description>a original list.</description>
          </param>
          <param>
            <name>size</name>
            <direction>in</direction>
            <type>int</type>
            <description>view size of given list</description>
          </param>
          <param>
            <name>list</name>
            <direction>out</direction>
            <type/>
           <description>a of view of the specified size within given list.</description>
          </param>
        </function>
      </members>
    </element>
  </elements>
</data>
```

Figure 76 -- Sample XML Data File

6. Generating Reports from Report Wizard

In addition to default document templates, Report Wizard allows you to create customized specification document templates. Templates may contain your own custom fields not related to model elements, as well as fields that correspond to model elements. Once customized, a Report Wizard template can be used with data from any project.

You can also format your template to create the style you want including tables of contents, headers and footers, and page numbers. You can apply most character and paragraph formatting available from your rtf editor for the rtf template or specify with html tags in the HTML template, including keywords. Once the template has been completed, you can run Report Wizard to create a report in a format that corresponds to the template.

Report Wizard templates can be in txt, rtf, html, odt, odf, odp, docx, xlsx, pptx, and xml for DocBook or FO files. If you prefer, you can work in Report Wizard Template Editor until you have finished the template file and use the Report Wizard template windows to save it in the template folder. The default templates are located in the *<MagicDraw home>/plugins/com.nomagic.magicdraw.reportwizard/data* folder.

6.1 Concepts

Report Wizard includes the following concepts:

- **Template**: specifies the document layout, format, corresponding model elements, and custom defined fields.
- **Report Data**: specifies a set of custom fields and data in the selected template. A template can have a number of different custom defined fields for a project or different projects organized in a report.
- **Report**: a generated user document with data from the project displayed in the selected template style.

6.2 Default Templates

There are 13 default templates that come with the report engine.

(i) Class Specification Report: describes the specification of class, interface, and enumeration type of the project.

(ii) Data Dictionary Report: enables you to print elements of MagicDraw data dictionary.

(iii) IEEE 1233: the standard template based on the IEEE 1233 recommendation.

(iv) Model Extension: describes common UML model extension mechanisms (stereotypes, tagged values, and constraints). It is useful for reporting custom UML profiles.

(v) Use Case (Simple): describes system functionalities and actors in a simple format.

(vi) Use Case (Modern): describes system functionalities and actors that are useful in support of use-case driven analysis.

(vii) Web Publisher (Simple HTML): enables you to publish MagicDraw models (including diagrams) in simple HTML. The report is viewable using a standard browser such as Internet Explorer or Firefox.

(viii) Web Publisher 2.0: enables you to publish MagicDraw models (including diagrams) in HTML with rich features. The report is viewable using a standard browser such as Internet Explorer or Firefox.

(ix) Activity Diagram: provides a standard report for activity diagrams. The content of the report is a list of all the nodes (ordered by node hierarchy in each activity diagram).

(x) Activity Diagram Specification: provides a standard report for activities. The content of the report is a list of all the nodes in each activity.

(xi) Sequence Diagram: provides a report that shows sequence diagrams and a list of all the messages in the diagrams.

(xii) Sequence Diagram Specification: provides a report that shows sequence diagrams, and a list of all the messages and lifelines in the diagrams.

(xiii) Web Publisher Collaboration: is implemented based on the Web Publisher 2.0 template. It improves features for text editing, element review, and XML messages. The J2EE compliance server is required to run the generated report.

6.3 Architecture Templates

There are five architecture report templates that come with the Report engine: (i) Behavioral, (ii) Environment, (iii) Implementation, (iv) Structural, and (v) Use Case report templates.

6.3.1 Behavioral Report

A Behavioral report is a standard report for the Behavioral view. This report will be generated by a selected package that consists of Sequence, Communication, State Machine, and Activity diagrams. The content of the report is a list of all packages which are ordered by a package hierarchy in the project. Each package consists of the specification of messages in the Sequence diagram, the specification of lifelines in the Communication diagram, the specification of states in the State Machine diagram, and the actions in the Activity diagram. All elements in the report are ordered by their names.

6.3.2 Environment Report

An Environment report is a standard report for the Environment view. The report will be generated by a selected package that consists of an Implementation diagram (Deployment diagram). The content of the report is the list of all packages which are ordered by a package hierarchy. Each package describes the specification of nodes, interfaces, components, and artifacts which are ordered by the element’s name.

6.3.3 Implementation Report

An Implementation report is a standard report for the Implementation view. The report will be generated by a selected package which consists of an Implementation diagram (Component diagram). The content of the report is a list of all packages which are ordered by a package hierarchy in the project. Each package consists of a list of interfaces, components, and artifacts. All elements in the report are ordered by their names.

6.3.4 Structural Report

A Structural report is a standard report for the Structural view. This report will be generated by a selected package which consists of a Class diagram. The report consists of a list of packages which are ordered by a package hierarchy. Each package describes the specification of interfaces, classes, and enumerations which are ordered by the element’s name.

6.3.5 Use Case Report

A Use Case report is a standard report for the Use Case view. This report will be generated by a selected package which consists of Use Case diagrams. The content of the report is a list of packages which are ordered by a package hierarchy. Each package describes the specification of actors and use case. All elements in the report are ordered by their names.

6.4 Generating Use Case Description Reports

The **Use Case Description** document template is designed to capture all the Use Case-related artifacts, for example, UseCases, actors, Use Case descriptions, etc. In this template, all the Use Case diagram-related keywords are included. Therefore, when you select the highest level of the packages (the highest level of the packages in MagicDraw is **Data**), only Use Case diagrams and their related elements will be added to the document.

To generate the Use Case description document:

1. Open the **Magic Library.mdzip** sample project from the **<MagicDraw_home>/samples/case studies** directory (Figure 77).

NOTE	This step is only necessary for illustration purposes (as well as other steps within this scenario).

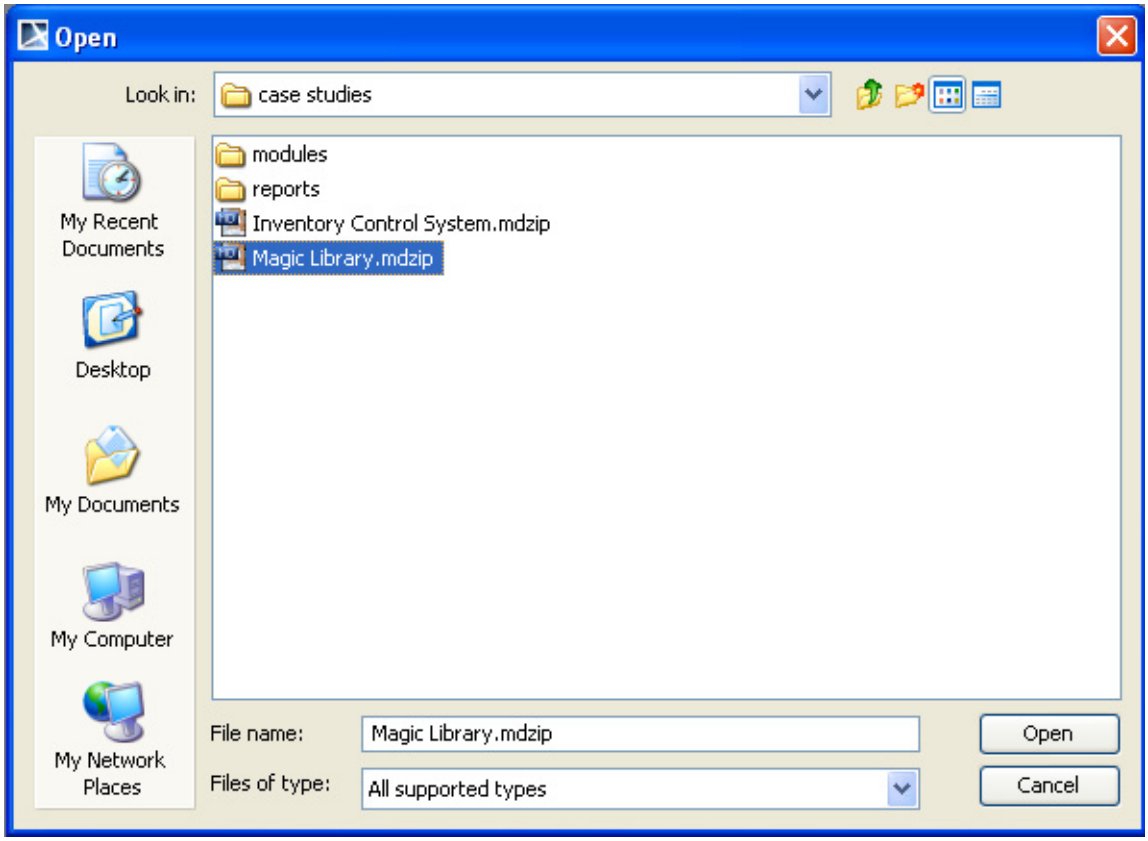


Figure 77 -- The Magic Library.mdzip Project File

2. On the **Tools** menu, select the **Report Wizard** command. The **Report Wizard** dialog will open. Select a template, for example, UseCase (Classic) and click **Next** (Figure 78).

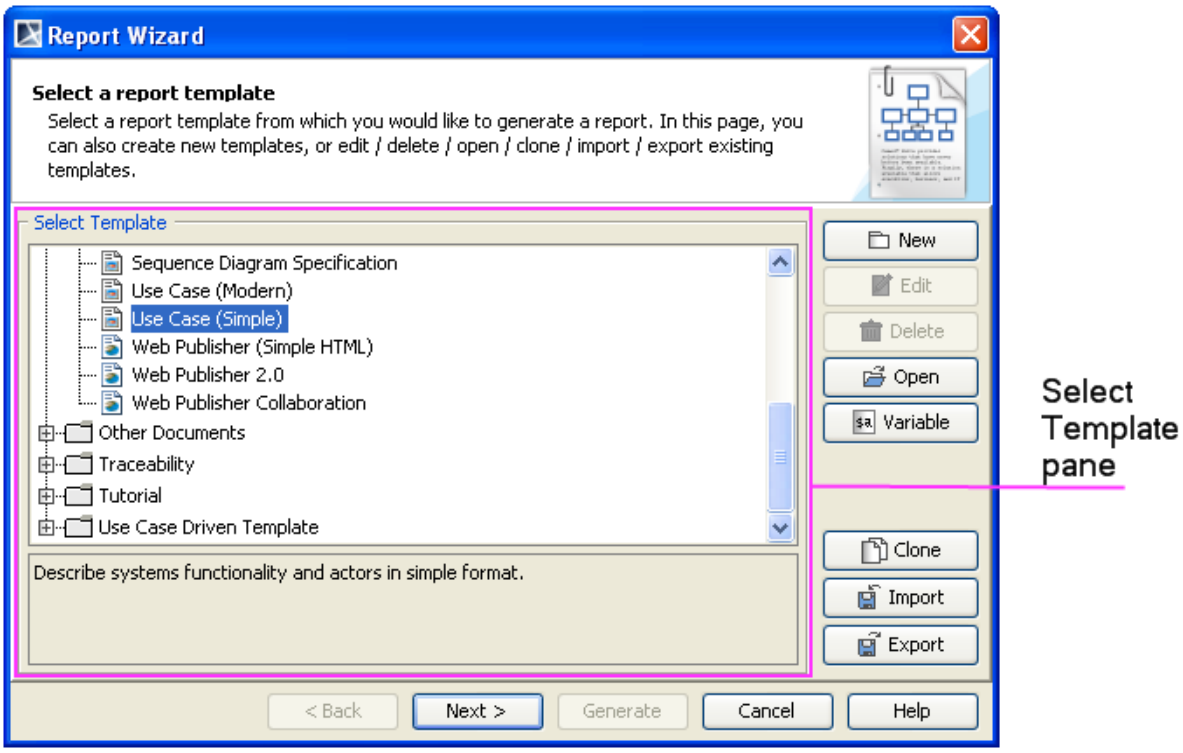


Figure 78 -- Report Wizard Dialog

3. Select the built-in report data (Figure 79) or create a new one (Figure 80), and then click **Next**.

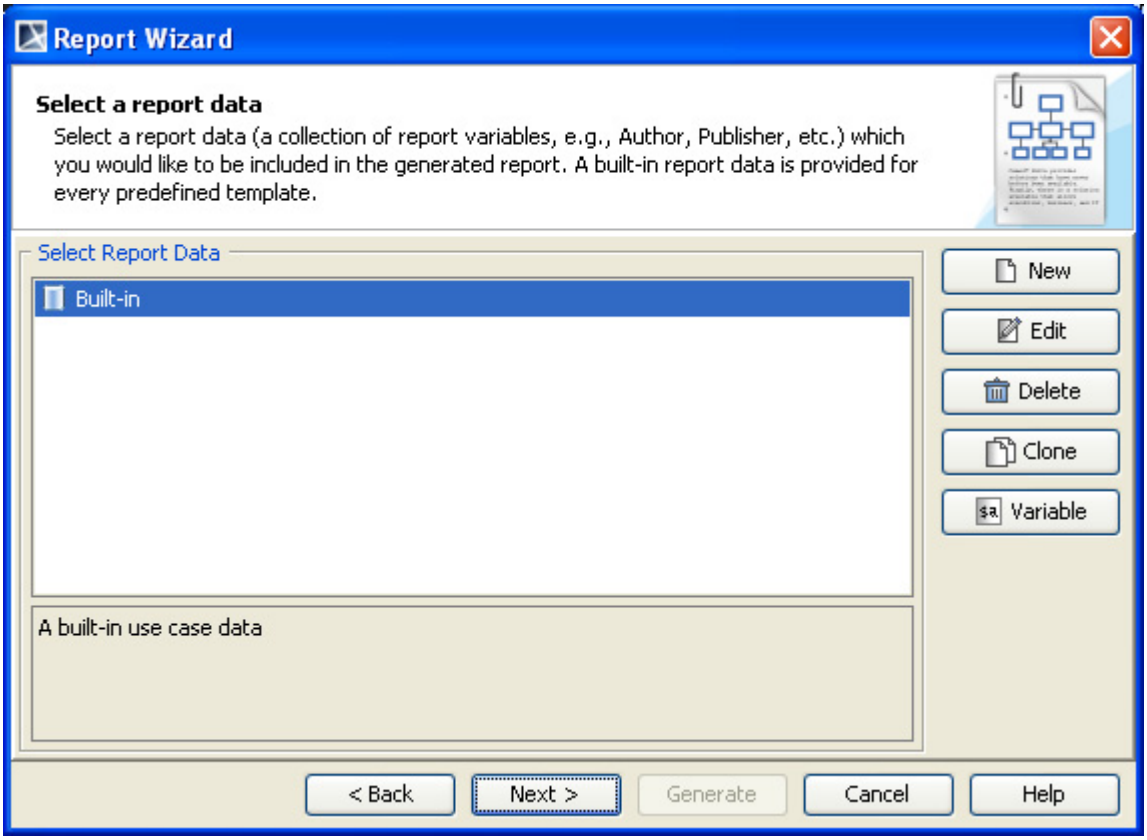


Figure 79 -- The Select Report Data Pane

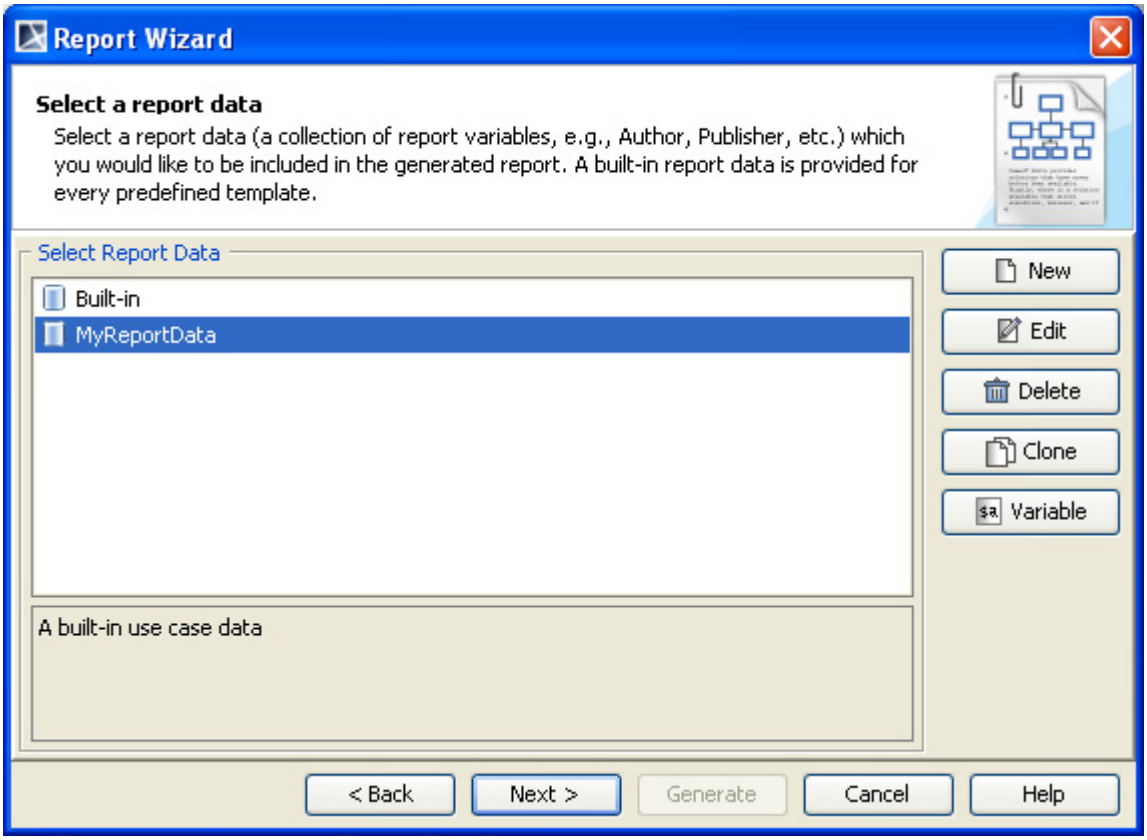


Figure 80 -- Selecting My Report Data

4. Click the **Variable** button (Figure 79) to create a new report variable, modify or delete an existing report variable with the use of the **Report Variable** dialog (Figure 81).

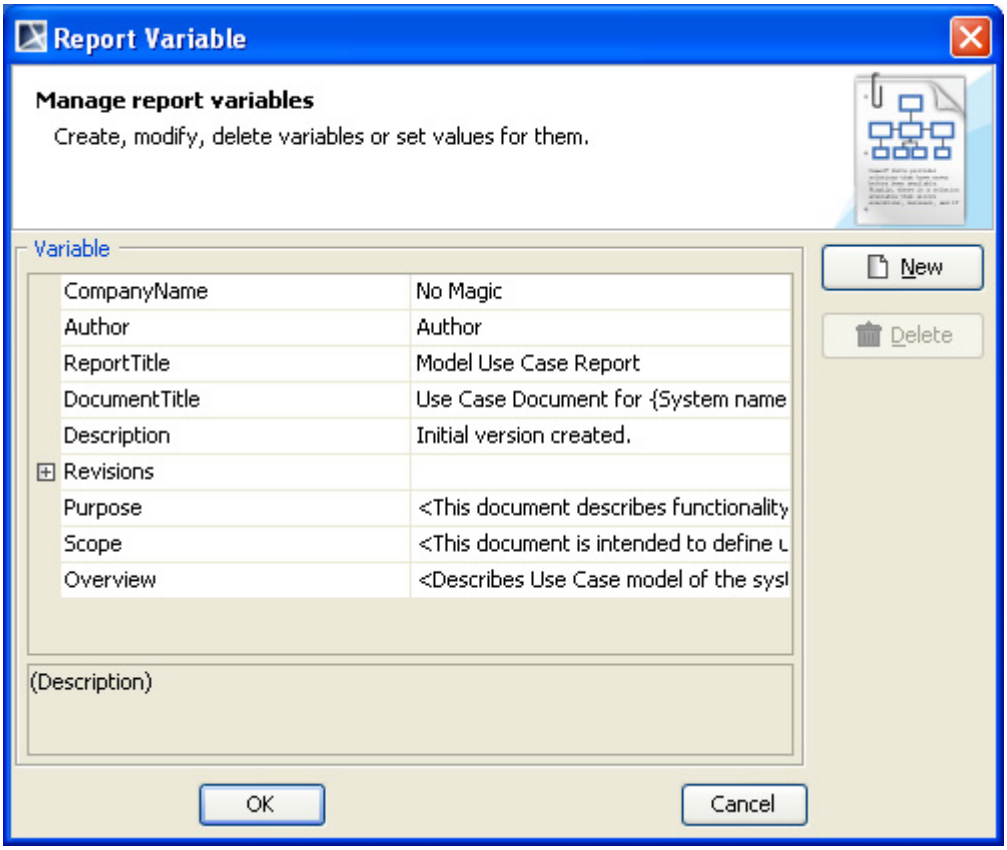


Figure 81 -- The Variable Pane

5. Select the scope of the report in the open package tree. In this case, it will be the **MagicLibrary Requirements** model package because all requirement of use cases are stored in this package. Click **Next** (Figure 82).

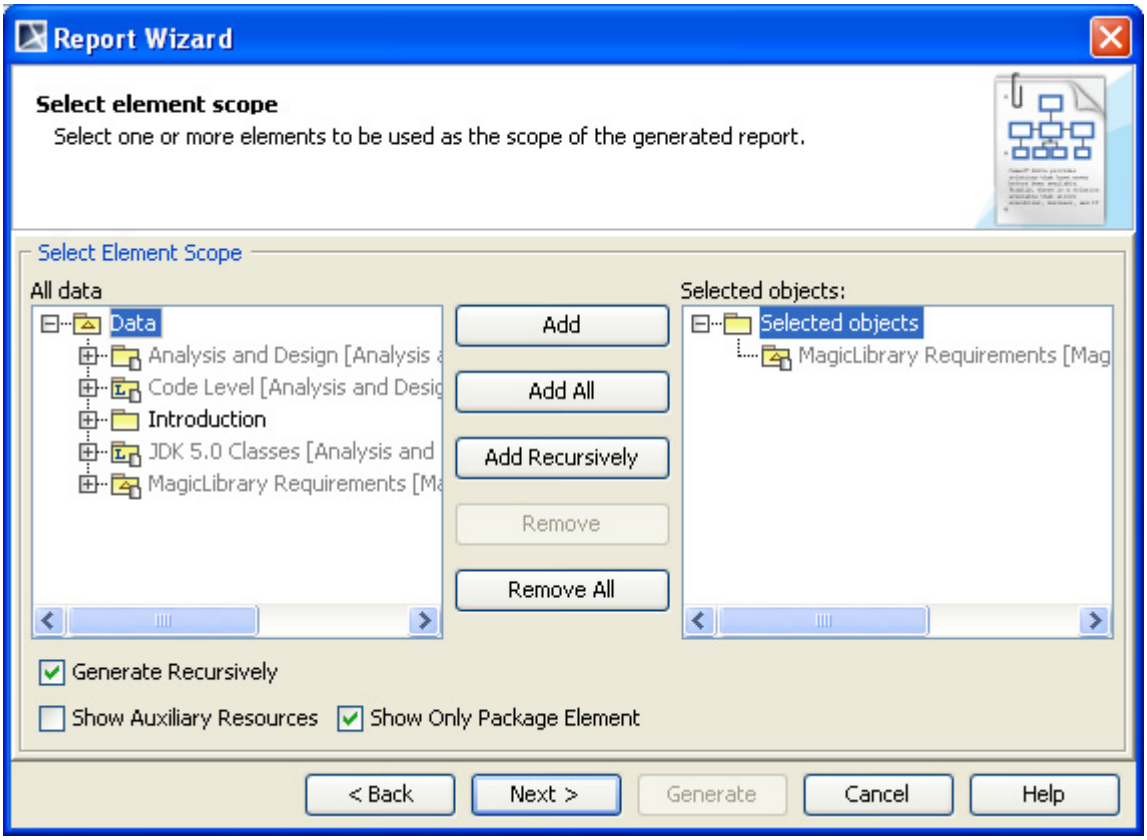


Figure 82 -- Selecting the Scope of the Report

6. Click the "**...**" button next to the **Report file** text box to locate the report file location (Figure 83). The **Save as** dialog will open.
7. Select the file location, enter the filename, and click **Save**. The report file format will depend on the template file format. For example, if the template file format is **.rtf,* the report file format will be **.rtf* as well.

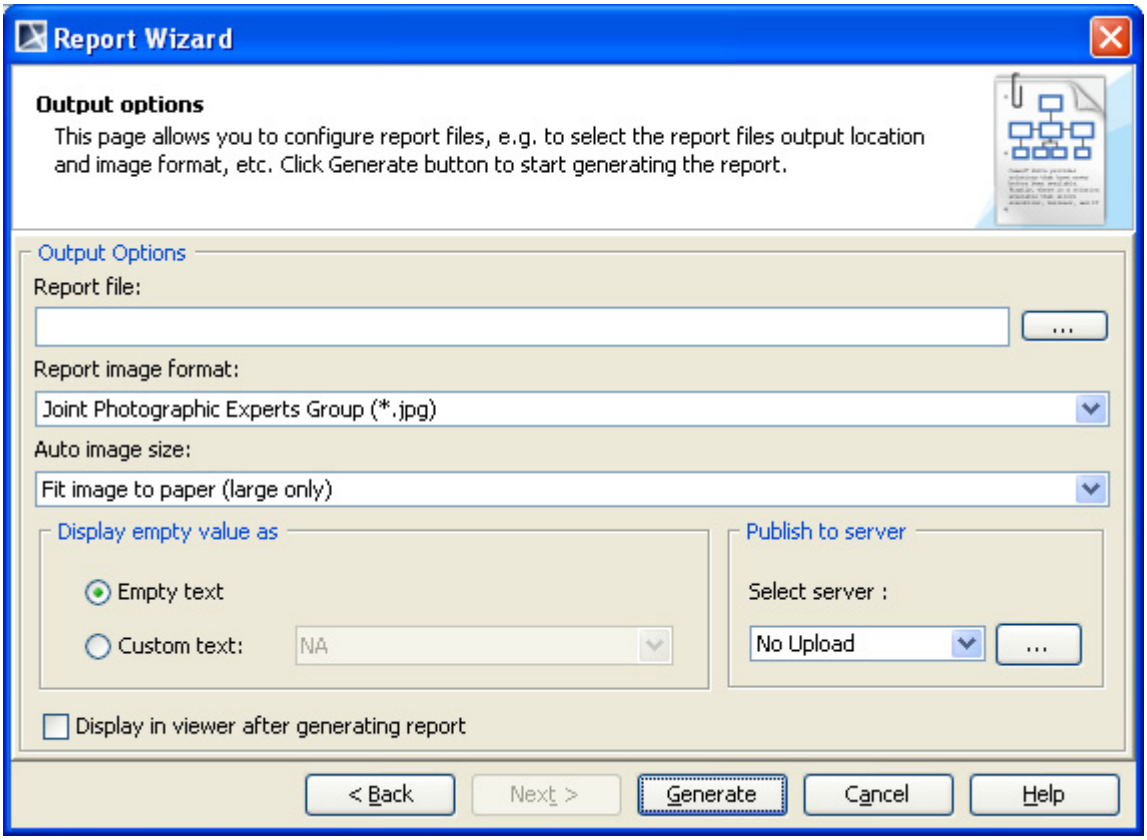


Figure 83 -- Selecting the Report File Location

8. Select the report image format, either **.png* or **.jpg,* and select an image size option (Figure 84).

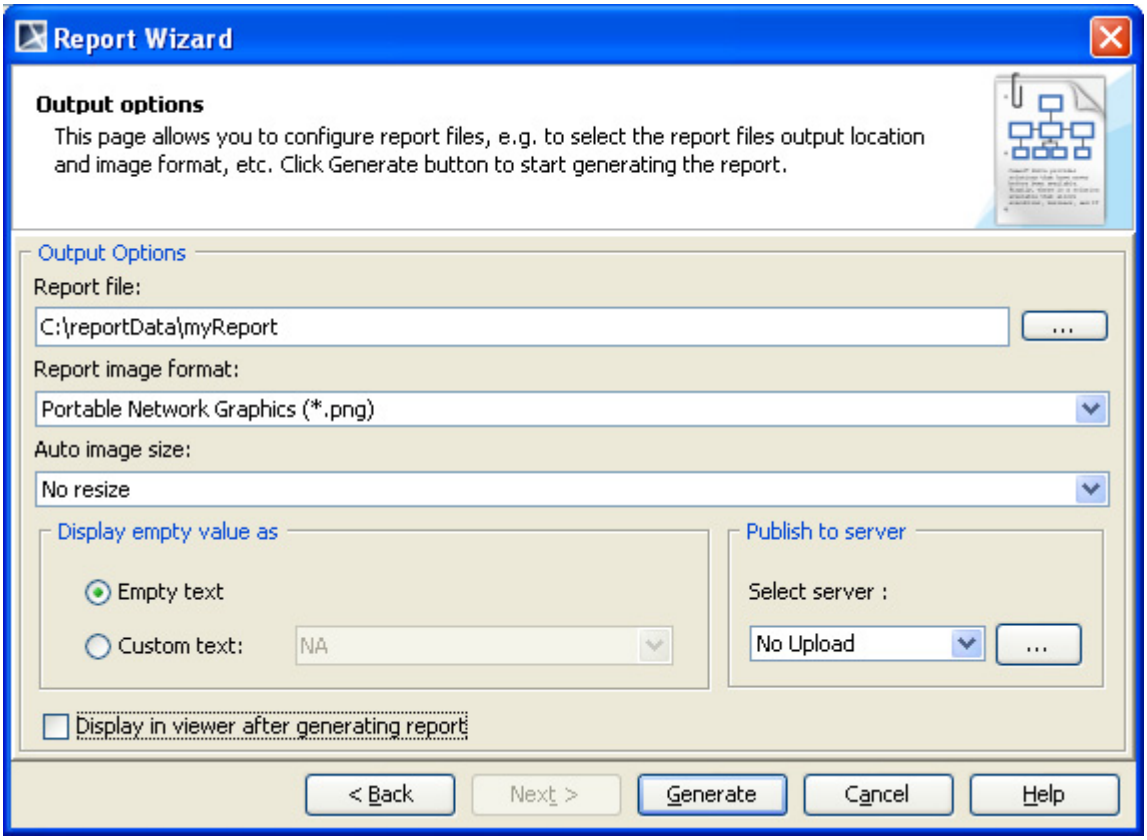


Figure 84 -- Selecting an Image Format

9. Select an option to display empty value information, either **Empty text** or **Custom text** (Figure 85).

NOTE	In some cases, the query may return an empty value that causes blank fields in the report. The **Display empty value as** option is useful when you have a standard representation for blank fields.

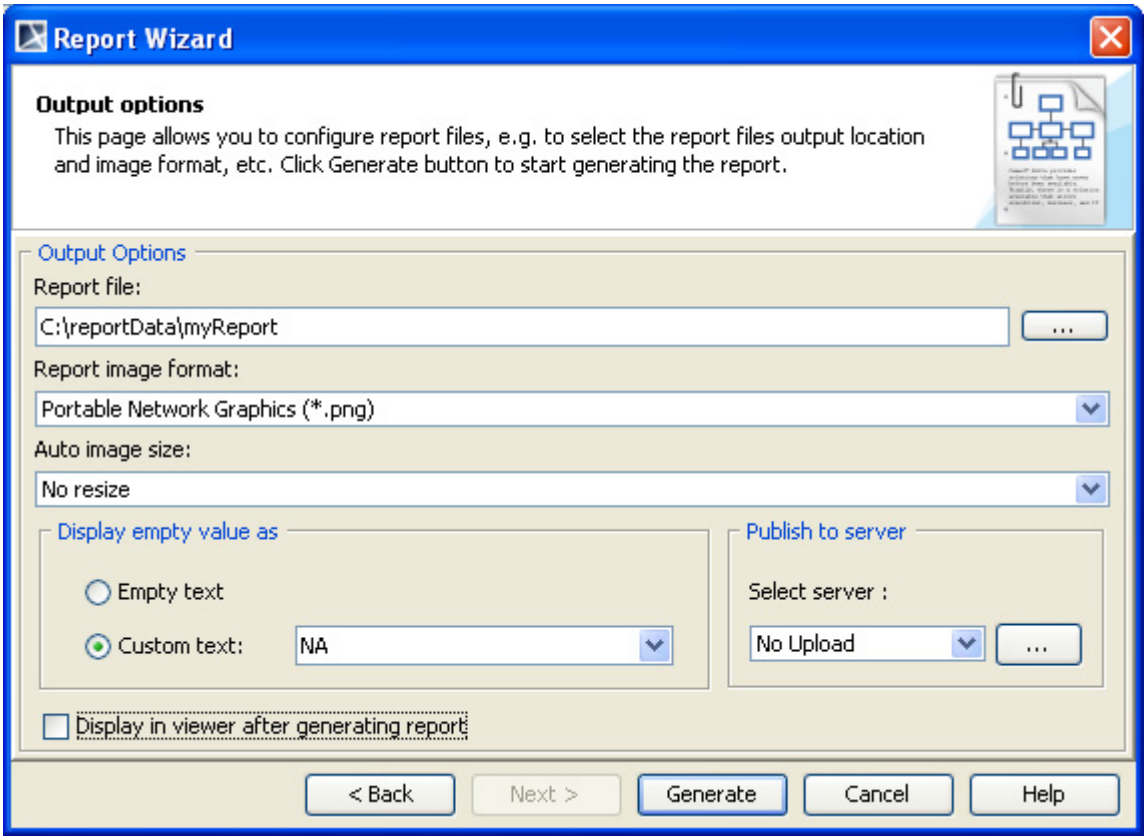


Figure 85 -- Selecting a Display Empty Value Option

10. Select the **Display in viewer after generating report** check box to open the report document with the default editor (Figure 86).

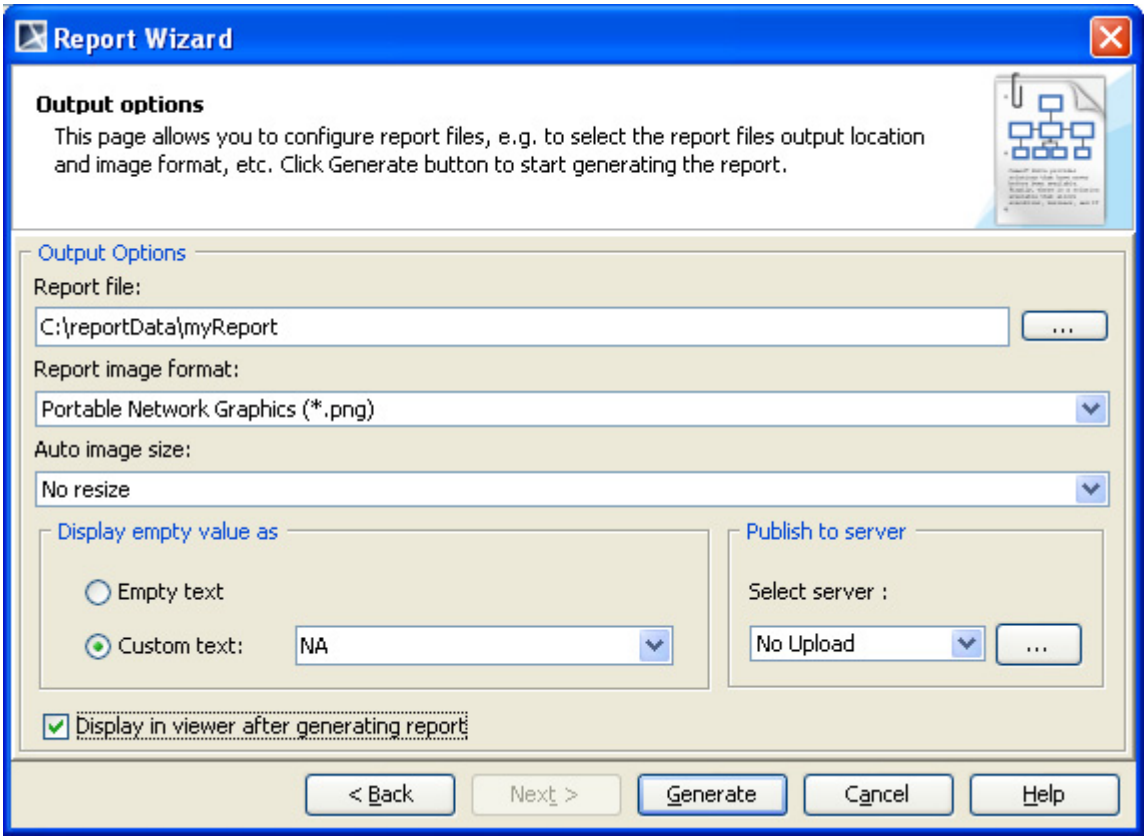


Figure 86 -- Selecting the Display in Viewer After Generating Report Option

11. Click **Generate**.

6.5 Web Publisher 2.0 Reports

6.5.1 Generating Reports

Web Publisher is a Java Doc like a report with clickable navigation and image map for all diagrams and elements.

To generate a Web Publisher 2.0 report:

1. Open the **Magic Library.mdzip** sample project from the **<MagicDraw_home>/samples/case studies** directory (see Figure 77 on page 122).
2. On the **Tools** menu, click **Report Wizard**. The **Report Wizard** dialog will open (Figure 87).
3. In the **Select Template** pane, select the **Web Publisher 2.0** template and click **Next** (Figure 87).

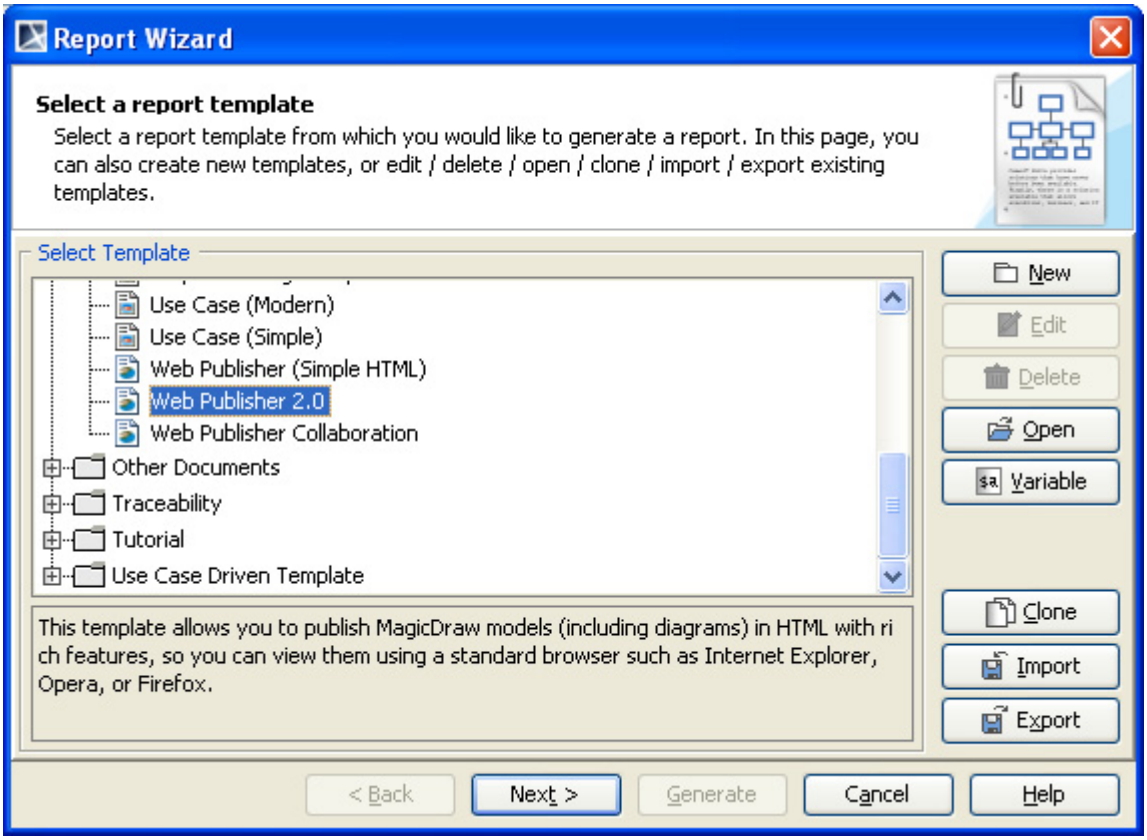


Figure 87 -- Selecting Web Publisher 2.0 Template

4. Click **New**. The **New Template** dialog will open. Type the report name and description and select the location of the template file. Click **Next**.
5. In the **Select Report Data** pane, select the built-in report data and click **Next**.

NOTE	In the **Select Report Data** pane, you can create a new set of report data for the Web Publisher template. The report data is a container for a set of custom-defined fields in the template. It can be used to group different report versions.

6. In the **Variable** pane, enter information for predefined custom fields or create a new one (Figure 88). Click **Next**.

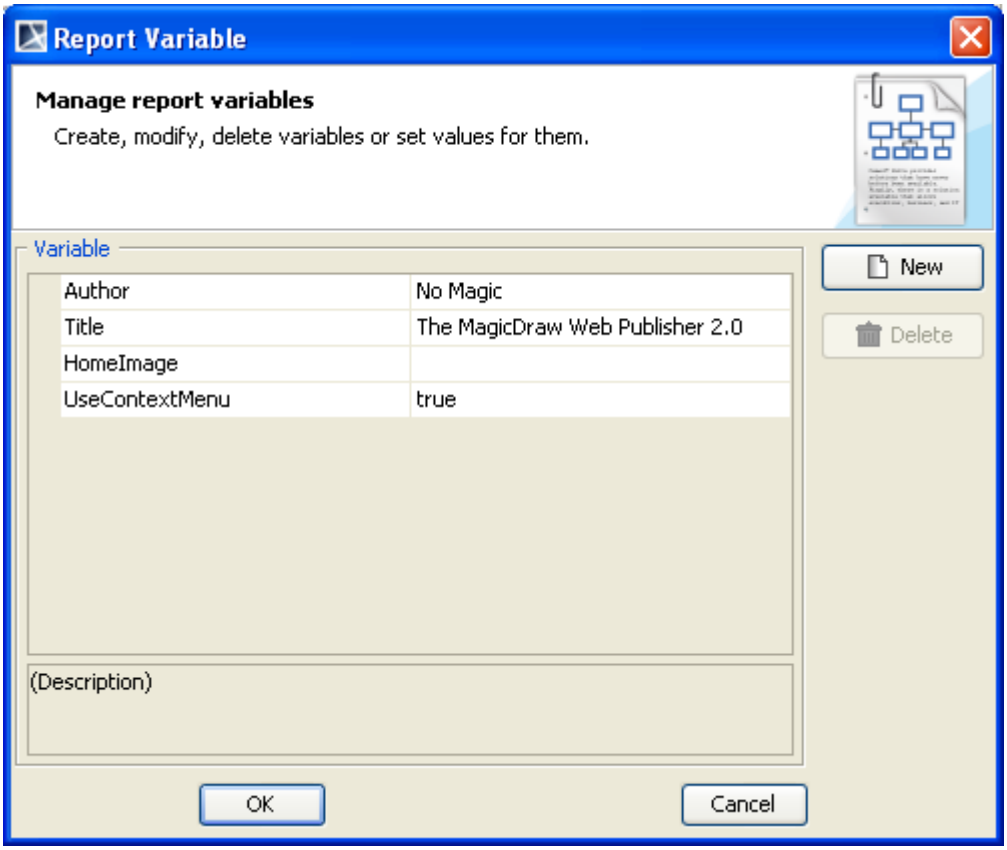


Figure 88 -- The Variable Pane

7. Select the scope of the report in the open package tree. In this case it will be the ***Data*** model package in the event that you want to have a web-based report of your entire project. Click **Next** (Figure 89).

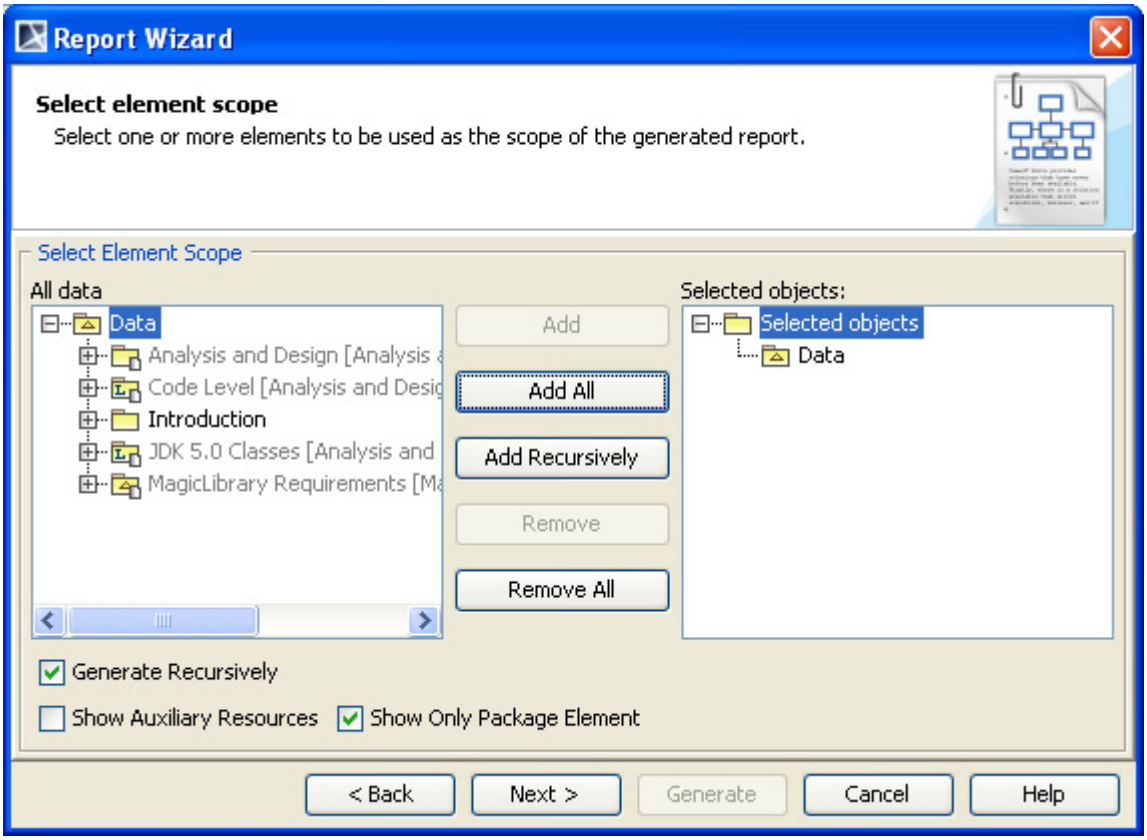


Figure 89 -- Selecting the Scope of the Web Publisher 2.0 Report

8. Click the "**...**" button to locate the report file location (see Figure 83 on page 128). The **Save as** dialog will open.
9. Select the file location, type the report name, and click **Save**. The generated web report will include a number of folders and files.
10. Select the report image format: **.png*, **.jpg* or **.svg* (see Figure 84 on page 129).
11. Select an option to display empty value information, either **Empty text** or **Custom text**.

NOTE	In some cases, the query may return an empty value that creates blank fields in the report. The **Display empty value as** option is useful when you have a standard representation for blank fields. (See Figure 85 on page 130).

12. Select the **Display in viewer after generating report** check box to open the report document with the default editor (see Figure 86 on page 131).
13. After all options have been selected, click **Generate**.

6.5.2 Web Publisher 2.0 Features

6.5.2.1 Report Layout

The Web Publisher report consists of 3 panels (see Figure 90):

(i) Containment panel: This panel shows the containment tree.

(ii) Content panel: This panel shows an element's content

(iii) Search panel: This panel contains a quick search box.

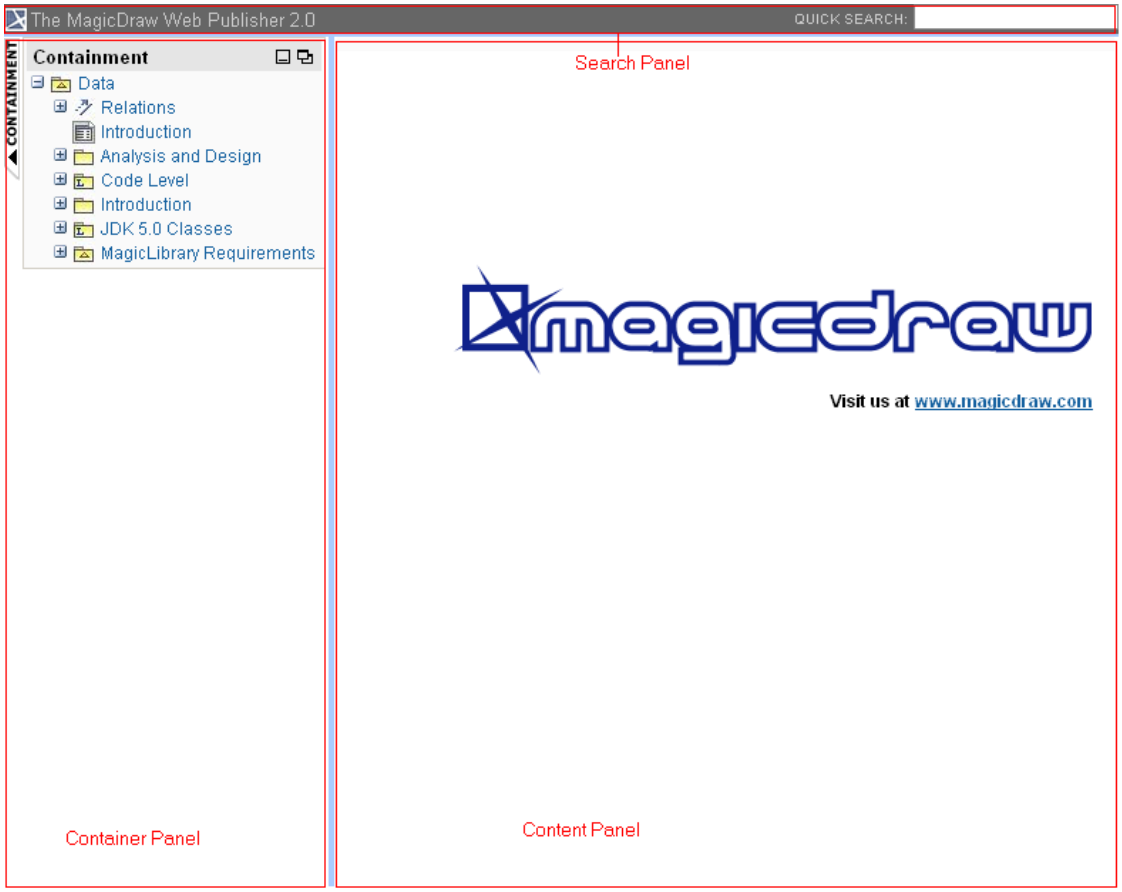


Figure 90 -- Web Publisher 2.0 Report Layout

6.5.2.2 Containment Menu

To hide or show the containment menu:

- Click and re-click the "**CONTAINMENT**" menu to hide and show it (see Figure 91).

Figure 91 -- Hiding the Containment Menu

To expand or collapse the containment tree:

- Click the “+” button on the containment tree to expand it or click the “-” button to collapse it (see Figure 92).

Figure 92 -- Expanding and Collapsing the Containment Tree

6.5.2.3 Contents Layout

Web Publisher contains two tabs: (i) Diagram and (ii) Specification.

(i) The Diagram tab shows diagram images (see Figure 93)

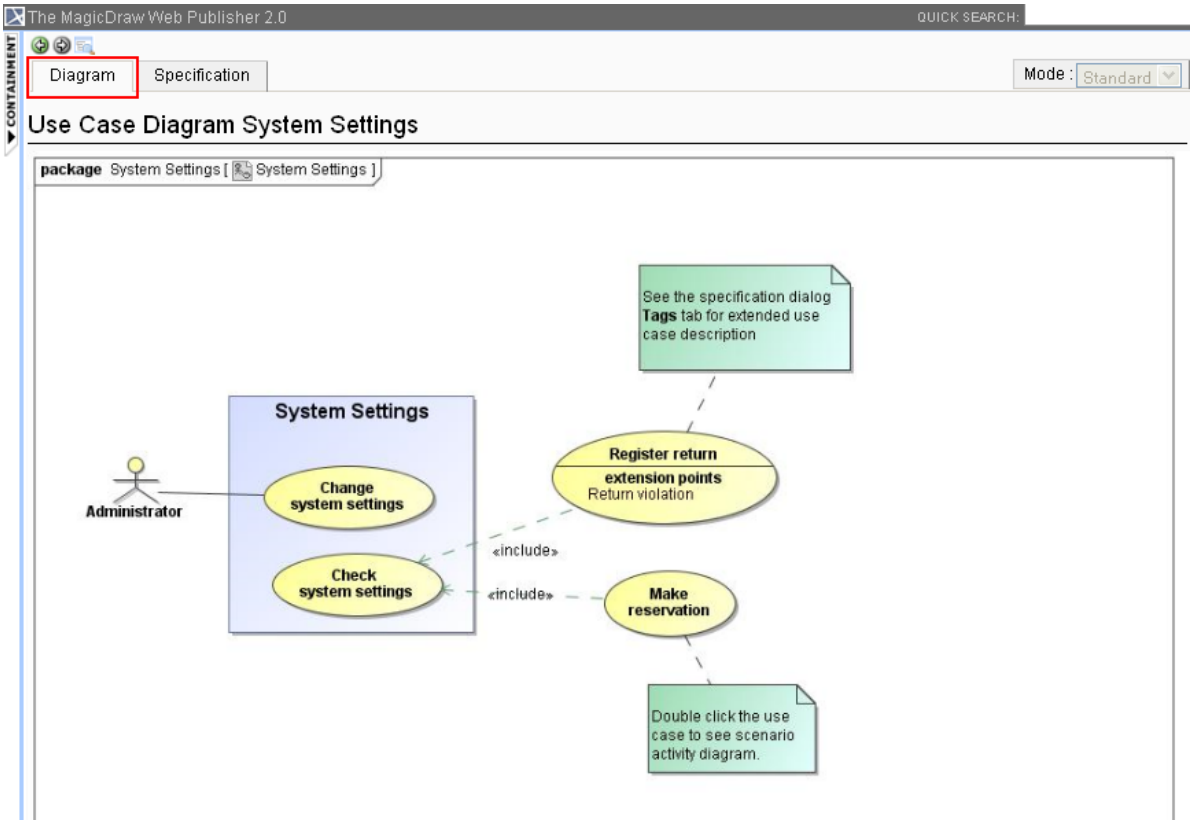


Figure 93 -- Diagram Tab

(ii) The Specification tab shows elements specification (see Figure 94).

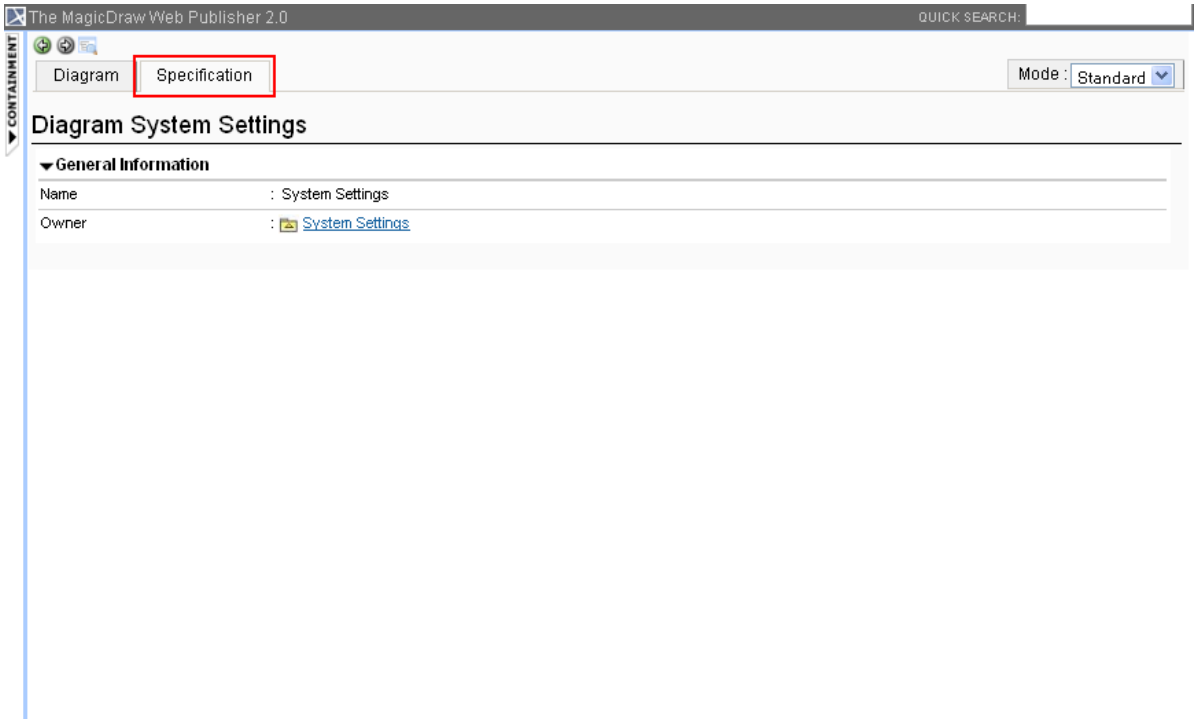


Figure 94 -- Specification Tab

To show or hide element contents:

- Click and re-click the arrow button to show and hide the contents (see Figure 95).

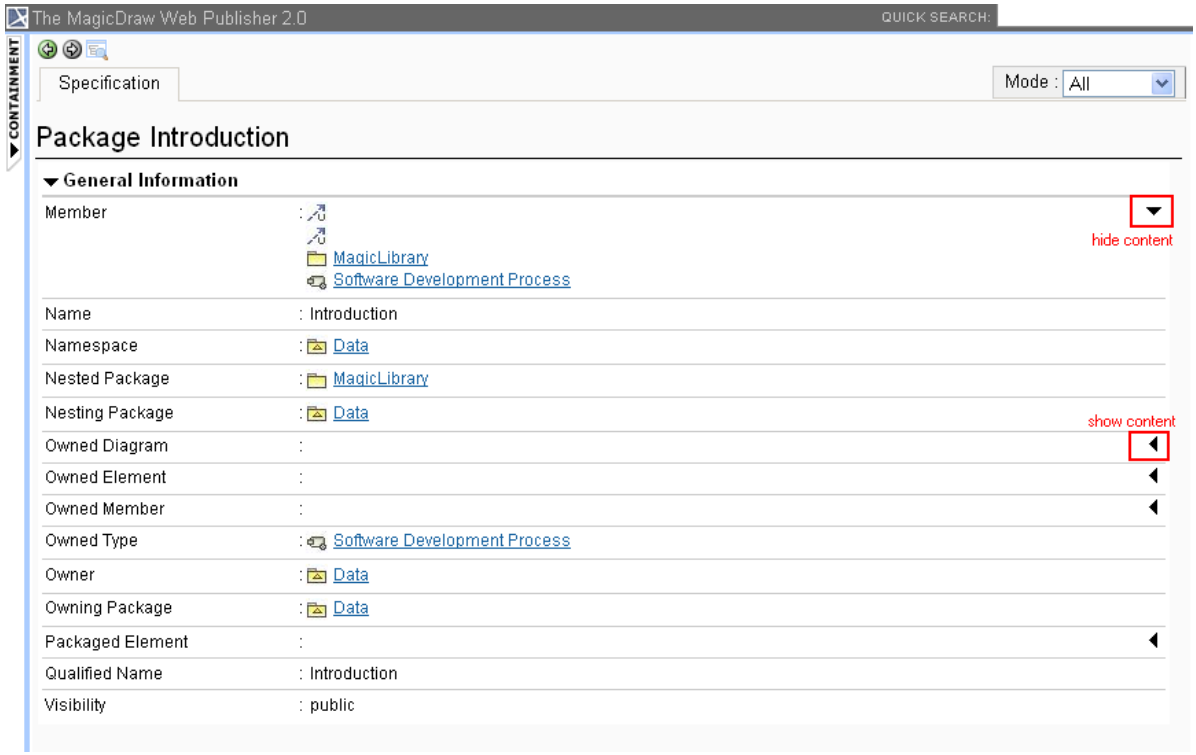


Figure 95 -- Showing and Hiding Element Contents

To show an element specification, Active Hyperlink, Hyperlink, submachine of state, or behavior of the call behavior action:

- Click a diagram's element to show the context menu for opening its specification, Active Hyperlink, Hyperlink, submachine of state, or behavior of the call behavior action (see Figure 96).

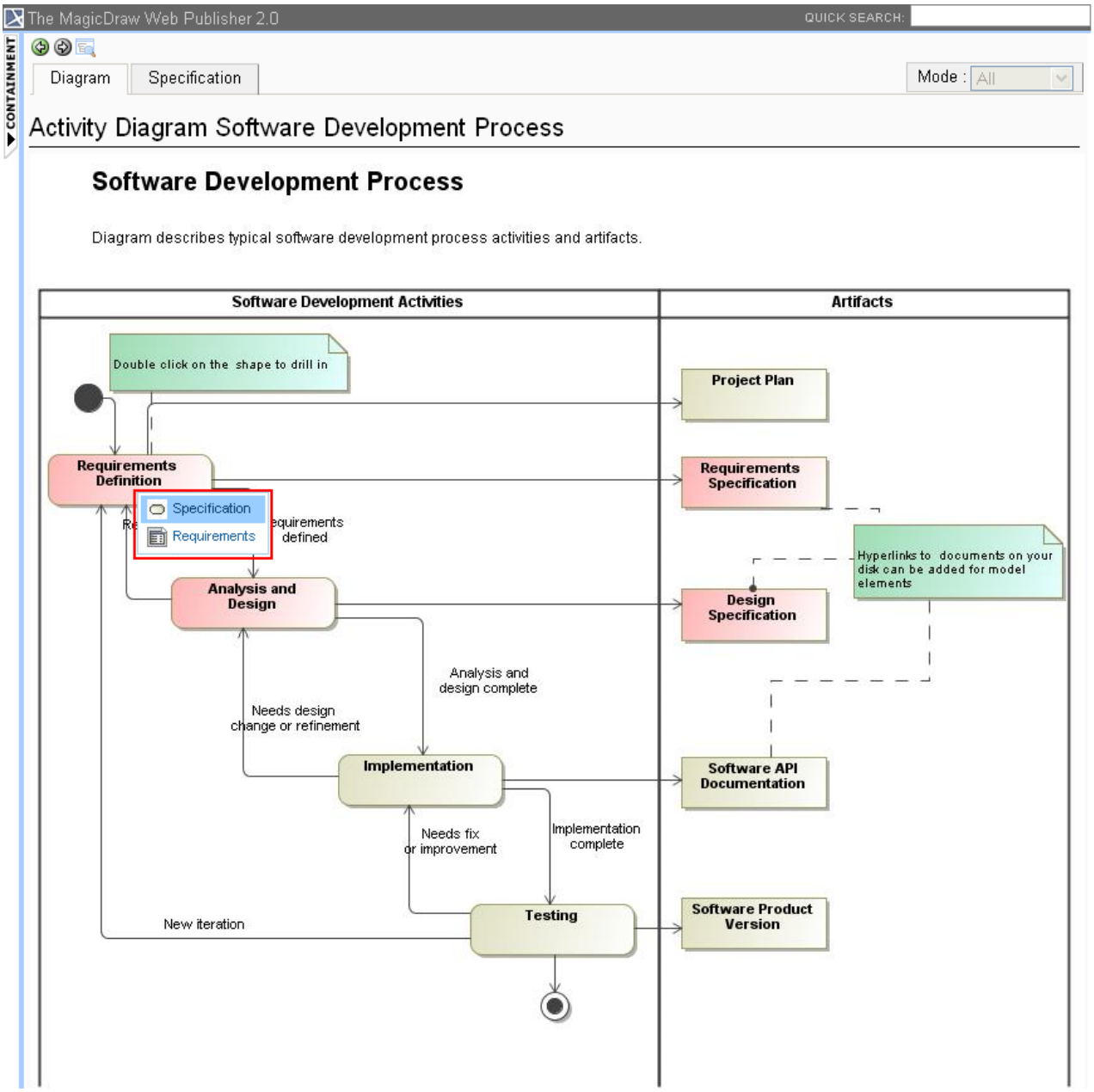


Figure 96 -- Web Publisher 2.0 Context Menu

To add an Active Hyperlink to a model:

- Double-click a diagram's element to open the Active Hyperlink and add it to a model element (see Figure 97).

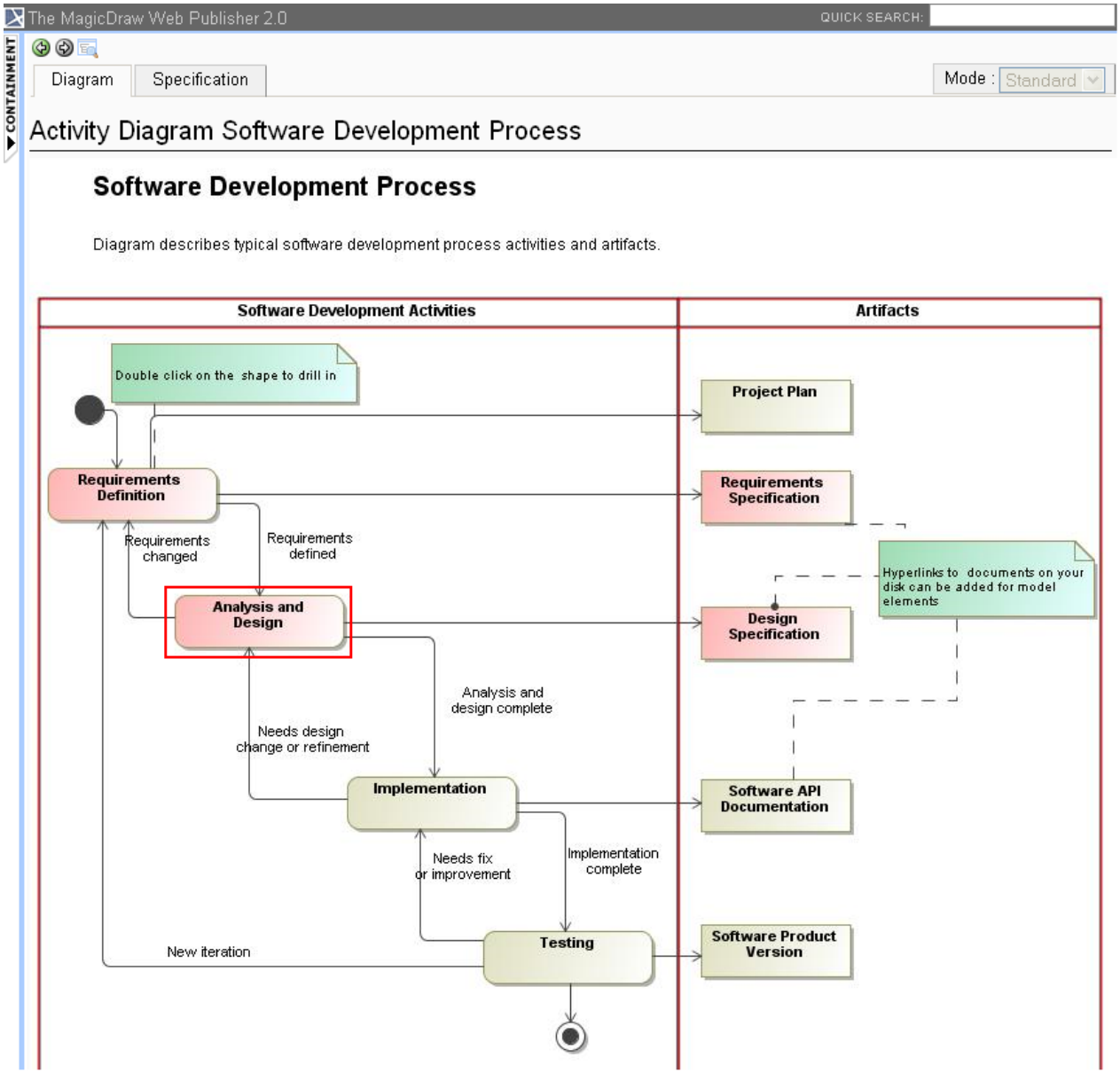


Figure 97 -- Adding Active Hyperlink

6.5.2.4 Quick Search Box

You can search for an element in a project by typing a specific keyword in the **Quick Search** box. You can also use a regular expression as a keyword (see Figure 98).

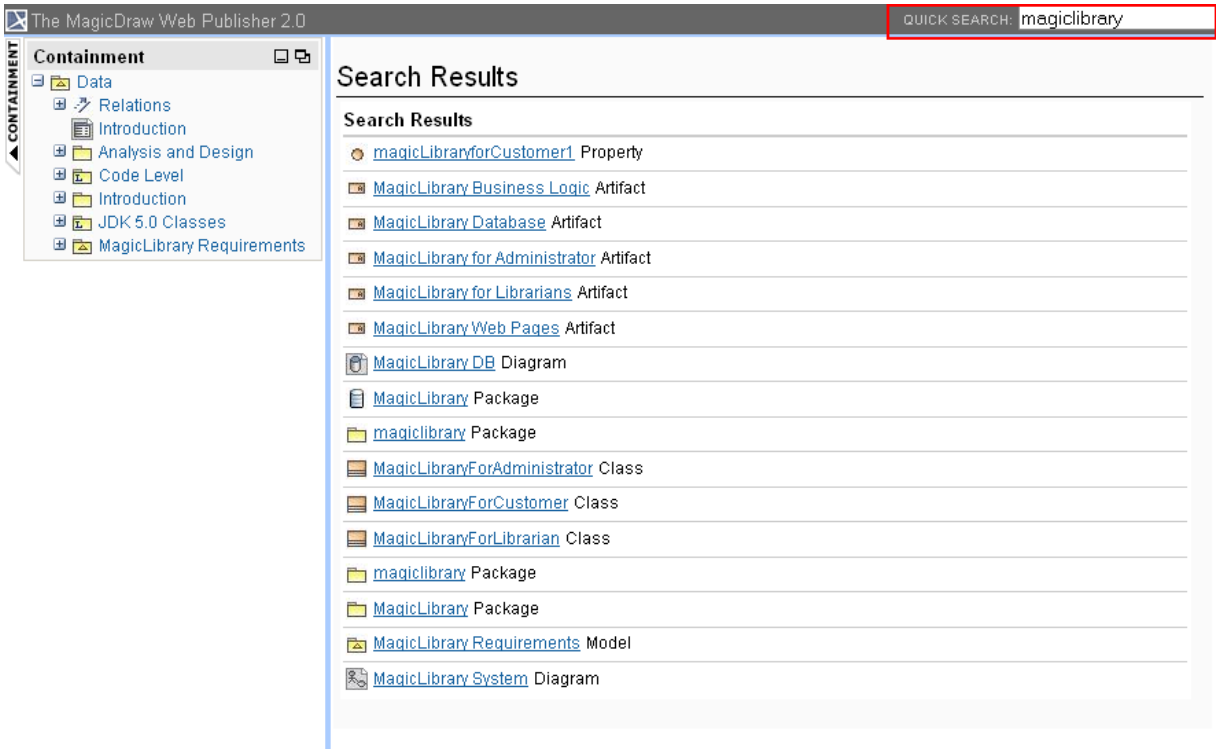


Figure 98 -- Searching for an Element

6.5.2.5 Changing a Homepage Image

Report Wizard enables you to change a homepage image. To change the homepage image, enter a specific image value in the **HomeImage** user-defined variable in the **Report Variable** dialog (see Figure 99).

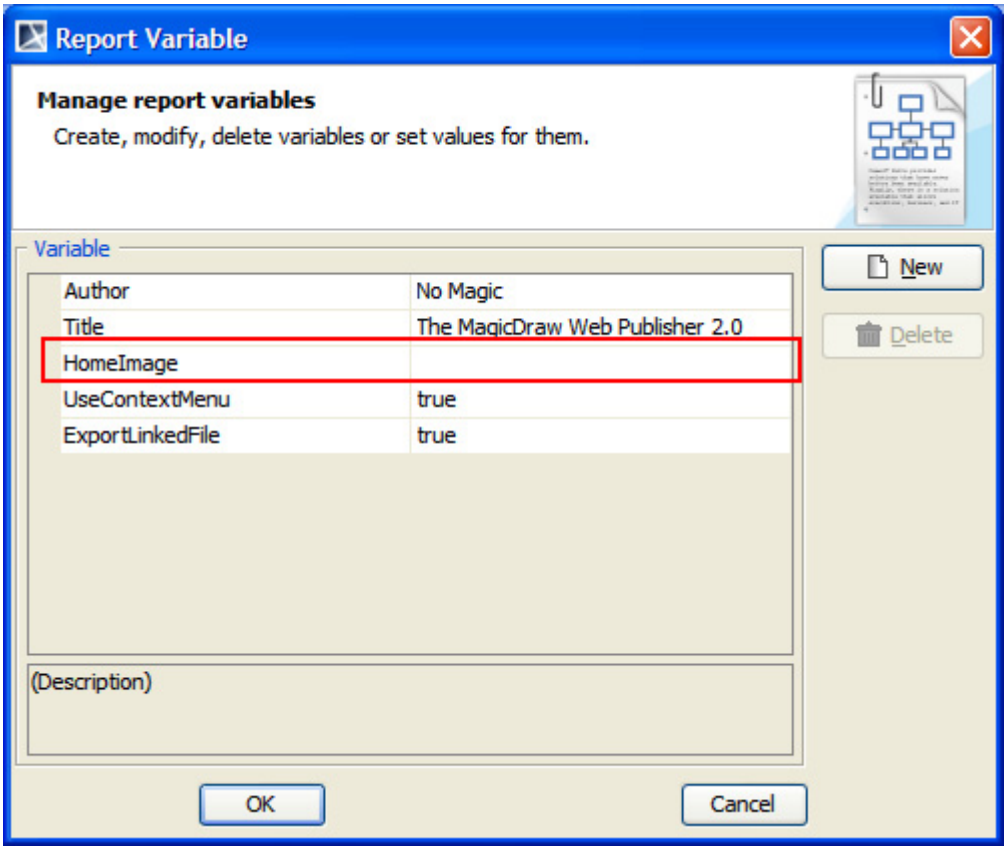


Figure 99 -- The Variable Pane in the Report Variable Dialog

See Table 16 for the possible values of the “HomeImage” variable.

Table 16 -- “HomeImage” User-defined Variable

Possible Value	Result
(not specified)	MagicDraw logo image will be displayed as the homepage image.
Diagram Name (plain text)	Enter a diagram name (in plain text) to display the diagram as the homepage image.
Element ID (mdel://)	Enter “mdel://” followed by a model element ID to display the element as the homepage image. For example, “mdel://_10_0EAPbeta2_8740266_1126593738250_35764_172”.
Local Image (file://)	Enter the location of an image file on your computer to display the image as the homepage image. For example, “file://d://picture//image.jpg”.
Web Image (http://)	Enter the location of an image on the Web to display the image as the homepage image. For example, “http://www.photobucket.com/image.jpg”.

Figure 100 demonstrates an example of specifying a diagram name (in plain text), in this case “Software Development Process”, in the “HomeImage” variable.

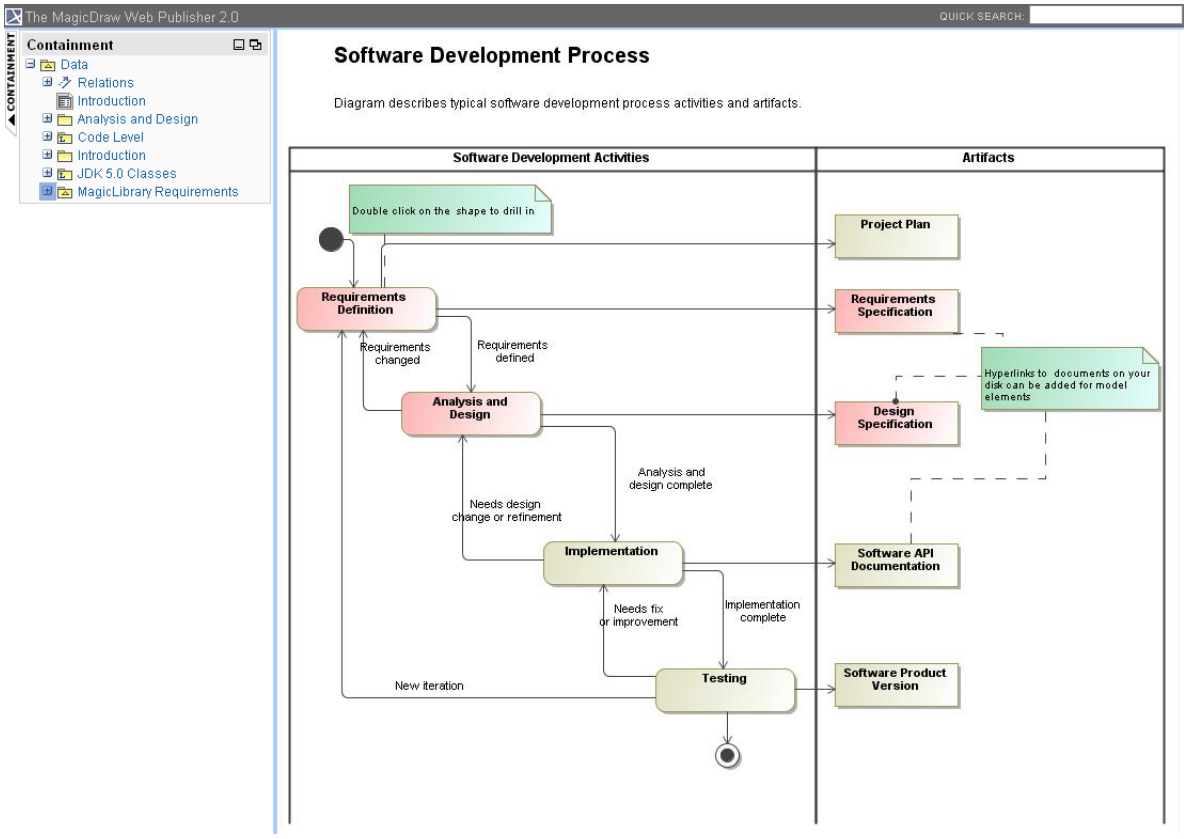


Figure 100 -- Example of Specifying Diagram Name (Plain Text) in "HomeImage" Variable

6.5.2.6 Element Description

The **Specification** tab shows a brief description of elements in a tooltip. Move your mouse over an element to see the description (see Figure 101).

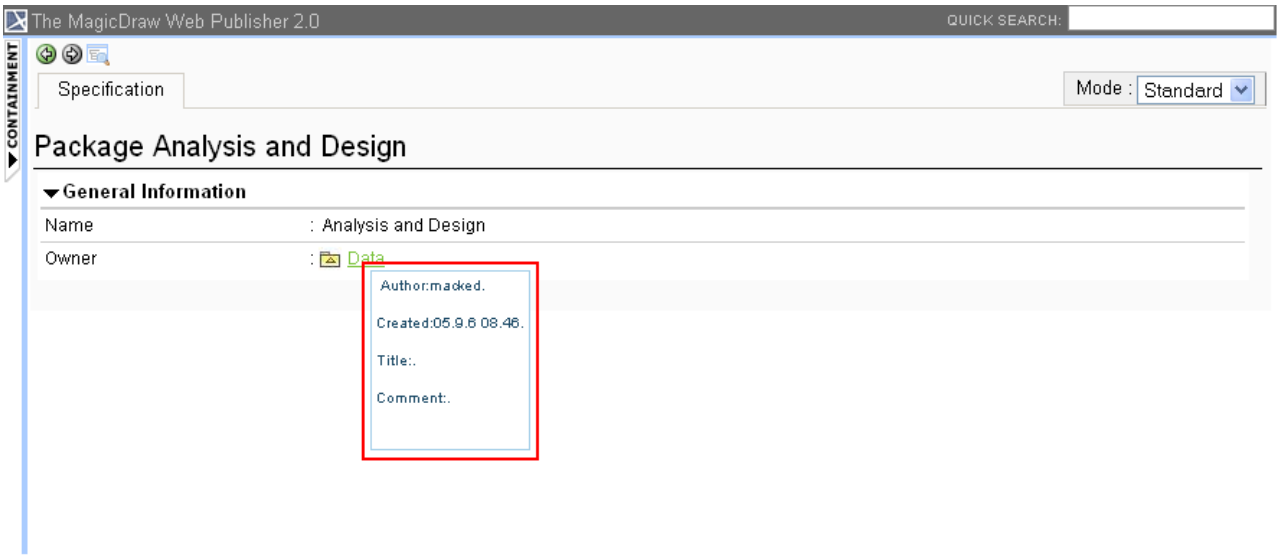


Figure 101 -- Showing Element Description

6.5.2.7 Shortcut to Homepage

You can click **The MagicDraw Web Publisher 2.0** at the top-left to go to the index page (see Figure 102).

Figure 102 -- Shortcut to Homepage

6.5.2.8 Property Visibility

The property visibility in the **Specification** tab has 3 mode types: (i) Standard, (ii) Expert, and (iii) All. The mode that will be shown in the property visibility depends on the mode that you have selected in MagicDraw (see Figure 103).

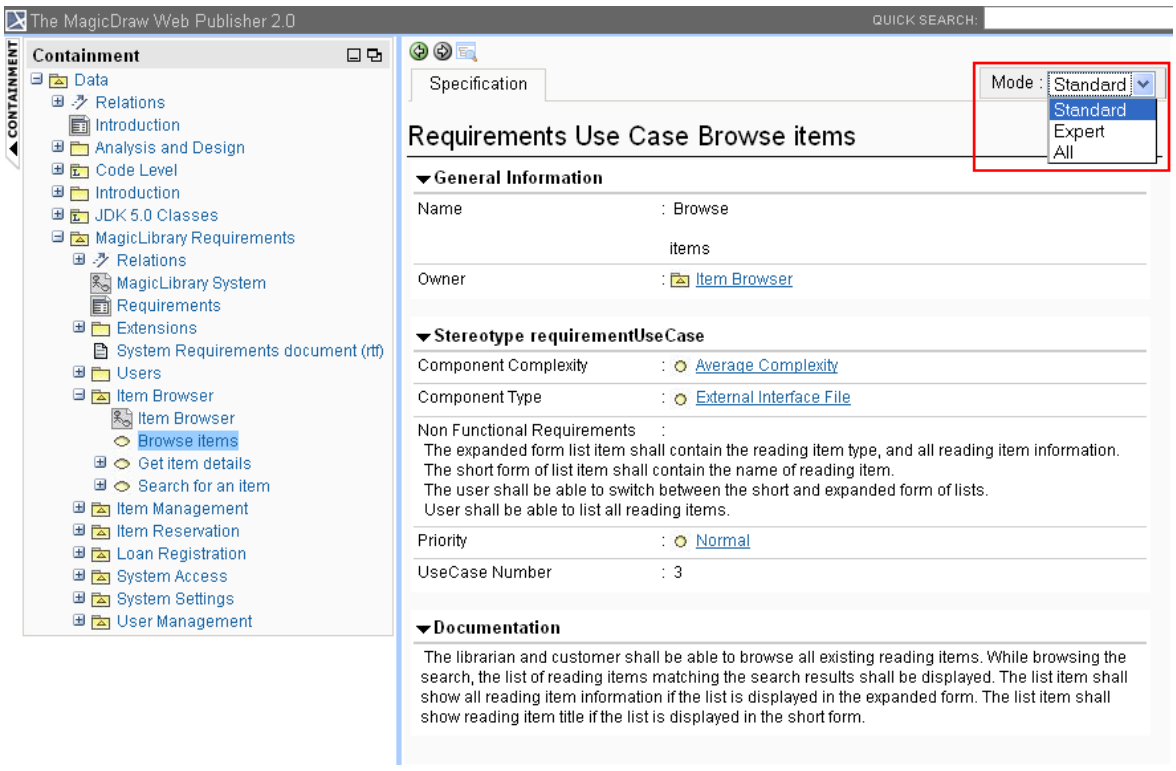


Figure 103 -- Property Visibility Mode Types

6.5.2.9 Showing or Hiding Context Menu

To show or hide a context menu:

1. Open the **Report Variable** dialog (Figure 104).
2. Set the value of the“UseContextMenu” user-defined variable to “true” or “false”.

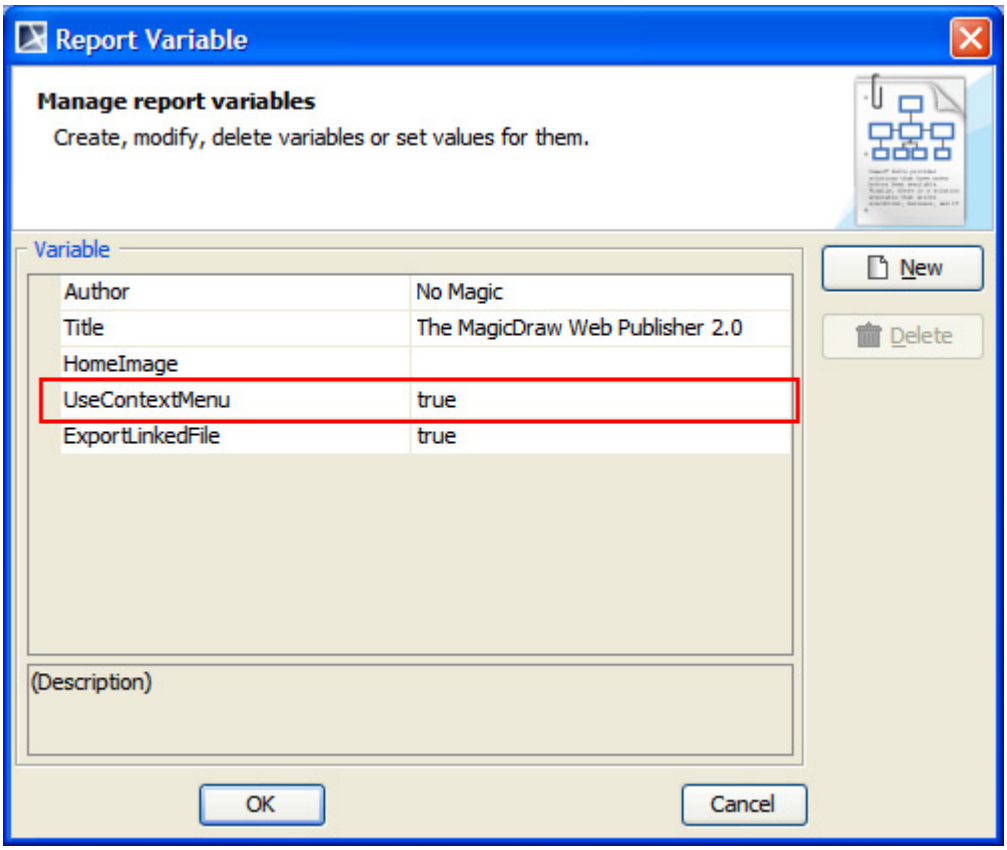


Figure 104 -- “UseContextMenu” User-defined Variable

UseContextMenu Value	Function
TRUE	To show the context menu on a right mouse click on a diagram element.
FALSE	To show an element specification or open an element/diagram/page specified in the existing active hyperlink, if any, on a right mouse click on a diagram element.

6.5.2.10 Exporting a Linked File into an Output Folder

To export a linked file into an output folder:

1. Open the **Report Variable** dialog (Figure 105).
2. Set the value of the“ExportLinkedFile” user-defined variable to “true” or “false”.

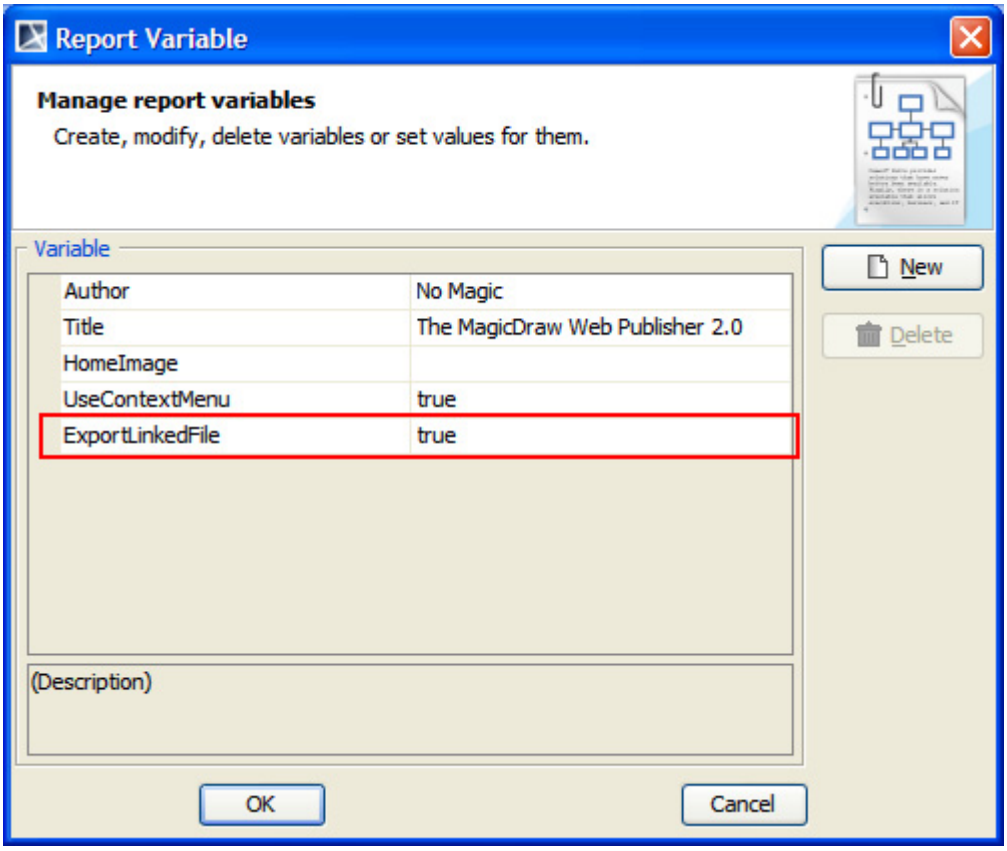


Figure 105 -- "ExportLinkedFile" User-defined Variable

ExportLinkedFile Value	Function
TRUE	To allow the template to copy a linked file from a model hyperlink into an output report folder.
FALSE	To keep only the link to an absolute path.

6.5.2.11 Opening an Activity, State Machine, Collaboration, or Interaction Diagram

Click an Activity, State Machine, Collaboration, or Interaction diagram to open a sub-diagram associated with an element.

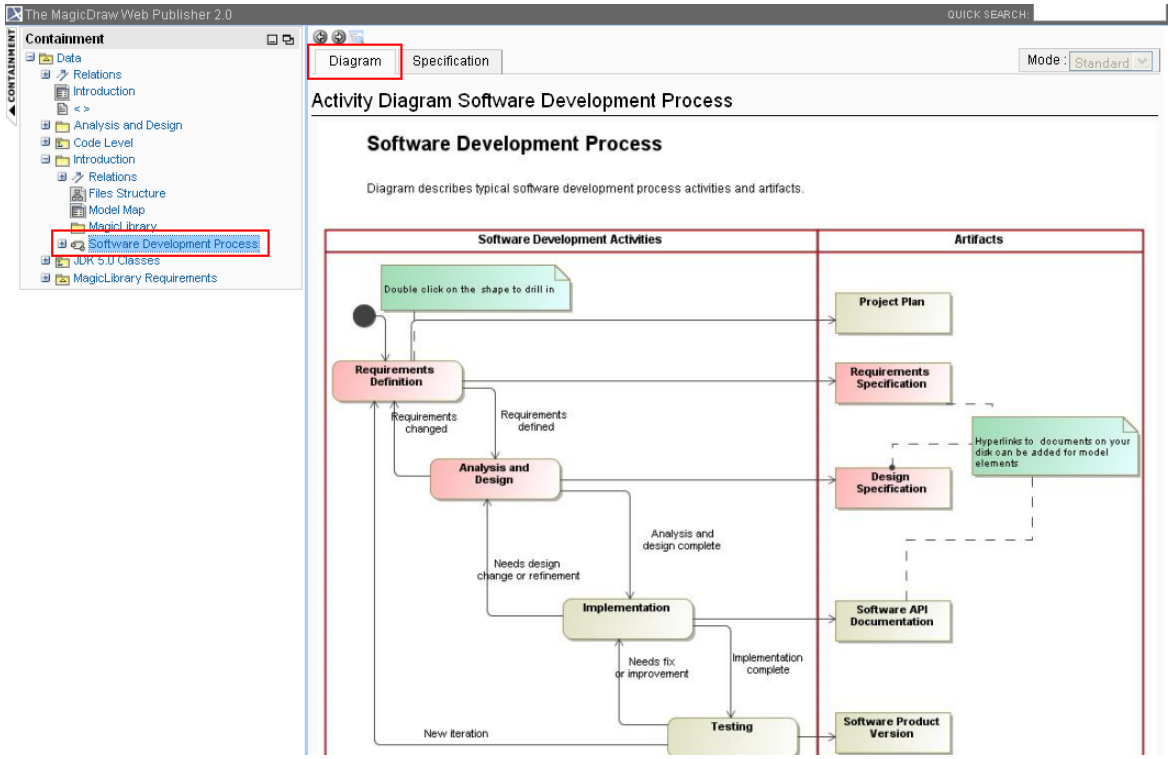


Figure 106 -- Opening Activity, State Machine, Collaboration, or Interaction Diagram

6.5.2.12 Opening the Sub-diagrams of a State with Submachine

Double-click a state with Submachine and Call behavior action with behavior to open its sub-diagram. For example, double-clicking the state “a1” (Figure 107) with the submachine “sub1” will show a diagram of the state machine “sub1” (Figure 108).

Figure 107 -- Double-clicking a State with Submachine

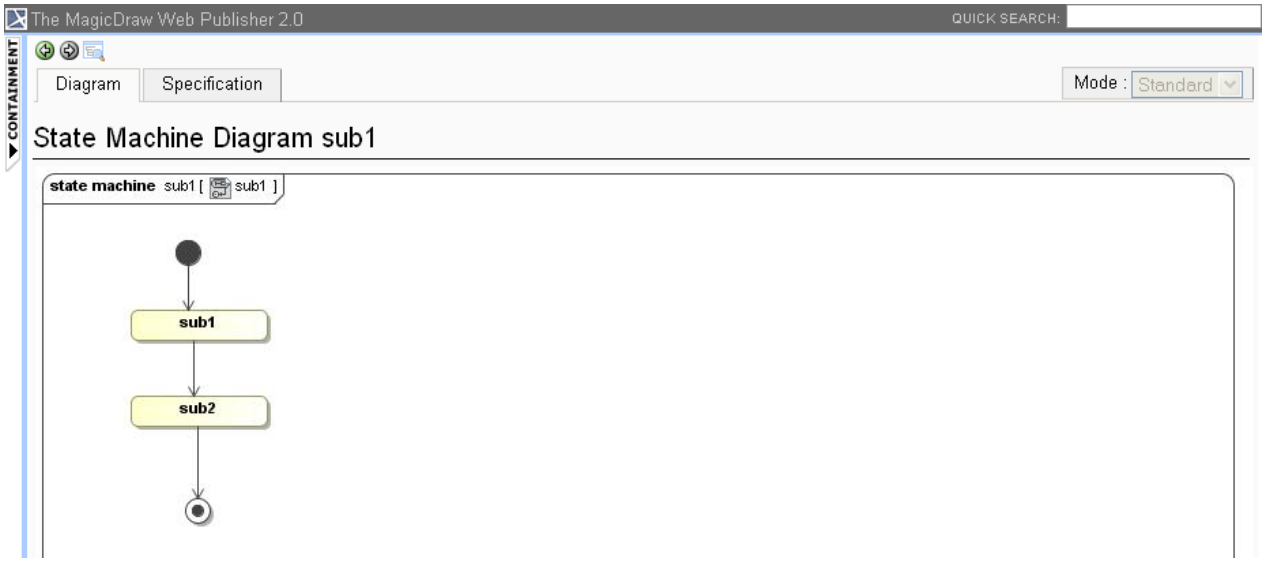


Figure 108 -- Diagram of State Machine "sub1"

6.6 Web Publisher Collaboration Reports

The Web Publisher Collaboration report is implemented based on the Web Publisher 2.0 report. The Web Publisher Collaboration report improves text editing, model review, and XML message features. The J2EE compliance server requires a running generated report.

6.6.1 Generating Reports

The Web Publisher Collaboration template is located under the Default Template category.

To generate a Web Publisher Collaboration template report:

1. On the **Tools** menu, click **Report Wizard**. The **Report Wizard** dialog will open.
2. Select **Web Publisher Collaboration** (Figure 109).

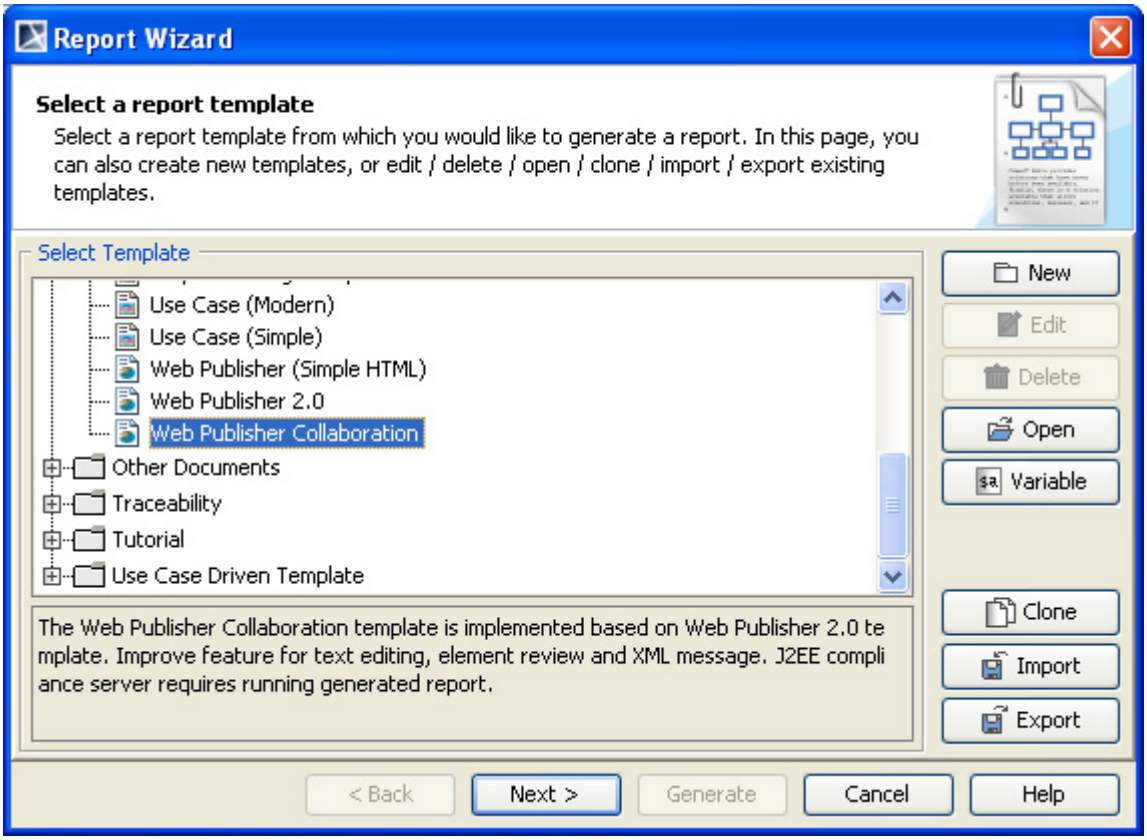


Figure 109 -- Selecting the Web Publisher Collaboration Template

- There are 3 user-defined variables ready for the default report data.

  (i) **Author**: The author of the report

  (ii) **Title**: The title of the web page

  (iii) **UseElementLink**: If “true”, it will allow the element hyperlink to work with diagrams. If “false”, the element specification will be shown on the web page.

3. Before generating a report, choose the option to upload the generated report to the server (Figure 110). (See **6.7 Uploading Reports to Remote Locations** for details).

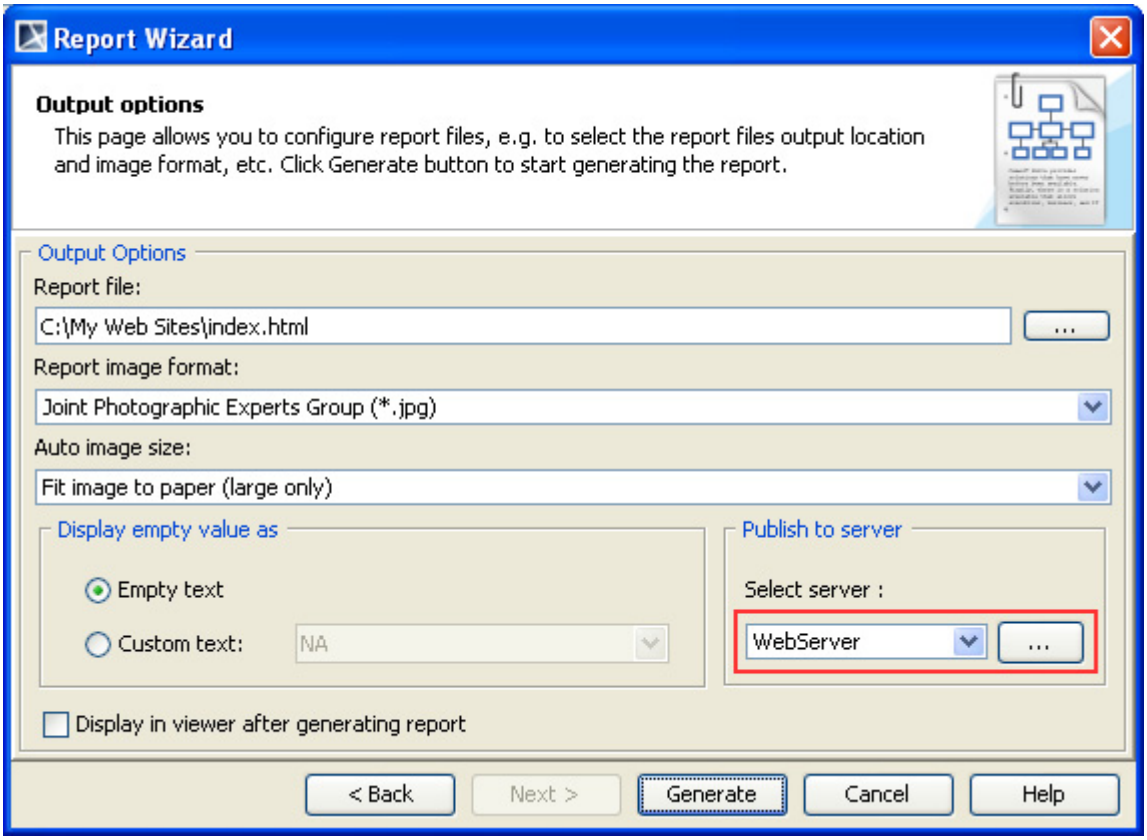


Figure 110 -- Selecting the Server to Upload the Report

6.6.2 Web Publisher Collaboration Feature

The main feature of the **Web Publisher Collaboration** report is for reviewing tasks. The generated report will allow you to (6.6.2.1) add comments and (6.6.2.2) alter model contents including documentation.

6.6.2.1 Adding Comments

To add comments:

- Click **Add Review** to add review text (Figure 111). Review text can be in a rich text format, for example, bold, italic, text color, etc. You cannot modify review text, but you can remove it with **Remove** clicked.

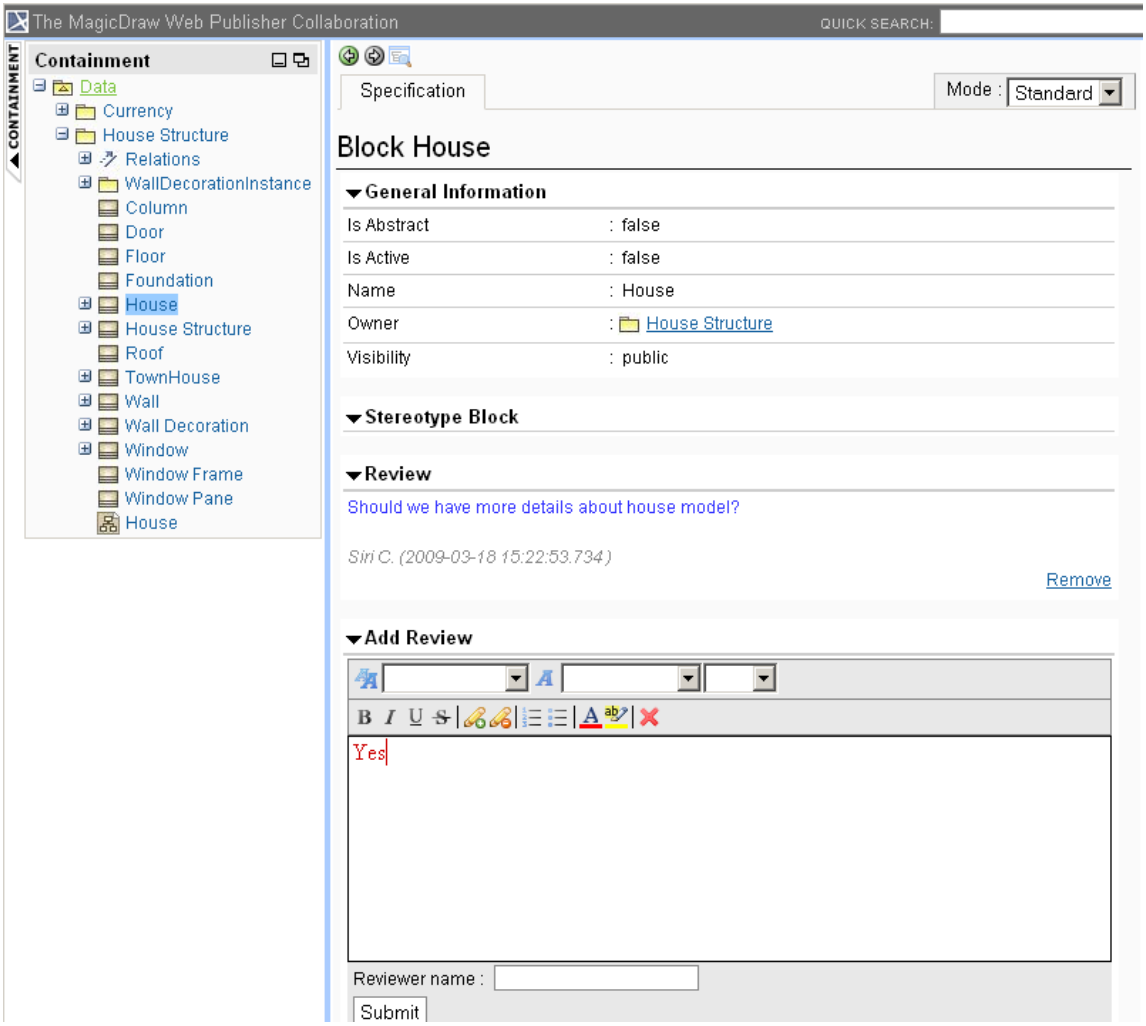


Figure 111 -- Adding Comments on the Web Publisher Collaboration Page

6.6.2.2 Altering Model Contents

To alter model contents:

- Click the field to open an input box (Figure 112). You can modify only the fields with the text value. Field editing will be completed once you have pressed enter or text input is out of focus. Press **ESC** to cancel the editing process.

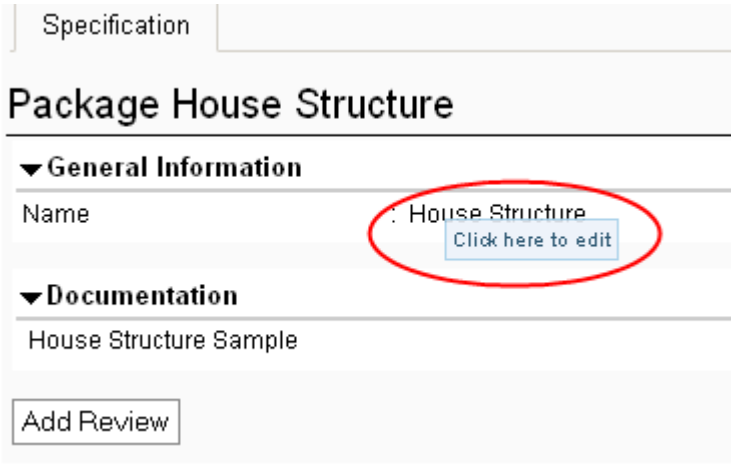


Figure 112 -- Editing Field

6.6.3 XML Integration

Web Publisher Collaboration opens an interface for retrieving model contents through XML or web services (Figure 113).

```
XML?refid=elementID
```

The endpoint service is “XML” with a query string for “refid”, for example:

```
http://127.0.0.1:8080/house/
XML?refid=_16_5beta1_2104050f_1233137219765_123064_4025
```

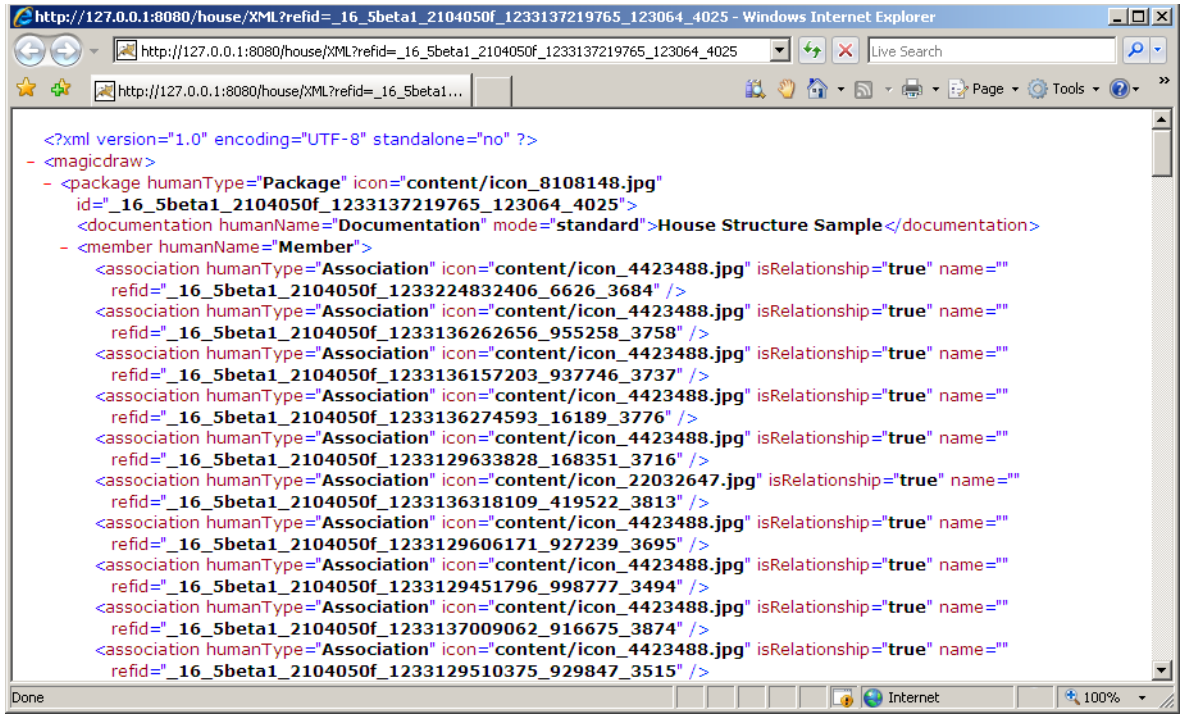


Figure 113 -- Example of Data Return from XML Service

6.7 Uploading Reports to Remote Locations

This feature allows you to upload a generated report to a predefined remote location. Each remote location is described in a “Profile” that holds all necessary information. You can use the **Profile Management** dialog to add, edit, or delete profiles.

This section contains (6.7.1) Quick Guide and (6.7.2) Detailed Explanations. In the Quick Guide section, you will learn how to create a profile step-by-step and how to use the created profile to upload a report to a remote server. The Detailed Explanations section describes what a valid Profile is and some of the palatial mistakes or errors when uploading a report.

6.7.1 Quick Guide

To create a profile:

1. In the **Report Wizard** dialog, click the "**...**" button (Figure 114). The **Profile Management** dialog will open (Figure 115). You can add, edit, or remove profiles.

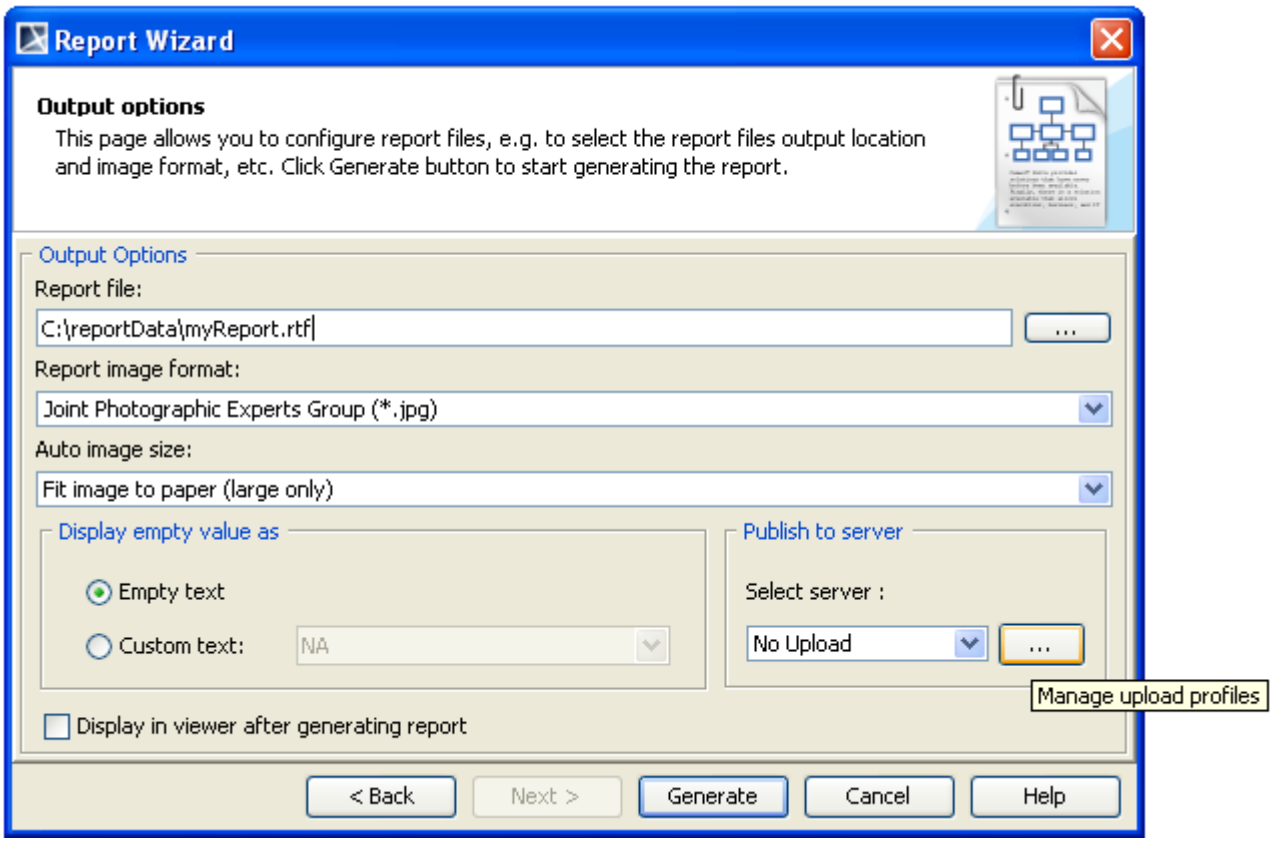


Figure 114 -- Dialog for Uploading Report to Remote Server

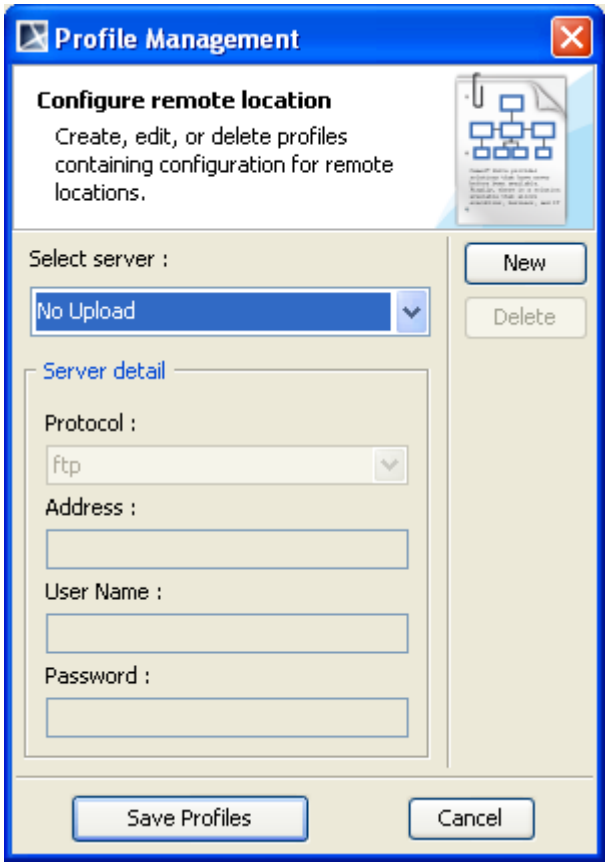


Figure 115 -- Profile Management Dialog

2. Click the **New** button to create a new Profile. The **New Server Profile** dialog will open (Figure 116). Enter the name of the Profile to be created. The name must neither be empty nor already be used in an existing set of Profiles. Click **OK**.

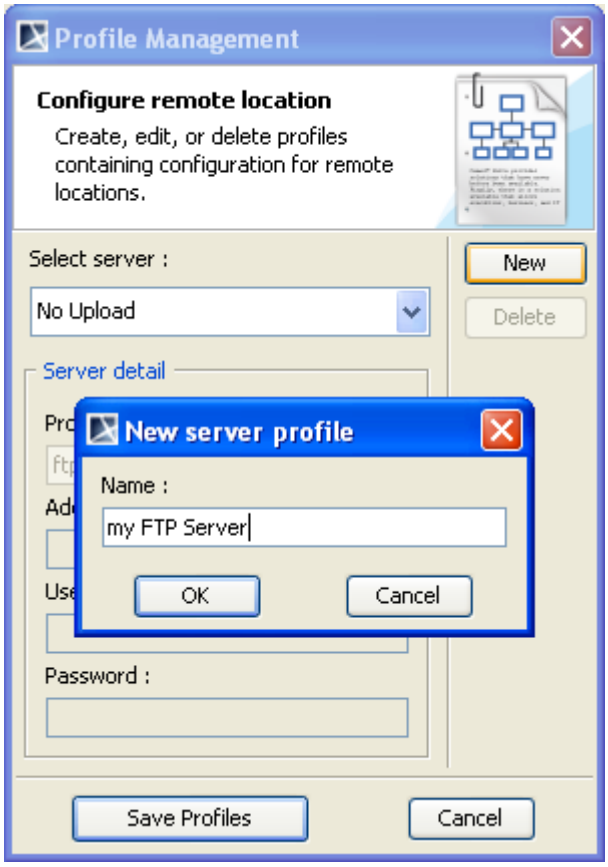


Figure 116 -- New Server Profile Dialog

3. To fill in the details of the profile, choose a Protocol and filling in a URL address. The User Name and Password fields can be left empty if desired.
4. Click the **Save Profiles** button to save the newly-added Profile and close the **Profile Management** dialog (Figure 117).

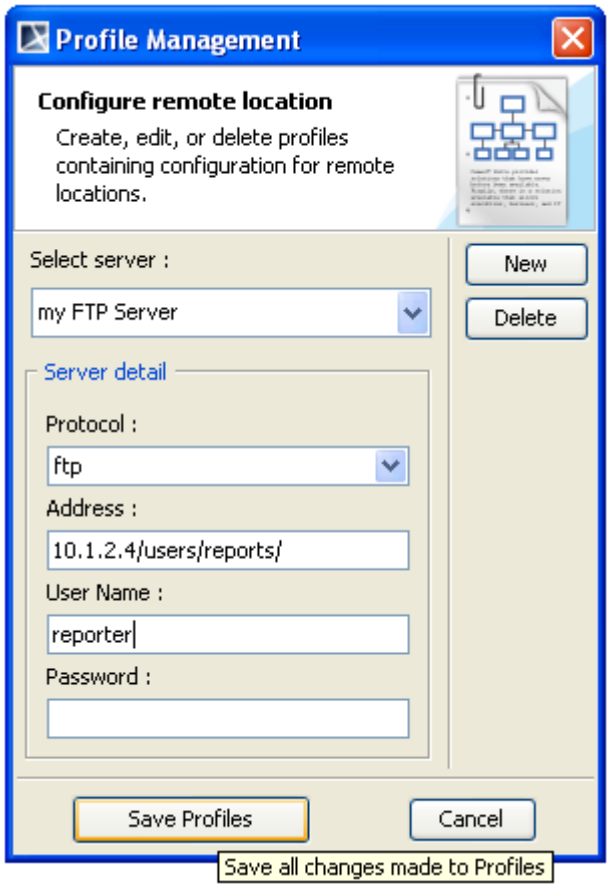


Figure 117 -- Enter Server Details

5. You are now back to the Report Wizard dialog and are ready to select the newly-created Profile (Figure 118). Once the report has been generated, it will be uploaded to the location specified in the Profile created earlier.

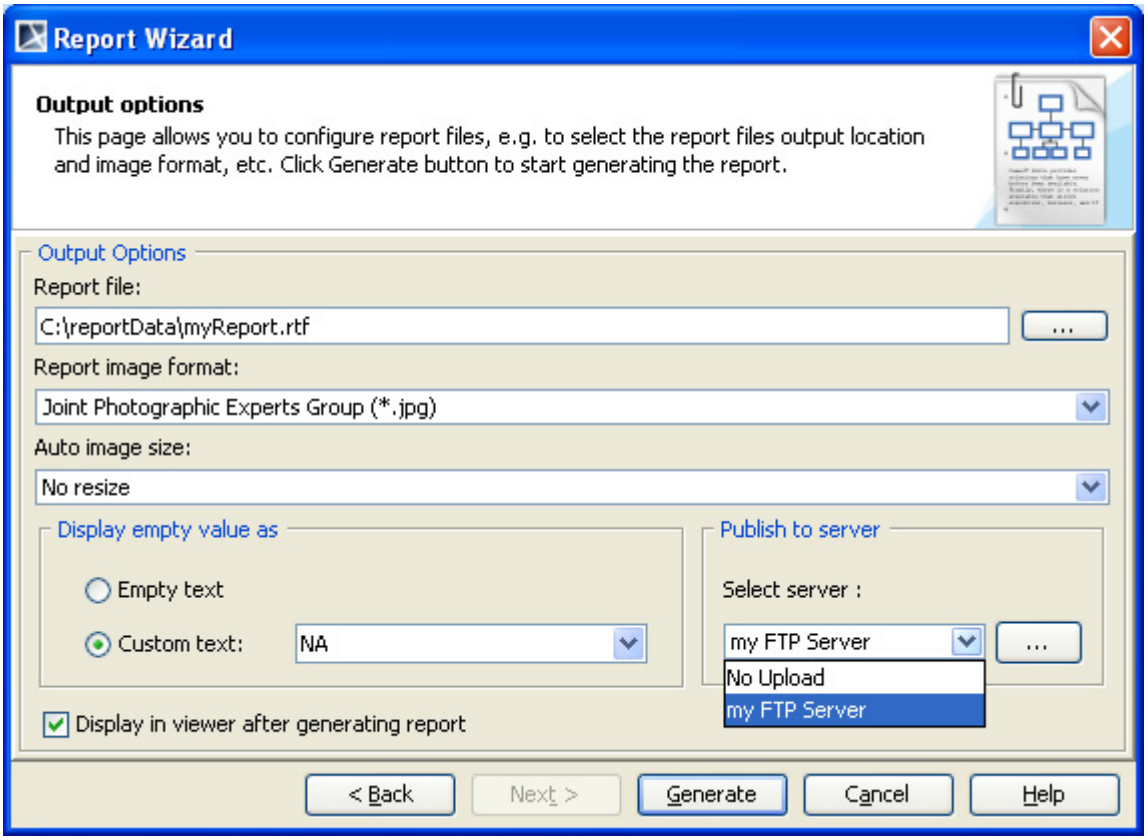


Figure 118 -- Selecting Server Profile

6. Enter authentication information in the **Authentication** dialog and click **OK** (Figure 119).

Figure 119 -- Authentication Dialog

7. If the report has been successfully uploaded, a message dialog will open showing the following message (Figure 120). Click **OK**.

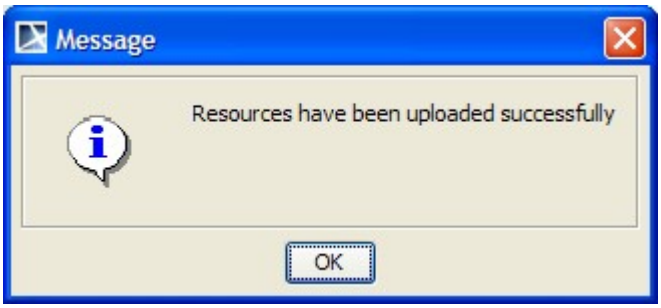


Figure 120 -- Successful Report Upload

6.7.2 Detailed Explanations

This section contains two parts: (6.7.2.1) Using the **Profile Management** dialog (Figure 115) and (6.7.2.2) Upload Problems illustrating problems that may arise when uploading a report to a remote location.

6.7.2.1 Using Profile Management Dialog

This section contains explanation about (i) Profile Management dialog buttons and (ii) Profile Management dialog fields, describing the dialog in Figure 115.

(i) Profile Management Dialog Buttons

The Profile Management dialog contains 5 buttons:

(a) New: to add a new profile to the list of existing Profiles. You cannot create a blank name profile or with a profile name that has already been used. You can add and edit multiple Profiles. However, these profiles will not be saved until you click the **Save Profiles** button.

(b) Remove: to remove the currently selected profile. Note that you cannot remove the "No Upload" profile.

(c) Save Profiles: to save all profiles. If new profiles have been added or existing Profiles have been modified, then these changes will be saved. All profiles will be checked for consistency or information correctness. You will be asked to fix issues, if any. It is not possible to save profiles as long as invalid information is detected. See Section **(ii) Profile Management Dialog Fields** below for the description of "valid profile".

(d) Cancel: to abandon all newly-added profiles and all changes made to existing profiles. If no changes were made, click this button to simply bring you back to the **Output Options** pane of the Report Wizard dialog. If changes are found, the **Confirm Cancel** dialog (Figure 121) will open. In this dialog, you will see with all profile names whose content has been changed. You can choose to either abandon all changes or return to the **Profile Management** dialog and save the changes.

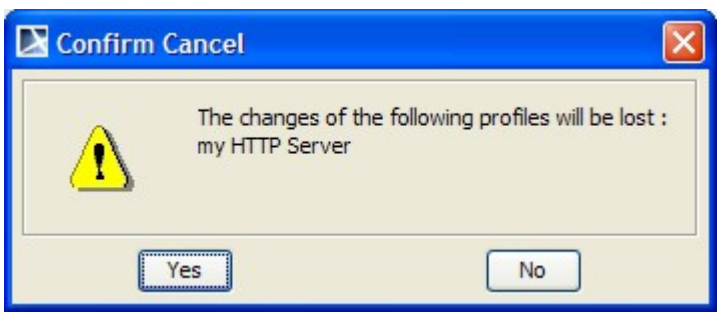


Figure 121 -- Cancel Confirmation Dialog

(e) Exit: this button  is located in the upper-right corner of the Profile Management pane. It works in a manner similar to the **Cancel** button.

(ii) Profile Management Dialog Fields

Profiles describe information necessary for sending a report to a remote location. The report will be saved remotely with the same name as the locally-generated report and will overwrite any file with the same name on the remote location.

The Address field is URL sensitive, and can hold protocol as well as username information. If the protocol or username information is found in the Address field, the corresponding fields, namely the Protocol and User-name fields, will be updated with the values from the Address field. At the very least, a profile must have a host-name in the Address field.

You cannot save a profile unless this condition is met. The Username and Password fields contain sensitive data and therefore are optional. If these fields are empty in a profile, you will be prompted for the username and password when the server asks for your authentication.

- **Profile name** : This drop-down list holds all the existing Profiles. They are alphabetically sorted. The "No upload" profile is always available and you can neither modify nor remove it. Select the"No Upload" profile if you do not want to upload a report.
- **Protocol** : Five different protocols are supported: FTP, FTP over SSL (Secure Socket Layer), Webdav, Webdav over SSL, and SSH (Secure Shell). You can either choose to select a protocol from the protocol drop-down list or fill in the scheme in the address part of the profile. Certificates that are sent by servers using secure traffic (FTP over SSL, Webdav over SSL, and SSH) are silently accepted. The scheme identifiers and default port numbers of the five supported protocols are listed in the following table.

Table 17 -- Scheme Identifier and Default Port Numbers of Five Supported Protocols

	Protocol Name	Scheme	Default Port Number	Traffic Mode
1	ftp	ftp://	21	Plain text
2	webdav	http://	80	Plain text
3	ftp + ssl	ftps://	990	Encrypted & Secure
4	webdav + ssl	https://	443	Encrypted & Secure
5	ssh	ssh://	22	Encrypted & Secure

- **Address** : The Address field is the central part of a profile. Other fields, except the Password field, can be set by it. An Address contains five different parts: scheme, username, hostname, port (number), and (remote) path. Except hostname, every part is optional. Depending on the circumstances, you might need to add a special port number and/or the remote path where you have write access. Even though an address is valid without the path part, most servers will require a path and will not let you write to the root directory. In every case, scheme and username parts will take precedence over the selection in the Protocol Field or the content in the Username field. The following is the basic pattern for the address field.

  [scheme://][username@]hostname[:port][path]

  - scheme: The supported protocols are ftp, ftps, http, https, or ssh. Must be followed by "**://**".
  - username: The username needed for the authentication. It must be followed by "**@**".

- hostname: Must be either an IP address (eg. 10.1.1.195) or a human readable URL (eg. www.ftpserver.com) and present to be a valid address.
- port: The port number where the server is listed. It is only necessary if the port number is different from the default port for well-known services. It must be preceded by "**:**".
- path: The remote location where the report will be saved. Paths are Unix style paths and therefore use "*/*" (forward-slash) as a delimiter. Optional.

Examples of valid addresses :

- 10.1.1.195 (hostname only)
- 10.1.1.195/companyDav (hostname and path)
- 10.1.1.195:80/companyDav (hostname, port number, and path)
- ftp://www.ftpserver.com (scheme and hostname)
- john@10.1.1.195 (username and hostname)
- john@www.ftpserver.com/users/john/reports (username, hostname, and path)
- ssh://john@10.1.2.35:22/reports (all parts)

- **Username** : This field specifies the name used for the authentication. If the address contains a username part, the content of the Username field will be ignored. It will be either filled in or overwritten with the username from the Address. A Profile with an empty username field is valid, but you will be prompted for a username and password every time you upload to the server using this profile.
- **Password** : A profile with a password must always be accompanied by a username. A Profile with a password but no username is invalid and therefore cannot be saved. The password field represents password characters as '*'. Saving a profile with a password will save the password in an encrypted form. For example, assume the password is 'test', this might result in an encrypted password to be '0013BCA5590DE'. A Profile with an empty password field is valid, but you will be prompted for a password every time you upload reports to the server in this profile.

6.7.2.2 Upload Problems

There are three main problems that can arise when uploading reports to a remote location.

(i) The server that you are trying to connect is not responding, or the valid address you have entered in the Profile is not the address of the server that responds to your request (Figure 122).

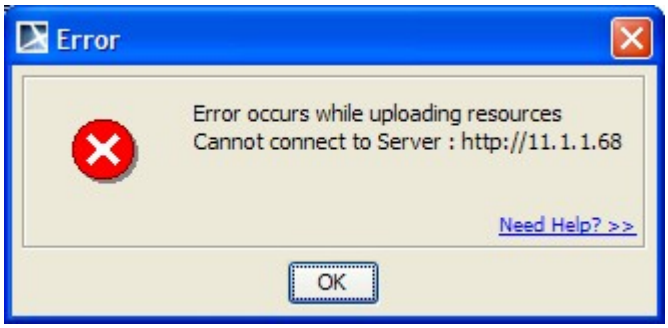


Figure 122 -- Connection Error Message Dialog

(ii) Your username and/or password are incorrect, and the server does not allow you to proceed (Figure 123).

Figure 123 -- Authentication Error Message Box Dialog

(iii) Various causes can lead to an unsuccessful attempt to upload a report. For example, the server runs out of space and thus cannot save the report, or the connection could have been interrupted, etc. Whatever the cause might be, the report HAS NOT BEEN transmitted sucessfully (Figure 124). For more details on error messages, see MagicDraw logs.

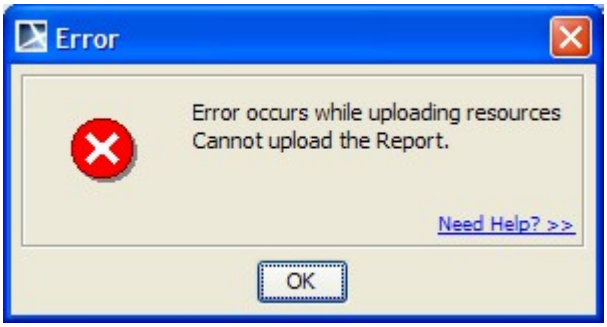


Figure 124 -- Unsuccessful Upload Error Message Dialog

6.8 FAQs

To generate an output report document from Report Wizard:

1. On the **Tools** menu, click **Report Wizard**. The **Report Wizard** dialog will open.
2. Select the template or create a new one. Click **Next**.
3. Specify **Report Data** and add information for the variables. Click **Next**.
4. Select the document scope from the corresponding packages. Click **Next**.
5. Select the output location, images format, and text for blank fields.
6. Click **Generate**.

To add a new report template:

1. On the **Tools** menu, click **Report Wizard**. The **Report Wizard** dialog will open.
2. Click **New**. The **New** dialog will open.
3. Enter the template name and description. Click the "**...**" button to specify the template file location.
4. Click **Create**.

NOTE	Once a new template has been created, **Report Wizard** will add a folder with the entered template name and save the selected template in this folder.

To remove a template:

1. On the **Tools** menu, click **Report Wizard**. The **Report Wizard** dialog will open.
2. Select a template from the list and click **Delete**.

NOTE	Once removed, the selected template with its folder and reports cannot be recovered.

To modify a template file:

1. On the **Tools** menu, click **Report Wizard**. The **Report Wizard** dialog will open.
2. Select a template from the list and click **Open**. The default editor and a template file for editing will open.
3. Modify the template and perform the save command in the editor.

NOTE	A different editor will be used for each template format. For example, MS Word could be used for **.rtf* template modification, or Macromedia Dreamweaver could be used for **.html* template editing.

To add a report data into the template:

1. On the **Tools** menu, click **Report Wizard**. The **Report Wizard** dialog will open.
2. Select a template or create a new one. Click **Next**. The **Select Report Data** pane will appear.
3. Click **New**. The **New** dialog will appear.
4. Enter the report data name and description. Click **Create**. A new report data will be created. In the next step, you may add new fields or delete the existing ones.

To remove report data from the template:

1. On the **Tools** menu, click **Report Wizard**. The **Report Wizard** dialog will open.
2. Select a template or create a new one. Click **Next**. The **Select Report Data** pane will appear.
3. Select the report data from the list and click **Delete**.

To set the default viewer option for the report file:

1. On the **Tools** menu, click **Report Wizard**. The **Report Wizard** dialog will open.
2. Select a template and report data and specify the variables and package scope. Click the **Next** button to proceed.
3. At the last wizard step, select the **Display in viewer after generating report** check box. The report output will be displayed in the default editor or browser.

To change an output image format:

1. On the **Tools** menu, click **Report Wizard**. The **Report Wizard** dialog will open.
2. Select a template and report data and specify the variables and package scope. Click the **Next** button to proceed.
3. At the last wizard step, select the output image format from the **Report Image Format** box.

NOTE	Supported image formats include: • ***.PNG** and ***.JPG**: for all supported report templates. • ***.SVG**: for HTML, XML and Text report templates. • ***.EMF** and ***.WMF**: for RTF, OOXML and Text report templates.

To change an empty value configuration:

1. On the **Tools** menu, click **Report Wizard**. The **Report Wizard** dialog will open.
2. Select a template and report data and specify the variables and package scope each time.Click the **Next** button to proceed.
3. At the last wizard step, select an option for output on blank fields:
   - Select **Display empty value** or **Display value as**, and select **NA** or
   - Enter the text to represent other than null value when the template query fields return empty.

To add a new variable:

1. On the **Tools** menu, click **Report Wizard**. The **Report Wizard** dialog will open.
2. Select a template and report data. Click the **Next** button to proceed.
3. Click the **New** button when the **Variable** pane opens. The **Variables** dialog will open.
4. Enter the name of a new variable and its value (The value can be modified later after the variable has been created).
5. Click **Create**.

To delete a variable:

1. On the **Tools** menu, click **Report Wizard**. The **Report Wizard** dialog will open.
2. Select a template and report data. Click the **Next** button to proceed.
3. Select a field and click **Delete** when the **Variable** step opens.

NOTE	A deleted report data cannot be recovered.

To modify a variable:

1. On the **Tools** menu, click **Report Wizard**. The **Report Wizard** dialog will open.
2. Select a template and report data. Click the **Next** button to proceed with the steps.
3. Select a field and modify its value when the **Variable** step opens. The value can be modified in the properties list or in the **Field Value** box below the properties list.

To select a package for report generation:

1. On the **Tools** menu, click **Report Wizard**. The **Report Wizard** dialog will open.
2. Select a template and report data and specify the variables. Click the **Next** button to proceed.
3. The package tree will open. Select the package of the project and click the **Add** button to add the elements from the package to the report.

NOTE	Clicking the **Add** button will add only the selected element, not its children, to the report scope. In order to include all children, click the **Add Recursively** button instead.

To generate a normal text output format:

1. Create the MagicDraw query in the text file using the text editor (Figure 125).

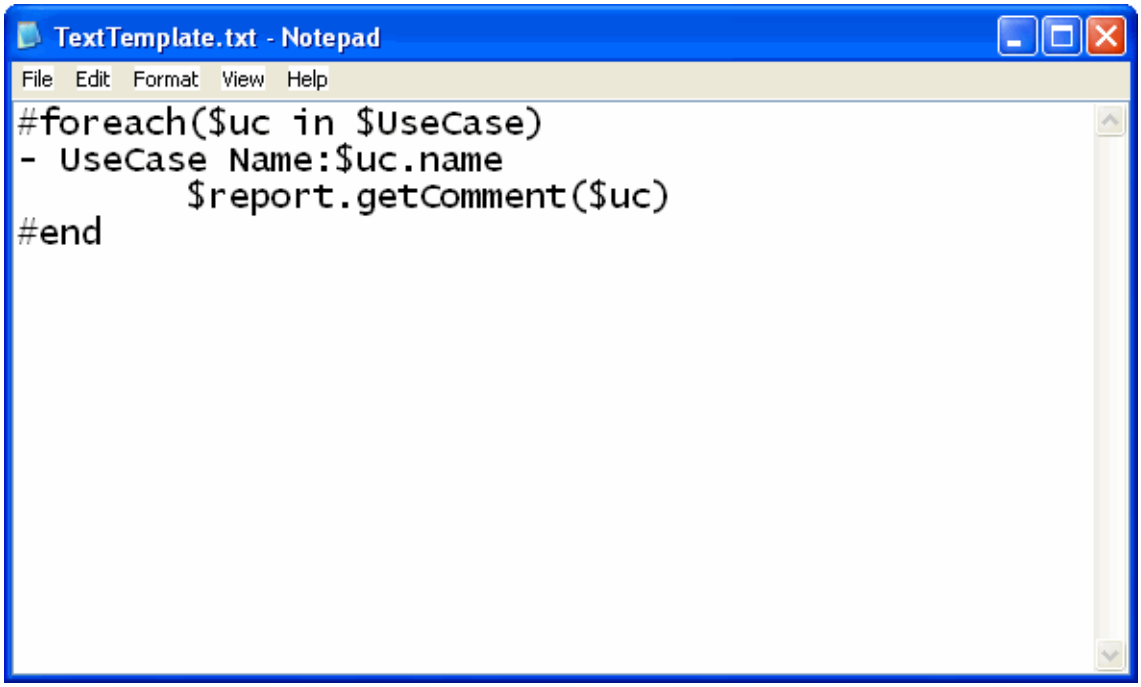


Figure 125 -- Entering Query in the Normal Text Template

2. Create a new template in the **Report Wizard** dialog via the **New Template** Dialog (Figure 126).

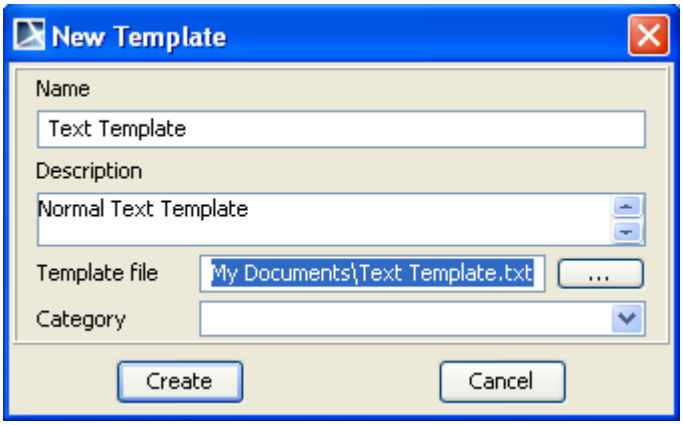


Figure 126 -- Creating New Text Template

3. Select **Text Template** (Figure 127) and generate the output report (Figure 128).

.

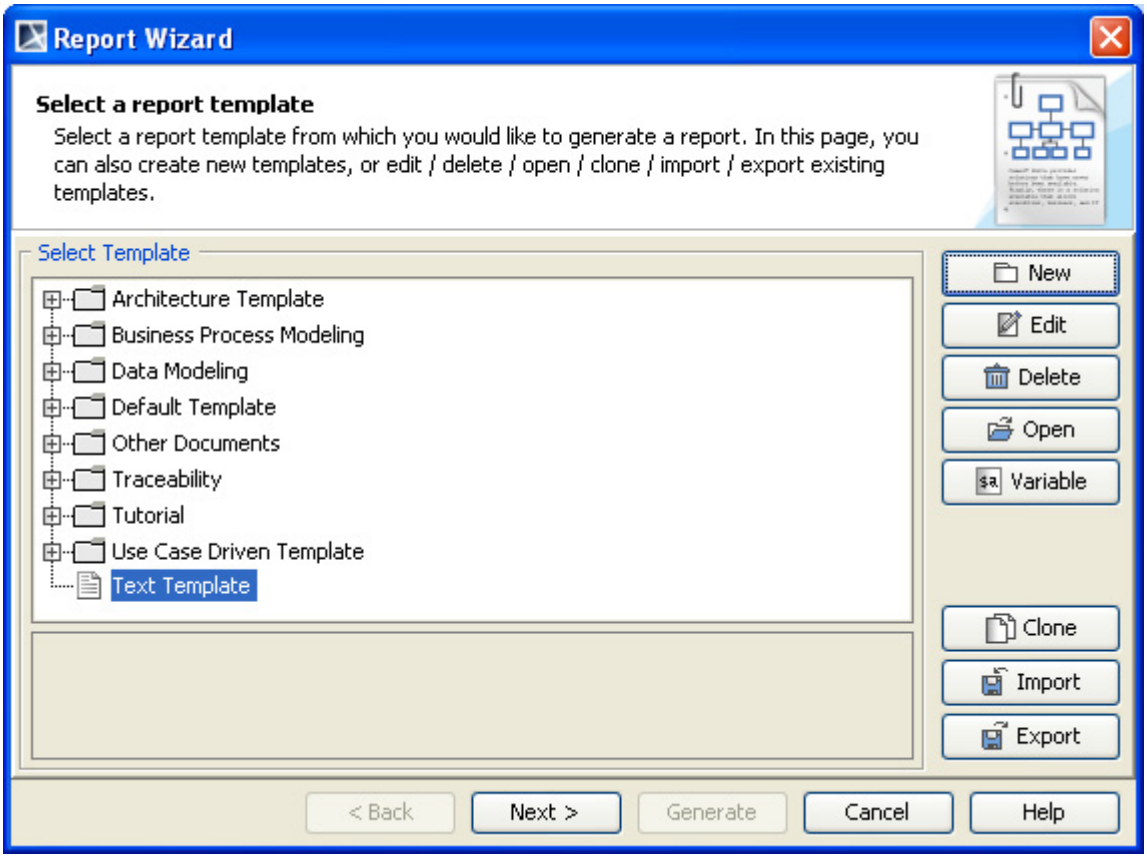


Figure 127 -- Selecting Text Template to Generate Report Output

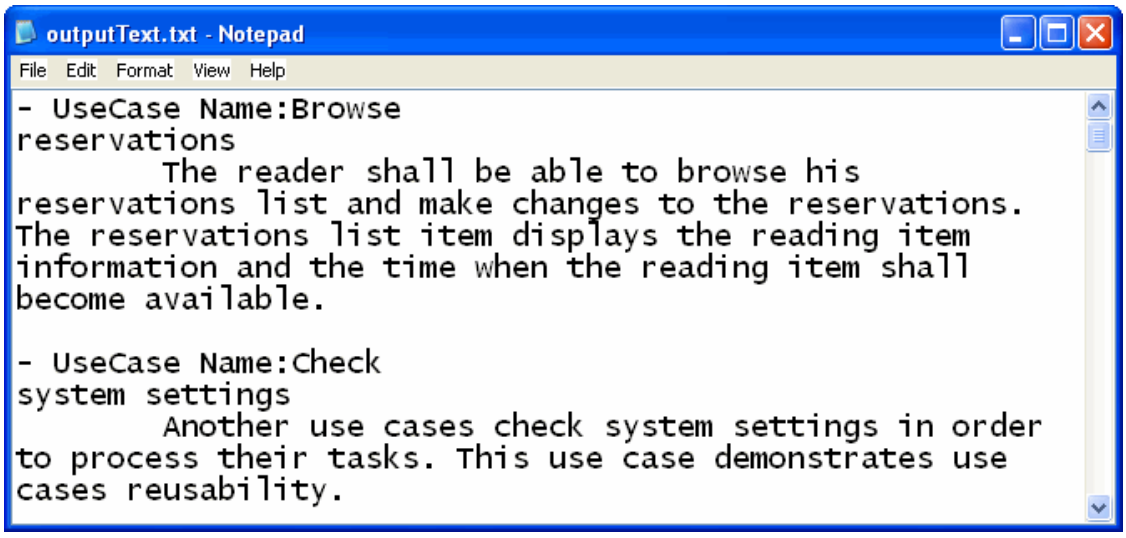


Figure 128 -- Text Output Report

7. Generating Reports from the Containment Tree

You can generate reports directly from the Containment tree in MagicDraw.

To generate reports from the Containment tree:

1. Either:

   Right-click a package in the Containment tree to open the shortcut menu, as displayed in Figure 129. Select **Generate Report...**, and then select a template from the category list or template item.

   or

   Right-click a report data in the Containment tree to open the shortcut menu, as displayed in Figure 130. Select **Quick Generate Report...**.

NOTE	You must specify the template and data in the report data before using the **Quick Generate Report...** shortcut menu.

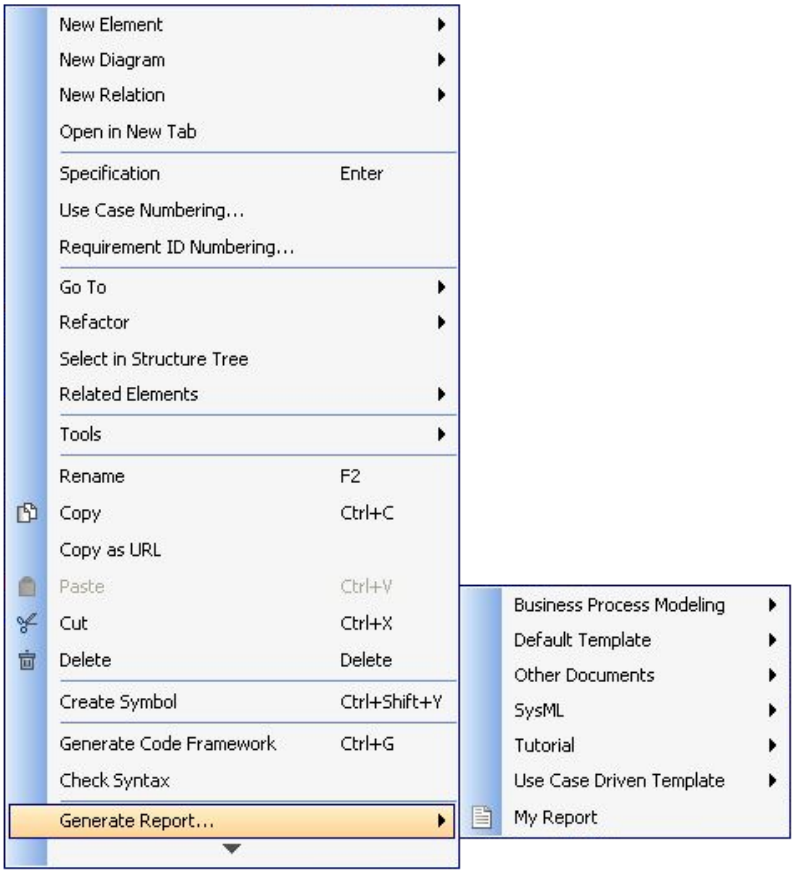


Figure 129 -- Containment Tree Shortcut Menu - Generate Report

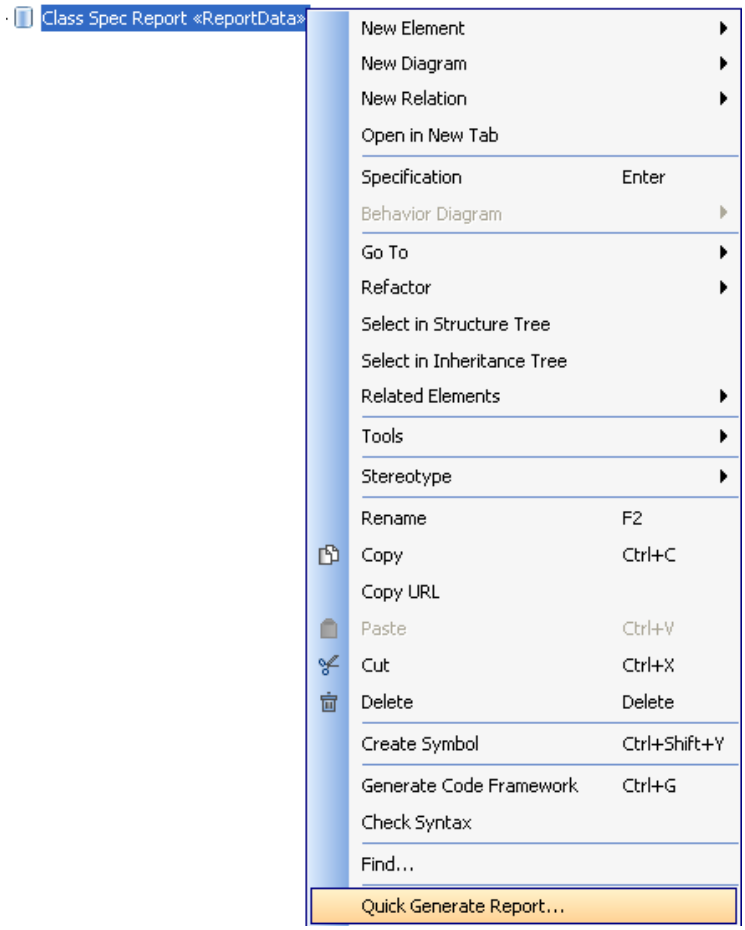


Figure 130 -- Containment Tree Shortcut Menu - Quick Generate Report

2. Select the save location and type the filename.
3. A dialog will open, asking if you want to open the report in the default viewer after the report generating process is complete (Figure 131).

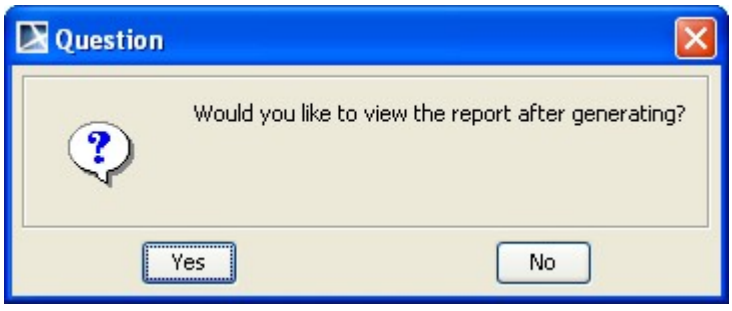


Figure 131 -- Question Dialog - View Generated Report

NOTE	• To open the Report Wizard dialog and modify the data or output properties, click the **Report Wizard...** on the shortcut menu. • By default, the quick generate report command will select the default settings from the last changes saved to the report data.

8. Generating Reports from the Command Line

To generate reports and schedule reports printing, type the command in the terminal. This feature allows you to generate reports without opening the MagicDraw application and using Report Wizard.

8.1 Generate - the Command to Generate Reports

To generate a report from the package scope:

- **generate** - `project` *`projectFileName`* `-output` *`outputFileName`* `-template` *`templateName`* `-package` *`packageNameList`*

To generate a report from the element scope:

- **generate** `-project` *`projectFileName`* `-output` *`outputFileName`* `-template` *`templateName`* `-element` *`elementNameList`*

To generate a report from the option specified in the properties file:

- **generate** `-properties` *`propertiesFileName`*

To generate a Teamwork password for a properties file:

- **generate** `-generatepassword` *`password`*

8.1.1 Synopsis

- `-project` *`projectFileName`*

  This command specifies a filename with the MagicDraw project path.

- `-output` *`outputFileName`*

  This command specifies a filename with the output file path.

- `-template` *`templateName`*

  This command specifies a template name to be used for generating a report document. The template must exist in MagicDraw prior to using this command, otherwise an error will occur. If the template name is not specified, the template specified in the report data (specified in **-report** `reportName`, see Section 8.2 Options) will be used instead.

- `-package` *`packageNameList`*

  This command specifies the name of one or more packages from a MagicDraw project that will be included in the report. Package entries must be separated by a semicolon (;).

- `-element` *`elementNameList`*

  This command specifies the name of one or more elements from a MagicDraw project that will be included in the report. Element entries must be separated by a semicolon (;).

- `[options]`

  The command-line options. See Section **8.2 Options** for more details.

- `-properties` *`propertiesFileName`*

  This command specifies the name of a properties file to utilize. Use only with `-properties`.

- `-generatepassword` *`password`*

  This command generates an encrypted password to be used with a properties file.

8.1.2 Description

8.1.2.1 Generating a Report

The Generate command creates a report document with a received set of information from arguments and parameters. The information will then be generated as a report document to the specified output file. By default, an argument is the specified data of the parameters that are invoked. If the `-properties` option is specified, the argument is the name of a properties file. A properties file contains other parameters with the specified data of each parameter.

8.1.2.2 Generating a Report from Teamwork Server (and Cameo Team Server)

The command arguments available include:

- `-server`, `-login`, and `-password` are key arguments to log on to Teamwork Server.
- You will be prompted for a password when trying to enter `-server` and `-login` without `-password`.
- If `-spassword` is used with a properties file, an encrypted password must be used for this property.
- `-cameoserver` flag must be set to “true” if you are going to connect to a Cameo Team Server instead.

8.2 Options

The command line feature supports a set of options that are useful in adding more configuration possibilities.

- **-report** `reportName`

  This command specifies the name of a report data file or report data element. The report data file must exist in MagicDraw prior to using this command, otherwise an error will occur. If the `reportName` is not specified, the previous report data will be set and used instead.

- **-autoImage** `0|1|2|3`

  This command specifies how an image will be shown in a report document. The default value for this argument is 1.

  0 – No resize.

  1 – Fit image to paper (large only).

  2 – Fit and rotate image (clockwise) to paper (large only).

  3 – Fit and rotate image (counter-clockwise) to paper (large only)

- **-imageFormat** `jpg|png|svg|emf|wmf`

  This command specifies an image format in a report document. The default value for this argument is jpg.

  jpg – Joint Photographic Expert Group

  png – Portable Network Graphics

  svg – Scaling Vector Image

  emf – Microsoft Windows Enhanced Metafile

  wmf – Windows Metafile

- **-recursive** `true|false`

  This command specifies how to select a package in a MagicDraw project. The default value for this argument is true.

  true – select the specified package and its recursive package.

  false – select only the specified package.

- **-includeAuxiliary** `<true|false>`

This command is used to select packages including auxiliary packages. The default value is false.

- **-outputOnBlankField** `stringValue`

  This command will show a string value in a blank field in a report document. The default value for this argument is “”.

- **-category** `categoryName`

  This command specifies a template category.

- **-fields** `[name=value]`

  This command specifies a variable.

- **-server** `serverName|IPAddress`

  This command specifies the name or IP Address of a Teamwork Server project.

- **-login** `loginName`

  This command specifies the login name to log on to Teamwork Server.

- **-password** `password`

  This command specifies the password to log on to Teamwork Server.

- **-spassword** `encryptedPassword`

  This command specifies the encrypted password (Teamwork password) to log on to Teamwork Server. This option is available only in a properties file.

- **-cameoserver** `<true|false>`

  This command must be set to true if you are going to generate a report from a Cameo Team Server.

8.3 Properties Filename

When `-properties` is used, other command actions are not necessary anymore. The configuration information is contained in the properties file. The information in the properties file is the same as you would put in the command line.

There are four parameters needed to specify data for a report: (i) project, (ii) output, (iii) template, and (iv) package. There are also other five parameters called options which are additional configuration information such as report, autoImage, imageFormat, recursive, and outputOnBlankField. These nine parameters are described in the previous sections **8.1 Generate - the Command to Generate Reports** and **8.2 Options**.

A properties file is a simple text file. You can create and maintain a properties file with any text editors. For example:

```
#---- main argument ----#
project = C:\\MagicDraw\\samples\\diagrams\\class diagram.mdzip
output = C:\\output\\output.rtf
template = Class Specification Report
package = java;Library System;magiclibrary;objects

#---- optional ----#
report = Built-in
autoImage = 1
imageFormat = png
recursive = false
outputOnBlankField = #NA
```

Figure 132 -- Sample of Properties File

In Figure 132, the command lines begin with a pound sign (#). The other lines contain key-value pairs. The key is on the left side of the equal sign, and the value is on the right. For instance, `template` is the key that corresponds to the value of `Class Specification Report`.

The string that represents the key and value in the properties file has a special character. The special character is an escape character and needs to be used to prevent interpretation errors. Each escape character starts with a backslash.

Table 18 -- Available Character Sequences

Escape Sequence	Character
\n	new line
\t	tab
\b	backspace
\f	form feed
\r	return
\"	" (double quote)
\'	' (single quote)
\\	\ (backslash)
\uDDDD	character from the Unicode character set (DDDD is four hex digits)

8.3.1 XML Properties File

J2SE version 1.5 and Report Wizard version 15.5 allow you to use XML files to load and save key-value pairs.

Properties DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- DTD for properties -->
<!ELEMENT properties ( comment?, entry* ) >
<!ATTLIST properties version CDATA #FIXED "1.0">
<!ELEMENT comment (#PCDATA) >
<!ELEMENT entry (#PCDATA) >
<!ATTLIST entry key CDATA #REQUIRED>
```

For example:

```
<?xml  version = "1.0" encoding = "UTF-8"?>
<!DOCTYPE properties SYSTEM
"http://java.sun.com/dtd/properties.dtd">
<properties>
     <entry key = "project">C:\MagicDraw\samples\diagrams\class
diagram.mdzip</entry>
     <entry key = "output">C:\\output\\output.rtf</entry>
     <entry key = "template">Class Specification Report</entry>
     <entry key = "package">java;Library
System;magiclibrary;objects</entry>
</properties>
```

Figure 133 -- Sample of XML Property

8.4 Using Command Line

You can use the command line on Windows, Linux, or Unix.

To use the command line on Windows:

1. Run a command line console. In the Console window, go to the local installation of the MagicDraw application.
2. Go to the **plugins\com.nomagic.magicdraw.reportwizard** folder and type the command line there.

To use the command line on Linux or Unix:

1. Run a terminal (command line console). In the Console window, go to the local installation of the MagicDraw application.
2. Go to the **plugins\com.nomagic.magicdraw.reportwizard** folder and type the command line there.

Examples:

```
generate -properties "C:\\output\\prop.properties"
```

```
generate -project "C:\\MagicDraw\\samples\\diagrams\\class
diagram.mdzip" -output "C:\\output\\output.rtf" -template "Class
Specification Report" -package "Data" -report "Built-in" -
autoImage 1 -imageFormat png -recursive false -
outputOnBlankField "#NA"
```

Figure 134 -- Samples of Command Line

8.5 Syntax Rules

- Command parameters containing spaces must be enclosed in quotes, for example, "UML Standard Profile".
- Command parameters are case sensitive, except `project`, `imageFormat` and `recursive` which are not case-sensitive.
- Blank spaces ( ) separate commands and parameters.
- To specify a package name that contains a semicolon (;), you can use a backslash (\). For example, the package name “com;nomagic” can be typed by using “`com\;nomagic`”. The backslash followed by a semicolon (\;) is not considered to be a character separator.
- For a parameter that contains unicode characters such as Thai, you can use the xml properties file to generate a report. The properties file does not support the unicode encoding.
- A field entry consists of a name-value pair. The name and value formats are specified by [name=value] pattern strings. A valid name must start with a letter (a-z or A-Z) and can be followed by any letter, digit, or underscore.
- You can specify a field name that consists of brackets ([ ]) by using backslashes (\ ). For example, "[author=NoMagic][Revision=[1.0]]" will be typed as "`[author=NoMagic][Revision=\[1.0\]]`".
- Either `-package` or `-element` can be used with the Generate command. For example, you can use `-package` without `-element`.

9. Report Wizard Quick Print

Report Wizard provides 5 keyboard shortcuts to create a report, with last used options, from a recently-used template.

Table 19 -- The predefined keyboard shortcuts

Keyboard Shortcut	Quick Print Template
Alt + 1	Template's name A
Alt + 2	Template's name B
Alt + 3	Template's name C
Alt + 4	Template's name D
Alt + 5	Template's name E

To change a keyboard shortcut:

1. On the main menu, click **Options** > **Environment** to open the **Environment Options** dialog.
2. Select **Category: Tools** in the **Keyboard** pane.
3. Press the **New** key and click **Assign**.
4. For more details, see MagicDraw User Manual, Setting Environment Options.

To use Report Wizard Quick Print:

1. After a report has been generated, a new menu will appear below the **Report Wizard...** menu (Figure 135). The name of the new menu is the name of the recently generated template.

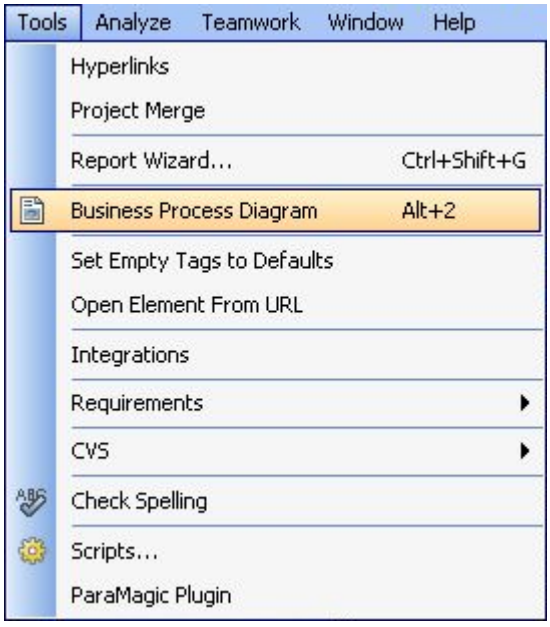


Figure 135 -- Newly-generated Report as a Quick Option in Main Menu

2. Either click this menu or press **Alt+1** to generate this template with the last used options.
3. Select the saved location and filename (Figure 136).

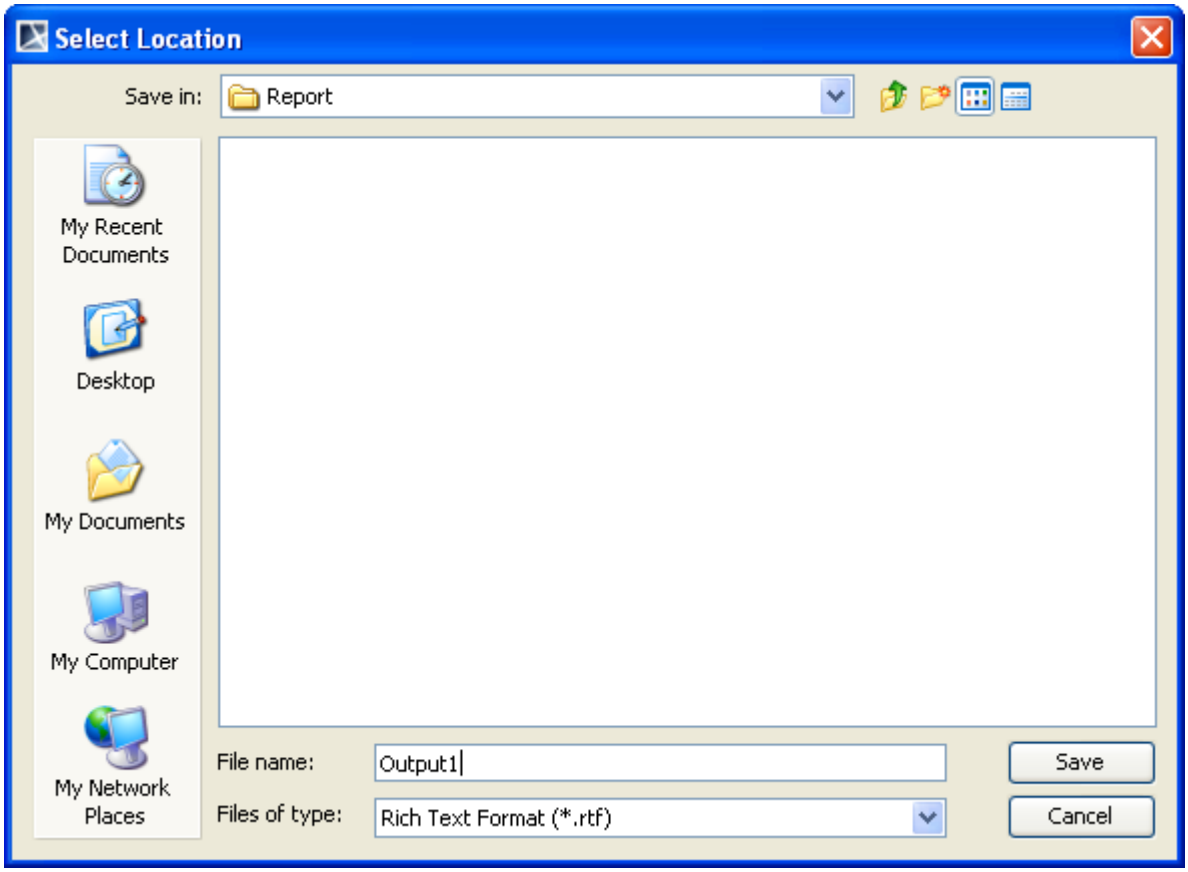


Figure 136 -- Select the Location to Save the Report

4. A message box will open, asking if you want to open the report in the default viewer once the generating process has been completed (Figure 137).

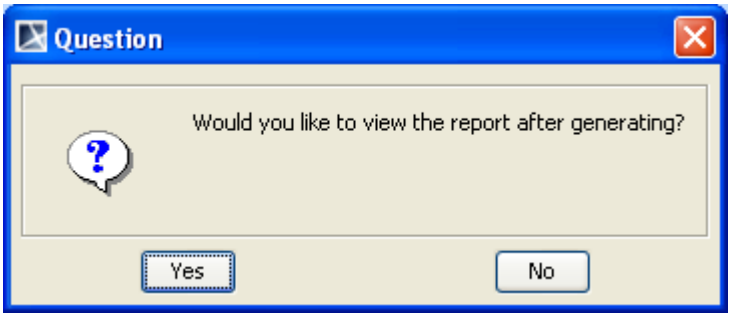


Figure 137 -- Question Dialog - View Report.

5. The next time you generate a report, a new menu will create and map the keyboard shortcuts until the keyboard shortcuts list is full. The number of keyboard shortcuts is limited to 5 only.
6. If the keyboard shortcuts list is full and the new template has been generated, the keyboard shortcuts will be reused.

For example, Report Wizard Quick Print provides the following keyboard shortcuts:

Table 20 -- Sample of Report Wizard Quick Print Keyboard Shortcuts

Shortcut Key	Template Name
Alt + 1	Tutorial 01 – Print Element
Alt + 2	Tutorial 02 – Print Diagram
Alt + 3	Tutorial 03 – Print Project
Alt + 4	Tutorial 04 – Print Report Data
Alt + 5	Tutorial 05 – Macro

Table 21 shows that the **Tutorial 01 – Print Element** keyboard shortcut is the most outdated template and the following report has been completed using the *Business Process Diagram* template. Report Wizard Quick Print will reuse the keyboard shortcut of the most outdated template.

Table 21 -- Reusing Keyboard Shortcuts

Shortcut Key	Template Name
Alt + 1	Tutorial 01 – Print Element
Alt + 2	Tutorial 02 – Print Diagram
Alt + 3	Tutorial 03 – Print Project
Alt + 4	Tutorial 04 – Print Report Data
Alt + 5	Tutorial 05 – Macro

If the next template to be generated has already been in the Quick Print list, the list will not change.

10. Report Wizard Environment Options

The Report Wizard environment options in the MagicDraw Environment Options dialog allow you to configure the report engine. There are three things that you can configure: (i) Report engine properties, (ii) Template mappings, and (iii) Template folder.

(i) Report Engine Properties

The properties that you can configure are as follows:

a. Template process size (int). This value defines the size of allocated memory for processing a template. If the template is larger than the allocated size, the engine will create temporary files on a local disk and process the template from these files.

b. Template pool size (int). The number of processing threads for evaluating a template. An increase in the number of threads may improve the performance of the engine, which will also depend on your hardware and JVM.

c. Velocity properties. These properties are a key-value pair. You can add any number for these properties. You will need a basic knowledge of Velocity to enter the value. (See http://velocity.apache.org/engine/devel/apidocs/org/apache/velocity/runtime/RuntimeConstants.html)

d. Debug report template. You can enable or disable any invalid properties, references, or exception messages on the message window of MagicDraw. (See 11 Debug Report Template)

(ii) Template Mapping

Template mappings associate a template type with the template engine. The Report engine allows you to determine how to handle different types of files, such as JS or DOCBOOK file. All you have to do is specify a filename extension and select an engine name. The Report engine also allows developers to add a user custom engine into this option.

(iii) Template Folder

You can either select or clear the **Monitor template folder** check box (Figure 138) in the Report Wizard environment options in the MagicDraw Environments Options dialog to enable or disable the option to automatically deploy an MRZIP template file. If you select the check box, Report Wizard will automatically deploy the MRZIP template file to the Report Wizard dialog whenever the file is added to your folder. (See 1.2 MRZIP File Automatic Deployment)

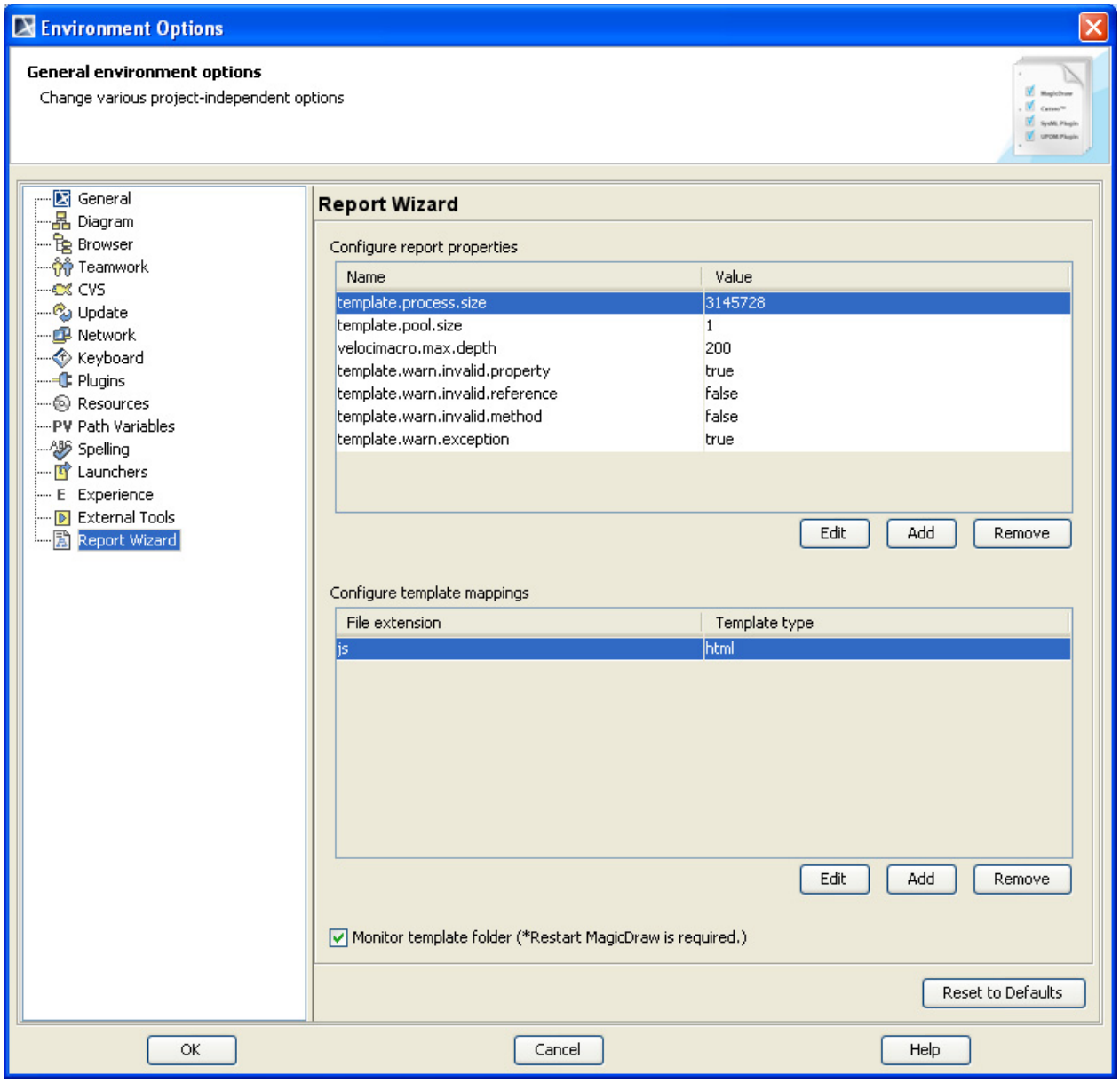


Figure 138 -- Report Wizard GUI Configuration in MagicDraw Environment Options Dialog

10.1 Configuring Report Engine Properties

Figure 139 shows the Configure report properties pane.

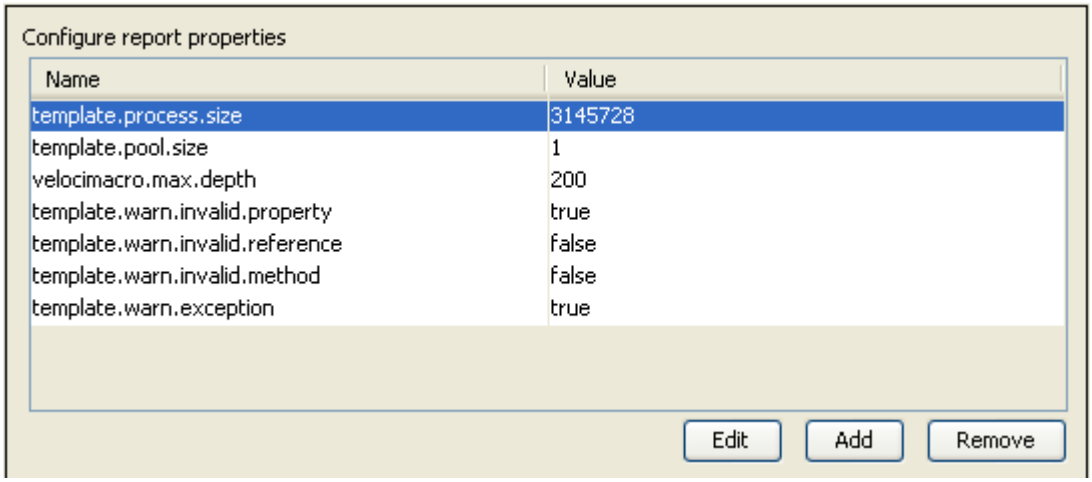


Figure 139 -- Configure Report Properties Pane

This pane contains 3 buttons: (i) Add, (ii) Edit, and (iii) Remove.

(i) Add button

To create a new report property:

1. Click the **Add** button in the **Configure Report Properties** pane. The **Report properties** dialog will open (Figure 140).

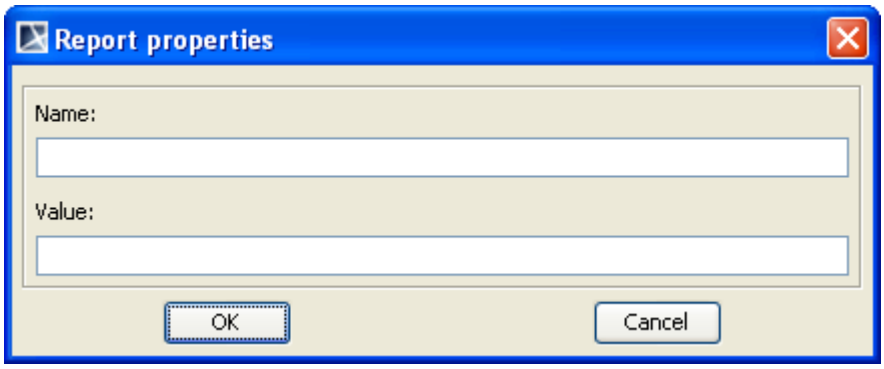


Figure 140 -- Creating a New Report Property in the Report Properties Dialog

2. Enter the property name and value.

Table 22 -- Report Engine Properties Dialog Fields

Field Name	Function	Default Value	Possible Value	Required
Name	To enter the property name	Blank	Text	Yes
Value	To enter the property value	Blank	Text	No

(ii) Edit button

To edit a report engine property:

1. Select a property in the **Configure Report Properties** pane and click the **Edit** button. The **Report properties** dialog will appear (Figure 141).

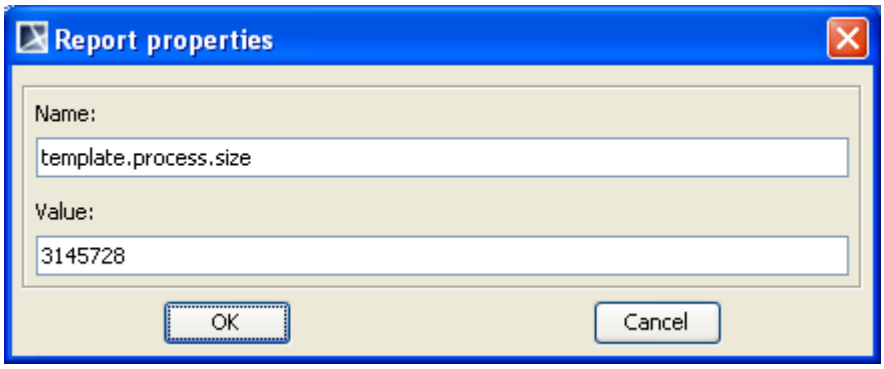


Figure 141 -- Editing a Report Property in the Report Properties Dialog

2. Edit the property name and value. (See Table 22 Report Engine Properties Dialog Fields)

(iii) Remove button

To delete a report property:

1. Select a property in the **Configure Report Properties** pane and click the **Remove** button.

10.2 Configuring Template Mappings

Figure 142 shows the **Configure Template Mappings** pane.

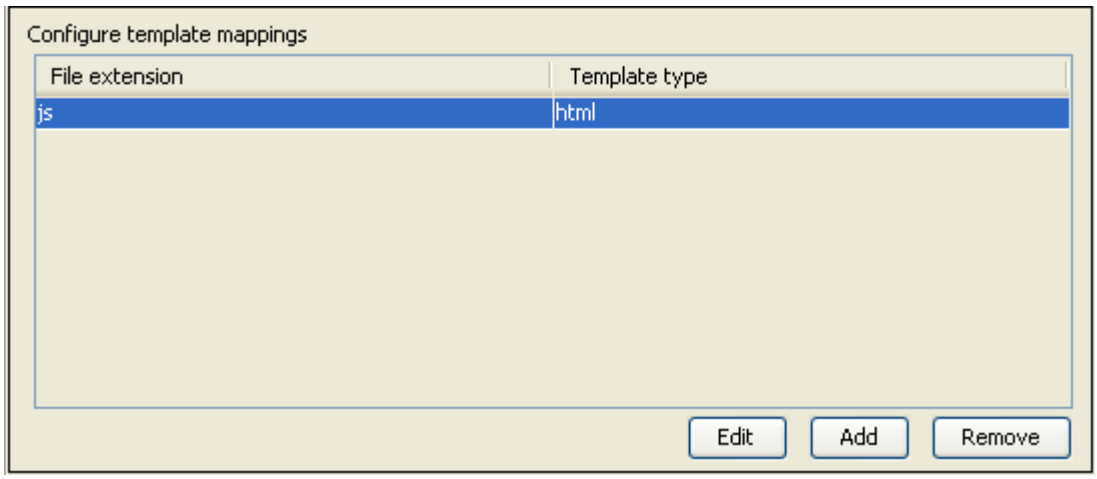


Figure 142 -- Configure Template Mappings Pane

This pane contains 3 buttons: (i) Add, (ii) Edit, and (iii) Remove.

(i) Add button

To create a new template mapping:

1. Click the **Add** button in the **Configure Template Mappings** pane. The **Template mappings** dialog will open (Figure 143).

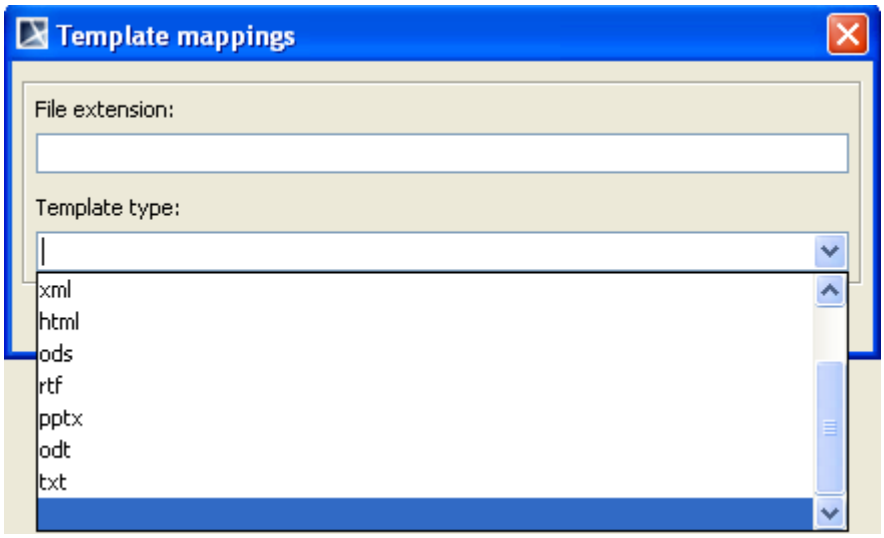


Figure 143 -- Adding Template Mapping in the Template Mappings Dialog

2. Enter a filename extension and template type.

Table 23 -- Template Mappings Dialog Fields

Field Name	Function	Default Value	Possible Value	Required
File extension	To enter a filename extension. The filename extension is case-insensitive	Blank	Text	Yes
Template type	To enter a template type. The template type is case-sensitive	Blank	Text	Yes

(ii) Edit button

To edit a template mapping:

1. Select a template mapping in the **Configure Template Mappings** pane and click the **Edit** button. The **Template mappings** dialog will open (Figure 144).

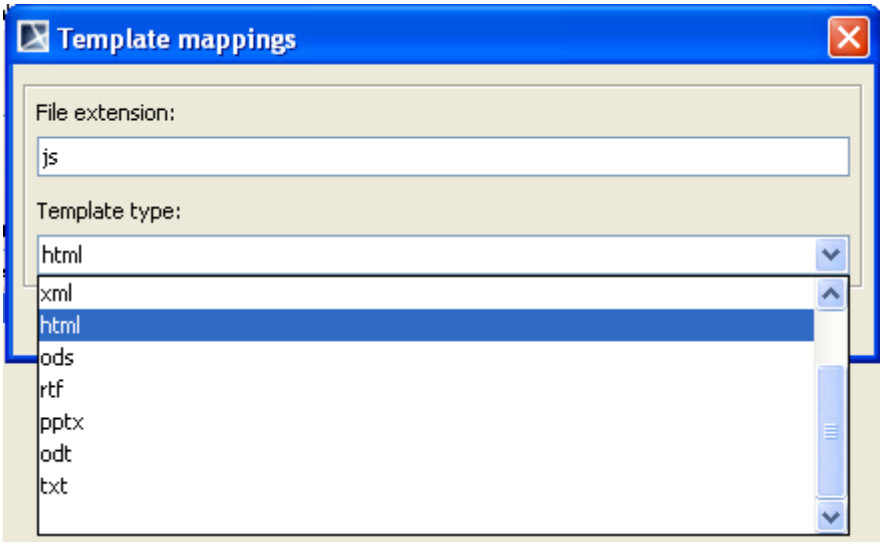


Figure 144 -- Editing Template Mapping in the Template Mappings Dialog

2. Edit the filename extension and template type. (See Table 23 Template Mappings Dialog Fields)

(iii) Remove button

To delete a template mapping:

1. Select a template mapping in the **Configure Template Mappings** pane and click the **Remove** button.

NOTE	When template mapping is removed from environment option, the file extension mapping will be removed from MagicDraw. To restore default setting, you have to restart MagicDraw.

10.3 Monitor Template Folder Option

You can either enable or disable the MRZIP template file automatic deployment option in the Report Wizard environment options (See 1.2 MRZIP File Automatic Deployment) by selecting or clearing the Monitor template folder check box (Figure 145).

Figure 145 -- Monitor Template Folder Pane.

10.4 Reset to Defaults Option

Click the **Reset to Defaults** button to reset data to the default settings (Figure 146).

Figure 146 -- Reset to Defaults Button

NOTE	• MagicDraw allows you to configure the report engine settings in three levels: (i) Environment option, (ii) global config.xml, and (iii) template config.xml. • The template config.xml configuration settings will override the environment option configuration settings, which will override the global config.xml configuration settings.

11. Debug Report Template

The Debug Report template features warning messages that describe template errors. A warning message will appear in **Messages Window** while the report template is being processed. All warning messages have no effect on the output report, thus can be ignored. They are intended for the informative purpose only.

There are 5 warning message types:

11.1 Invalid Property

The warning (Figure 147) will open when an invalid property is encountered during the report generation. For example, getting a text property from the `$package` variable, when the text property is not a valid property for `$package`, is a variable with an invalid property that will trigger a warning.

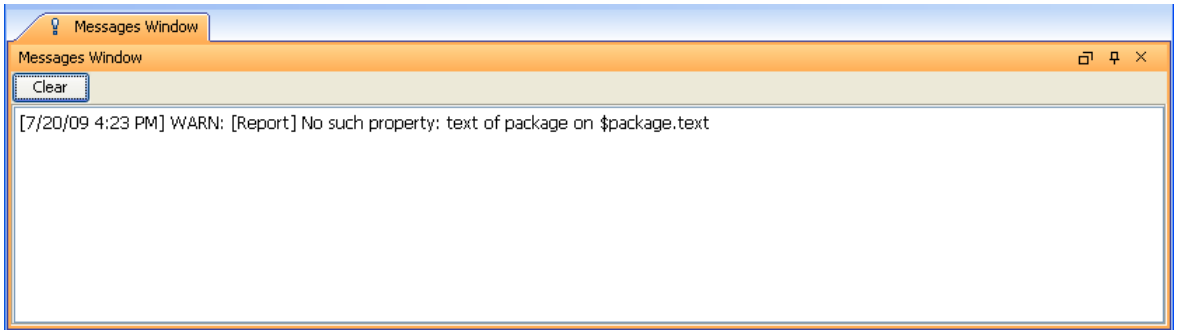


Figure 147 -- Invalid Property Warning Message

11.2 Invalid Reference

The warning will open (Figure 148) whenever using a variable, for example, `$empty` that has not been specified.

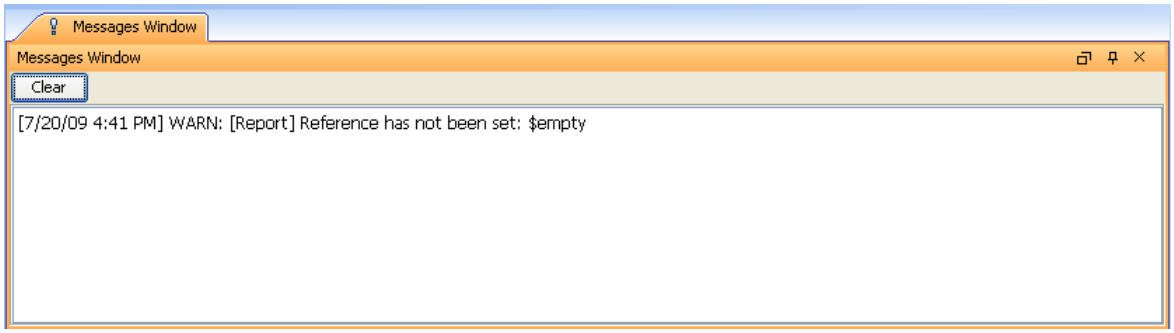


Figure 148 -- Invalid Reference Warning Message

11.3 Invalid Method Reference

The warning message will open (Figure 149) when the use of an invalid method from a variable, such as using the “getStereotypeProperty” method from the `$report` variable whose “getStereotypeProperty” method is not a valid method for `$report`, is encountered during the report generation.

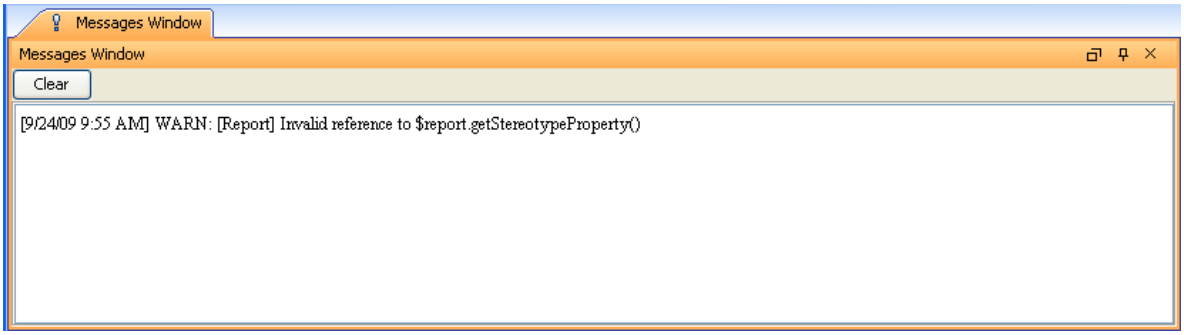


Figure 149 -- Invalid Method Reference Warning Message

11.4 Exception

The warning message (Figure 150) will open whenever an exception, such as IndexOutOfBoundException, occurs during the report generation.

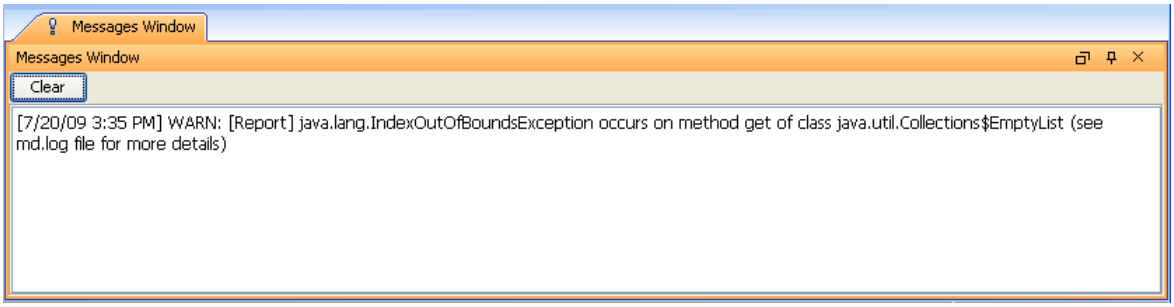


Figure 150 -- Exception Warning Message

11.5 Invalid Syntax

The warning message (Figure 151) will open whenever a template contains any invalid syntax, such as using `#forrow` without `#endrow`.

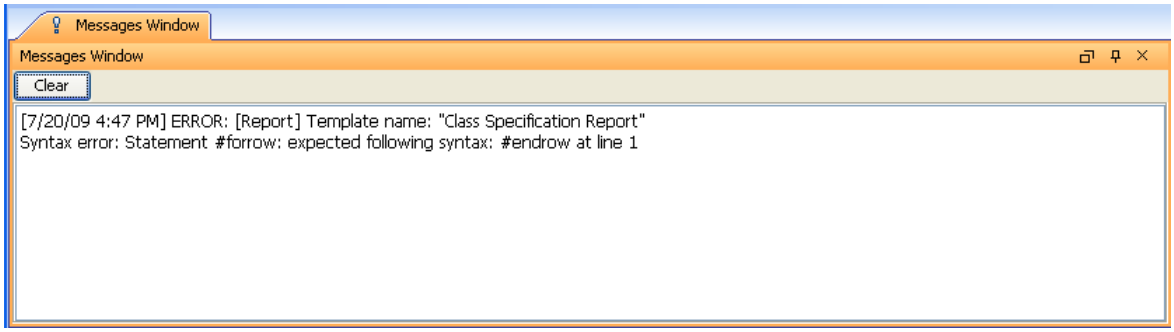


Figure 151 -- Invalid Syntax Warning Message

11.5.1 Enabling or Disabling Warning Messages

You can enable or disable any invalid properties, references, and exception messages except invalid syntax messages by editing the config.xml file and tags and values. Invalid syntax messages are always enabled.

11.5.1.1 Modifying config.xml

To modify the config.xml file:

- Either (i) edit the config.xml file in the **...\plugins\com.nomagic.magicdraw.reportwizard\data** folder to enable or disable all templates' warning messages or (ii) edit the config.xml file or create a file in the **...\plugins\com.nomagic.magicdraw.reportwizard\data\templatefolder** folder for each template.

For example, to enable or disable a warning message of Class Specification Report you need to edit the config.xml file in the **...\plugins\com.nomagic.magicdraw.reportwizard\data\Class Specification Report** folder.

11.5.1.2 Adding Tags and Values

```
<tag>value</tag>
```

Figure 152 -- A Tag for a Warning Message Type

Table 24 -- Warning Message Types

Warning Message Type	Tag
Invalid Property	<template.warn.invalid.property>
Invalid Reference	<template.warn.invalid.reference>
Invalid Method Reference	<template.warn.invalid.method>
Exception	<template.warn.exception>

The value is a boolean value:

- the boolean value is "true" when enabling warning messages
- the boolean value is "false" when disabling warning messages

An example of how an Invalid Reference warning message of Class Specification Report can be enabled is shown in Figure 153.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<configuration>
<template.warn.invalid.reference>true</template.warn.invalid.reference>
</configuration>
```

Figure 153 -- The Invalid Reference Warning Message Enabled through File config.xml

12. Use Case Driven

Report Wizard provides a set of templates to support the Use Case Driven approach methodology. A MagicDraw project that creates the Use Case Driven approach development steps can generate a report to capture elements.

Report Wizard provides 3 templates for the Use Case Driven approach:

12.1 Use Case Specification Report

12.2 Method Specification Report

12.3 Use Case Project Estimation Report

You can create a new MagicDraw project using the Use Case Driven approach either as:

(i) A blank project with the **UseCase_Profile.xml** module or

(ii) A Use Case project

(i) To create a new project as a blank project using **UseCase_Profile.xml**:

1. On the main menu, click **File**.
2. Click **Use Module…** . The **Use Module** dialog will open.
3. Select the **UseCase_Profile.xml** module.
4. Click **Finish**. The MagicDraw project will apply the Use Case Description profile.

(ii) To create a new project as a Use Case Project:

- A MagicDraw project will apply the Use Case Description profile automatically.

12.1 Use Case Specification Report

The Use Case Specification report shows the details of actors and use cases in the project. The first part, which is the actor summary, shows the list of all actors and their associated use cases. The second part presents the use case specification in general information of relations and scenarios. The report supports use case and SysML use case diagrams.

12.2 Method Specification Report

The purpose of the Method Specification report is to show the method design and its related use cases in details to developers.

The report shows a class diagram and a list of all classes in the diagram. The list contains attributes and methods for each class. The report also shows the method specification, parameters, and algorithms.

12.3 Use Case Project Estimation Report

The use case project estimation or 'Use Case Points' is a method for sizing and estimating projects developed with the object-oriented method, developed by Gustav Karner of Objectory (now Rational Software). The method is an extension of Function Point Analysis and Mk II Function Point Analysis (an adaption of FPA mainly used in the UK).

The Use Case Project Estimation report is intended to estimate the project size and duration of hourly manpower required based on the use case model. The report supports use case and SysML use case diagrams.

The use case point technique is applied to reestimate the project size. All the use cases used in the example below are based on the use case diagram. The following weight criteria are used:

12.3.1 Classifying Actors

Table 25 -- Actors Classification

Actor complexity	Litmus Test to recognize classifications	Weight
Low complexity	An external system with a well-defined API.	1
Average complexity	Either a human with a command line interface or an external system via some protocols or a database.	2
High complexity	A human with a GUI or a web page.	3

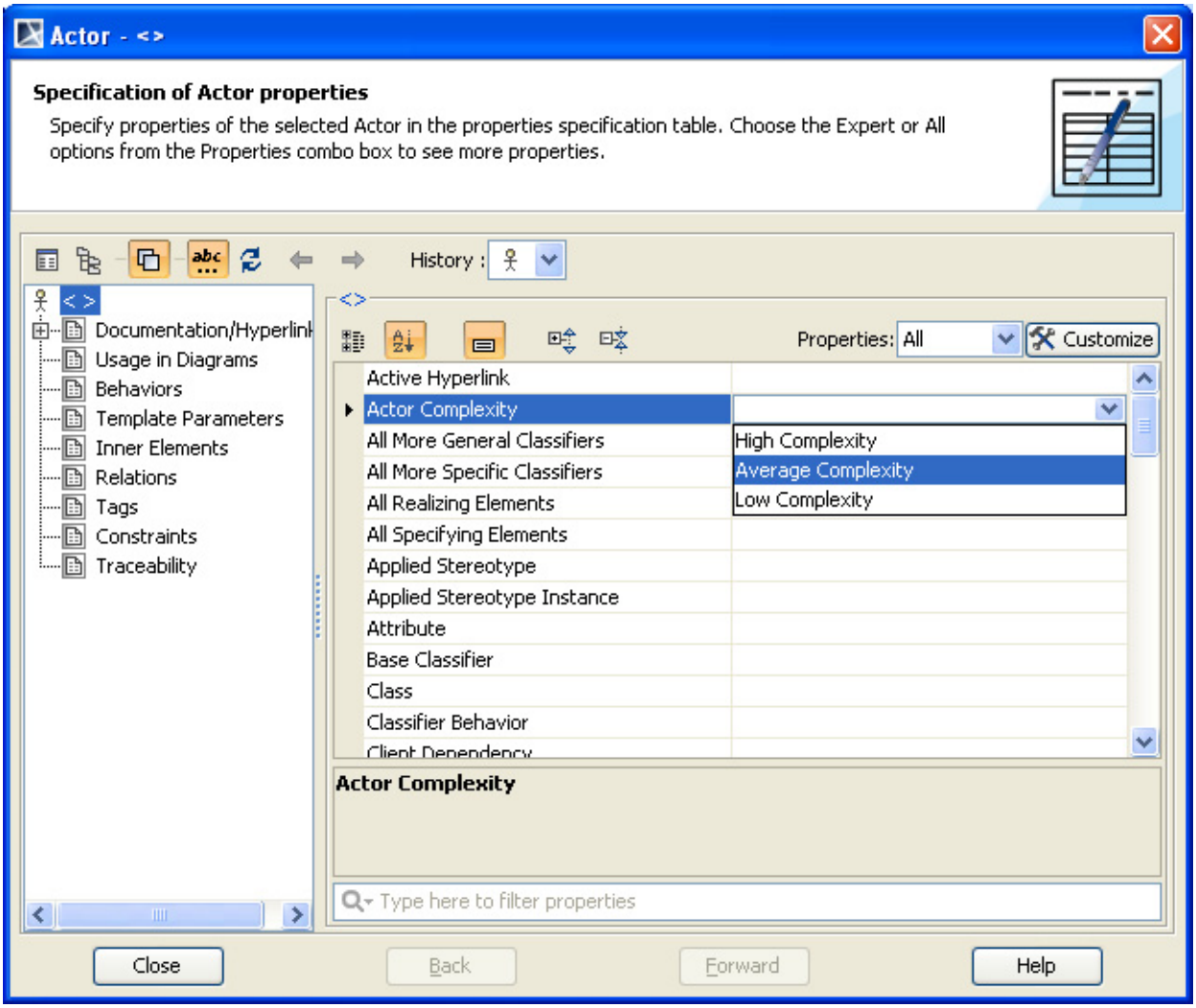


Figure 154 -- Use Case Driven - Actor Classification

12.3.2 Unadjusted Actor Weights

Unadjusted Actor Weights (UAW) is obtained by counting how many actors there are in each category, multiplying each total by its weight, and adding up the products.

12.3.3 Determining Scenarios and Transactions of Use Cases

Table 26 -- Use Case Complexity

Use Case Complexity	Litmus Test To Decide Classifications	Weight
Low complexity	1 - 3 transactions	5
Average complexity	4 - 7 transactions	10
High complexity	> 7 transactions	15

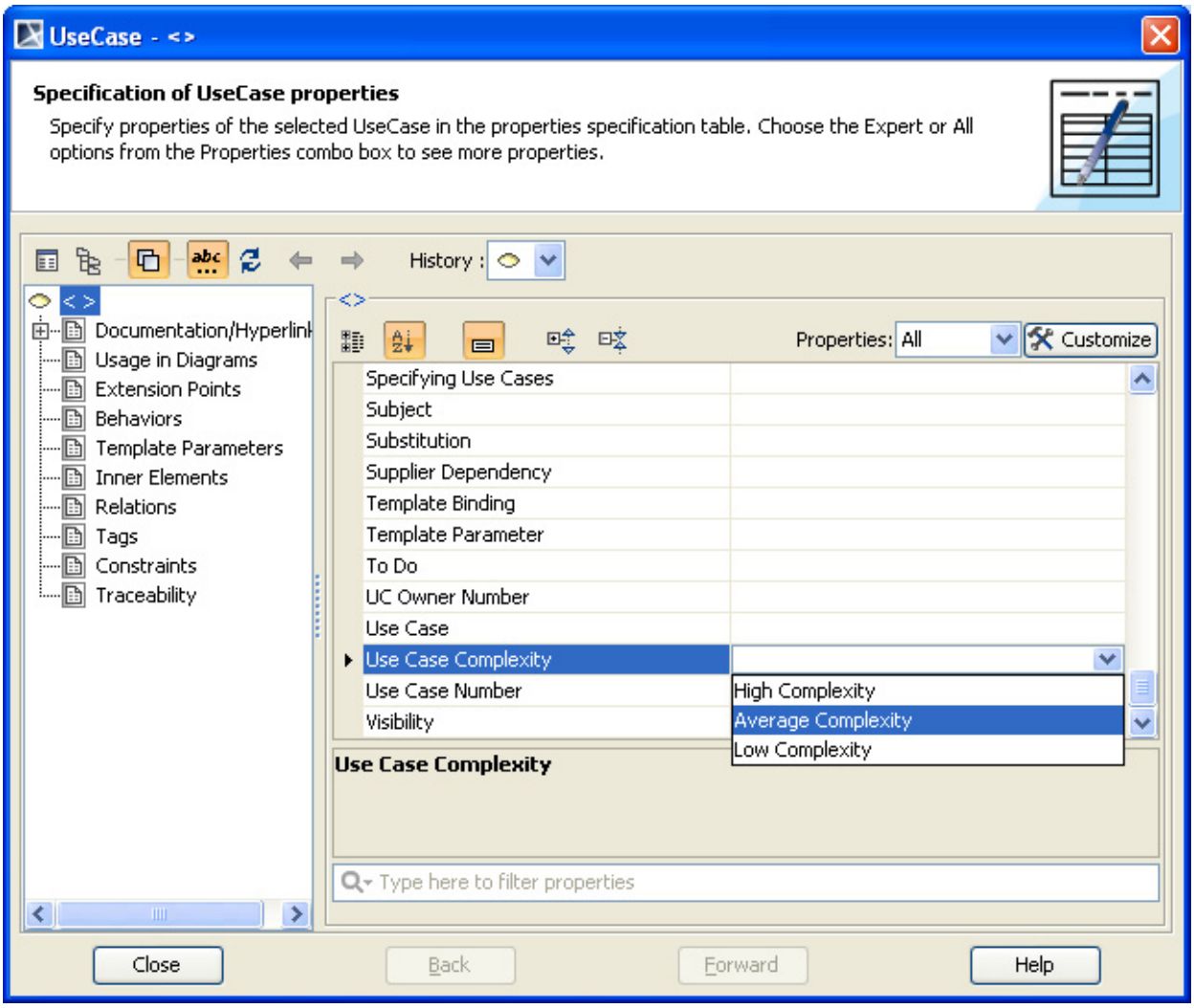


Figure 155 -- Use Case Driven - Use Case Complexity

12.3.4 Unadjusted Use Case Weights

A transaction is a set of activities, which is either performed entirely or not at all. To determine the number of transactions, you need to count the use case steps. Next, multiply each use case type by the weighting factor and add up the products to get **Unadjusted Use Case Weights (UUCW)**.

12.3.5 Unadjusted Use Case Point

Add UAW and UUCW to get **Unadjusted Use Case Point (UUCP)**: *UUCP = UAW + UUCW.*

Reference Documents

- The Estimation of Effort Based on Use Cases, published by Rational software

  ftp://ftp.software.ibm.com/software/rational/web/whitepapers/2003/finalTP171.pdf
- COCOMO II

  http://sunset.usc.edu/csse/research/COCOMOII/cocomo_main.html

12.4 Project Characteristics

The Technical and Environmental factors are associated with the weight of the project. You have to assign a value to each factor. The value assigned to a particular factor depends on the degree of influence a factor has. The relevance values range from 0 to 5, where 0 means no influence, 3 is the average level of influence, and 5 means a strong influence.

12.4.1 Technical Factors

Table 27 -- Technical Factors

Factor	Technical Factors	Weight	Description
T1	Distributed system	2.0	The system distributed architecture or centralized architecture.
T2	Response adjectives	1.0	The response time is one of the important criteria.
T3	End-user efficiency	1.0	The end-user efficiency.
T4	Complex processing	1.0	The business process is very complex.
T5	Reusable code	1.0	The project maintains high reusability. The design is complex.
T6	Easy to install	0.5	The project requires simple installation.
T7	Easy to use	0.5	User-friendly is one of the important criteria.
T8	Portable	2.0	The project is cross-platform implementation.
T9	Easy to change	1.0	The project is highly customizable in the future. The architectural design is complex.
T10	Concurrent	1.0	The project has a large numbers of users working with locking support. The architecture increases the project complexity.
T11	Security features	1.0	The project has heavy security.
T12	Access for third parties	1.0	The project is dependent on the third party's control. Studying and understanding the third party is required.
T13	Special training required	1.0	The application is so complex for the user that training must be provided.

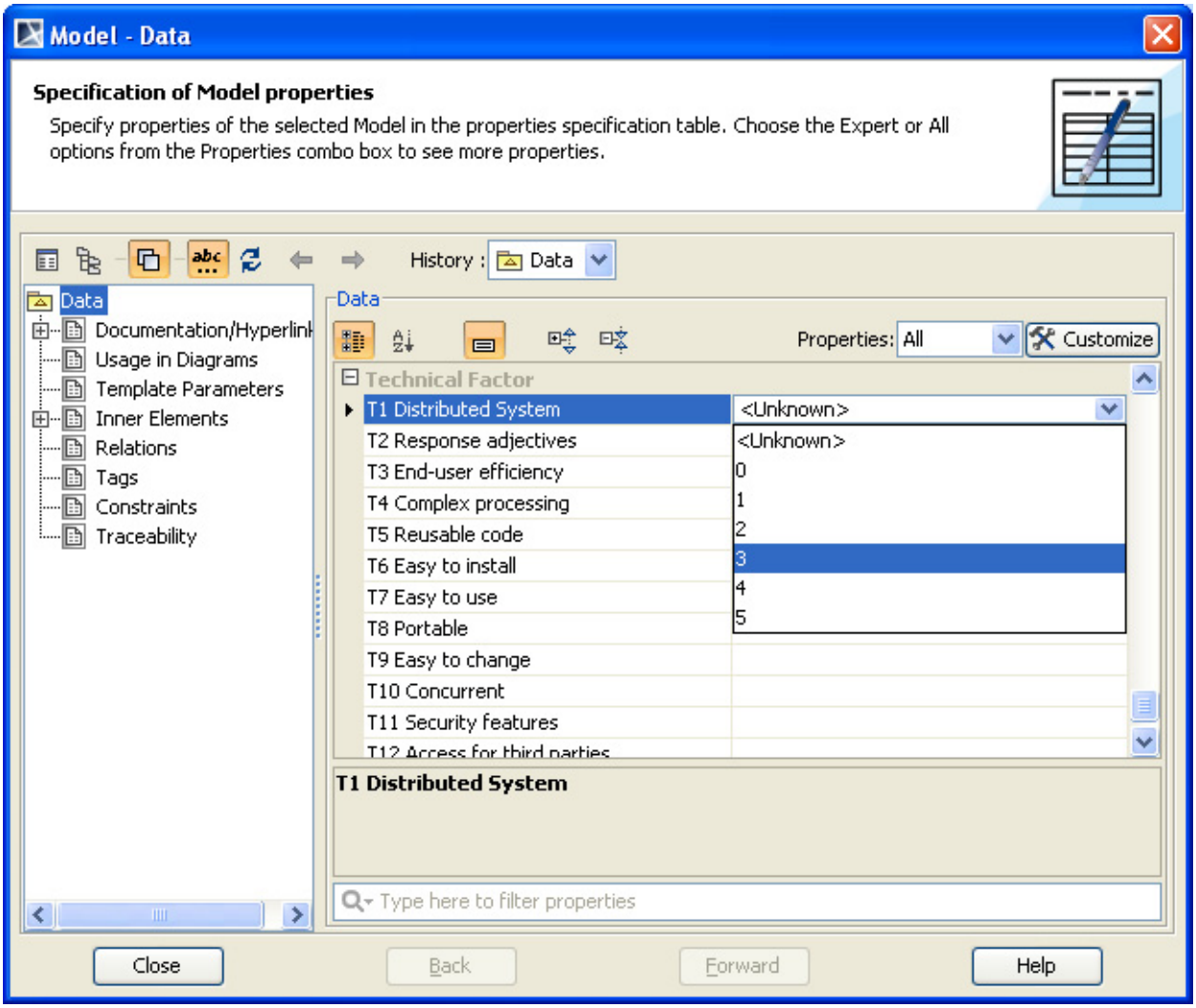


Figure 156 -- Use Case Driven Model Data T1

12.4.1.1 Technical Factor Value

Technical Factor Value (TFactor) is obtained with the multiplication of the value of each Technical factor by its weight.

TFactor = *value* x *weight*

12.4.1.2 Technical Complexity Factor

Technical Complexity Factor (TCF) is obtained with the addition of *0.6* to the sum of **TFactor** multiplied by *0.01*.

*TCF = 0.6 + (0.01 * TFactor)*

12.4.2 Environmental Factors

Table 28 -- Environmental Factors

Factor	Enviromental Factors	Weight	Description
E1	Familiar with RUP	1.5	Staff in the project are familiar with domain and technical details of the project.
E2	Application experience	0.5	The application experience level.
E3	Object-oriented experience	1.0	Staff in the project have basic knowledge of the OOP concept. The project is implemented on the object oriented design.
E4	Lead analyst capability	0.5	The analyst who is leading the project has enough domain knowledge.
E5	Motivation	1.0	The project motivates staff to work including how the software industry is going on
E6	Stable requirements	2.0	The requirements are clear, stable, and unlikely to change in the future.
E7	Part-time workers	-1.0	Part-time staff are working on the project.
E8	Difficult programming language	-1.0	Complexity of a programming language.

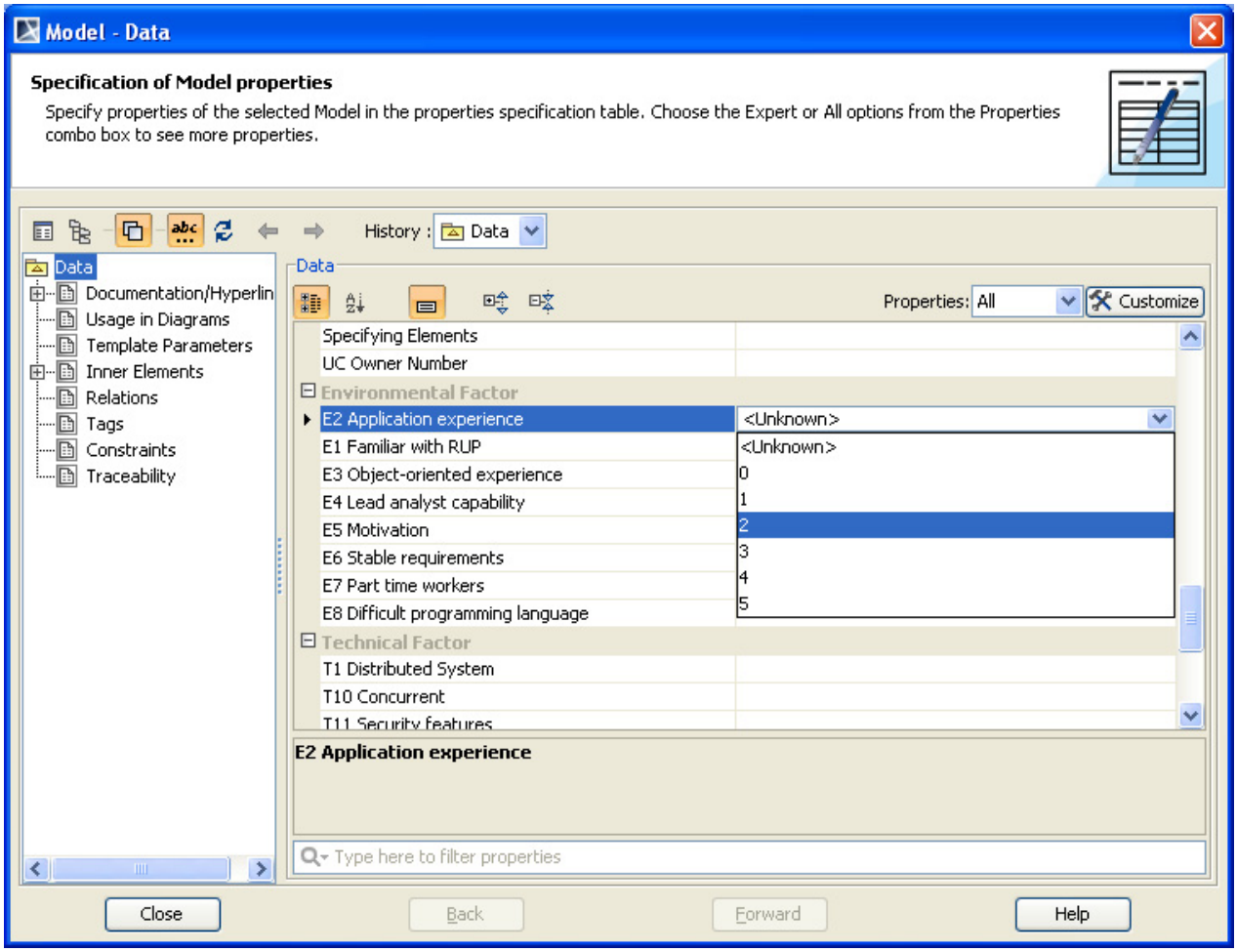


Figure 157 -- Use Case Driven Model Data E2

12.4.2.1 Environmental Factor Value

Environmental Factor Value (EFactor) is obtained with the multiplication of the value of each Environmental factor by its weight.

12.4.2.2 Environmental Factor

Environmental Factor (EF) is obtained with the addition of *1.4* to the sum of **EFactor** multiplied by *-0.03*.

*EF = 1.4 + (-0.03 * EFactor)*

12.4.3 Project Estimation

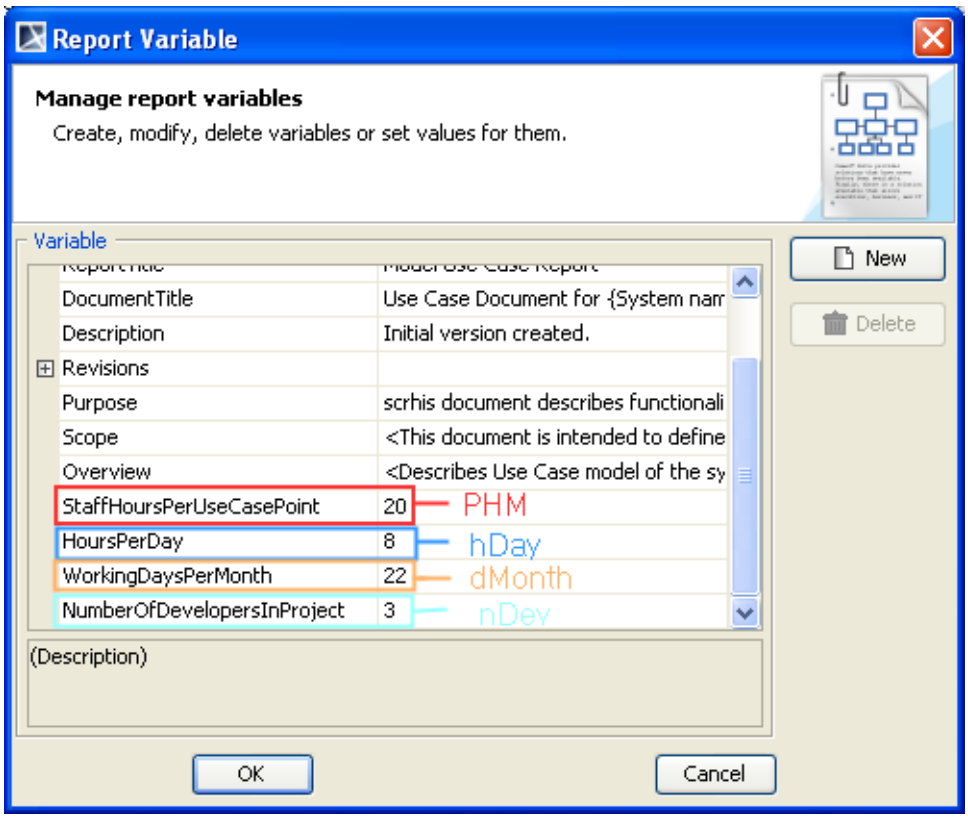


Figure 158 -- Variables of Use Case Project Estimation

12.4.3.1 Adjusted Use Case Points

The Adjusted Use Case Point (UCP) is determined with the multiplication of Unadjusted Use Case Point (UUCP) by Technical Complexity Factor (TCF) and Environmental Factor (EF).

*UCP = UUCP * TCF * EF*

12.4.3.2 Estimated Effort in Person Hours

The person hours multiplier or *X* hours is a ratio of the number of man hours (PHM) per Use Case Point based on past projects. If no historical data has been collected, industry experts suggest using a figure between 15 and 30. A typical value is 20.

X hours = *UCP * PHM*

12.4.3.3 Estimated Effort in Scheduled Time

Divide *X* hours by the number of developers working on the project and working hours per day to determine Estimated Effort in Scheduled Time or *Y* days. This means that with *n*Dev developers, it would take *Y* days to complete the project.

Y days = *X* / *n*Dev / *h*ay

12.4.3.4 Estimated Effort in Working Days

Divide *Y* days by working days per month to get the estimated effort in working days or *Z* months. This means that with *n*Dev developers, it would take *Z* month to complete the project.

Z months = *Y* / *d* Month

13. Javadoc Syntax Tool

Javadoc is the industry standard for documenting Java classes. Javadoc can be documented inside the documentation field (Figure 159).

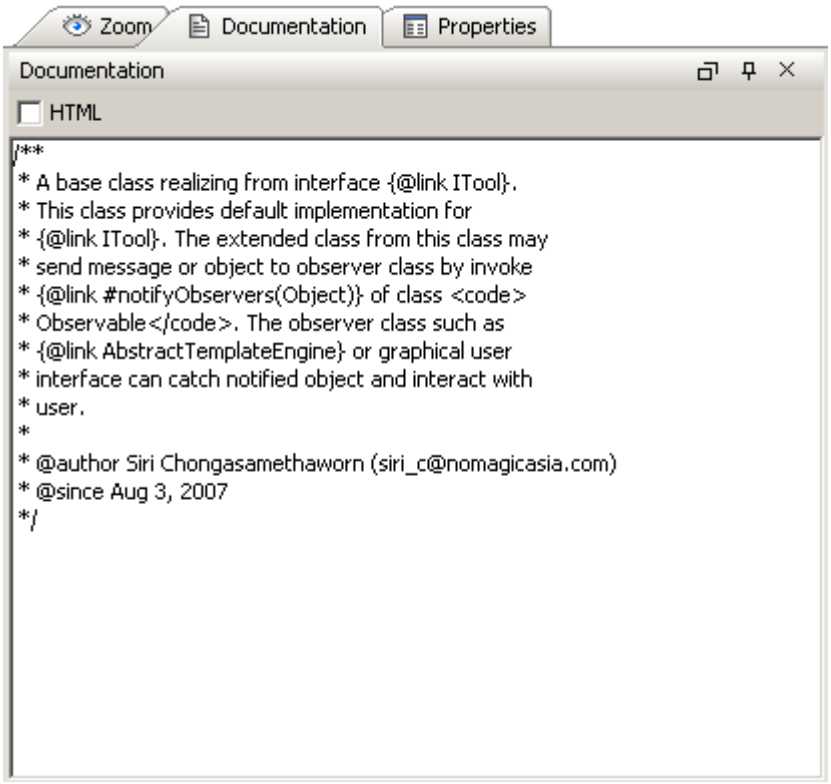


Figure 159 -- Documentation Field

Report Wizard provides a template tool for translating Javadoc text into Javadoc properties. Javadoc Tool will be loaded and used inside the template upon the user's request.

An example of Javadoc text is shown below:

```
/**
* A base class realizing from interface {@link ITool}.
* This class provides default implementation for
* {@link ITool}. The extended class from this class may
* send message or object to observer class by invoke
* {@link #notifyObservers(Object)} of class <code>
* Observable</code>. The observer class such as
* {@link AbstractTemplateEngine} or graphical user
* interface can catch notified object and interact with
* user.
*
* @author Siri Chongasamethaworn (siri_c@nomagicasia.com)
* @since Aug 3, 2007
*/
```

Print the Javadoc comment and author inside the template:

```
#import ("javadoc",
"com.nomagic.reportwizard.tools.doc.JavaDocTool")
#set ($doc = $javadoc.create($class.documentation))
Comment:
    $doc.comment
Author:
    $doc.author
```

The output from this example will be:

```
Comment:
        A base class realizing from interface ITool. This class
        provides default implementation for ITool. The extended
        class from this class may send message or object to
        observer class by invoke #notifyObservers(Object) of class
        Observable. The observer class such as
        AbstractTemplateEngine or graphical user interface can
        catch notified object and interact with user.
Author:
  Siri Chongasamethaworn (siri_c@nomagicasia.com)
```

The Javadoc Syntax Tool is implemented based on Custom Tool (See Appendix A: Report Extensions).

13.1 Javadoc Syntax

A doc comment consists of characters between the characters /** that begin the comment and the characters */ that end it. Leading asterisks are allowed on each line and are described below. Text in a comment can continue to multiple lines.

- **Leading asterisks**

  When Javadoc Tool parses a doc comment, leading asterisk (*) characters on each line are discarded.

- **First sentence**

  The first sentence of each doc comment is a summary sentence. This sentence can be retrieved from `$doc.firstSentenceTags`. The first sentence tags are a collection of inline tags until full stops `(.)`.

- **Comment and tag sections**

  A comment is the main description which begins after the starting delimiters /** and continues until the tag section. A tag section starts with the first block tag. The comment can be retrieved by $doc.comment

- **Comments written in HTML**

  Text is written in HTML and will be rendered as HTML support before being printed to a report. (See Appendix D: HTML Tag Support)

- **Block and in-line tags**

  A tag is a special keyword within a doc comment that Javadoc Tool can process. There are two kinds of tags:

  (i) Block tags

  They can be placed only in the tag section. The block tag form is @tag. The block tag can be accessed directly from the root document such as $doc.param and $doc.author.

  The $doc.tags will provide a collection of all the tags appearing in this Javadoc.

  (ii) Inline tags

  They can be placed anywhere in the main description or in the block tags. The inline tag form is {@tag}.

Table 29 -- Current Supported Tags

Tag	JDK	Report Wizard Support
@author	1.0	Yes
{@code}	1.5	Render as <code>text</code>
{@docRoot}	1.3	Not supported
@deprecated	1.0	Yes
@exception	1.0	Yes
{@inheritDoc}	1.4	Not Supported
{@link}	1.2	Yes, with conditions (External link, model support, and class/ method link will be ignored)
{@linkplain}	1.4	Yes, see {@link link}
{@literal}	1.5	Yes
@param	1.0	Yes
@return	1.0	Yes
@see	1.0	Yes, with conditions (External link, model support, and class/ method link will be return as plain text)
@serial	1.2	Yes
@serialData	1.2	Yes
@serialField	1.2	Yes
@since	1.1	Yes
@throws	1.2	Yes
{@value}	1.4	Not Supported
@version	1.0	Yes

13.2 Javadoc Tool API

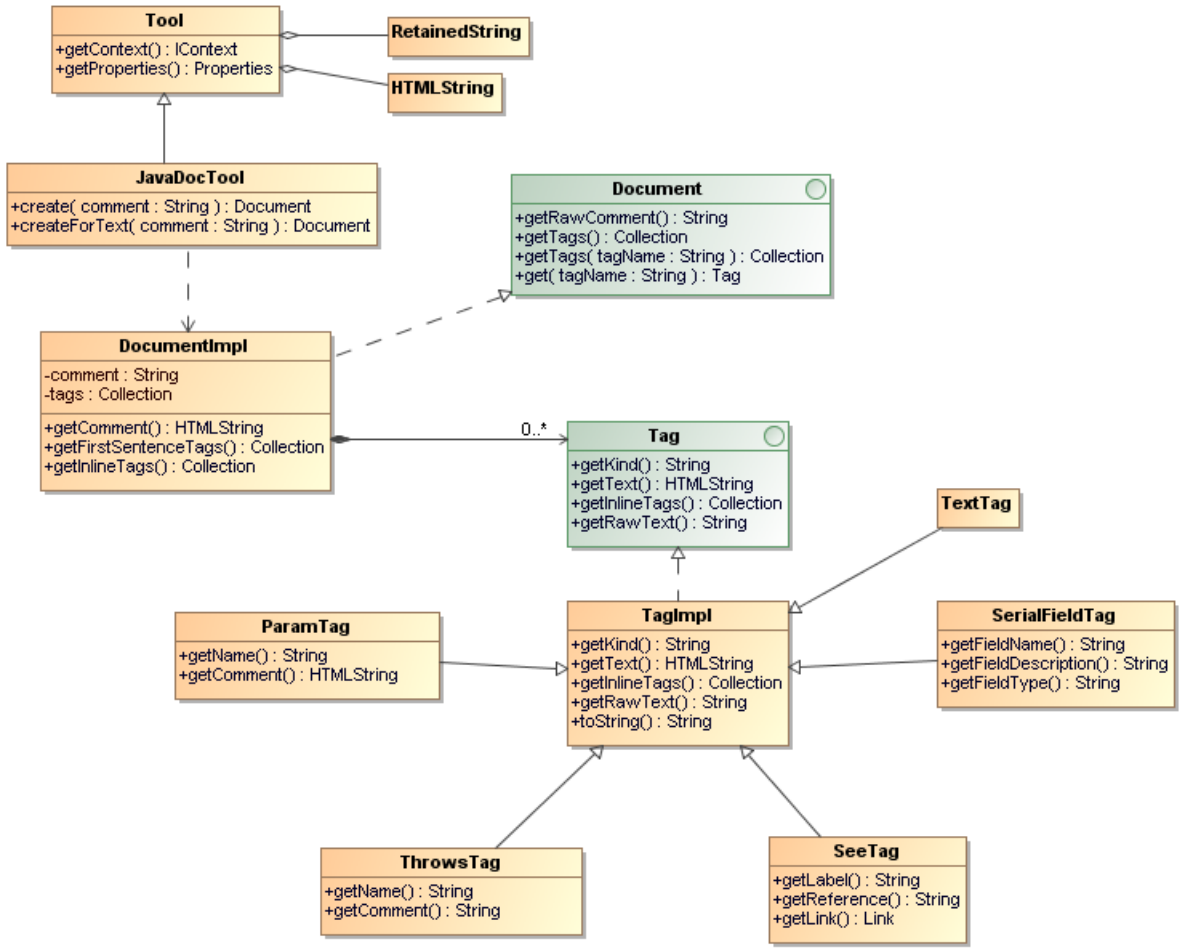


Figure 160 -- Javadoc Tool API

13.2.1 JavaDocTool

JavaDoc Tool is a custom tool class used for creating a Document node.

- create() - Returns a new instance of a Document node. Each JavaDoc comment must start with /** and ended with */.
- createForText() - Returns a new instance of a Document node. This method allows in-complete JavaDoc (comments without /** and */) to be loaded with JavaDocTool.

13.2.1.1 Document

Document is an interface representing a Comment Document.

- getRawComment() - Returns the full unprocessed text of the comment.
- getTags() - Returns all tags in this document item.
- getTags(tagName : String) - Returns a tag of the specified kind of this document item.
- get(tagName : String) - Returns a tag of the specified kind of this document item. If any tags with the same kind are found, the first tag of that kind will be returned.

13.2.1.2 DocumentImpl

DocumentImpl is a default JavaDoc document implementation.

- getComment() - Returns formatted text for this document.
- getInlineTags() - Returns comment as a collection of tags.
- getFirstSentenceTags() - Returns the first sentence of the comment as a collection of tags.

13.2.1.3 Tag

Tag is an interface representing a simple documentation tag, such as @since, @author, and @version

- getKind() - Returns the kind of this tag.
- getText() - Returns the text of this tag.
- getInlineTags() - Returns a collection of tags for a documentation comment with embedded {@link} tags.

13.2.1.4 TagImpl

TagImpl is a default tag implementation.

- getRawText() - Returns the full unprocessed text of this tag.
- toString() - Returns the text of this tag. Usually calls getText().

13.2.1.5 ParamTag

ParamTag represents a @param documentation tag.

- getName() - Returns the name of the parameter associated with this tag.
- getComment() - Returns the parameter's comment.

13.2.1.6 ThrowsTag

ThrowsTag represents a @throws or @exception documentation tag.

- getName() - Returns the name of the exception associated with this tag.
- getComment() - Returns the exception's comment.

13.2.1.7 SeeTag

SeeTag represents a user-defined cross-reference to related documentation. The reference can be either inline with the comment using {@link} or a separate block comment using @see.

- getLabel() - Returns the label of the @see tag.
- getReference() - Returns the MODEL or URL of the @see reference.
- getLink() - Returns the Link object representing this tag.

13.2.1.8 SerialFieldTag

SerialFieldTag represents a @serialField tag.

- getFieldName() - return the serial field name.
- getFieldDescription() - return the field comment.
- getFieldType() - return the field type string.

For examples:

- Import the Javadoc tool and create an instance of Javadoc document:

```
#import ("javadoc",
"com.nomagic.reportwizard.tools.doc.JavaDocTool")
#set ($doc = $javadoc.create($comment))
```

- Print a Javadoc comment:

```
$doc.comment
```

- Print the first encountered author:

```
$doc.author
```

- Print all authors:

```
#foreach ($author in $doc.getTags("author"))
$author.text
#end
```

- Print the first sentence:

```
#foreach ($tag in $doc.firstSentenceTags)
$tag.text
#end
```

- Print the inline tags:

```
#foreach ($tag in $doc.inlineTags)
$tag.kind
$tag.text
#end
```

- Print a raw comment (plain text without a document parser or HTML renderer):

```
$doc.rawComment
```

- Print all block tags:

```
#foreach ($tag in $doc.tags)
$tag.kind : $tag.text
#end
```

- Print all param tags:

```
#foreach ($tag in $doc.getTags("param"))
$tag.name - $tag.comment
#end
```

- Print all throws tags:

```
#foreach ($tag in $doc.getTags("throws"))
$tag.name - $tag.comment
#end
```

- Print all see tags:

```
#foreach ($tag in $doc.getTags("see"))
$tag.label - $tag.reference
#end
```

Reference Documents

- Javadoc specification

  http://java.sun.com/j2se/1.5.0/docs/tooldocs/windows/javadoc.html#javadoctags

- Writing Javadoc

http://java.sun.com/j2se/javadoc/writingdoccomments/index.html

- Javadoc Tool API

  <install.dir>/plugins/com.nomagic.magicdraw.reportwizard/api/javadoc.zip

14. Import Tool

Import Tool enables you to dynamically import documents or parts of them into reports, giving you greater flexibility when generating reports that require dynamic resources. You can now include documents whose location is only known at the actual translation time.

Import Tool allows you to:

- parse and import RTF, HTML, and text templates from any location in the file system;
- reuse the **#sectionBegin** and **#sectionEnd** syntax to import any part of a document. ImportTool makes use of the predefined syntaxes: **#sectionBegin** and **#sectionEnd**, which were introduced with the **#includeSection** directive. (See #includeSection Directive in the Report Wizard Custom Language section for **#sectionBegin** and **#sectionEnd** usage)

14.1 Import Syntax

To use the Import tool syntax:

1. To declare Import Tool in the template, type: **#import ('import', 'com.nomagic.reportwizard.tools.ImportTool')**. The directive **#import** will load Import Tool so that it can be accessed inside the template.
   - The first parameter **'import'** is for identifying the alias of Import Tool (in this case, **'import'**, but it can be named anything).
   - The second parameter **'com.nomagic.reportwizard.tools.ImportTool'** has to be copied as it is (do not rename it). Otherwise, Import Tool will not be loaded. For further details, see Ruby Script Tool.
2. To import any content from a child template, type:
   - **$import.include('child.rtf')** to import a complete document at the current position of the template, and/or
   - **$import.includeSection('child.rtf', 'Section A')** to include only the text contained in the named section of the document.

$import

This part names the alias assigned to Import Tool when declaring it to the template.

$import.include(FileName)

A 'FileName' specifies the location of a file in the file system. This filename can be either an absolute or a relative path. The 'FileName' parameter can be set either statically or dynamically. See the Import Usage section for an illustration on how to use it.

$import.includeSection(FileName, SectionName)

A ‘FileName' functions the same as in the sub-section above. A 'SectionName' denotes a paragraph in the document named 'FileName', which is delimited by the **#sectionBegin(SectionName)** and **#sectionEnd** identifiers. The SectionName variable can also be set either statically or dynamically. See the Import Usage section for an illustration on how to use it.

14.2 Import Usage

14.2.1 Preparatory Step

Define a child template that you will use in examples 1, 2, and 3. This child template contains the following header line and named sections ('Section A' and 'Section B') :

```
This is the child template.

#sectionBegin(Section A)
This is Section A.
#sectionEnd

#sectionBegin(Section B)
This is Section B.
#sectionEnd
```

14.2.2 Usage in Example 1

This example shows how to statically include the complete child template as shown in the Preparatory Step. To include it, call the **$import.include('child.rtf')** method. Note that there is no specified path, this path to the child template tells you that the master and child templates reside in the same directory.

```
#import ('import', 'com.nomagic.reportwizard.tools.ImportTool')

This is the 1st master template
Include child template
$import.include('child.rtf')
```

The output right below will include the entire text from the master template (right above) and from the child template (see the Preparatory Step) minus the template directives. The included content will be then parsed and stripped of all velocity directives. The output is as follows:

```
This is the 1st master template
Include child template
This is the child template.

This is Section A.

This is Section B.
```

14.2.3 Usage in Example 2

This example shows how to statically include a section of the child template as shown in the Preparatory Step. To include the section, named 'Section A' from the child template, call the **$import.includeSection('templates/child.rtf', 'Section A')** method. In this example, the master and child templates reside in different directories, which means that the child template will reside in a subdirectory called 'templates'. This 'templates' directory will thus reside in the same directory as the master template.

```
#import ('import', 'com.nomagic.reportwizard.tools.ImportTool')

This is the 2nd master template
Include Section A
$import.includeSection('templates/child.rtf', 'Section A')
```

The output will be:

```
This is the 2nd  master template
Include Section A
This is Section A.
```

14.2.4 Usage in Example 3

This example shows how to dynamically include the section of a child template as shown in the Preparatory Step. Before dynamically including the section of the child template named 'Section B', set a variable for the file location of the child template [**#set ($child = "C:/ImportTool/child.rtf")**] and another one for the section name to be included [**#set ($section = "Section B")**]. To include the 'Section B' section from the child template, call the **$import.includeSection($child, $section)** method. In this example, the child's template location is provided by an absolute path. Note that dynamic values must not be quoted, otherwise the section will not be included.

```
#import ('import', 'com.nomagic.reportwizard.tools.ImportTool')

This is the 3rd master template
Include Section B
#set ($child = "C:/ImportTool/child.rtf")
#set ($section = "Section B")
$import.includeSection($child, $section)
```

The output will be:

```
This is the 3rd master template
Include Section B
This is Section B.
```

NOTE	The Import Tool supports RTF, HTML, XML, and text files only.

15. JavaScript Tool

JavaScript Tool enables report templates to evaluate or run JavaScript codes from templates and external JavaScript files (Figure 161).

The general concept behind this JavaScript feature is to separate complex business logic from presentation logic. Complex logic should be executed by JavaScript, and presentation logic should be executed by Velocity code.

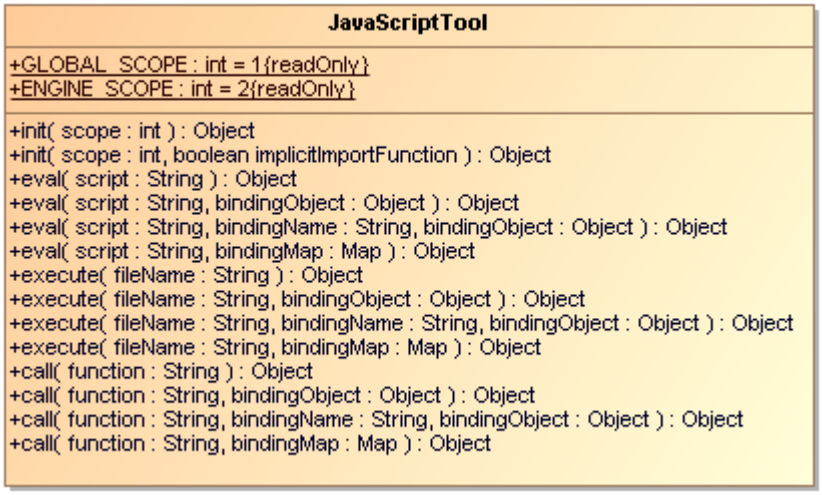


Figure 161 -- Class Diagram: JavaScript Tool

JavaScript Tool includes 3 methods:

(i) 'eval' Method: this method will evaluate JavaScript text, and then return the result.

(ii) 'execute' Method: this method will execute a JavaScript file, and then return the result.

(iii) 'call' Method: this method will call a JavaScript function, and then return the result.

Like any other Custom Tools, the JavaScript Tool 'scripttool.jar' must be installed in the 'extensions' folder of the Report Wizard plugin, and 'js.jar' must be included in any class path or the 'extensions' folder. For further information about Custom Tool and the installation, see **20.1 Custom Tool**.

To import JavaScript Tool to a template type, for example:

```
#import('js', 'com.nomagic.reportwizard.tools.script.JavaScriptTool')
```

15.1 JavaScript Tool API

15.1.1 'eval' Method

eval(String script)

This method will evaluate a JavaScript code from a string and return the result.

```
$js.eval('script')
```

eval(String script, Object bindingObject)

This method will evaluate a JavaScript code with a single binding object. The code will be evaluated from a string. The "importer" name will be used as a binding name for this object.

```
#foreach ($class in $Class)
   $js.eval('importer.getName()', $class)
#end
```

eval(String script, String bindingName, Object bindingObject)

This method will evaluate a JavaScript code with a single binding object and specified binding name. The code will be evaluated from a string. The binding name will be used as the name for this object.

```
#foreach ($class in $Class)
   $js.eval('cls.getName()', 'cls', $class)
#end
```

eval(String script, Map bindingMap)

This method will evaluate a JavaScript code with a set of binding arguments (a name and an object). The code will be evaluated from a string. The binding map consists of key-value pairs for this binding name and binding object.

```
#set ($dict = $map.createHashMap())
#set ($void = $dict.put("first", "foo"))
#set ($void = $dict.put("last", "bar"))
$js.eval("first + ' ' + last", $dict)
```

15.1.2 'execute' Method

execute(String filename)

This method will execute a JavaScript file. The filename parameter is a file path to the JavaScript file.

```
$js.execute('filename.js')
```

execute(String filename, Object bindingObject)

This method will execute a JavaScript file with a single binding object. The filename parameter is a file path to the JavaScript file. The "importer" name will be used as the binding name for this object.

```
#foreach ($class in $Class)
   $js.execute('filename.js', $class)
#end
```

```
// filename.js
importer.getName();
```

execute(String filename, String bindingName, Object bindingObject)

This method will execute a JavaScript file with a single binding object and specified binding name. The filename parameter is a file path to the JavaScript file. The binding name will be used as the name for this object.

```
#foreach ($class in $Class)
   $js.execute('filename.js', 'cls', $class)
#end
```

```
// filename.js
cls.getName();
```

execute(String filename, Map bindingMap)

This method will execute a JavaScript file with a set of binding arguments (a name and an object). The filename parameter is a file path to the JavaScript file. The binding map consists of key-value pairs for this binding name and binding object.

```
#set ($map = $map.createHashMap())
#set ($void = $map.put("first", "foo"))
#set ($void = $map.put("last", "bar"))
$js.execute('filename.js', $map)
```

```
// filename.js
first + ' ' + last;
```

NOTE	• *Absolute Path:* If a 'filename' is provided with an absolute path, JavaScript Tool will read the JavaScript file from an absolute location such as $js.execute('c:/mycode/readclass.js'). • *Relative Path*: If a 'filename' is provided with a relative path, JavaScript Tool will read the template from a relative location. This relative location starts from the current directory in which the template is located such as $js.execute('readclass.js').

15.1.3 'call' Method

call(String function)

This method will provide a short and reusable method to call a JavaScript function. This function must be defined before calling this method. This function can be defined by 'eval' or 'execute'.

```
$js.execute('javascript.js')
$js.call('calc(1, 3)')
```

```
// javascript.js
function calc(var1, var2)
{
   return var1 + var2;
}
```

call(String function, Object bindingObject)

This method will provide a short and reusable method to call a JavaScript function with a single binding object. The "importer" name will be used as the binding name for this object. This function must be defined before calling this method. This function can be defined by 'eval' or 'execute'.

```
$js.execute('javascript.js')
$js.call('calc(1, 3)', 10)
```

```
// javascript.js
function calc(var1, var2)
{
   var f = Number(importer ? importer : 0);
   return f + var1 + var2;
}
```

call(String function, String bindingName, Object bindingObject)

This method will provide a short and reusable method to call a JavaScript function with a single binding object and specified binding name. This binding name will be used as the name for this object. This function must be defined before calling this method. This function can be defined by 'eval' or 'execute'.

```
$js.execute('javascript.js')
$js.call('calc(1, 3)', 'factor', 10)
```

```
// javascript.js
function calc(var1, var2)
{
   var f = Number(factor ? factor : 0);
   return f + var1 + var2;
}
```

call(String function, Map bindingMap)

This method will provide a short and reusable method to call a JavaScript function with a set of binding arguments (a name and an object). The binding map consists of key-value pairs for this binding name and binding object.

```
#set ($map = $map.createHashMap())
#set ($void = $map.put("first", "foo"))
#set ($void = $map.put("last", "bar"))
$js.call('hello()', $map)
```

```
// filename.js
function hello()
{
   return 'hello ' + first + ' ' + last;
}
```

15.2 References to Elements

References to elements are implicitly inserted into the JavaScript context when calling **eval()**, **execute()**, or **call()**. Examples of implicit variables include, for instance, **$Class**, **$UseCase**, **$sorter**, etc.

Example :

A "functions.js"

```
importPackage(java.util)

// variable $Dependency and $sorter can be accessed directly inside
function getSupplier()
{
   var supplierList = new ArrayList();
     var sortedDependencyList = $sorter.sort($Dependency);
   for (var i=0; i<sortedDependencyList.size(); i++)
   {
      var dependency = sortedDependencyList.get(i);
      supplierList.addAll(dependency.getSupplier());
   }
   return supplierList;
}
```

A template code

```
#import ('js', 'com.nomagic.reportwizard.tools.script.JavaScriptTool')

$js.execute("functions.js")
#set ($supplierList = $js.eval("getSupplier()"))
#foreach ($supplier in $supplierList)
  $supplier.name
#end
```

Example:

A "functions.js"

```
var qualifiedName = "";
function packageQualifiedName(element)
{
   qualifiedName = "";
     packageName(element);
     return qualifiedName==""?"Defualt":qualifiedName;
}

function packageName(element)
{
     var parent = element.owner;
   if (parent.classType.simpleName == 'Package') {
        if (qualifiedName == "")
              qualifiedName = parent.name;
           else
           qualifiedName = parent.name + '.' + qualifiedName;
           packageName(parent);
     }
}
```

A template code

```
#import ('js', 'com.nomagic.reportwizard.tools.script.JavaScriptTool')

$js.execute("functions.js")
#foreach ($c in $Class)
   Package: $js.eval('packageQualifiedName($c)')
     Name: $c.name
#end
```

Reference Documents

- Mozilla JavaScript Tool

  http://developer.mozilla.org/en/Rhino_documentation
- Sun JavaScript programming guide

  http://java.sun.com/javase/6/docs/technotes/guides/scripting/programmer_guide/index.html

16. Groovy Script Tool

Groovy Script Tool enables report templates to evaluate or run Groovy codes from templates and external Groovy files.

The Groovy Tool includes two methods:

(i) 'eval' method: This method will evaluate Groovy text, and then return the result.

(ii) 'execute' method: This method will execute a Groovy file, and then return the result.

Like any other Script Tools, the Groovy Script Tool 'scripttool.jar' must be installed in the 'extensions' folder of the Report Wizard plugin, and 'groovy-all-1.7.9.jar' must be included in any class path or the 'extensions' folder. For further information about Custom Tool and the installation, see **20.1 Custom Tool**.

To import Groovy Script Tool to a template, type the following code:

```
#import ('groovy', 'com.nomagic.reportwizard.tools.script.GroovyTool')
```

16.1 Groovy Script Tool API

16.1.1 'eval' Method

eval(String script)

This method will evaluate a Groovy code from a string and return the result.

```
$groovy.eval("println 'Hello World!'")
```

eval(String script, String bindingName, Object bindingObject)

This method will evaluate a Groovy code with a single binding object and specified binding name. The code will be evaluated from a string. The binding name will be used as the name for this object.

```
#foreach ($c in $Class)
 $groovy.eval("println classname", "classname", $c.name)
#end
```

eval(String script, Map bindingMap)

This method will evaluate a Groovy code with a set of binding arguments (a name and an object). The code will be evaluated from a string. The binding map consists of key-value pairs for the binding name and the binding object.

```
#set ($dict = $map.createHashMap())
#set ($void = $dict.put("first", "foo"))
#set ($void = $dict.put("last", "bar"))
$groovy.eval("println first + last", $dict)
```

Or

```
$groovy.eval("println first + ' '+ last", {"first":"foo", "last":"bar"})
```

NOTE	The second code contains curly brackets; '{' and '}' characters, which are not allowed to be used in any RTF template. For the RTF template, use the first code instead.

16.1.2 'execute' method

execute(String filename)

This method will execute a Groovy file. The 'filename' parameter refers to a name of the Groovy file or an absolute path to the Groovy file.

```
$groovy.execute("filename.groovy")
```

execute(String filename, String bindingName, Object bindingObject)

This method will execute a Groovy file with a single binding object and specified binding name. The 'filename' parameter is a file path to the Groovy file. The binding name will be used as the name for this object.

File filename.groovy

```
"Class name is $c.name"
```

The template code

```
#foreach ($c in $Class)
  $groovy.execute("filename.groovy", 'c', $c)
#end
```

execute(String filename, Map bindingMap)

This method will execute a Groovy file with a set of binding arguments (a name and an object). The 'filename' parameter is a file path to the Groovy file. The binding map consists of key-value pairs for the binding name and the binding object.

File filename.groovy

```
first + " " + last
```

The template code

```
#set ($dict = $map.createHashMap())
#set ($void = $dict.put("first", "foo"))
#set ($void = $dict.put("last", "bar"))
$groovy.execute("filename.groovy", $dict)
```

NOTE	• *Absolute Path*: If the 'filename' is provided with an absolute path, Groovy Tool will read the Groovy file from an absolute location such as $groovy.execute('c:/mycode/readclass.groovy'). • *Relative Path*: If the 'filename' is provided with a relative path, Groovy Tool will read the template from a relative location. This relative location starts from the current directory in which the template is located such as $groovy.execute('readclass.groovy').

16.2 References to Elements

References to elements are similar to ones in JavaScript Tool. The elements are implicitly inserted into the Groovy context when calling “eval()”, or “execute()”. Examples of implicit variables include $Class, $UseCase, $sorter, etc.

File 'AllAbstractClass.groovy'

```
// variable $Class can be accessed directly inside Groovy script

def list = []
for (c in $Class) {
    if (c.isAbstract()) {
        list.add(c)
    }
}
return list
```

The report template code is:

```
#import ('groovy', 'com.nomagic.reportwizard.tools.script.GroovyTool')
#set ($abstractClassList = $groovy.execute('AllAbstractClass.groovy'))
#foreach ($cls in $abstractClassList)
  $cls.name
#end
```

17. Ruby Script Tool

Ruby Script Tool enables report templates to evaluate or run Ruby codes from templates and external Ruby files.

The Ruby Tool includes 2 methods:

(i) 'eval' method: This method will evaluate Ruby text, and then return the result.

(ii) 'execute' method: This method will execute a Ruby file, and then return the result.

Like any other Script Tools, the Ruby Script Tool 'scripttool.jar' must be installed in the 'extensions' folder of the Report Wizard plugin, and 'jruby-complete-1.6.3.jar' must be included in any class path or the 'extensions' folder. For further information about Custom Tool and the installation, see **20.1 Custom Tool**.

To import Ruby Script Tool to a template, type the following code:

```
#import ('ruby',
'com.nomagic.reportwizard.tools.script.RubyScriptTool')
```

17.1 Ruby Script Tool API

17.1.1 'eval' Method

17.1.1.1 eval(String script)

This method will evaluate a Ruby code from a string and return the result.

```
$ruby.eval("puts 'Hello World!'")
```

17.1.1.2 eval(String script, String bindingName, Object bindingObject)

This method will evaluate a Ruby code with a single binding object and specified binding name. The code will be evaluated from a string. The binding name will be used as the name for this object.

```
#foreach ($class in $Class)
  $ruby.eval('"Class name is " + c.name', 'c', $class)
#end
```

17.1.1.3 eval(String script, Map bindingMap)

This method will evaluate a Ruby code with a set of binding arguments (a name and an object). The code will be evaluated from a string. The binding map consists of key-value pairs for the binding name and binding object.

```
#set ($dict = $map.createHashMap())
#set ($void = $dict.put("first", "foo"))
#set ($void = $dict.put("last", "bar"))
$ruby.eval("puts first + last", $dict)
```

Another alternative is as follows:

```
$ruby.eval("puts first + last", {"first":"foo", "last":"bar"})
```

The second code contains curly brackets; "{" and "}" characters, which are not allowed to be used in any RTF template. For the RTF template, use the first code instead.

17.1.2 'execute' Method

17.1.2.1 execute(String filename)

This method will execute a Ruby file. The 'filename' parameter is a name of the Ruby file or an absolute path to the Ruby file.

```
$ruby.execute("filename.rb")
```

After executing the Ruby file, the result of the execution will be stored in the single context for each report generation. If the Ruby file contains Ruby functions, you can recall the functions with the use of 'eval()' methods.

For example, File 'String.rb':

```
def deCamelCase(str)
  return str.gsub!(/(.)([A-Z])/,'\1 \2')
end
```

Indicated below is the template code.

```
$ruby.execute("String.rb")
#foreach ($c in $Class)
  $ruby.eval('deCamelCase($c.name)')
#end
```

17.1.2.2 execute(String filename, String bindingName, Object bindingObject)

This method will execute a Ruby file with a single binding object and specified binding name. The 'filename' parameter is a file path to the Ruby file. The binding name will be used as the name for this object.

File 'filename.rb'

```
puts c.name
```

The template code

```
#foreach ($c in $Class)
  $ruby.execute("filename.rb", 'c', $c)
#end
```

17.1.2.3 execute(String filename, Map bindingMap)

This method will execute a Ruby file with a set of binding arguments (a name and an object). The 'filename' parameter is a file path to the Ruby file. The binding map consists of key-value pairs for the binding name and binding object.

File 'filename.rb'

```
puts first+' '+last
```

The template code

```
#set ($dict = $map.createHashMap())
#set ($void = $dict.put("first", "foo"))
#set ($void = $dict.put("last", "bar"))
$ruby.execute("filename.rb", $dict)
```

NOTE	• *Absolute Path*: If the 'filename' is provided with an absolute path, Ruby Tool will read the Ruby file from an absolute location such as $ruby.execute('c:/mycode/readclass.rb'). • *Relative Path*: If the 'filename' is provided with a relative path, Ruby Tool will read the template from a relative location. This relative location starts from the current directory at which the template is located such as $ruby.execute('readclass.rb').

17.2 References to Elements

References to elements are similar to JavaScript Tool and Groovy Tool. The elements are implicitly inserted into the Ruby context when "eval()" or "execute()" is called. Examples of implicit variables include $Class, $UseCase, $sorter, etc.

For example, File 'Functions.rb':

```
def supplierList
   result = [];
   $Dependency.each do |item|
      item.supplier.each do |supplier|
         result << supplier;
      end
   end
   return result;
end
```

Indicated below is the template code.

```
#import ('ruby',
'com.nomagic.reportwizard.tools.script.RubyScriptTool')
$ruby.execute('Functions.rb')
#set ($supplierList = $ruby.eval('supplierList'))
#foreach ($e in $supplierList)
  $e.name
#end
```

18. Dialog Tool

The purpose of Dialog Tool is to enable report templates to call functions for creating dialogs to interact with users during the report generation process. To modify the report templates, you can use **Dialog Tool**.

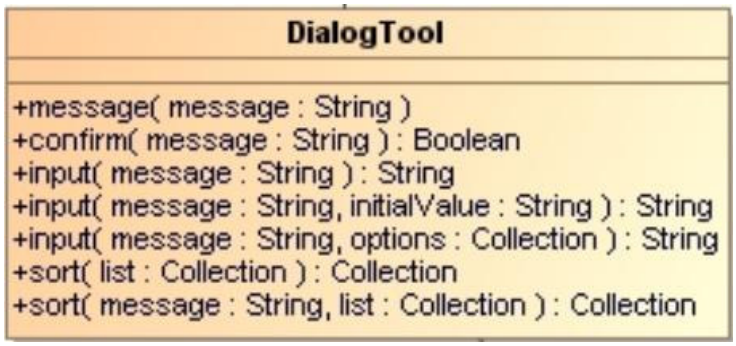


Figure 162 -- Class Diagram: Dialog Tool

Dialog Tool uses 4 methods:

(i) '**message**': This method will create message dialogs.

(ii) '**confirm**': This method will create confirmation dialogs and return the Boolean value 'true' when you click the **OK** button or return 'false' when you click the **Cancel** button.

(iii) '**input**': This method will create input dialogs and return the input value as a string.

(iv) '**sort**': This method will create Sort and Enable dialogs and return a collection of new sorted results.

Like any other Custom Tools, Dialog Tool (dialogtool.jar) must be installed in the 'extensions' folder of the Report Wizard plugin.

To import Dialog Tool to a template type, for example:

```
#import('dialog','com.nomagic.reportwizard.tools.DialogTool')
```

18.1 Dialog Tool API

18.1.1 'message' Method

`message(String message) : void`. This method will create messages into message dialogs. For example:

```
$dialog.message("Click OK to close dialog")
```

The output will be:

Figure 163 -- Message Dialog

18.1.2 'confirm' Method

`confirm(String message) : boolean`. This method will create messages into confirmation dialogs and return the Boolean value 'true' when you click the **OK** button or 'false' when you click the **Cancel** button in the dialog. For example:

```
#if($dialog.confirm("Do you want to geanerate section A?"))
#end
```

The output will be:

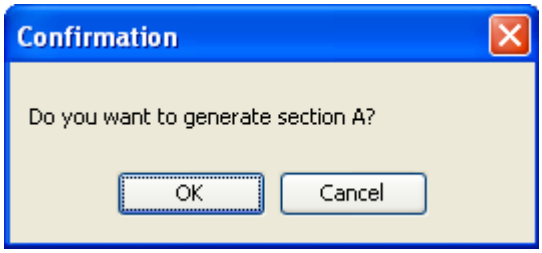


Figure 164 -- Confirmation Dialog

18.1.3 'input' Method

18.1.3.1 Input Dialogs with Text

`input(String message) : String`. This method will create input dialogs from messages, and then return the input values as strings when you click the **OK** button or 'null' when you click the **Cancel** button in the dialog. For example:

```
#set ($userText = $dialog.input("Enter your name"))
#if ($userText)
Your name is $userText
#else
Please specify your name
#end
```

The output will be:

Figure 165 -- Input Dialog with Text

18.1.3.2 Input Dialogs with Text and Initial Value

`input(String message, String initialValue) : String`. This method will create messages and initial values into input dialogs and return the input values as strings when you click the **OK** button or 'null' when you click the **Cancel** button in the dialog. For example:

```
#set ($userText = $dialog.input("Enter a date",
"July10,2009"))
Date is $userText
```

The output will be:

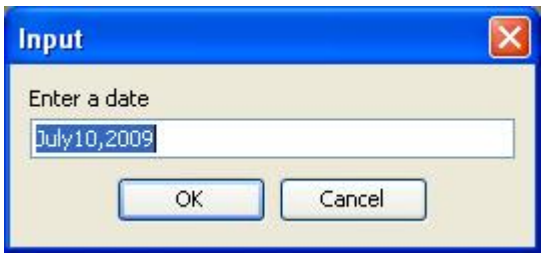


Figure 166 -- The Input Dialog with Text and an Initial Value

18.1.3.3 Input Dialog with Text and Initial Value Array

`input(String message, Collection options) : String`. This method will create messages and initial value arrays into input dialogs and return the input values as strings when you click the **OK** button or 'null' when you click the **Cancel** button in the dialog. For example:

```
#set ($selectedOption = $dialog.input("Choose your favorite
fruit", ["Apple", "Orange", "Banana"]))
Your favorite fruit is $selectedOption
```

The output will be:

Figure 167 -- Input Dialog with Text and Initial Value Array

18.1.4 'sort' Method

18.1.4.1 Sort and Enable Dialogs

`sort(Collection list) : Collection`. This method will create Sort and Enable dialogs from collections and return a collection of the newly-sorted results when you click the **OK** button or return the original collection when you click the **Cancel** button in the dialog. For example:

```
#foreach ($diagram in $dialog.sort($Diagram))
$diagram.name
#end
```

The output will be:

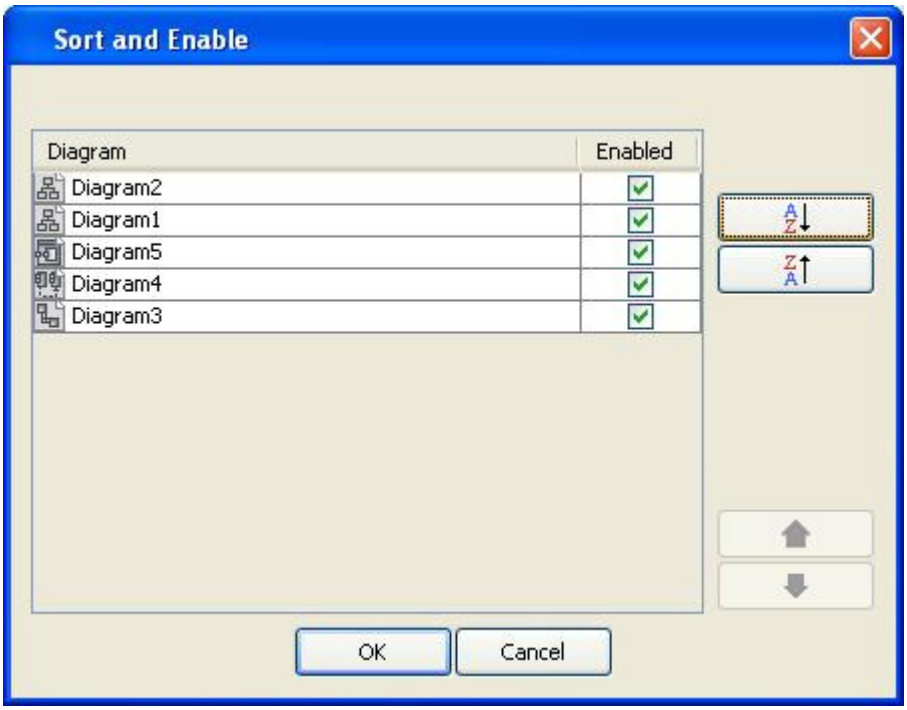


Figure 168 -- Sort and Enable Dialog

18.1.4.2 Sort and Enable Dialogs with Text

`sort(String message, Collection list)`. This method will create the Sort and Enable dialogs from text and collections and return a collection of the new sorted results when you click the **OK** button or return the original collection when you click the **Cancel** button in the dialog. For example:

```
#foreach ($diagram in $dialog.sort("Rearrange diagram for pre-
sentation report",$Diagram))
$diagram.name
#end
```

The output will be:

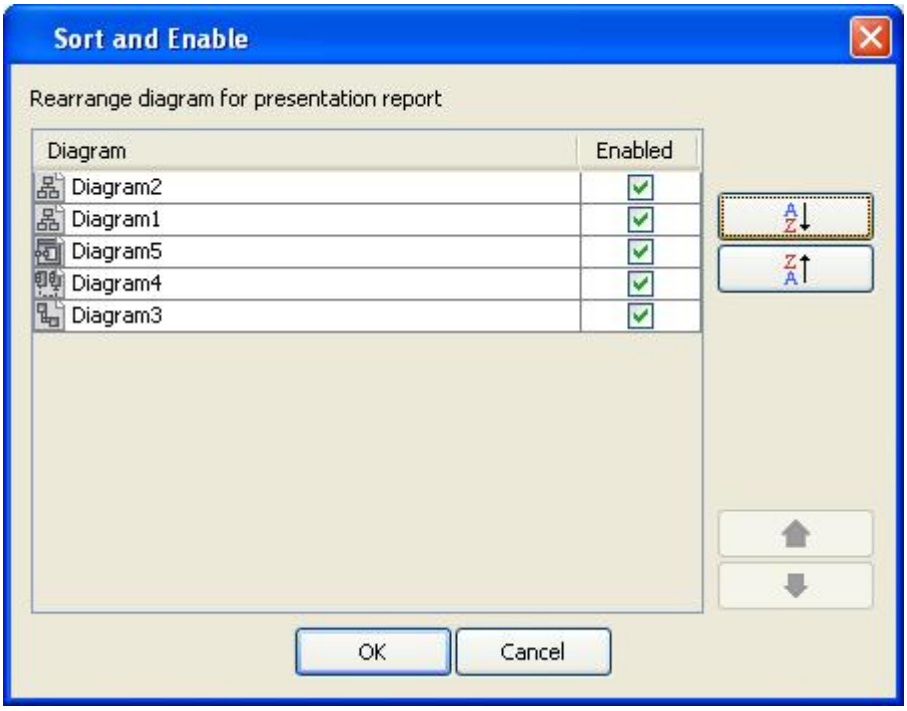


Figure 169 -- Sort and Enable Dialog with Text

19. Text Tool

Text tool provides extra functions to convert or format text.

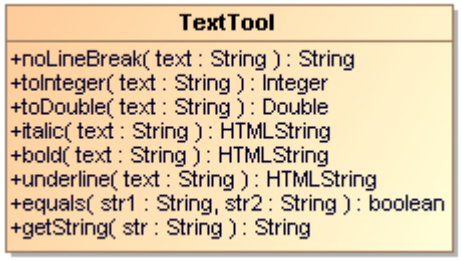


Figure 170 -- Class Diagram: Text Tool

Like any other Custom Tools, Text Tool (text.jar) must be installed in the 'extensions' folder of the Report Wizard plugin. For further information on Custom Tool and installation, see 20.1 Custom Tool.

To import Text Tool to a template type, for example:

```
#import('text', 'com.nomagic.reportwizard.tools.TextTool')
```

19.1 Text Tool API

noLineBreak(String text)

This method will remove all control characters such as U+0009 (Tab), U+000A (Line Feed), and U+000D (Carriage Return) from a given text. This method will also filter text under the category "Cc" in the Unicode specification. An example of a use case that contains multiple lines has the code as shown below.

```
$usecase.name
```

The output will be:

```
Line 1
Line 2
Line 3
```

With the **$text.noLineBreak()** method, the code will be:

```
$text.noLineBreak($usecase.name)
```

The output will be:

```
Line 1 Line 2 Line 3
```

toInteger(String text)

Converts a string argument to a signed decimal integer. For example:

```
#set ($int = $text.toInteger('2'))
#set ($result = $int + 1)
Result is $result
```

toDouble(String text)

Converts a string argument to a signed decimal double. For example:

```
#set ($d = $text.toDouble('10.1'))
#set ($result = $d + 2)
Result is $result
```

italic(String text)

Forces a report to render a given text in the italic style. For example:

```
The $text.italic($class.name) must provide interface for web
services.
```

The output will be:

```
The BusinessServices must provide interface for web services.
```

bold(String text)

Forces a report to render a given text in the bold style. For example:

```
Method $text.bold($operator.name) must be implemented by
implementation class.
```

The output will be:

```
Method doService must be implemented by implementation class.
```

underline(String text)

Forces a report to render a given text in the underlined style. For example:

```
Attribute $text.underline($attribute.name) contains a service
name.
```

The output will be:

```
Attribute serverName contains a service name.
```

html(String text)

Forces a report to render a given text as HTML. For example:

```
$text.html('<ul><li>A</li><li>B</li></ul>')
```

The output will be:

```
·A
·B
```

For more information on supported HTML tags, see Appendix D: HTML Tag Support.

getString(String text)

Converts text from RTF to a Java String. For example:

```
#set ($str = $text.getString("Übersetzer"))
$str
```

The output will be:

```
Übersetzer
```

equals(String str1, String str2)

Compares two strings. The result will be true if and only if **str1** represents the same sequence of characters as **str2** does. This method supports RTF text comparison. For example:

```
#set ($str = $text.getString("Übersetzer"))
#if ($text.equals($str, "Übersetzer"))
   true
#end
```

20. Appendix A: Report Extensions

Report Wizard has been enhanced with an extension that provides additional functionalities (for example, Custom Tool that encapsulates Java functions in context). The report extensions are Java class and related resources collected into JAR archives, which are merely ZIP files that contain a specially formatted “manifest” file describing the contents of the archive.

To install the extension, copy the `.jar` file into the extensions directory under the Report Wizard plugin directory or template directory. The archives stored within subdirectories will not be loaded in Report Wizard (Figure 171).

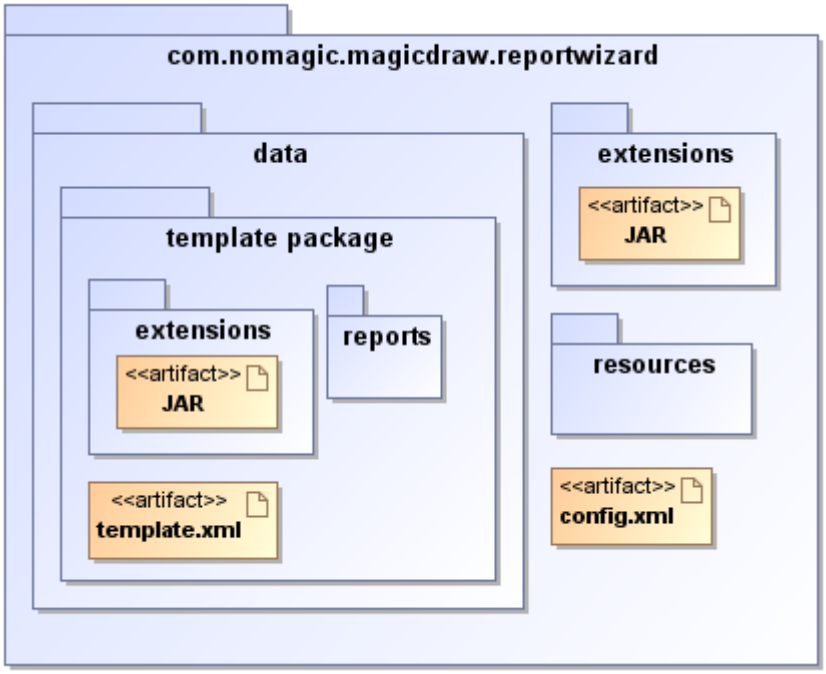


Figure 171 -- Report Wizard Plug-in Directory Structure.

The extension deployed under the global extensions directory will affect all templates, which means you can use the deployed Custom Tool in all templates. Extensions deployed under a template package will affect only the specific template.

20.1 Custom Tool

Custom Tool is a template library written in Java to encapsulate functions in context. Custom Tool (Figure 172) is extended from the report API. The tool can be used to develop custom context properties that are presentation centric or that can take advantage of the existing ‘Tool’ extensions or by document authors familiar with Java.

Report Wizard uses the word 'Tool' meaning an object encapsulating dynamic functions on the JavaBean. A Tool is a "carrier" of data between the Java and template layers. In other words, the Tool simplifies template development and maintenance.

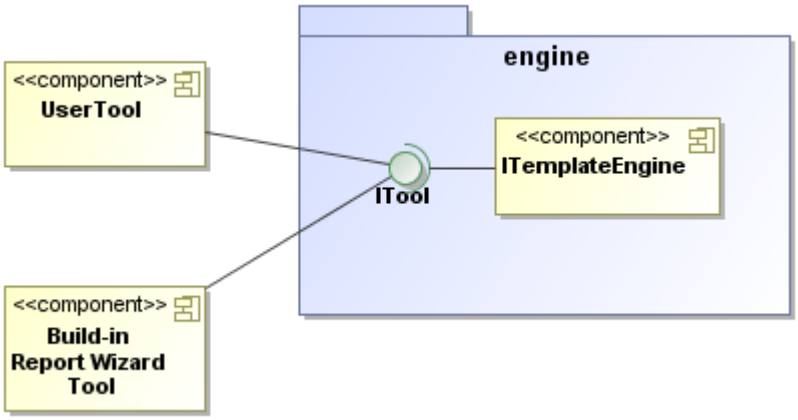


Figure 172 -- Custom Tool API Structure.

20.1.1 Context Name

A Context name is a name for variables or references in the template. Report Wizard uses the Velocity Template Language (VTL) as a standard syntax for the template. VTL uses "`$`" as a leading character followed by an identifier for the variable.

For example:

```
$UseCase
$foo.name
```

20.1.2 Context Object

A Context object is a Java Object representing a variable. A Context object is modeled on the JavaBean specifications defined by Sun Microsystems.

For example, if a String represents the variable `$foo`.name in context, the value of the String will be printed out in the output report as follows.

```
Foo Name
```

20.2 Tool Interface

Custom Tool is written in the Java language. The tool implements a specialized interface called `ITool`. The Report API provides both an interface and a class to support interface realization and class generalization. As mentioned earlier, a Tool is modeled on the JavaBean specifications. Functions implemented in this class must be defined in a series of setter or getter methods. The following sample shows the source code for the `ITool` interface that you must implement:

```
public interface ITool extends IBean
{
   // attributes
   Void VOID = new Void();
   // inner classes
   public class Void{}
   public class RetainedString implements CharSequence{}
   // methods
   void setContext(IContext context);
   String getContext();
   void setProperties(Properties properties);
   Properties getProperties();
}
```

All tools must implement the `ITool` interface (or one of its subclasses) as it defines the methods the template engine calls to execute. Table 30 provides a description of the methods in the `ITool` interface.

Table 30 -- Description of Methods in ITool interface.

Method	Description
setContext(Context context)	Once the class has been initialized, this method is called by the template engine runtime to set the template context. Overriding the current context will affect the code after this tool.
getContext()	Return the template context that is assigned to the current runtime.
setProperties(Properties properties)	This method is called by the template engine runtime, once the class has been initialized, to set the template properties. Overriding the current properties will not affect the engine.
getProperties()	Return the current template properties.
VOID	Represents the Void class. VOID is used to make sure that returning data from the setter method is absolutely nothing. In general, Velocity considers the return of `void` from the setter method as a 'null' value. This causes the word 'null' to be printed out in the report. To avoid this problem, the setter method may use VOID as a return object.
class Void	A void class is used as a return in Tool when you want to be sure that nothing is returned to context.
class RetainedString	A direct command report formatter to keep the referenced String format. The formatter and other reference insertion handlers should maintain Strings directed by this class.

The other classes which can be extended are `Tool` and `ConcurrentTool`.

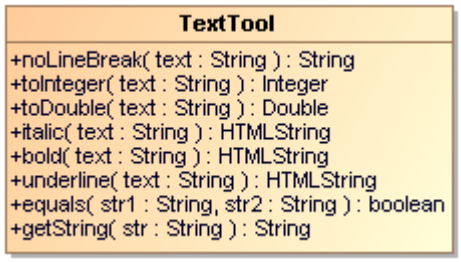


Figure 173 -- ITool Interface And Related Class.

20.2.1 Class Tool

Class Tool is a base class realizing the interface `ITool`. This class provides the default implementation for `ITool`.

This class also provides methods derived from the **java.util.Observable** class, which can be used to notify the observers. An observer class, such as a template engine or a graphical user interface class, can receive the notification message from the tool and manage to display or interact with the user.

20.2.2 Concurrent Tool

Concurrent Tool provides concurrent tasks running in the template engine. This class implements an unbounded thread-safe queue, which arranges the elements in a first-in-first-out (FIFO) order. New tasks are inserted at the tail of the queue, and the queue retrieval operations obtain elements at the head of the queue.

The tool extends from this class and is ideal for processing a long task that does not want other tasks to wait until the process is complete. The following code shows the sample usage of Concurrent Tool.

```
import com.nomagic.magicreport.engine.ConcurrentTool;

public class LongTaskTool extends ConcurrentTool
{
   public String longTask() {
      // enqueue object
      offer(new ConsumeObject(referent));
   }

   public void consume(ConsumeObject consumeObject) {
      Object referent = consumeObject.get();
      // process long task
   }
}
```

An example of Concurrent Tool is `FileTool`. The root template that calls the **file tool** method will create a subprocess, offer the subprocess into a queue, and continue the root template until finished. The template engine will later call the **consume** method to complete the subprocess once the consumer queue is available. The number of concurrent processes available for the execution is declared in the template engine property (See **TemplateConstants.TEMPLATE_POOL_SIZE** for detail.).

20.3 Creating Custom Tool

There are a few steps involved in developing a custom tool. These steps can be summarized as follows:

(20.3.1) Developing a Tool Class

(20.3.2) Creating an Extension Package

20.3.1 Developing a Tool Class

Developing a Tool class requires setting a class path to **magicreport.jar**. You can find **magicreport.jar** from the library folder under the Report Wizard plugin directory. The following sample shows the source code for `Hello.java`.

```
package mytool;
import com.nomagic.magicreport.engine.Tool;

public class HelloTool extends Tool
{
   public String getHello() {
      return "Hello World";
   }
   public String getHello(String name) {
      return "Hello " + name;
   }
}
```

The sample shows two methods following the getter concept defined by the JavaBean specification.

20.3.2 Creating an Extension Package

The extension package is delivered in a JAR file. JAR (Java ARchive) is a file format based on the popular ZIP file format and is used for aggregating many files into one. To create a JAR file, you can store `*.class` with the Java package folder structure in a ZIP file format or creating from a JAR tool.

To combine files into a JAR file (in this case), here is the sample code fragment::

```
jar cf MyTool.jar *.class
```

In this example, all the class files in the current directory are placed in the file named "`MyTool.jar`". A manifest file entry named META-INF/MANIFEST.MF is automatically generated by the JAR tool. The manifest file is the place where any meta-information about the archive is stored as named: Value pairs. Please refer to the JAR file specification for more details.

```
jar cf MyTool.jar mytool
```

The example above shows that all the class files in the directory mytool are placed in the file named "`MyTool.jar`".

A complete JAR tool tutorial can be found at:

- http://java.sun.com/docs/books/tutorial/deployment/jar/
- http://java.sun.com/javase/6/docs/technotes/tools/windows/jar.html

20.4 Installing Custom Tool

To install the extension, copy the JAR file containing a custom tool class into the extensions directory under the Report Wizard plugin directory or template directory. Report Wizard will automatically load a new JAR file when a new report is generated.

20.5 Importing Custom Tool to Template

Report Wizard uses the import directive for importing a Custom Tool to a template.

An example of a directive:

```
#import('prefix', 'name')
```

The import directive declares that the template uses the custom tool, names the tool that defines it, and specifies its tag prefix before the custom tool is used in a template page. You can use more than one import directive in a template, but the prefix defined in each template page must be unique.

20.5.1 Attributes

- name (type:String)

  The uniform or full qualified class name that locates the Tool class. For example, `mytool.HelloTool`

- prefix (type:String)

  The prefix that precedes the custom tool name such as a string in `$hello.getHello()`. Empty prefixes are not allowed. When developing or using custom tools, do not use tag prefixes such as bookmark, sorter, template, file, array, group, map, iterator, list, date, report, exporter, profiling, and project, as they are reserved by Report Wizard.

The code fragment indicated below declares a custom tool.

```
#import('hello', 'mytool.HelloTool')
Get Hello
$hello.getHello()
Get Hello Foo
$hello.getHello('foo')
```

The output from the above template is:

```
Get Hello
Hello World
Get Hello Foo
Hello foo
```

21. Appendix B: Office Open XML Format Template

Office Open XML (also referred to as OOXML or Open XML) refers to standardized file formats for representing spreadsheets, charts, presentations and word processing documents for Microsoft Office. These standardized file formats become free and open as ECMA-376 Office Open XML File Formats - 2nd edition (December 2008) and ISO/IEC 29500:2008.

The most common Open XML file formats supported by Report Wizard include:

- DOCX - for word processing (text) documents
- XLSX - for spreadsheets
- PPTX - for presentations

21.1 Microsoft Office Word Document (DOCX)

Report Wizard supports most DOCX features. You can place the VTL codes inside core (properties) and content of any DOCX file. **All syntax usable in RTF can also be used in DOCX**.

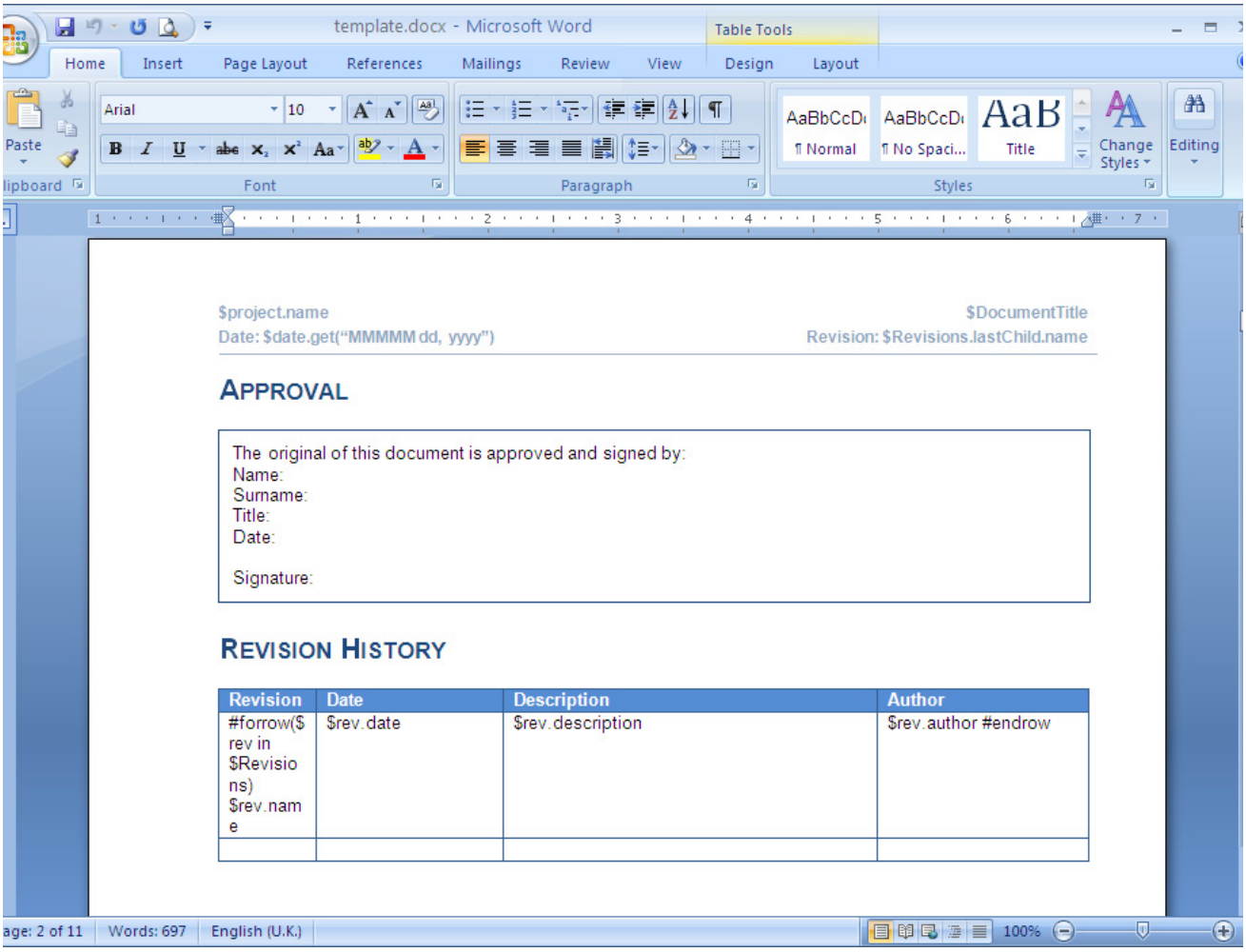


Figure 174 -- Sample of DOCX, Converted from an RTF Document

21.1.1 Limitations When Used in Microsoft Office Word Document

1. Cannot use #forcol in DOCX. If you try to use #forcol in a DOCX report template (Figure 175),

1 Using Forcol
2

#forcol($c in $Class)$c.name	#endcol

3

Figure 175 -- Using #forcol in DOCX

the error message in Figure 176 will then be shown.

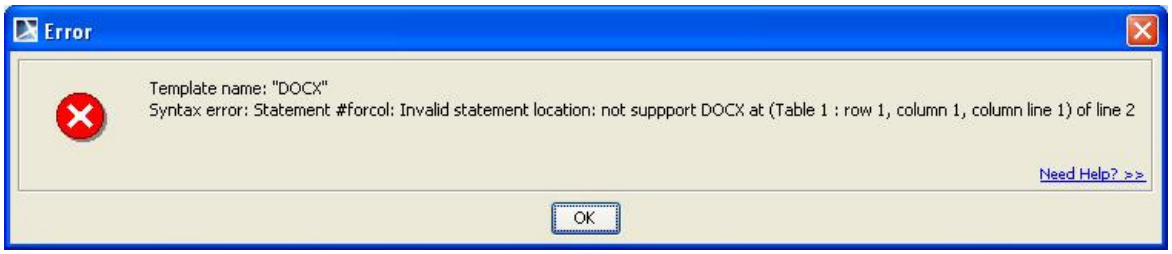


Figure 176 -- Error Message When Using #forcol in DOCX

2. Cannot use multi-line statements in different objects. If you try to use multi-line statements in DOCX (Figure 177),

1 **Using directive in different object**

2 #forpage($c in $Class)

$c.name #endpage

3

Figure 177 -- Invalid Usage of Multi-line Statement in DOCX

the error message in Figure 178 will then be shown.

Figure 178 -- The Error Message of Invalid Usage of the Multi-line Statements in DOCX

21.2 Microsoft Office Excel Worksheet (XLSX)

21.2.1 Multi-line Statements in XLSX

All multi-line directives such as `#if-#else-#elseif`, `#foreach` and `#macro`, must be used under the following conditions.

1. The beginning and ending statements must be declared within a single cell. Figure 179 and 180 below show samples of invalid usage of the `#if` and `#foreach` statements between cells respectively.

	A	B
1	#if($e.elementType == "usecase")	
2	$e.name	
3	#end	
4		

Figure 179 -- Invalid Usage of the Multi-line #if Statement in XLSX

	A	B	C	D	E
1					
2		**Use Case**			
3		#foreach($uc in $UseCase)	$uc.name	#end	
4					

Figure 180 -- Invalid Usage of the Multi-line #foreach Statement in XLSX

In Figure 179, since the body of the `#if` statement ($e.name) resides in the A2 cell, not in the A1 cell, the body will not be generated when generating a report, regardless of the evaluation of the `#if` statement.

In Figure 180, this code will break the structure of spreadsheet document.

Figure 181 and 182 demonstrate samples of valid usage of the `#if` and `#foreach` statements, respectively.

	A	B
1	#if($e.elementType == "usecase")$e.name#end	
2		

Figure 181 -- Valid Usage of the Multi-line #if Statement in XLSX

	A	B	C
1			
2		**Use Case**	
3		#foreach($uc in $UseCase) $uc.name #end	
4			

Figure 182 -- Valid Usage of Multi-line #foreach Statement in XLSX

2. VTL Macro must be declared within a single cell. Do not insert the multi-cell recorded macros in a single cell (Figure 183).

	A	B	C
1			
2		**Macro**	
3		#macro(insertCell $e)	
4		#if($e == "red") $e.name	
5		#else $e.name	
6		#end	
7		#end	
8			
9		**Use Macro**	
10		#foreach($e in $elements)	
11		#insertCell($e)	
12		#end	
13			

Figure 183 -- Invalid Usage of #macro Statement in XLSX

The macro will copy all contents between `#macro` and `#end`. Cells and rows will be included in the macro as well. Once this record has been inserted, the macro content will break the document structure.

Figure 184 demonstrates a sample of valid usage of the `#macro` statement.

	A	B	C
1			
2		**Macro**	
3		#macro(insert $e) #if($e == “red”) $e.name #else $e.name #end #end	
4			
5		**Use Macro**	
6		#foreach($e in $elements) #insertCell($e) #end	
7			

Figure 184 -- Valid Usage of #macro Statement in XLSX

21.2.2 Creating Data for Multiple Rows

The `#foreach` directive can only be used in a single cell record. To create data for multiple rows, use the `#forrow` directive instead (Figure 185).

	A	B	C	D
1	#forrow($uc in $UseCase)	$uc.name	#endrow	
2				

Figure 185 -- Usage of #forrow in XLSX

The output of the above code is shown in Figure 186.

	A	B	C
1		Use Case A	
2		Use Case B	
3		Use Case C	
4		Use Case D	
5			

Figure 186 -- Output of #forrow in XLSX

21.2.3 Creating Data for Multiple Columns

`#forcol` is used for creating data for multiple columns (Figure 187). This statement can be used in conjunction with the `#forrow` statement.

	A	B
1	#forcol($uc in $UseCase)$us.name#endcol	
2		

Figure 187 -- Usage of #forcol in XLSX

The output of the above code is shown in Figure 188.

	A	B	C	D	E
1	Use Case A	Use Case B	Use Case C	Use Case E	
2					

Figure 188 -- Output of #forcol in XLSX

21.2.4 Displaying Content in a Cell

Texts in any generated report are always wrapped. Also, the cells' width in the generated report depends on the cells' width in the report template used. For example, an XLSX report template in Figure 189 will generate an output report as shown in Figure 190.

	A	B	C	D
1	#forrow($uc in $UseCase)	$uc.name	#endrow	
2				

Figure 189 -- Sample of Wrapped Text (Column B) in an XLSX Report Template

	A	B	C
1		this_is_u secase1	
2		this_is_u secase2	
3		this_is_u secase3	
4			

Figure 190 -- Wrapping Text Output in XLSX

21.2.5 Limitation When Used in Microsoft Office Excel Worksheet

`#sectionBegin` and `#includeSection` cannot be used in XLSX. If you try to use `#sectionBegin` (or `#includeSection`) in an XLSX report template (Figure 191),

	A	B
1		
2		
3		
4	#sectionBegin(SectionA)	
5		
6	This is section begin	
7		
8		
9	#sectionEnd	
10		

Figure 191 -- Using #sectionBegin in XLSX

the error message in Figure 192 will then be shown.

Figure 192 -- Error Message When Using #sectionBegin in XLSX

21.3 Microsoft Office PowerPoint Presentation (PPTX)

A presentation document requires a special document template. This template does not contain any content order, and each text content is always placed inside a text box. A text box is an image structure (an image structure keeps the position of each image in x, y coordinates). Text box positions can be changed. Text boxes can also be placed in the same positions as others (Figure 193).

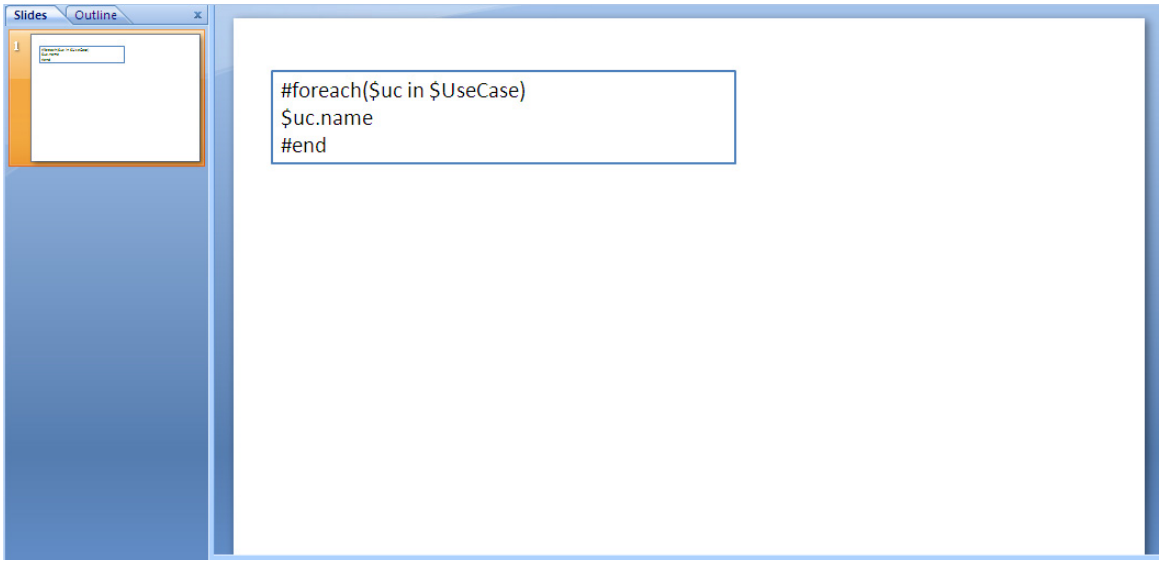


Figure 193 -- Sample of PPTX Template

21.3.1 Multi-line Statements in PPTX

Similar to XLSX, all multi-line directives such as `#if-#else-#elseif`, `#foreach`, and `#macro` must be used under the following conditions.

1. The beginning and ending statements must be declared within a single text box. Figure 194 below shows the sample of invalid usage of the`#foreach` statement between the text boxes.

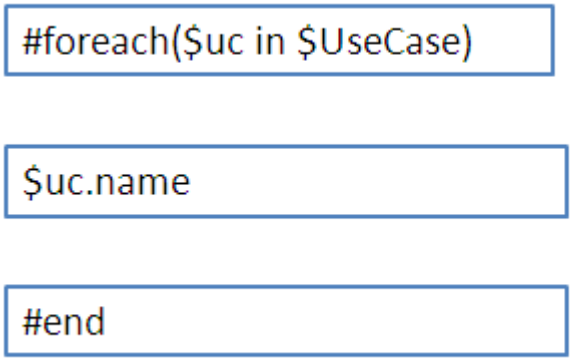


Figure 194 -- Invalid Usage of Multi-line Statement #foreach in PPTX

Since the PPTX template does not provide the statement order, the template will not be interpreted as the order of the displayed images (text boxes). For example, `$uc.name` may not be processed after `#foreach($uc in $UseCase)` has been completely processed. Figure 195 below demonstrates a sample of valid usage of the `#foreach` statement.

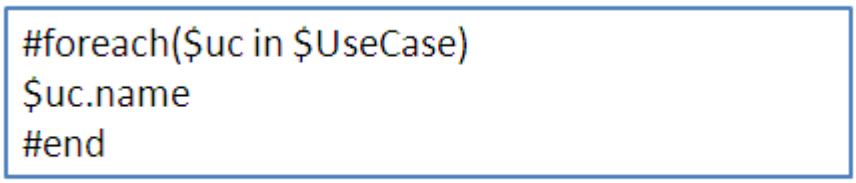


Figure 195 -- Valid Usage of Multi-line Statement #foreach in PPTX

2. VTL Macro must be declared within a single text box. Do not insert the multi-cell recorded macros in a single text box (Figure 196).

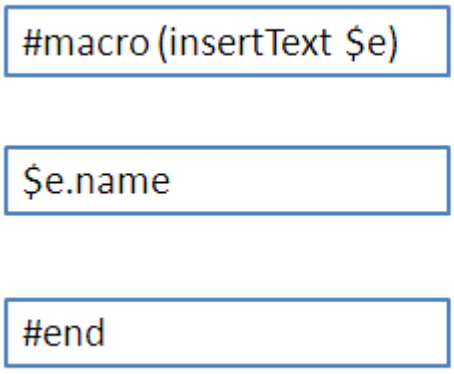


Figure 196 -- Invalid Usage of #macro Statement in PPTX

Since each text box does not have any sequence order, the macro cannot record any content between the text boxes. Figure 197 demonstrates a valid usage of the `#macro` statement.

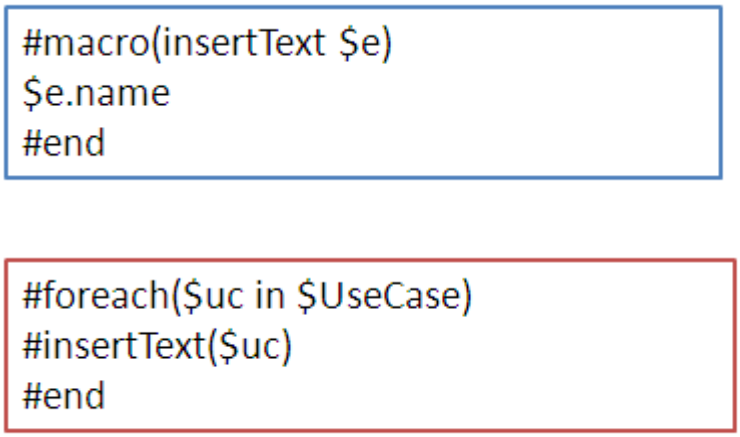


Figure 197 -- Valid Usage of #macro Statement in PPTX

21.3.2 Creating Data for Multiple Slides

In PPTXreport templates, you can use the `#forpage` directive to create additional slide(s) in your presentation. You can use `#forpage` and `#endpage` directives in any text box. However, the `#endpage` directive must be in same slide as the `#forpage` directive is or in the following slide, but not before the slide which contains the `#forpage` directive. All directives contained in the slides between the slide the `#forpage` directive appears and the slide the `#endpage` directive appears will be included within the `#forpage` statement. For example, the template in Figure 198 will produce the output in Figure 199.

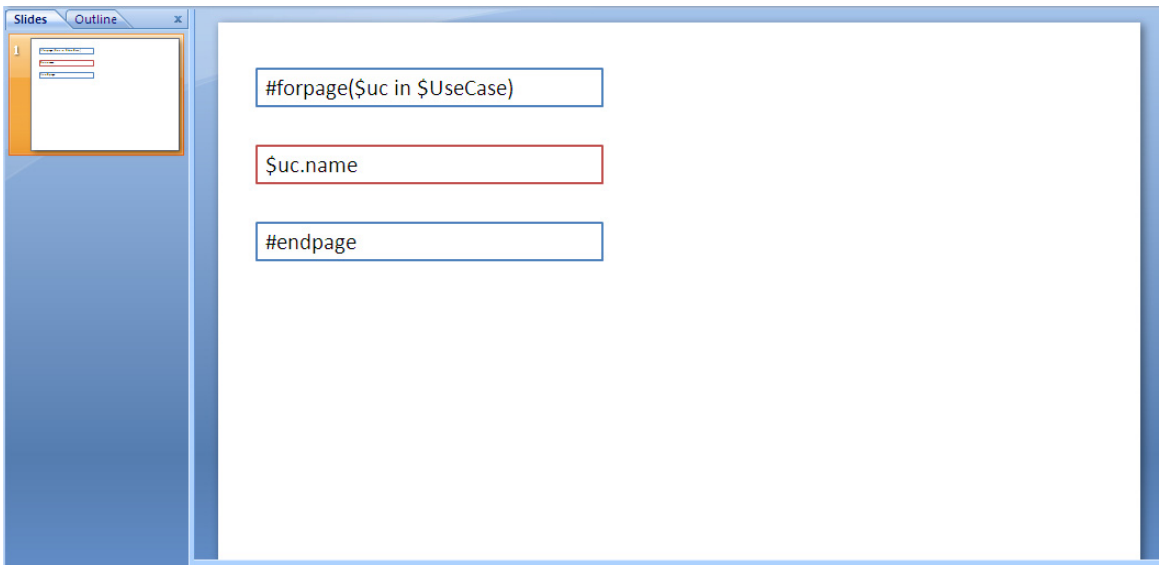


Figure 198 -- Sample of Valid Usage of #forpage in PPTX

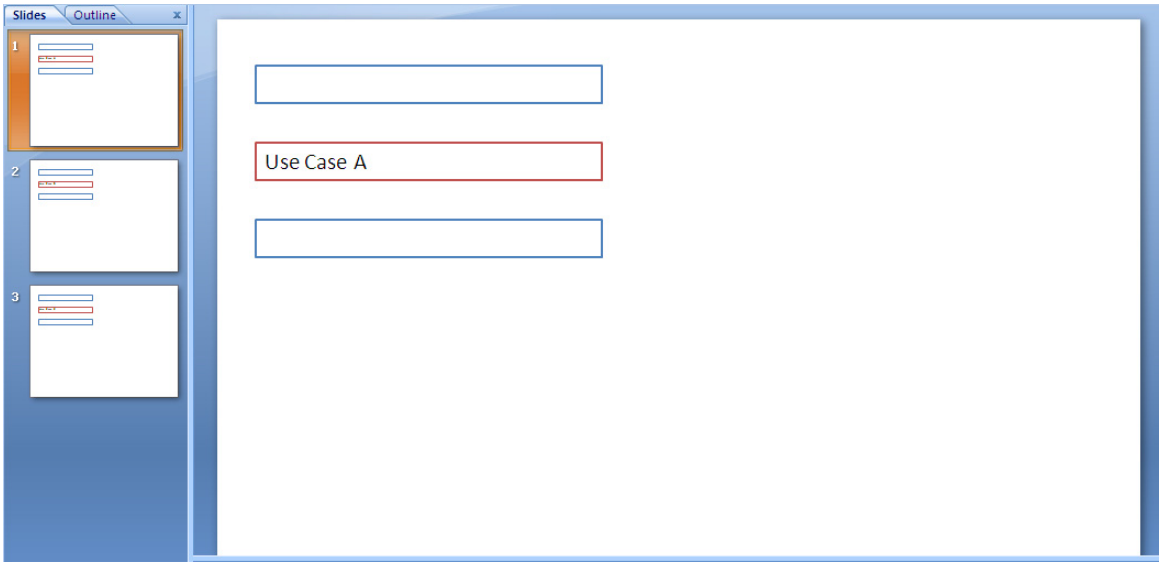


Figure 199 -- Sample Output of Valid Usage of #forpage in PPTX

21.3.3 Creating a Page with Conditions

Since a directive in the PPTX report template does not provide any statement order (unless residing in the same text box), the `#if` directive cannot be used with the `#forpage` directive (Figure 200).

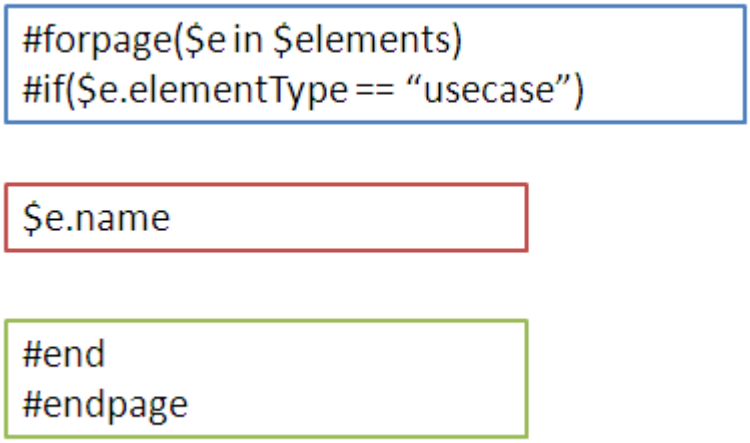


Figure 200 -- Sample of Looping With Condition in General Style

The code in Figure 200 will not produce the output report as expected, because the `#forpage` directive automatically covers all directives in the current page regardless of the statement order. Consequently, the `#if` directive may not be interpreted after the `#forpage` directive.

To avoid this problem, you can use the `$report.filterElement($elements, $types)` method. This helper method provides the element filter for the specified type. In this case, use the following code (Figure 201).

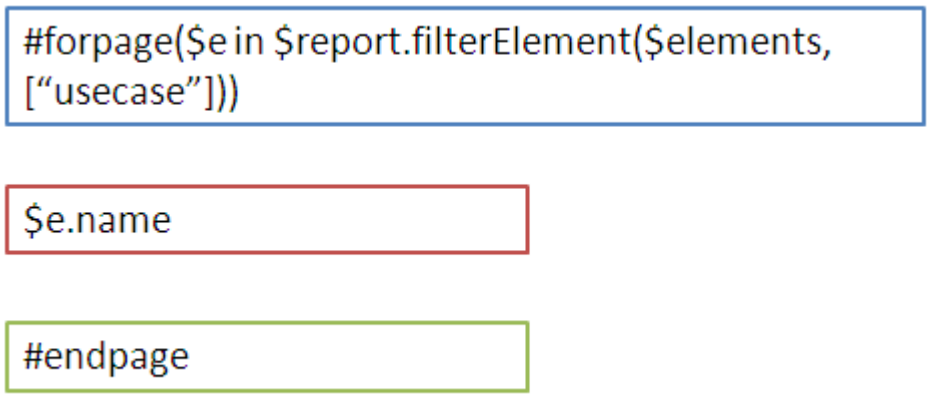


Figure 201 -- Sample of Looping With Conditional Filter

For more details on `$report.filterElement($elements, $types)`, see Section 4 Helper Modules.

21.3.4 Limitation When Used in Microsoft Office PowerPoint Presentation

`#sectionBegin`, `#includeSection`, `#forrow` and `#forcol` cannot be used in any PPTX report template. If you try, for example, to use `#sectionBegin` in a PPTX report template (Figure 202),

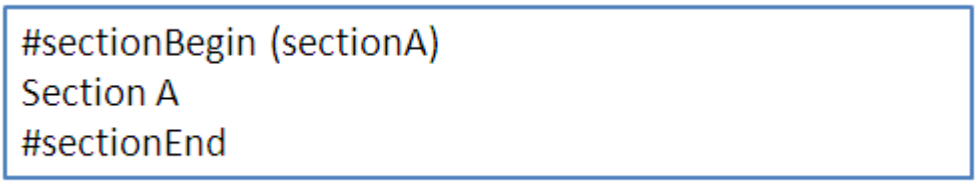


Figure 202 -- Using #sectionBegin in PPTX

the error message in Figure 203 will then be displayed.

Figure 203 -- Error Message When Using #sectionBegin in PPTX

22. Appendix C: OpenDocument Format Template

The **OpenDocument** format (ODF) is an open file format for office documents, such as spreadsheets, presentations, and word processing documents. The standard was developed by the Open Office XML technical committee of the Organization for the Advancement of Structured Information Standards (OASIS) consortium.

The version 1.0 manifestation was published as an **ISO/IEC International Standard, ISO/IEC 26300:2006 Open Document Format for Office Applications (OpenDocument) v1.0**.

The most common files supported by Report Wizard are:

- ODT - for word processing (text) documents
- ODS - for spreadsheets
- ODP - for presentations

22.1 OpenDocument Text

Report Wizard supports most OpenDocument Text (ODT) features. It enables you to input the VTL codes inside meta-data, styles, and text content. All syntax usable inside RTF can be used with ODT.

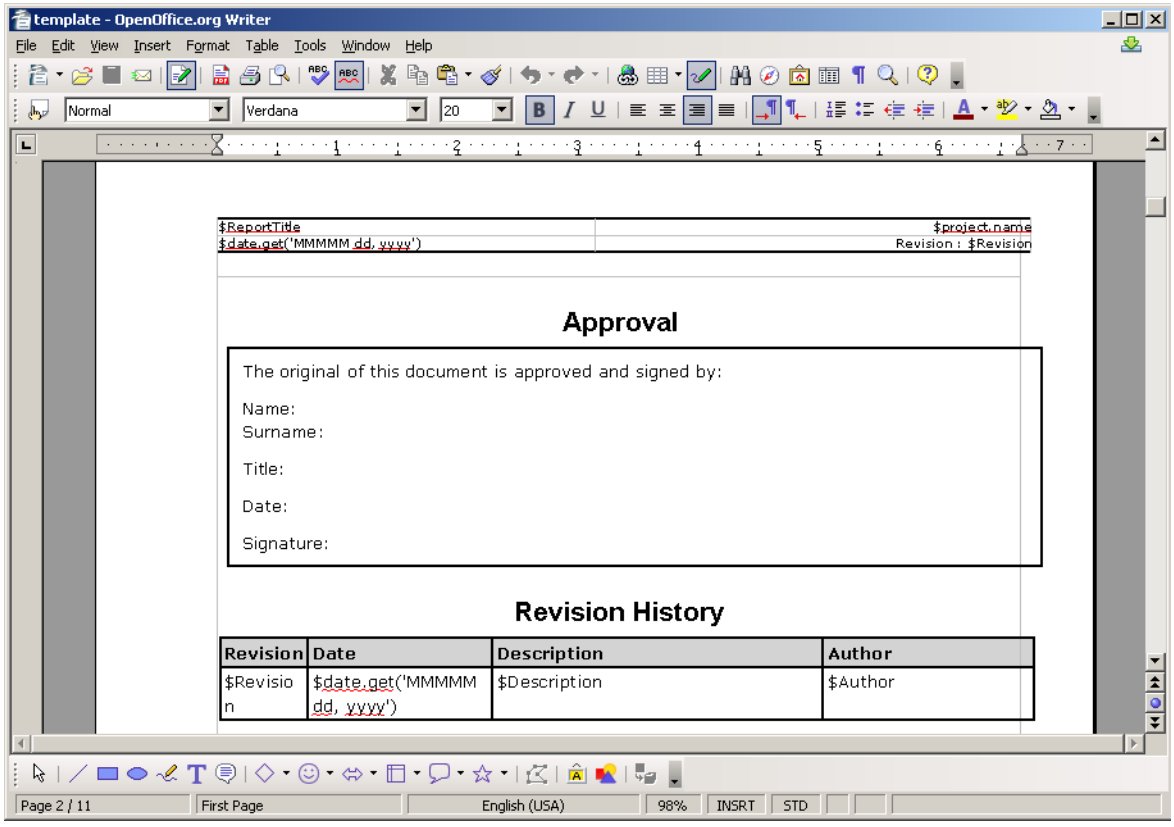


Figure 204 -- Sample of OpenDocument Text Converted From RTF Document

22.2 OpenDocument Spreadsheet

All multiline directives such as `#if-#else-#elseif`, `#foreach`, and `#macro` must be used under the following conditions.

1. The beginning and ending statements must be declared within a single cell. Figure 205 and 206 below show two samples of invalid usage of the multiline statements between cells.

	A	B
1	#if ($e.elementType == "usecase")	
2	$e.name	
3	#end	
4		

Figure 205 -- Invalid Usage of the Multiline #if Statement in ODS

	A	B	C	D
1				
2		**Use Case**		
3		#foreach ($uc in $UseCase)	$name	#end
4				

Figure 206 -- Invalid Usage of Multiline #foreach Statement in ODS

Since the body of the `#if` statement is contained in the cell A2, this cell will not be generated if the condition is not true (the element type is not “usecase”). Due to the constraints of spreadsheet document structure, the number of cells in a column must be equal to the number of cells in all columns, and the number of cells in a row must be equal to the number of cells in all rows.

The codes in Figure 206 will break the structure of a spreadsheet document. Figure 207 and 208 demonstrate two samples of valid usage of the multiline statements.

	A	B
1	#if ($e.elementType == "usecase")$e.name#end	
2		

Figure 207 -- Valid Usage of Multiline #if Statement in ODS

	A	B	C
1			
2		**Use Case**	
3		#foreach ($uc in $UseCase) $name #end	
4			

Figure 208 -- Valid Usage of Multiline #foreach Statement in ODS

2. VTL Macro must be declared within a single cell. Do not insert the multi-cell recorded macros in a single cell (Figure 209).

	A	B	C
1			
2		**Macro**	
3		#macro(insertCell $e)	
4		#if ($e=="red") $e.name	
5		#else $e.name	
6		#end	
7		#end	
8			
9		**Use Macro**	
10		#foreach ($e in $elements)	
11		#insertCell($e)	
12		#end	
13			

Figure 209 -- Invalid Usage of Macro Statement in ODS

The macro will copy all contents between `#macro` and `#end`. Cells and rows will be included in the macro as well. Once this record has been inserted, the macro content will break the document's structure.

22.2.1 Creating Data for Multiple Rows

The `#foreach` directive can only be used in a single cell record. To create data for multiple rows, use the `#forrow` directive instead (Figure 210).

	A	B	C
1	#forrow($uc in $UseCase)	$uc.name	#endrow

Figure 210 -- Usage of #forrow

As shown in Figure 210, the output will be:

	A	B	C
1		Use Case A	
2		Use Case B	
3		Use Case C	
4		Use Case D	
5			

Figure 211 -- Output of #forrow

22.2.2 Creating Data for Multiple Columns

`#forcol` is used to create data for multiple columns (Figure 212). This statement can be used with `#forrow` (See the sample from the "Other Document" template).

	A	B
1	#forcol($uc in $UseCase)$uc.name#endcol	
2		

Figure 212 -- Usage of #forcol

From the usage of `#forcol` shown in Figure 212, the output will be:

	A	B	C	D	E
1	Use Case A	Use Case B	Use Case C	Use Case D	
2					

Figure 213 -- The Output of #forcol

22.3 OpenDocument Presentation

A presentation document is a special document template. This template does not contain a content order. The text content used within this document is inserted inside a text box. A text box is an image structure (An image structure keeps the position of each image in x, y coordinates).

Text box positions can be changed. Text boxes can also be placed in the same positions as others (Figure 214).

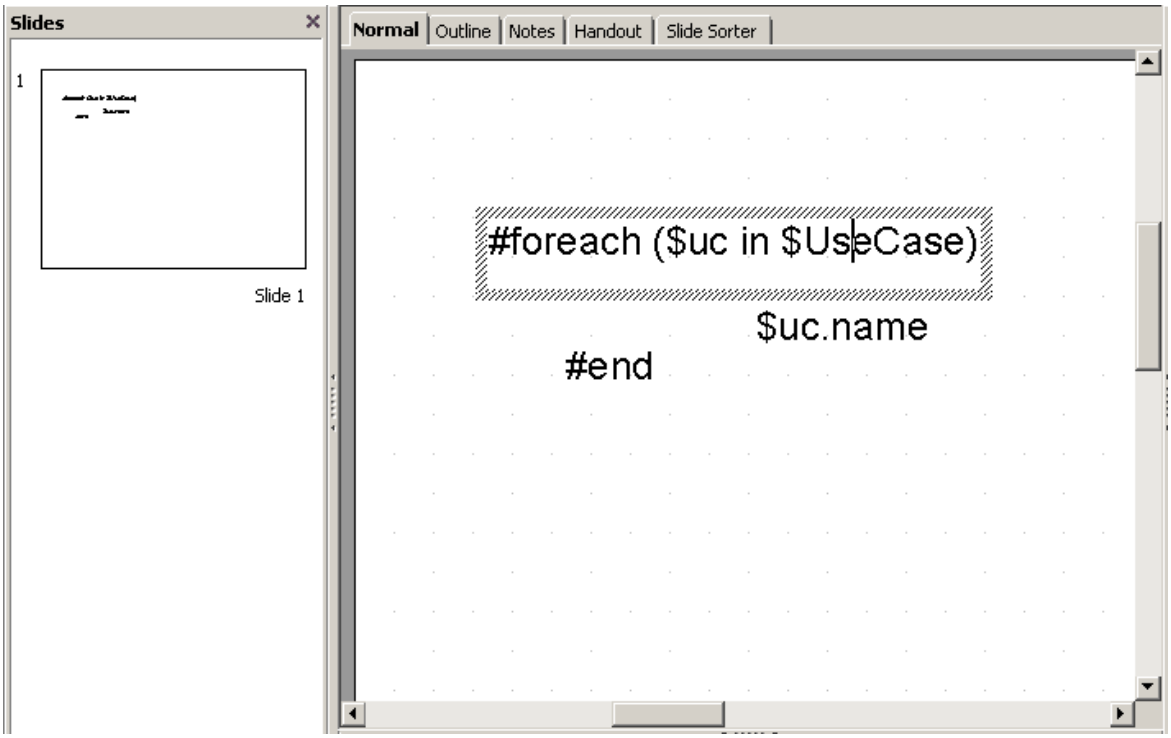


Figure 214 -- Sample of ODP Template

Using the same concept as ODS, all multi-line directives such as `#if-#else-#elseif`, `#foreach`, and `#macro` must be used under conditions.

1. The beginning and ending statements must be declared within a single text box. Figure 215 below shows the sample of invalid usage of the multiline `#foreach` statement between the text boxes.

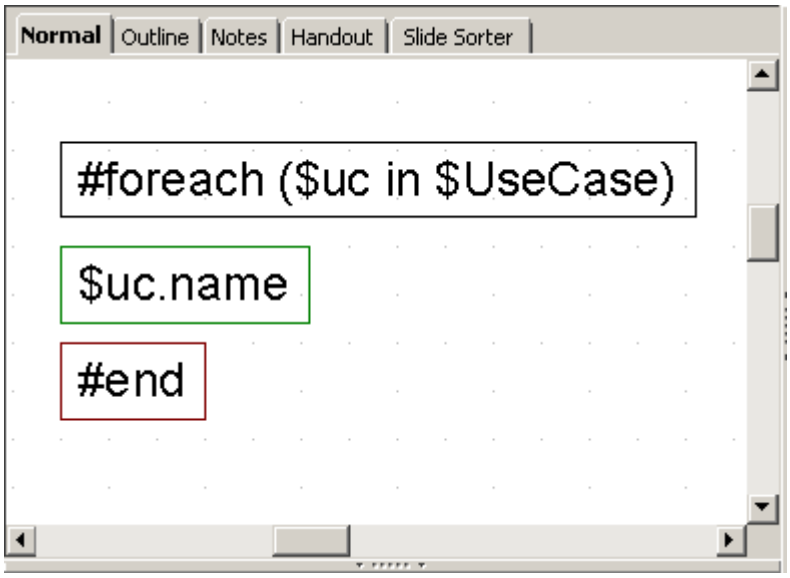


Figure 215 -- Invalid Usage of Multiline Statement in ODP

Since the ODP template does not provide the statement order, the template will not be interpreted in the order of the displayed images. For example, `$uc`.name may not be processed after `#foreach($uc in $UseCase)` has been completed.

Figure 216 below shows the sample of valid usage of the `#foreach` statement.

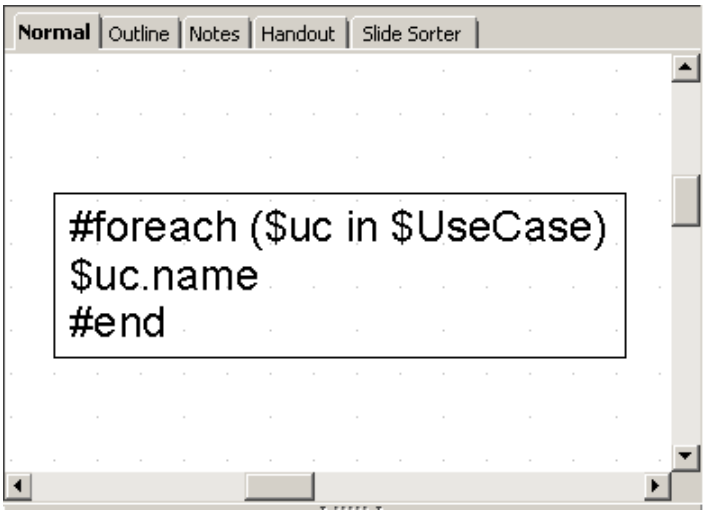


Figure 216 -- Valid Usage of Multiline #foreach Statement in ODP

2. VTL Macro must be declared within a single text box. Do not insert the multi-cell recorded macros in a single text box (Figure 217).

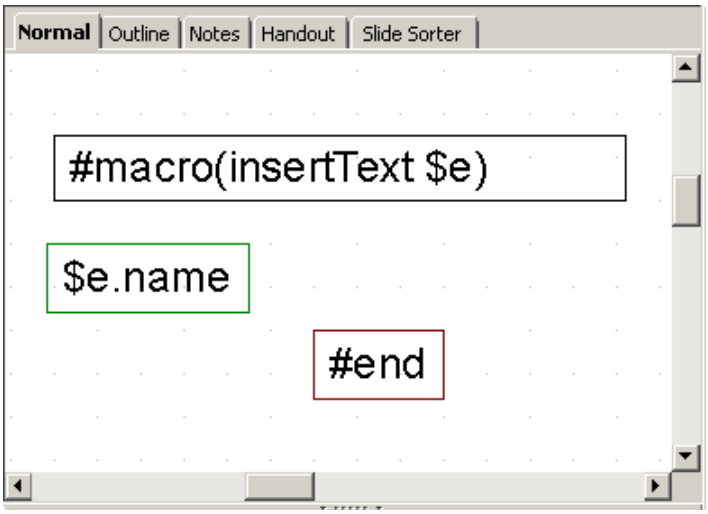


Figure 217 -- Invalid Usage of Macro Statement in ODP

The text box does not have a sequence order; therefore, macro cannot record any content between the text boxes (Figure 218).

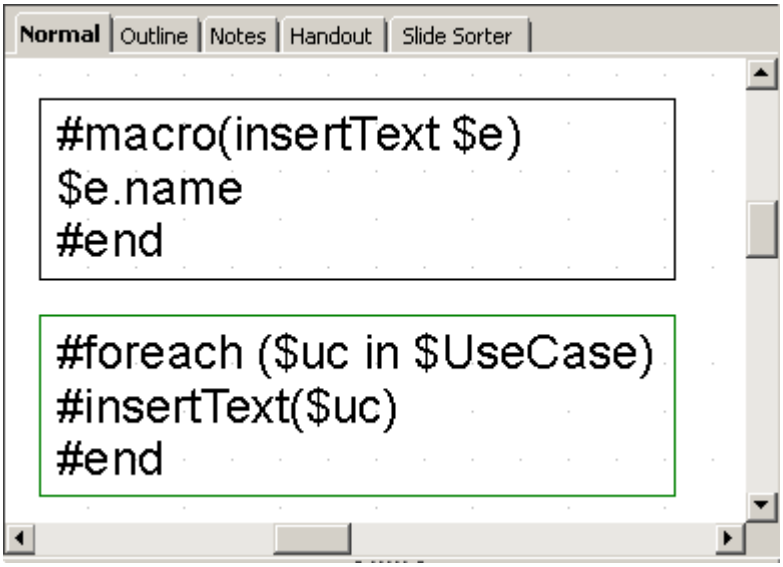


Figure 218 -- Valid Usage of Macro Statement in ODP

22.3.1 Creating Data for Multiple Slides

ODP uses the `#forpage` directive to create a slide for each data. The `#forpage` directive does not contain any order. You can use `#forpage` and `#endpage` in any text boxes. All directives inside the slide will be included within the `#forpage` statement (Figure 219).

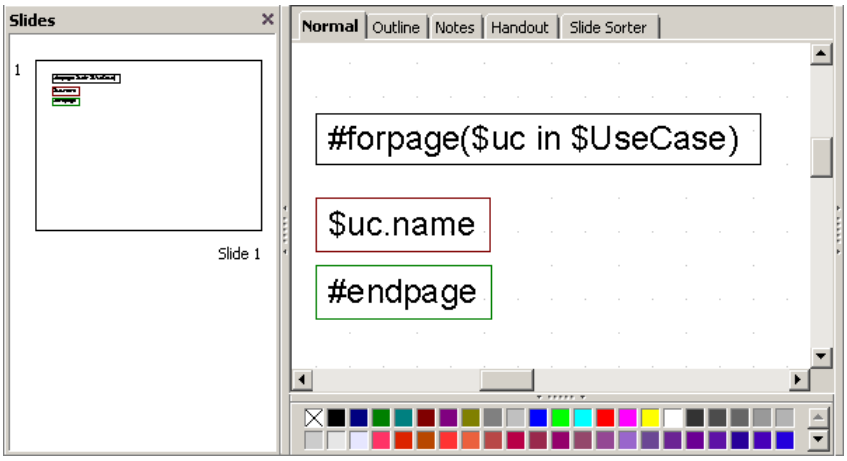


Figure 219 -- Sample of #forpage Usage

The output from the code in Figure 220 is an ODP with a single use case name for each slide.

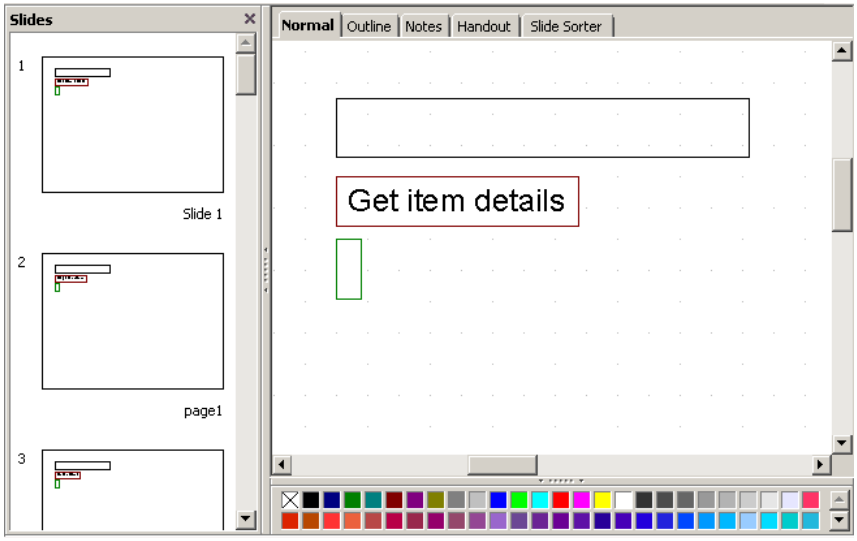


Figure 220 -- Sample Output from #forpage

For more samples of ODP reports, see the “Other Document” template.

22.3.2 Creating Page with Conditions

The ODP directive does not provide any statement order; therefore, the `#if` directive will not be attached to the `#forpage` statement (Figure 221).

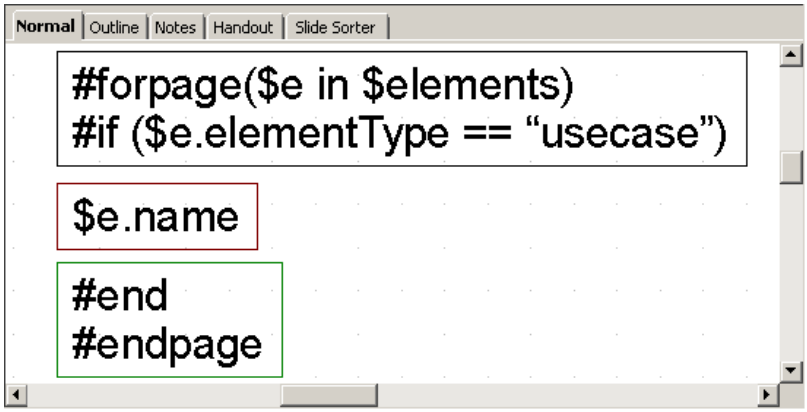


Figure 221 -- Sample of Looping with Condition in General Style

The codes in Figure 221 may not produce the report exactly as expected. `#forpage` automatically covers all directives in the current page without any statement order. Therefore, the `#if` directive may not be interpreted after `#forpage`.

To solve this problem you can use the `$report.filterElement($elements, $types)` method. This helper method provides the element filter for the specified type. The codes are shown in Figure 222:

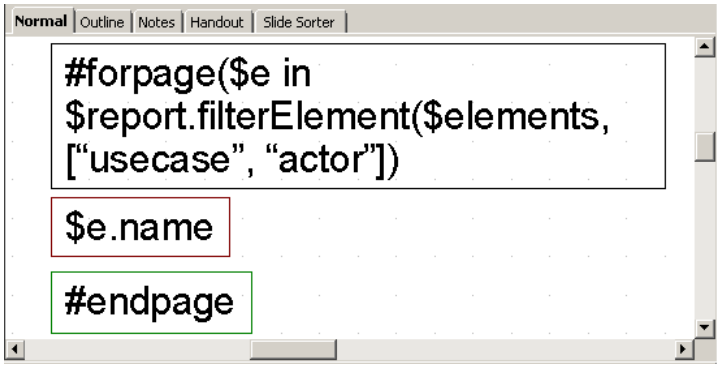


Figure 222 -- Sample of Looping with Conditional Filter

For more details on `$report.filterElement($elements, $types)`, see Section **4 Helper Modules**.

22.4 OpenDocument Conversion Tool

The following tools can convert RTF documents to ODF documents.

22.4.1 Microsoft Office ODF Extensions

This Microsoft Office add-on allows Microsoft Office to open ODF documents and save Microsoft Office documents in the OpenDocument format.

Download Microsoft Office ODF Extensions from http://odf-converter.sourceforge.net/

22.4.2 OpenOffice.org

OpenOffice.org is an office suite that can open and save documents in many formats. This tool can also open RTF documents and save them as ODF documents.

Download OpenOffice.org from http://www.openoffice.org.

23. Appendix D: HTML Tag Support

Report Wizard supports HTML code conversion to RTF, ODF, and OOXML file formats. The HTML code is created from the Element Documentation pane (Figure 223).

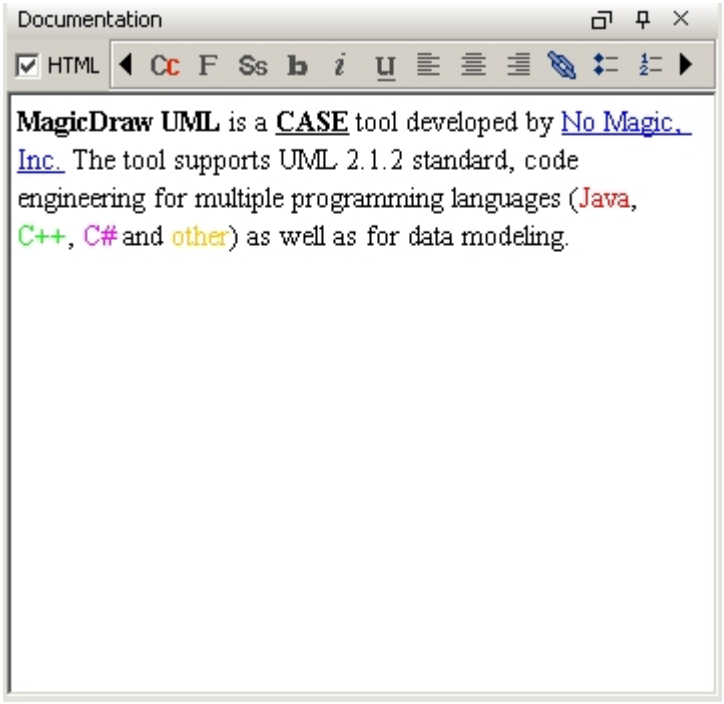


Figure 223 -- MagicDraw Documentation Pane

The element documentation can also be retrieved by the following VTL code:

```
#foreach ($e in $elements)
$e.documentation
#end
```

Whenever the report engine encounters HTML content, it will automatically convert the content into a valid output format style.

23.1 Supported HTML Tags

23.1.1 Font Tags

A font tag consists of three attributes: (23.1.1.1) Size, (23.1.1.2) Face, and (23.1.1.3) Color.

23.1.1.1 Size

The Size attribute determines the font size. Possible values are integers from 1 to 7. The default base font size is 3. The greater the value is, the larger the size becomes.

- The base font size for RTF documents is 24 dot (equivalent to 12 pt).
- The base font size for ODF documents is 12 pt.
- The base font size for OOXML documents is 12 pt.
- The base font size for HTML documents is determined by the web browser.

Each value will be multiplied by two. Shown below is an example of the size attribute:

```
<font size="3">
It will be rendered as font size 12 pt
<font size="5">
It will be rendered as font size 16 pt
<font size="1">
It will be rendered as font size 8 pt
```

If the size attribute is specified without the face attribute, the default font will be determined by the template or document editor, unless the font tag is covered by other HTML elements such as `<code>` or `<tt>`.

23.1.1.2 Face

The Face attribute defines the font name. If the face attribute is specified without the size attribute, the default size will be determined by the template or the document editor.

23.1.1.3 Color

The Color attribute specifies the text color. A color value can be either a hexadecimal number (prefixed with a hash mark) or one of the following sixteen colors. Colors are case-insensitive.

Table 31 -- Font Colors

Color	Hexadecimal code
Black	#000000
Silver	#C0C0C0
Gray	#808080
White	#FFFFFF
Maroon	#800000
Red	#FF0000
Purple	#800080
Fuchsia	#FF00FF
Green	#008000
Lime	#00FF00
Olive	#808000
Yellow	#FFFF00
Navy	#000080
Blue	#0000FF
Teal	#008080
Aqua	#00FFFF

For example:

```
<font face="Arial"> This is Arial text</font><br>
<font face="Comic Sans MS">This is Comic Sans text</font><br>
<font face="Arial" size="1">This is small Arial</font><br>
<font face="Arial" size="7">This is large Arial</font><br>
<font face="Arial" color="Red">This is red Arial</font><br>
<font face="Arial" color="#FF0000">and this is red Arial too</font><br>
```

Figure 224 -- Sample of Font Tags

As shown in Figure 225, the outputs in RTF, ODF, or HTML will be as follows:

This is Arial text

This is Comic Sans MS text

This is small Arial

This is large Arial

This is red Arial

and this is red Arial too

Figure 225 -- Sample of RTF, ODF, or HTML Document Outputs

23.1.2 Font Style Tag

Table 32 -- Font Style Elements

Tag name	Description
TT	Renders teletyped or monospaced text.
I	Renders italic text.
B	Renders bold text.
BIG	Renders text in large font.
SMALL	Renders text in small font.
STRIKE and S	Renders strikethrough text.
U	Renders underlined text.

- TT – This tag will be rendered as `<font face="Courier New">`
- I – This tag is supported by the existing HTML conversion component.
- B – This tag is supported by the existing HTML conversion component.
- BIG – This tag will be rendered as `<font size="5">`
- SMALL – This tag will be rendered as `<font size="1">`
- STRIKE and S – This tag will be rendered as a strikethrough text.
- U – This tag is supported by the existing HTML conversion component.

23.1.3 Phrase Elements

Table 33 -- Phrase Elements

Tag name	Function
EM	Indicates emphasis.
STRONG	Indicates stronger emphasis.
CITE	Contains a citation or a reference to other sources.
DFN	Indicates that this is the defining instance of the enclosed term.
CODE	Designates a fragment of computer code.
SAMP	Designates a sample output from programs, scripts, etc.
KBD	Indicates the text to be entered by the user.
VAR	Indicates an instance of a variable or program argument.
ABBR	Indicates an abbreviated form such as WWW, HTTP, URI, and Mass.
ACRONYM	Indicates an acronym such as WAC and radar.

- EM – This tag will be rendered as `<i>`
- STRONG – This tag will be rendered as `<b>`
- CITE – This tag will be rendered as `<i>`
- DFN – This tag will be rendered as `<i>`
- CODE – This tag will be rendered as `<font face=”Courier New”>`
- SAMP – This tag will be rendered as `<font face=”Courier New”>`
- KBD – This tag will be rendered as `<font face=”Courier New”>`
- VAR – This tag will be rendered as `<i>`
- ABBR – This tag will be rendered as normal text.
- ACRONYM – This tag will be rendered as normal text.

23.1.4 Ordered and Unordered Lists and List Item Tags

Ordered and unordered lists are rendered in an identical manner, except that ordered list items are numbered.

The report engine supports both unordered and ordered lists without attributes. The list tag attributes will be ignored in the report output. The list tag attributes are type, start, value, and compact.

Both unordered and ordered lists are not supported in both XLSX and ODS templates.

23.1.4.1 Ordered Lists

An Ordered List is defined by the OL element. The element contains one or more LI elements that define the actual items of the list.

Unlike unordered lists (UL), items in an ordered list have a definite sequence. A conversion will render each item in the list with a number. All OL attributes will be ignored in the report output. An example of the Ordered List tag is shown in Figure 226:

```
<OL>
   <LI>Apple</LI>
   <LI>Orange</LI>
   <LI>Banana</LI>
</OL>
```

Figure 226 -- Sample of Ordered List Tag

As shown in Figure 227, the outputs in RTF, ODF, or HTML will be as follows:

1. Apple
2. Orange
3. Banana

Figure 227 -- The Sample of RTF, ODF, or HTML Document Outputs

23.1.4.2 Nested Ordered Lists

HTML conversion will indent nested lists with respect to the current level of nesting. A number for each level will restart at the value 1. An example of the Nested Ordered List tag is shown in Figure 228:

```
<OL>
  <LI>Apple</LI>
  <LI>Orange</LI>
  <OL>
    <LI>Banana</LI>
    <OL>
      <LI>Grape</LI>
      <OL>
        <LI>Mango</LI>
      </OL>
    </OL>
  </OL>
</OL>
```

Figure 228 -- Sample of Nested OL Tags

As shown in Figure 229, the outputs in RTF, ODF, or HTML will be as follows:

1. Apple
2. Orange
    1. Banana
        1. Grape
            1. Mango

Figure 229 -- The Sample of RTF, ODF, or HTML Document Outputs

23.1.4.3 Unordered Lists

An Unordered List is defined by the UL element. The element contains one or more LI elements that define the actual items of the list.

A conversion will render the UL element with a bullet preceding each list item. All UL attributes will be ignored in the report output. Figure 230 shows an example of an Unordered List tag:

```
<UL>
  <LI>Apple</LI>
  <LI>Orange</LI>
  <LI>Banana</LI>
</UL>
```

Figure 230 -- Sample of UL Tags

As shown in Figure 231, the outputs in RTF, ODF, or HTML will be as follows:

- Apple
- Orange
- Banana

Figure 231 -- Sample of RTF, ODF, or HTML Document Outputs

23.1.4.4 Nested Unordered Lists

Lists can also be nested. HTML conversion will indent nested lists with respect to the current level of nesting.

HTML conversion should attempt to present a small filled-in circle to the first level, a small circle outline to the second level, and a filled-in square to the third level. Bullets after the third level are filled-in squares. An example of the Nested Unordered List tag is shown in Figure 232:

```
<UL>
  <LI>Apple</LI>
  <LI>Orange</LI>
  <UL>
    <LI>Banana</LI>
    <UL>
      <LI>Grape</LI>
      <UL>
        <LI>Mango</LI>
      </UL>
    </UL>
  </UL>
</UL>
```

Figure 232 -- Sample of Nested UL Tags

As shown in Figure 233, the outputs in RTF, ODF, or HTML will be as follows:

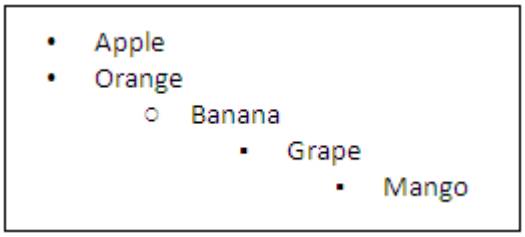


Figure 233 -- Sample of RTF, ODF or HTML Document Outputs

23.1.5 Definition List Tags

A Definition List is defined by the DL element. An entry in the list is created using the DT element for the term being defined and the DD element for the definition of the term.

A definition list can have multiple terms for a given definition as well as multiple definitions for a given term. Authors can also give a term without a corresponding definition, and vice versa, but such a structure rarely makes sense.

A conversion will render DT as a non-indent item and DD as a single indent item. The Definition List tag is not supported in the XLSX template. Figure 234 shows an example of a Definition List tag:

```
<DL>
  <DT>Dweeb</DT>
  <DD>young excitable person who may mature into a Nerd or Geek</DD>
  <DT>Hacker</DT>
  <DD>a clever programmer</DD>
  <DT>Nerd</DT>
  <DD>technically bright but socially inept person</DD>
</DL>
```

Figure 234 -- Sample of DL Tags

As shown in Figure 235, the outputs in RTF, ODF or HTML will be as follows:

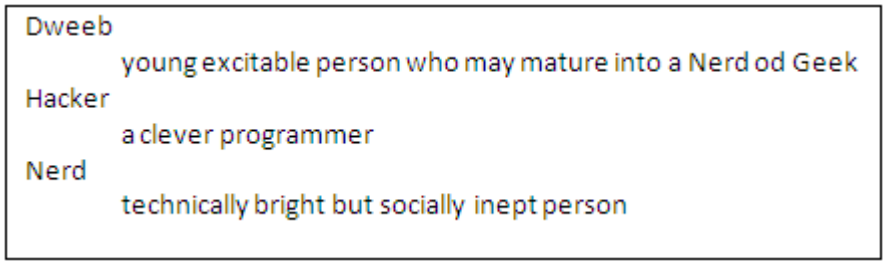


Figure 235 -- Sample of RTF, ODF, or HTML Document Outputs

23.1.6 Line and Paragraph Tags

A line is defined by the
 element. The element inserts a single line break. It is an empty tag. This means that it has no end tag. The line attributes will be ignored in the report output.

A paragraph is defined by the <P> element. The element automatically creates some space before and after itself. The paragraph attributes will be ignored in the report output. Figure 236 shows an example of a Line and Paragraph tag:

```
This is first text
This is second text <br>
This is third text
<p> This is paragraph </p>
```

Figure 236 -- Sample of Line and Paragraph Tags

As shown in Figure 237, the outputs in RTF, ODF, or HTML will be as follows:

This is first text This is second text

This is third text

This is paragraph

Figure 237 -- The Sample of RTF, ODF, or HTML Document Outputs

23.1.7 Preformatted Text

A preformatted text is defined by the <PRE> element. All the space and carriage returns are rendered exactly as you type them. The preformatted attributes will be ignored in the report output. Figure 238 shows an example of the preformatted text:

```
<p>
  The PRE element tells visual
     user agents that
        the enclosed text
      is "preformatted"
</p>
<pre>
  The PRE element tells visual
     user agents that
        the enclosed text
      is "preformatted"
</pre>
```

Figure 238 -- Sample of Preformatted Text

As shown in Figure 239 and 240 respectively, the outputs in RTF/ ODF and HTML will be as follows:

```
The PRE element tells visual user agents that the enclosed text is "preformatted"

    The PRE element tells visual
       user agents that
          the enclosed text
        is "preformatted"
```

Figure 239 -- Sample of RTF / ODF Document Output

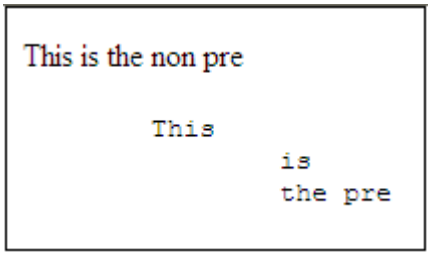


Figure 240 -- Sample of HTML Document Output (Open in Internet Explorer 7.0)

23.1.8 Heading Tags

A heading is defined by the <H1>, <H2>, <H3>, <H4>, <H5>, or <H6> element. In this report, all heading tags will be rendered as <b> for ODT, RTF, and OOXML document outputs. The heading attributes will be ignored in the report output. Figure 241 shows an example of Heading tags.

```
<h1>Heading 1</h1>
<h2>Heading 2</h2>
<h3>Heading 3</h3>
<h4>Heading 4</h4>
<h5>Heading 5</h5>
<h6>Heading 6</h6>
```

Figure 241 -- Sample of Heading Tags

As shown in Figure 242 and 243 respectively, the outputs in RTF / ODF and HTML will be as follows:

```
Heading 1
Heading 2
Heading 3
Heading 4
Heading 5
Heading 6
```

Figure 242 -- Sample of RTF / ODF Document Output

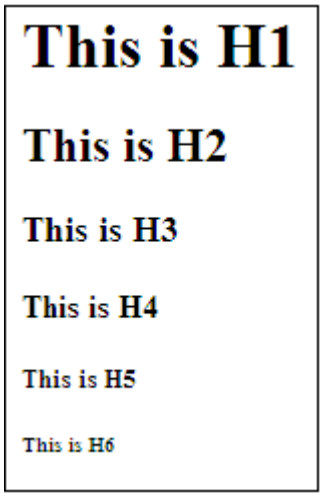


Figure 243 -- Sample of HTML Document Output (Open in Internet Explorer 7.0)

23.1.9 Link Tags

A Link tag is defined by the <A> element. This element is used to create a link to another document with the href attribute. The href attribute specifies the destination of the link. Link tag is not supported in the XLSX template. Figure 244 shows an example of a Link tag:

```
<A href="http://www.google.co.th">
This is a link to www.google.com
</A>
```

Figure 244 -- Sample of Link Tag

As shown in Figure 245, 246, and 247 respectively, the outputs in RTF, ODF, and HTML will be as follows:

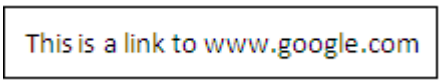


Figure 245 -- Sample of RTF Document Output

Figure 246 -- Sample of ODF Document Output

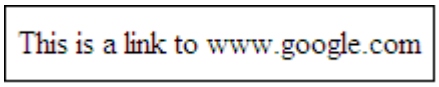


Figure 247 -- Sample of HTML Document Output (Open in Internet Explorer 7.0)

23.1.10 Table Tags

A table is defined by the <TABLE> element. A table consists of multi-dimensional data arranged in rows and columns.

23.1.10.1 Table Elements

The <TABLE> element takes a number of optional attributes to provide presentational alternatives in a document. The table attributes will be ignored in the report output except the following attributes:

(i) border - specifies the width in unit of the border of a table.

(ii) bgcolor - specifies table background color. Apply the background color here will affect the whole table.

Table elements are not supported in XLSX, PPTX, ODS, and ODP templates.

Figure 248 shows an example of table tags:

```
<TABLE border="1" bgcolor="Silver">
    <TR>
        <TH>Abbreviation</TH>
        <TH>Long Form</TH>
    </TR>
    <TR>
        <TD>AFAIK</TD>
        <TD>As Far As I Know</TD>
    </TR>
    <TR>
        <TD>IMHO</TD>
        <TD>In My Humble Opinion</TD>
    </TR>
    <TR>
        <TD>OTOH</TD>
        <TD>On The Other Hand</TD>
    </TR>
</TABLE>
```

Figure 248 -- Sample of Table Tags

As shown in Figure 249 and 250 respectively, the outputs in RTF / ODF and HTML will be as follows:

Abbreviation	Long Form
AFAIK	As Far As I Know
IMHO	In My Humble Opinion
OTOH	On The Other Hand

Figure 249 -- Sample of RTF Document Output

Abbreviation	Long Form
AFAIK	As Far As I Know
IMHO	In My Humble Opinion
OTOH	On The Other Hand

Figure 250 -- Sample of HTML Document Output (Open in Internet Explorer 7.0)

(i) Border

Border width in HTML is specified in pixels. When the table attributes are converted into RTF, ODF, or OOXML, 1 pixel will be equal to 1 pt.

(ii) Color

The attribute value type "bgcolor" refers to color definitions as specified in [SRGB]. A color value may be either a hexadecimal number (prefixed by a hash mark) or one of the following sixteen colors. Colors are case-insensitive.

Table 34 -- Table Element Colors

Color	Hexadecimal Code
Black	#00000000
Silver	#C0C0C0
Gray	#808080
White	#FFFFFF
Maroon	#800000
Red	#FF0000
Purple	#800080
Fuchsia	#FF00FF
Green	#008000
Lime	#00FF00
Olive	#808000
Yellow	#FFFF00
Navy	#000080
Blue	#0000FF
Teal	#008080
Aqua	#00FFFF

23.1.10.2 Row Elements

The <TR> elements act as a container for a row of table cells. The <TR> elements must be contained within <TABLE>.

<TR> contains <TH> or <TD> elements, which in turn contain the actual data of the table. <TR> takes presentational attributes for specifying the alignment of cells within the row and the row's background color. The row attributes will be ignored in the report output except for the following attributes:

(i) align - Specifies the horizontal alignment for each cell in a row.

(ii) valign - Specifies the vertical position of a cell's content.

(iii) bgcolor - Specifies the table background color. A background color will apply to rows only (See Color in 23.1.10.1 Table Elements for more details).

Row elements are not supported in XLSX, PPTX, ODS, and ODP templates.

(i) Align

This attribute specifies the alignment of data and the justification of text in a cell. Possible values are:

- left - Left-flushed data/Left-justified text. This is the default value for table data.
- center - Centered data/Center-justified text. This is the default value for table headers.
- right - Right-flushed data/Right-justified text.
- justify - Double -justified text.
- char - No text alignment set.

(ii) Valign

This attribute specifies the vertical position of data within a cell. Possible values are:

- top: Cell data is flushed with the top of a cell.
- middle: Cell data is centered vertically within a cell. This is the default value.
- bottom: Cell data is flushed with the bottom of a cell.
- baseline: No text alignment set.

Figure 251 shows an exampleof TR tags:

```
<TABLE border="1">
    <TR align="center">
        <TD>This is center tr</TD>
        <TD>This center tr</TD>
    </TR>
    <TR bgcolor="Gray">
        <TD>This is gray tr</TD>
        <TD>This gray tr</TD>
    </TR>
    <TR valign="bottom">
        <TD>This is bottom tr<br>This is bottom tr</TD>
        <TD>This bottom tr</TD>
    </TR>
    <TR>
        <TD>This is normal tr, This is normal tr</TD>
        <TD>This is normal tr, This is normal tr</TD>
    </TR>
</TABLE>
```

Figure 251 -- Sample of TR Tags

As shown in Figure 252 and 253 respectively, the outputs in RTF / ODF and HTML will be as follows:

This is center tr	This center tr
This is gray tr	This gray tr
This is bottom tr This is bottom tr	This bottom tr
This is normal tr, This is normal tr	This is normal tr, This is normal tr

Figure 252 -- Sample of RTF Document Output

This is center tr	This center tr
This is gray tr	This gray tr
This is bottom tr This is bottom tr	This bottom tr
This is normal tr, This is normal tr	This is normal tr, This is normal tr

Figure 253 -- Sample of HTML Document Output (Open in Internet Explorer 7.0)

23.1.10.3 Cell Elements

The <TD> elements define a data cell in a table. <TD> elements are contained within a <TR> element (a table row). The cell attributes will be ignored in the report output except for the following attributes:

- align - specifies the horizontal alignment for each cell in the row. (See Align in 23.1.10.2 Row Elements for details)
- valign - specifies the vertical position of a cell's contents. (See Valign in 23.1.10.2 Row Elements for details)
- bgcolor - specifies the table background color. A background color will apply only to cells. (See Color in 23.1.10.1 Table Elements for details)
- rowspan - rows spanned by the cell
- colspan - columns spanned by the cell

Cell elements are not supported in XLSX, PPTX, ODS, and ODP templates.

Row Span

This attribute specifies the number of rows spanned by the current cell. The default value of this attribute is one ("1"). For an RTF output, the result of row span (*.rtf) is readable only in MS Word and Word on Mac.

Column Span

This attribute specifies the number of columns spanned by the current cell. The default value of this attribute is one ("1"). Figure 254 shows an example of a column span:

```
<TABLE border="1">
    <TR>
        <TD align="right">This is align right<br>This is align right</TD>
        <TD align="center">This is align center</TD>
        <TD valign="top">This is valign top</TD>
        <TD valign="bottom">This is valign bottom</TD>
        <TD>This is normal td</TD>
    </TR>
    <TR>
        <TD>This is normal td</TD>
        <TD colspan="2">This is colspan</TD>
        <TD rowspan="2">This is rowspan</TD>
        <TD>This is normal td</TD>
    </TR>
    <TR>
        <TD bgcolor="red">This is red td</TD>
        <TD>This is normal td, This is normal td</TD>
        <TD>This is normal td, This is normal td</TD>
        <TD>This is normal td</TD>
    </TR>
</TABLE>
```

Figure 254 -- Sample of TD Tags As Column Spans

As shown in Figure 255 and 256 respectively, the outputs in RTF / ODF and HTML will be as follows:

This is align right This is align right	This is align center	This is valign top	This is valign bottom	This is normal td
This is normal td	This is colspan		This is rowspan	This is normal td
This is red td	This is normal td, This is normal td	This is normal td, This is normal td		This is normal td

Figure 255 -- Sample of RTF / ODF Document Output

This is align right This is align right	This is align center	This is valign top	This is valign bottom	This is normal td
This is normal td	This is colspan		This is rowspan	This is normal td
This is red td	This is normal td, This is normal td	This is normal td, This is normal td		This is normal td

Figure 256 -- Sample of HTML Document Output (Open in Internet Explorer 7.0)

23.1.10.4 Header Elements

The <TH> elements define a header cell in a table. <TH> elements are contained within a <TR> element (a table row). The header attributes will be ignored in the report output, except for the following attributes:

- align - specifies the horizontal alignment for each cell in the row. (See Align in 23.1.10.2 Row Elements for details)
- valign - specifies the vertical position of a cell's contents. (See Valign in 23.1.10.2 Row Elements for details)
- bgcolor - specifies the table background color. A background color will apply to the whole table. (See Color in 23.1.10.1 Table Elements for details)
- rowspan - rows spanned by the cell. (See Row span in 23.1.10.3 Cell Elements for details)
- colspan - columns spanned by the cell. (See Column span in 23.1.10.3 Cell Elements for details)

The default alignment for <TH> is center and the default font style for <TH> is bold. Header elements are not supported in XLSX, PPTX, ODS, and ODP templates. Figure 257 shows an example of header elements:

```
<TABLE border="1" bgcolor="Silver">
    <TR>
        <TH>Name</TH>
        <TH>Cups</TH>
        <TH>Type of Coffee</TH>
        <TH>Sugar?</TH>
    </TR>
    <TR>
        <TD>T. Sexton</TD>
        <TD>10</TD>
        <TD>Espresso</TD>
        <TD>No</TD>
    </TR>
    <TR>
        <TD>J. Dinnen</TD>
        <TD>5</TD>
        <TD>Decaf</TD>
        <TD>Yes</TD>
    </TR>
</TABLE>
```

Figure 257 -- Sample of TH Tags As Header Elements

As shown in Figure 258 and 259 respectively, the outputs in RTF / ODF and HTML will be as follows:

Name	Cups	Type of Coffee	Sugar?
T. Sexton	10	Espresso	No
J. Dinnen	5	Decaf	Yes

Figure 258 -- Sample of RTF / ODF Document Output

Name	Cups	Type of Coffee	Sugar?
T. Sexton	10	Espresso	No
J. Dinnen	5	Decaf	Yes

Figure 259 -- Sample of HTML Document Output (Open in Internet Explorer 7.0)

23.1.11 Image Tags

The <img> tag embeds an image in a document. The <img> tag consists of three attributes: (23.1.11.1) src, (23.1.11.2) width, and (23.1.11.3) height

23.1.11.1 src

The src attribute specifies the location of an image resource. The value of this attribute can be one of the following types:

- A URL. The recognized scheme types are HTTP, HTTPS, and FILE
- An absolute path such as c:/user/image.png. The path separator can be either /(slash) or \(blackslash)

An output image format will depend on the format of the image source.

23.1.11.2 Width

The width attribute specifies the width of an image in pixel units. For example, width = “100” or width = “100px”

23.1.11.3 Height

The height attribute specifies the height of an image in pixel units. For example, height = “100”, or height = “100px”

NOTE	If the width or height attribute of an image is not specified, the size of the image will be calculated according to the following rules: • For an image file that contains the width and height properties such as JPG, PNG, and GIF, the size of the image output will be calculated from the size of the image. • For an image file that has no width and height properties such as SVG, EMF, and WMF, the size of the image output will be calculated from the size of the paper.

In the event that the HTML code is as follows:

```
<img src="file:/D:/Waterlilies.jpg" width="200" height="200"/>
```

The image will be as shown in Figure 260:

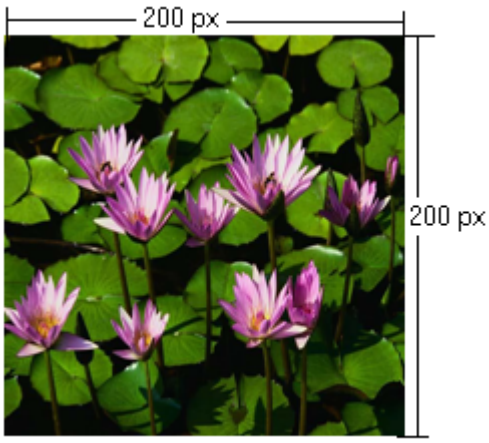


Figure 260 -- Sample of IMG Tag

References

ISO/IEC 26300:2006 Information Technology - Open Document Format for Office Applications (OpenDocument) v1.0.

- http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=43485

Application Supports for the OpenDocument Format

- http://opendocumentfellowship.com/applications

Microsoft expands List of Formats Supported in Microsoft Office

- http://www.microsoft.com/Presspass/press/2008/may08/05-21ExpandedFormatsPR.mspx

OpenOffice.org

- http://www.openoffice.org/

Microsoft Office ODF Extensions

- http://odf-converter.sourceforge.net/

23.2 Supported CSS

23.2.1 Background

The shorthand property for the background property is 'background', which is used for setting an individual background style. Only the ‘background-color’ attribute is supported. Such attribute can be used in RTF, ODT, and DOCX report templates.

23.2.1.1 Color Specification

Color attributes such as background-color, can be specified by either a keyword or a numerical RGB specification. The list of color keywords is displayed in Table 35.

Table 35 -- Color Keywords and Corresponding RGB Specification

Color	RGB Specification - Hexadecimal Code
Maroon	#800000
Red	#ff0000
Orange	#ffA500
Yellow	#ffff00
Olive	#808000
Purple	#800080
Fuchsia	#ff00ff
White	#ffffff
Lime	#00ff00
Green	#008000
Navy	#000080
Blue	#0000ff
Aqua	#00ffff
Teal	#008080
Black	#000000
Silver	#c0c0c0
Gray	#808080

The format of an RGB specification value in form of the hexadecimal notation is a '#' immediately followed by either three or six hexadecimal characters. The three-digit RGB notation (#rgb) is converted to the six-digit form (#rrggbb) by replicating digits. For example, #fb0 is converted to #ffbb00.

For example:

```
<p style = “background-color:#fb0”>
<p style = “background-color:#ffbb00”>
```

23.2.1.2 Supported Tags

The ‘background-color’ property can be used in the following tags: `b, i, u, h1, h2, h3, h4, h5, h6, tt, code, samp, kbd, pre, big, small, strike, s, em, cite, dfn, var, strong, font, a, dl, dt, dd, ul, ol, li, table, tr, td, th, p, div,` and `span`.

For example:

```
background color : <span style="background-color:red;">red </span>
<span style="background-color:#008000;">green </span>
<span style="background-color:blue;">blue </span>
<span style="background-color:#FF8040;">orange </span>
<span style="background-color:yellow;">yellow </span>
```

Figure 261 -- Sample of Background Style

Figure 262 -- Sample of Document Output (Resulted from Figure 261)

23.2.2 Border

The shorthand property for the border property is 'border', which is used for setting the same width, color, and style of all four borders on four sides of a box. Border can be used in RTF, ODT, and DOCX report templates.

23.2.2.1 Border Width

Border width properties specify the width of border areas. The value of the border property is length. The border width properties that can be used with a longhand property are as follows:

- `'border-top-width'`
- `'border-right-width'`
- `'border-bottom-width'`
- `'border-left-width'`

23.2.2.2 Length Specification

The format of a length value is a real number immediately followed by a unit identifier. There are two types of length units: *relative* and *absolute*. Relative length units specify a length relative to another length property of the displaying device.

Relative units are:

- **px**: pixels, relative to the displaying device.

Absolute units are:

- **in**: inches - 1 inch is equal to 2.54 centimeters.
- **cm**: centimeters.
- **mm**: millimeters.
- **pt**: points - the points used by CSS 2.1 are equal to 1/72nd of an inch.
- **pc**: picas - 1 pica is equal to 12 points.

23.2.2.3 Border Color

Border color properties specify the border colors of a box. The value of the border color has two meanings:

1. **Color**: Specifies a border color. See 23.2.1.1 Color Specification on page 270 for more details on color.
2. **Transparent**: Referred to as a transparent border.

The border color properties that can be used with a longhand property are as follows:

- `'border-top-color'`
- `'border-right-color'`
- `'border-bottom-color'`
- `'border-left-color'`

23.2.2.4 Border Style

The Border style properties specify the style of a box's border lines. The border style may take one of the following 10 values:

1. **none**: This value will create no border because the computed border width is zero.
2. **hidden**: The same as 'none'.
3. **dotted**: This value will create a dotted border.
4. **dashed**: This value will create a dashed border.
5. **solid**: This value will create a single line segment border.
6. **double**: This value will create a two solid line border.
7. **groove**: This value will create a border that looks as though it were carved into the rendering surface.
8. **ridge**: The opposite of 'groove'. This value will create a border that looks as though it were coming out of the rendering surface.
9. **inset**: This value will create a border that looks as though it were embedded in the rendering surface.
10. **outset**: The opposite of 'inset'. This value will create a border that looks as though it were coming out of the rendering surface.

NOTE	
	• The supported border values for ODF documents are **none**, **hidden**, **solid**, and **double**. • The supported border values for RTF documents are **none**, **hidden**,**dotted**, **dashed**, **solid**, and **double**. • The supported border values for DOCX documents are **none**, **hidden**,**dotted**, **dashed**, **solid**, **double**, **outset**, and **inset** (**groove** and **ridge** will be rendered as **solid**).

The border style properties that can be used with a longhand property are as follows:

- `'border-top-style'`
- `'border-right-style'`
- `'border-bottom-style'`
- `'border-left-style'`

23.2.2.5 Supported Tags

Border properties can be used in the following tags: `h1, h2, h3, h4, h5, h6, td, th, p,` and `div`.

For example, if you want to have a red border, type the following line of code (Figure 263):

```
<p style="border:2px solid red;">This is 2px solid red border.</p>
```

Figure 263 -- Border Style Example

The result of using the above example will be as shown in Figure 264.

Figure 264 -- Red Border Output

23.2.3 Margin

Margin properties specify the width of margin area within a box. The 'margin' property is a shorthand property for setting the margin of all four sides while the other margin properties only set their respective side. The value of the margin property is length (See 23.2.2.2 Length Specification on page 272 for more details). Margin can be used in RTF, ODT, ODP, DOCX and PPTX report templates.

The margin properties that can be used with a longhand property are as follows:

- `'margin-top'`
- `'margin-right'`
- `'margin-bottom'`
- `'margin-left'`

23.2.3.1 Supported Tags

The supported tags for margins are: `h1, h2, h3, h4, h5, h6, table` (supported only in RTF, ODT, and DOCX), `p`, and `div`.

For example, if you want to set a specific margin width for your cells, type the lines of code as shown in Figure 265.

```
<p>This is first paragraph.</p>

<table width="100%" cellpadding="0" style="margin-top:30px;margin-right:30px;
margin-left:30px;margin-bottom:30px;" cellspacing="0" border="1">
    <tr>
      <td>This is cell1</td>
      <td>This is cell2</td>
    </tr>
    <tr>
      <td>This is cell3</td>
      <td>This is cell4</td>
    </tr>
 </table>

 <p>This is second paragraph.</p>

 <table width="100%" cellpadding="0" style="margin-top:50px;margin-right:50px;
 margin-left:50px;margin-bottom:50px;" cellspacing="0" border="1">
    <tr>
      <td>This is cell1</td>
      <td>This is cell2</td>
    </tr>
    <tr>
      <td>This is cell3</td>
      <td>This is cell4</td>
    </tr>
 </table>

  <p>This is third paragraph.</p>
```

Figure 265 -- Margin Style Example

The result of using the above example will be as indicated in Figure 266.

This is first paragraph.

This is cell1	This is cell2
This is cell3	This is cell4

This is second paragraph.

This is cell1	This is cell2
This is cell3	This is cell4

This is third paragraph.

Figure 266 -- Margin Style Outputs

23.2.4 Padding

Padding properties specify the width of padding area within a box. The 'padding' property is a shorthand property for setting the padding for all four sides while the other padding properties only set their respective side. The value of the padding property is length (See 23.2.2.2 Length Specification on page 272 for more details)adding can be used in RTF, ODT and DOCX report templates.

The padding properties that can be used with a longhand property are as follows:

- 'padding-top'
- 'padding-right'
- 'padding-bottom'
- 'padding-left'

23.2.4.1 Supported Tags

The supported tags for padding are: h1, h2, h3, h4, h5, h6, td, th, p, and div.

NOTE	• The supported border values of table tags for RTF documents are margin-left and margin-right. • The supported border values of table tags for DOCX documents are margin-left and margin-right. • The supported border values of all tags for PPTX documents are margin-left, margin-top and margin-bottom.

For example, if you want to create specific padding styles for your cells, type the lines of code shown in Figure 267.

```
<table width="100%"  border="1">
     <tr>
       <td  style="padding-bottom:20px;padding-top:20px;padding-left:20px;padding-right:20px;">This is cell1</td>
       <td>This is cell2</td>
    </tr>
    <tr>
       <td>This is cell3</td>
       <td>This is cell4</td>
    </tr>
</table>
```

Figure 267 -- Padding Style Example

The result of using the above example will be as shown in Figure 268.

This is cell1	This is cell2
This is cell3	This is cell4

Figure 268 -- The Padding Style Outputs

23.2.5 Color

The 'color' property specifies the foreground color of an element's text contents. Only color values are allowed in this property (See 23.2.1.1 Color Specification on page 270 for more details on color).

23.2.5.1 Supported Tags

The supported tags for colors are: `b, i, u, h1, h2, h3, h4, h5, h6, tt, code, samp, kbd, pre, big, small, strike, s, em, cite, dfn, var, strong, font, a, dl, dt, dd, ul, ol, li, table, tr, td, th, p, div,` and `span`.

For example, if you want to have a green text color, type the lines of code indicated in Figure 269.

```
<p style="color:green;">This text color is green.</p>
<span style="color:#800080;">This text color is purple. </span>
<span style="color:red;">This text color is red.</span>
```

Figure 269 -- Color Style Example

The result of using the above example will be as shown in Figure 270.

This text color is green.

This text color is purple. This text color is red.

Figure 270 -- Color Style Outputs

23.2.6 Display

The 'display' property specifies how text will be displayed. The possible display values are:

1. **block**: This value will make an element appear in the output report.
2. **none**: This value will make an element disappear from the output report.

23.2.6.1 Supported Tags

The supported tags for the display property are: `br, b, i, u, h1, h2, h3, h4, h5, h6, tt, code, samp, kbd, pre, big, small, strike, s, em, cite, dfn, var, strong, font, a, dl, dt, dd, ul, ol, li` (supported only in RTF, DOCX, and PPTX), `table, tr, p, div,` and `span`.

For example, if you want to display an element but do not want to display the other(s), type the lines of code as shown in Figure 271.

```
<p style="display:none;">This paragraph is none display.</p>
<p style="display:block;">This paragraph is block display.</p>
```

Figure 271 -- Display Style Example

The result of using the above example will be as indicated in Figure 272.

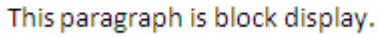


Figure 272 -- Display Style Output

23.2.7 Font

The 'font' property is a shorthand property for setting, except the following:

- `'font family'`
- `'font-style'`
- `'font-variant'`
- `'font-weight'`
- `'font-size'`

23.2.7.1 Font Family

Specify a prioritized list of font family names and/or generic family names.

23.2.7.2 Font Style

Specify either normal or italic face (within the specified font family).

23.2.7.3 Font Variant

Specify either normal or small-caps of variant (within the specified font family). Font variant is not supported in XLSX report templates.

23.2.7.4 Font Weight

Specifies the weight of the font. The following values are defined:

1. **normal**, **lighter**, **100**, **200**, **300**, and **400** will be rendered as ‘normal’.
2. **bold**, **bolder**, **500**, **600**, **700**, **800**, and **900** will be rendered as ‘bold’.

23.2.7.5 Font size

Specify the font size. Possible values include **xx-small**, **x-small**, **small**, **medium**, **large**, **xx-large**, and the integer number from **1** to **7**.

23.2.7.6 Supported Tags

The supported tags for the font property are: `b, i, u, h1, h2, h3, h4, h5, h6, tt, code, samp, kbd, pre, big, small, strike, s, em, cite, dfn, var, strong, font, a, dl, dt, dd, ul, ol, li, table, tr, td, th, p, div,` and `span`.

For example, if you want to have bold fonts in small caps, type the lines of code as displayed in Figure 273.

```
<font style="font-variant:small-caps;font-weight:bold;">
This font is bold and small-caps</font>
```

Figure 273 -- Font Style Example

The result of using the above example will be as indicated in Figure 274.

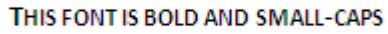


Figure 274 -- Font Style Output

23.2.8 Text Align

The 'text align' property describes how inline content of a block is aligned. Possible values of the text align property include: **left**, **right**, **center** and **justify**. The ‘text align’ property can be used in RTF, ODT, ODP, DOCX, and PPTX report templates.

23.2.8.1 Supported Tags

The supported tags for the text align property are: `table, tr, td, th, p,` and `div`.

For example, if you want to align your paragraph to the left, center, and right, type the lines of code as shown in Figure 275.

```
<p style="text-align:right;">This paragraph alignment is right.</p>
<p style="text-align:left;">This paragraph alignment is left.</p>
<p style="text-align:center;">This paragraph alignment is center.</p>
```

Figure 275 -- Text-align Style Example

The result of using the above example will be as displayed in Figure 276.

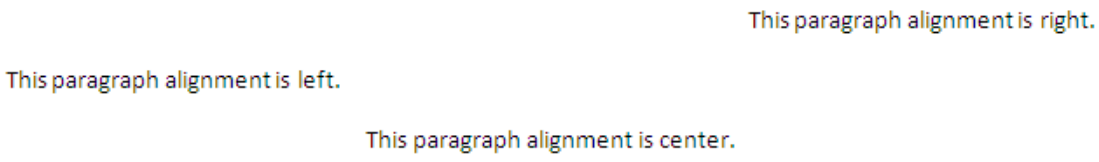


Figure 276 -- Text-align Outputs

23.2.9 Text Transform

The 'text transform' property controls capitalization effects of an element's text. Possible values of the 'text transform' property include:

1. **capitalize**: This will begin a word's first character in an uppercase letter. Other characters will not be affected.

2. **uppercase**: This will put all characters of each word in uppercase.
3. **lowercase**: This will put all characters of each word in lowercase.
4. **none**: This will not create any capitalization effects.

23.2.9.1 Supported Tags

The supported tags for the text transform property are: `b, i, u, h1, h2, h3, h4, h5, h6, tt, code, samp, kbd, pre, big, small, strike, s, em, cite, dfn, var, strong, font, a, dl, dt, dd, ul, ol, li, table, tr, td, th, p, div,` and `span`.

For example, if you want to make all letters in a paragraph become uppercases, type the line of code displayed in Figure 277.

```
<p style="text-transform:uppercase;">This is paragraph uppercase.</p>
```

Figure 277 -- Text-transform Style Example

The result of using the above example will be as shown in Figure 278.

THIS IS PARAGRAPH UPPERCASE.

Figure 278 -- Text-transform Style Output

23.2.10 White-space

The 'white-space' property affects the vertical position of white-space inside a lined box. White-space can be used in RTF, ODT, DOCX and PPTX report templates. Possible values of the 'white-space' property include:

1. **normal**: To collapse sequences of white space and break lines as necessary to fill line boxes.
2. **pre**: To prevent from collapsing sequences of white space. Lines are only broken at newlines in the source or at occurrences of "\A" in the generated content. Report Wizard will render this value as the **<pre>** tag.
3. **nowrap**: To collapse white space as for 'normal' but suppressing line breaks within text.
4. **pre-wrap**: To prevent user agents from collapsing sequences of white space. Lines are broken at newlines in the source, at occurrences of "\A" in the generated content, and as necessary to fill line boxes. This value will not be rendered as **pre**.
5. **pre-line**: To direct user agents to collapse sequences of white space. Lines are broken at newlines in the source, at occurrences of "\A" in the generated content, and as necessary to fill line boxes. This value will not be rendered as **pre**.

23.2.10.1 Supported Tags

The supported tags for the 'white-space' property are: `b, i, u, h1, h2, h3, h4, h5, h6, tt, code, samp, kbd, pre, big, small, strike, s, em, cite, dfn, var, strong, font, a, table, tr, td, th, p, div,` and `span`.

For example, if you want to preserve some white-space, type these lines of code as shown in Figure 279.

```
<p style="white-space:pre;">
 This    white space        is pre.
    This is        white space        pre.
</p>
```

Figure 279 -- White-space Style Example

The result of using the above example will be as indicated in Figure 280.

This white space is pre.

This is white space pre.

Figure 280 -- White-space Style Outputs

23.2.11 Width

The 'width' property specifies the contents width of each box generated by a block-level. The value of the width property is length (See 23.2.2.2 Length Specification on page 272 for more details). Width can be used in RTF, ODT, and DOCX report templates.

The supported tag for the width property includes `table`.

For example, if you want to have a specific table width, type the lines of code as described in Figure 281.

```
<table border=1 cellpadding="0" style="width:300px;" cellspacing="0">
  <tr>
     <td>This is cell1</td>
     <td>This is cell2</td>
  </tr>
  <tr>
     <td>This is cell3</td>
     <td>This is cell4</td>
  </tr>
</table>
<br>
<table style="width:6in;" border="1" cellpadding="0" cellspacing="0">
  <tr>
     <td>This is cell1</td>
     <td>This is cell2</td>
  </tr>
  <tr>
     <td>This is cell3</td>
     <td>This is cell4</td>
  </tr>
</table>
```

Figure 281 -- Width Style Example

The result of using the above example will be as shown in Figure 282.

This is cell1	This is cell2
This is cell3	This is cell4

This is cell1	This is cell2
This is cell3	This is cell4

Figure 282 -- Width Style Outputs

23.2.12 Text Decoration

The 'text decoration' property describes decorations that are added to text. Possible values of the text decoration property include:

1. **none**: To produce no text decoration.
2. **underline**: To underline each line of text.
3. **overline**: To add a line above each line of text.
4. **line-through**: To add a line that strikes through each line of text.
5. **blink**: To add the blinking effect to text (being visible and invisible alternatively). The blink text decoration supports only the ODT report template.

NOTE	Report Wizard does not currently support the **overline** text decoration.

23.2.12.1 Supported Tags

The supported tags for the text decoration property are: `b, i, u, h1, h2, h3, h4, h5, h6, tt, code, samp, kbd, pre, big, small, strike, s, em, cite, dfn, var, strong, font, a, dl, dt, dd, ul, ol, li, table, tr, td, th, p, div,` and `span`.

For example, if you want to create a line-through text, type the line of code as displayed in Figure 283.

```
This word is <span style="text-decoration:line-through;">line through.</span>
```

Figure 283 -- Text-decoration Style Example

The result of using the above example will be as indicated in Figure 284.

This word is ~~line through.~~

Figure 284 -- Text-decoration Style Output

23.2.13 Vertical Align

The 'vertical align' property affects the vertical position of an element inside a lined box. This property can be used in RTF, ODT and DOCX report templates. Possible values of the vertical align property include:

1. **baseline**: This value will align the baseline of a box with the baseline of the parent box. Leave the vertical alignment to be default when you encounter this value.
2. **middle**: This value will align the vertical midpoint of a box.
3. **top**: This value will align the top of the aligned sub-tree with the top of a line box.
4. **bottom**: This value will align the bottom of the aligned sub-tree with the bottom of a line box.

The supported tags for the vertical align property are: `table, tr, td,` and `th`.

For example:

```
<table width="100%" cellpadding="0" style="vertical-align:middle;" cellspacing="0" border=1>
  <tr>
    <td>
     This is cell1
     This is cell1
    </td>
    <td>
     This vertical align is middle.
    </td>
 </tr>
 <tr>
    <td>
     This is cell3
     This is cell3
    </td>
    <td>
     This vertical align is middle.
    </td>
 </tr>
</table>
```

Figure 285 -- Vertical-align Style Example

The result of using the above example will be as shown in Figure 286.

This is cell1 This is cell1	This vertical align is middle.
This is cell3 This is cell3	This vertical align is middle.

Figure 286 -- Vertical-align Style Output