

IBM® Rational® Rhapsody® Gateway Add On



Customization Guide

Rhapsody[®]

**IBM[®] Rational[®] Rhapsody[®]
Gateway Add On**

Customization Guide



License Agreement

No part of this publication may be reproduced, transmitted, stored in a retrieval system, nor translated into any human or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual or otherwise, without the prior written permission of the copyright owner.

The information in this publication is subject to change without notice, and Dassault Systèmes and its affiliates assume no responsibility for any errors which may appear herein. No warranties, either expressed or implied, are made regarding Rhapsody software and its fitness for any particular purpose.

Trademarks

Reqtify is a registered trademark of Dassault Systèmes or its affiliates in the US and/or other countries.

Rhapsody Gateway, IBM, the IBM logo, DOORS and Rhapsody are trademarks or registered trademarks of IBM Corporation.

All other product or company names mentioned herein may be trademarks or registered trademarks of their respective owners.

© Copyright 2001-2011 Dassault Systèmes. All rights reserved.

Contents

Contents.....	5
How to Use the Documentation?.....	9
Documentation Overview	9
Important Product Documentation	10
Documentation Conventions	10
Capture and Analysis Process	11
Capturing Information from Interfaced Tools.....	13
Intermediate File	14
Analysis Types	15
Using Customized Analysis Types	17
Capturing Requirements.....	17
Creating a Type of Analysis.....	17
Applying a Type of Analysis	19
Customizing Multi-Types	20
Capturing Coverage Information	22
Project Definition	26
Types Editor.....	27
General Aspects Definition	27
Contextual Menu.....	29
Edit Menu.....	30
Definition of a Type	30
Definition of the Analysis Tab	33
General Aspect of Analysis Tab	33
Specific Aspect of Analysis Tab	34
Definition of the Advanced Options of a Type	34
Exclusion of Text Areas.....	36
Exclusion of an Element	36
Definition of Type Elements from Text Files	37
Defining Sections	37
Defining Macro-Requirements.....	41
Defining Requirements	44
Defining Entities	48
Defining References.....	49
Defining Attributes	56
Defining Reference Attributes	58
Defining Links	60
Defining Texts.....	62

Defining Pictures.....	64
Defining Tables.....	64
Definition of Type Elements from XML Files.....	65
Creating a New XML Type.....	66
Creating the XML Structure.....	67
XML Structure Options	70
Creating Types for Added Elements	71
Declaration of Requirements to Be Modified	72
Creating References	73
Creating Links	74
Creating Attributes	75
Frequently Used Regular Expressions	76
Wildcard Meaning.....	76
Matching Explicit Strings	76
Matching Character Sets	76
Evaluating Occurrences	77
Specifying Location	77
Capturing Sub-expressions	77
Specifying Alternatives.....	77
Identifying Separators	77
Assertions.....	78
Matching Special Characters.....	79
Excluding Matching of Some Characters	79
Matching Information from a Table.....	79
Capturing Multi-lines	80
Examples of ? (question mark) Usage	80
Frequently Used Regular Expressions	82
Testing Regular Expressions	84
Accessing the Regular Expression Tester	84
Using the Regular Expression Tester	85
Importing a File.....	85
Importing a Directory.....	86
Importing an Intermediate File.....	87
Testing Regular Expressions.....	87
Updating Regular Expressions.....	89
Testing XML Syntaxes.....	90
Accessing the XML Tester.....	90
Using the XML Tester	91
Importing a File	91
Importing a Directory.....	92
Importing an Intermediate File.....	93
Testing XML Syntax.....	93
Updating XML Syntax.....	94
Customizing Reports.....	95
Accessing the Report Editor.....	95
Report Editor Window	96
Report List Area.....	96

Graphical Area	97
Report Structure Configuration Area	98
Report Elements	99
Report Editor Toolbar	99
Creating Reports	100
Creating a New Report	100
Duplicating an Existing Report	100
Editing a Report	101
Parameters Usage	101
Creating a Report Structure	103
Inserting Data in a Report	114
Customizing a Report	118
Templates	120
Template Description	120
Generation Options	122
Generating Reports	122
Reports Creation	124
Short Report Example	124
Defining my First Report	125
OTScript Language Essentials	134
OTScript Language Presentation	134
OTScript General Syntax	134
Reserved Words	135
Predefined Variables	135
Operators	135
Testing Belonging to a Class	136
Testing Belonging to a List	136
Testing Emptiness of a List	136
Testing String Matching	137
Control Structures	137
Frequently Used Statements	137
Writing OTScript in Rhapsody Gateway	138
Creating an OTScript Condition	138
Accessing the OTScript Methods	139
Frequently Used Methods	140
Example Analysis	140
Sub-part Analysis	141
Writing Condition Process	142
Go Further with OTScript	143
Management of Product Configuration	144
Re-use of Customization Work from one Project to another	144
Additional Configuration Directories	145
Defining Configuration Directories	145
Structuring Configuration Directory	145
Categories Creation	146
Document Categories	146
Cover Categories	147
Upgrades and New Releases	148

Contents

Installation: Local or Network?.....	148
Index	151

How to Use the Documentation?

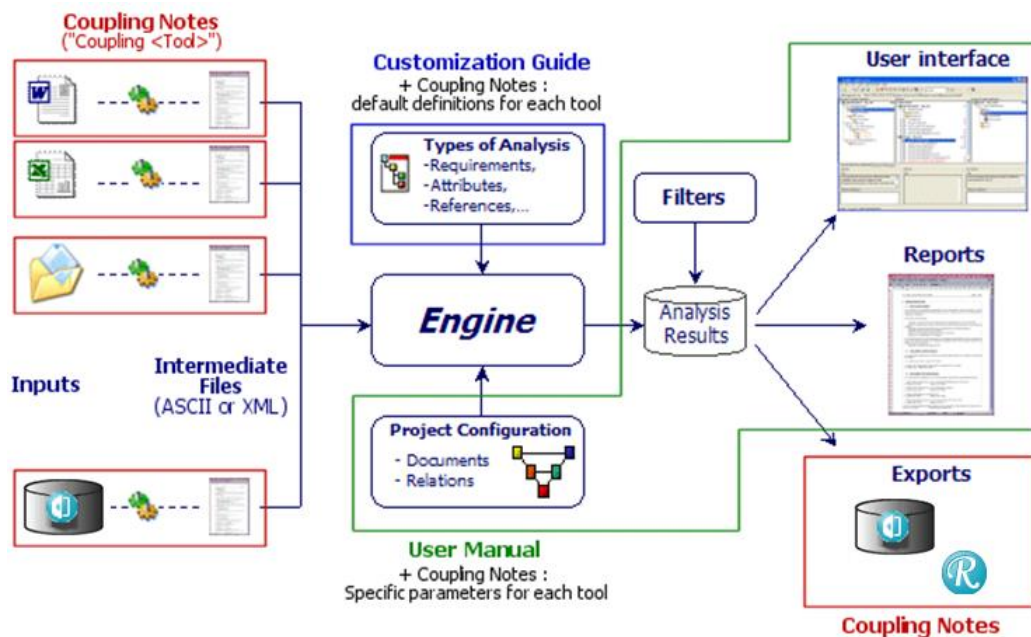
This chapter introduces the organization of the documentation.

You can read about these topics in:

- ◆ Documentation Overview
- ◆ Important Product Documentation
- ◆ Documentation Conventions

Documentation Overview

The Rhapsody Gateway documentation is organized as shown in the following figure:



There is one *Coupling Note* per tool interfaced with Rhapsody Gateway. These notes describe how Rhapsody Gateway brings the information to be analyzed to the engine, using either the third party tool API, or a dedicated converter, or any other convenient solution. This section is dedicated to administrators or users in charge of Rhapsody Gateway customization, in relation with the *Customization Guide*. The *Coupling Notes* also explain to users how Rhapsody Gateway interacts with their authoring and verification tool.

The *Customization Guide* explains how you indicate to Rhapsody Gateway the relevant information that should be picked up in the intermediate files. Such information can include requirements, attributes, coverage links, and so on. In other words, the *Customization Guide* explains how to implement your requirements standards in Rhapsody Gateway.

The *User Manual* explains how to use Rhapsody Gateway for your projects, such as how to describe your project's process, how to understand the analysis results, how to filter them and how to generate reports. Aspects directly linked to the use of authoring and verification tools are detailed in the *Coupling Note* for the concerned tool.

Important Product Documentation

Users need to read the *User Manual* and the Users part of the *Coupling Notes* for the tools they use (for example *Coupling Word*, *Coupling DOORS*, etc.)

Administrators or Users who need to implement their standards of requirements and to customize Rhapsody Gateway need to:

- ◆ Read the *Coupling Notes* for the tool used in the project or process. This will enable you to understand how the source information is converted and analyzed by the engine.
- ◆ Read the *Customization Guide* and play the Tutorial (direct links included in the *Customization Guide*, step by step).
- ◆ Read the *User Manual* for more information about requirements management aspects and display of analysis results by Rhapsody Gateway. This will enable you to understand users concerns and to properly support them.

Documentation Conventions

This document is dedicated to administrators, internal support teams, or users involved in preparing the Rhapsody Gateway customization for its application on projects for Users.

The screenshots and tutorial examples are based on Word or Excel, as these tools are often used and usually represent the highest effort in terms of customization.

Some interfaced tools that include the concept of requirements elements in their meta-model such as RequisitePro, usually do not request Rhapsody Gateway customization. Default types are already defined in compliance with these requirements models.

Note

This document provides direct links to several helpful training examples and/or animated demos.

When you have the opportunity to train yourself using an example, you will see the “hands-on” icon with direct links to relevant material for the current step.

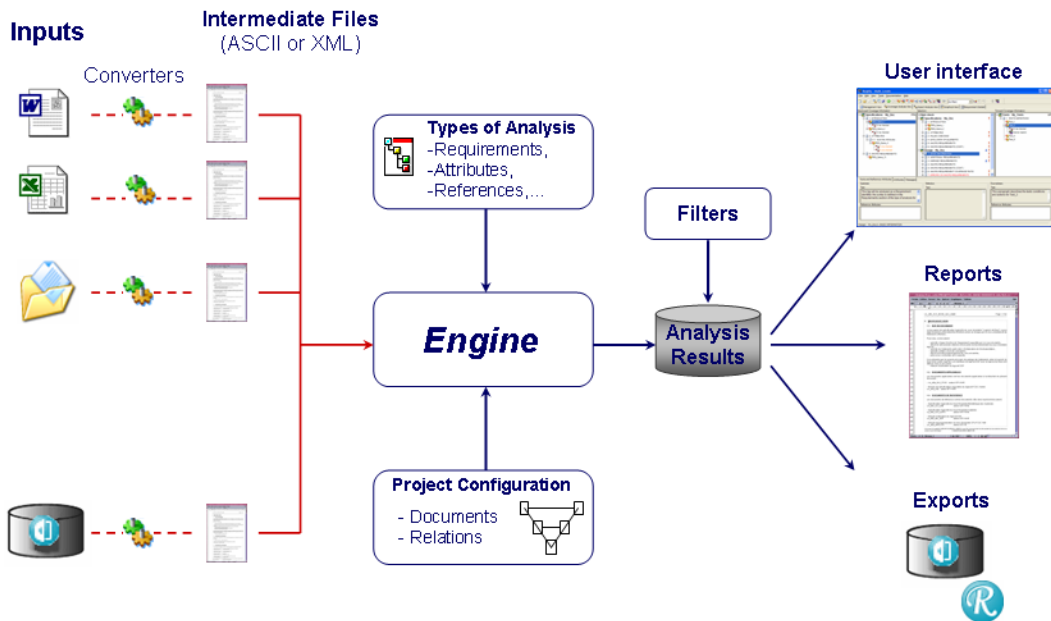
Capture and Analysis Process

In order for proper customization to take place, there needs to be a clear understanding of the analysis process.

As depicted below, several steps have to be considered:

- ◆ The converters bring the source information to the engine. These converters transform the information natively stored in the interfaced tool to an intermediate file containing the information in a format that the engine can analyze.
- ◆ The engine uses the standards of requirements definition (what is the expected format for requirements, attributes, etc.) to identify elements compliant with this definition in the intermediate file.
- ◆ The engine also uses the project definition, indicating how the intermediate files are supposed to be linked together. This allows the engine to calculate coverage ratios, check consistency rules, etc. The information is stored as “analysis results.”
- ◆ The results can be filtered to allow more oriented analysis.
- ◆ The results are displayed, produced in reports or can be exported in tools to automate parts of the requirements management process.

The Rhapsody Gateway capture and analysis is organized as shown in the following figure:



The relations between this process and Rhapsody Gateway windows usage can be summarized as follows:

- ◆ Neither the converters nor the contents of the intermediate files can be modified by users.
- ◆ The types are defined using the **Types Editor**.
- ◆ The types are applied using the **Project Editor**. The relationship between individual inputs is also defined in the **Project Editor**.
- ◆ The analysis results are displayed in the main window, containing several tabs and information areas.
- ◆ The filters are defined using the **Filters Editor**.
- ◆ The report templates are defined in the **Reports Editor** for the contents. Files created directly in the generation format are used as style sheets for their “look and feel”.
- ◆ Exports to interfaced tools are activated from the Tools menu; they can activate some dedicated additional windows described in the *Coupling Notes*.

Capturing Information from Interfaced Tools

The key points for this section are the **Intermediate file** and the **Types of Analysis**.

- ◆ The intermediate file is the output of a converter, the input being the native information stored in the interfaced tool (Word, Excel, DOORS, etc.). At this step the intermediate file is an ASCII or XML file.
- ◆ The "Types of Analysis" indicate how the information in the intermediate file is to be analyzed. Types are used to define your requirements, attributes, etc.

To illustrate, below is a simple example based on Word. The requirement descriptions are the following:

- ◆ The requirements are identified using syntax **REQ_nn**, nn being digits.
- ◆ The texts of requirements are the paragraphs with the Word style "Requirement_Text" applied.

This section describes how this information is captured by Rhapsody Gateway.

Intermediate File

The input is the following Word binary file (.doc file):

Heading1	1. <u>This is my first section</u>
Normal	
Normal	REQ_1
Requirement_Text	<i>Text of requirement 1</i>
Normal	This a general comment, or an additional description, but not part of the requirement.
Normal	
Normal	REQ_2
Requirement_Text	<i>Text of requirement 2</i>
Normal	
Heading1	2. <u>This is my second section</u>
Normal	
Normal	REQ_3
Requirement_Text	<i>Text of requirement 3</i>
Normal	
Normal	REQ_4
Requirement_Text	<i>Text of requirement 4</i>

The Word binary file is taken as input by the converter, which produces an ASCII intermediate file with the relevant information. The information is clearly equivalent to the original one, but in a format that is more readable as it is no more a binary or proprietary format:

Heading1	1. This is my first section
Normal	
Normal	REQ_1
Requirement_Text	Text of requirement 1
Normal	This a general comment, or an additional description, but not part of the requirement.
Normal	
Normal	REQ_2
Requirement_Text	Text of requirement 2
Normal	
Heading1	2. This is my second section
Normal	
Normal	REQ_3
Requirement_Text	Text of requirement 3
Normal	
Normal	REQ_4
Requirement_Text	Text of requirement 4

However, excluding human analysis and what we have as a hypothesis, we have not yet declared what is relevant in this file for our analysis.

- ◆ How can the tool know that REQ_nn has to be captured and has to be considered as requirement identifiers?
- ◆ How can the tool know that Requirement_Text paragraphs have to be captured and that they represent requirement text?

The Types of analysis have answers to these questions.

Analysis Types

The **Types of analysis** can be understood as the way to capture relevant information in the intermediate file, like you would do by highlighting a plain text file in order to indicate what is relevant for you.

Heading1	1. This is my first section
Normal	
Normal	REQ_1
Requirement_Text	Text of requirement 1
Normal	This a general comment, or an additional description, but not part of the requirement.
Normal	
Normal	REQ_2
Requirement_Text	Text of requirement 2
Normal	
Heading1	2. This is my second section
Normal	
Normal	REQ_3
Requirement_Text	Text of requirement 3
Normal	
Normal	REQ_4
Requirement_Text	Text of requirement 4
Normal	

Emphasizing is the following.

Hierarchy / Structure	Word styles “Heading XX” define the tree structure of my document
Requirement IDs	“REQ_” followed by digits are my requirements
Requirement Text	Text using the Word style “Requirement_Text” is a requirement text

Of course, you are supposed to also consider attributes, pictures / diagrams, coverage links and other links in a requirements management process.

A **Type of Analysis** is defined using the **Types Editor**. At least one default type of analysis is defined for each interfaced tool. This means that a default approach is defined

in order to capture requirements, text, and coverage links in interfaced tools. This default approach is defined in the *Coupling Note* concerning the interfaced tool.

Elements of the **Type of analysis** are presented in the *Types Editor* chapter.

Using Customized Analysis Types

This section introduces the use of customized types in a project:

- ◆ Capture requirements in a high level document.
- ◆ Capture coverage information in a low level document.
- ◆ Introduction to the concept of “Project” by explaining the need to define links between documents.

Capturing Requirements

Before going into details concerning **Types of Analysis**, it is helpful to give a first view of how a customized type is used in a project.

Take another look at the example based on the need to capture the following information:

- ◆ The requirement IDs use the syntax REQ_ followed by one or several digits.
- ◆ The requirement texts are paragraphs using the Word style “Requirement_text”.

If you read the *Coupling Word* document, you can see how the Word file is converted but also what the expected default syntax for requirements and other elements is.

For requirement texts, the default definition is the capture of paragraphs using the style “Requirement_Text”. But if the default definition for requirement identifiers does not comply with your needs, you need to create a customized type of analysis.

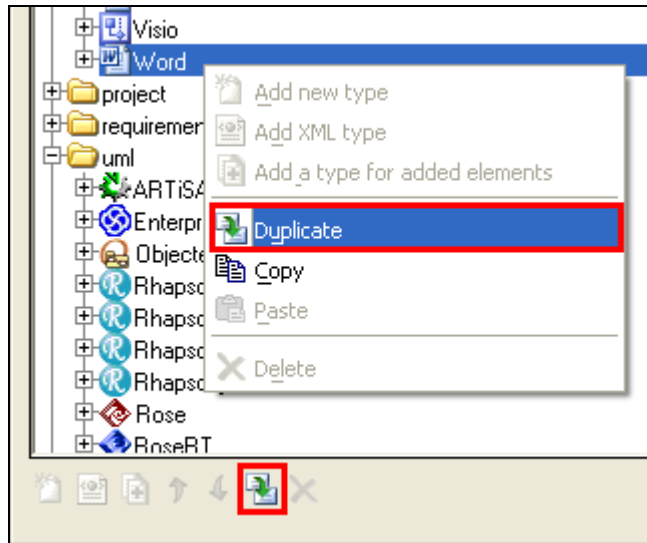
Creating a Type of Analysis

The default Word type of analysis can be fully re-used, except for requirements that have to be re-defined. Therefore there is no need to start from scratch as the default type can be re-used and modified (only for requirements).

To create your own type of analysis, follow these steps:

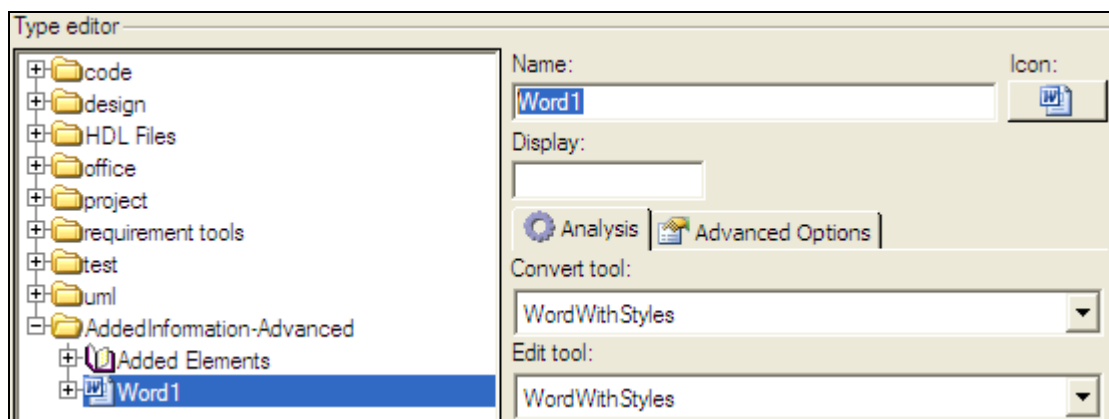
1. Open the **Types Editor** by choosing **Edit Types** from the **File** menu.
2. Expand the `office` folder, and select the **Word** type of analysis.
3. To duplicate the default Word type of analysis, you can either:

- ◆ Right click the default Word type and select **Duplicate** in the context menu.
- ◆ Click the **Duplicate** button.

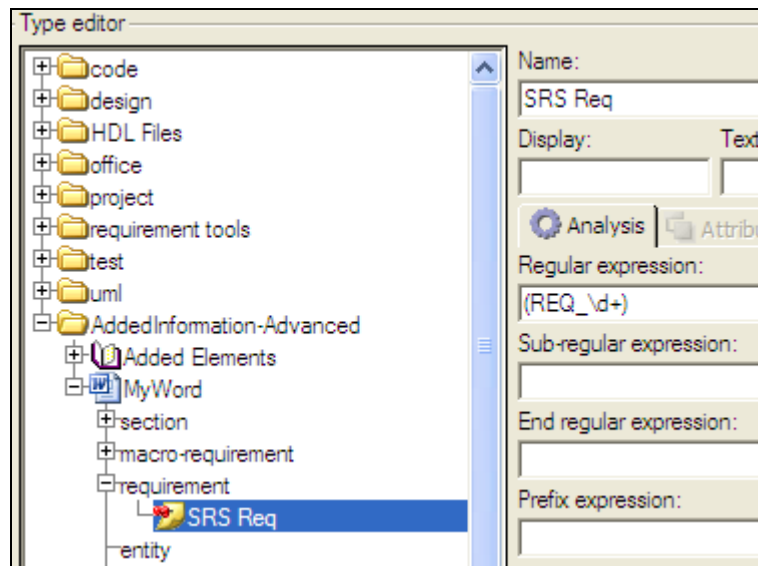


A new type of analysis is created. This type is located in a directory (automatically created) with the same name as your project.

4. You can rename this type using the **Name** field.



5. To define your requirements, expand the tree and select the **Requirement** element.



The **Regular expression** field is used to define the requirement IDs to be captured in the intermediate file.

Note

A summary of helpful syntaxes for these expressions is provided in this document. The Support Team and Application Engineers can also help you for advanced definitions.

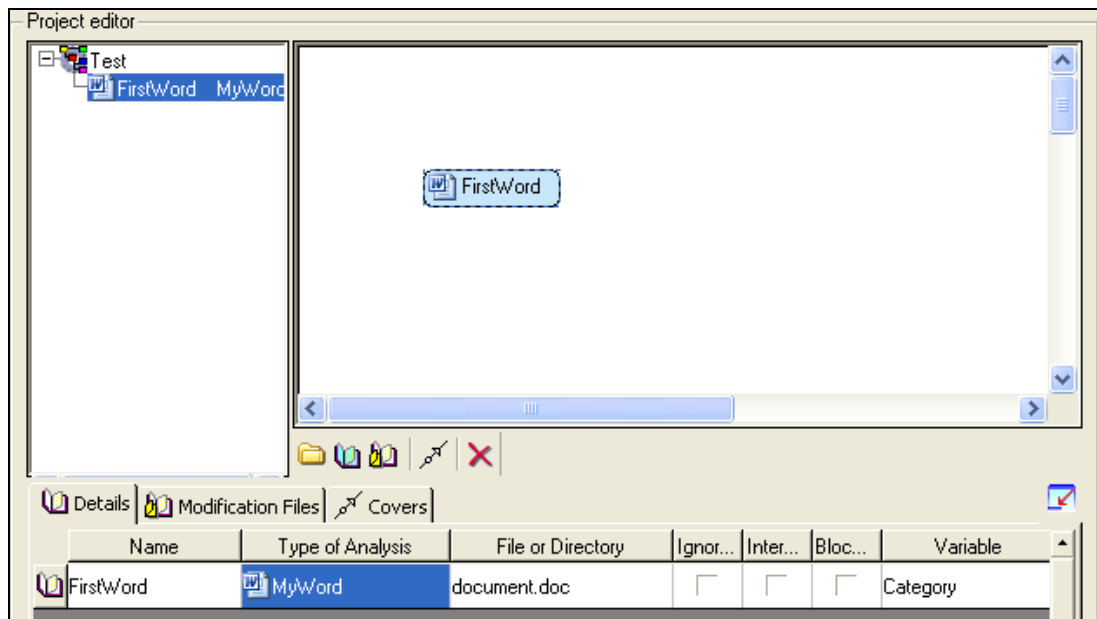
In our case, the expression is very simple: `(REQ_\d+)`

- (...) Brackets define the limits of the captured string
- `REQ_` is the fixed prefix for requirement IDs defined in our example
- `\d+` means “one or several digits”

Applying a Type of Analysis

Rhapsody Gateway is now able to analyze a document containing your requirements if they comply with the syntax defined for this example.

1. To use this new type, you simply have to include a document in the **Project Editor**, name it, and apply the customized type and to select the input Word file.



2. The new type is applied to the document and the analysis has to be executed another time.

Applying customized types to a document is as simple as applying default types to a document because customized types are available in the same drop-down list and can be selected.

For additional information and better understanding, see also:

- ◆ The [FirstRequirements](#) project, a ready to use example demonstrating these concepts.
- ◆ The [First Requirements animated demo](#), which presents the material that is explained in this section.

Customizing Multi-Types

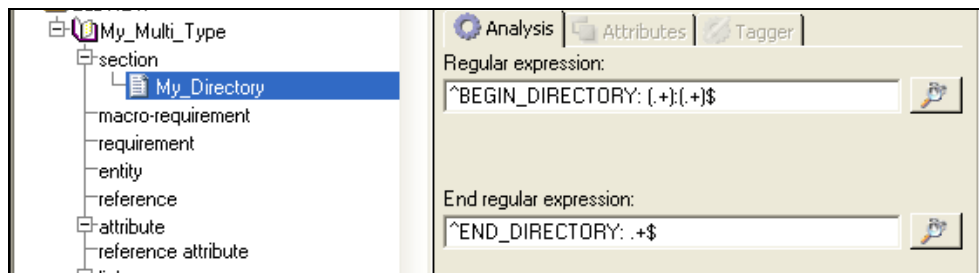
Multi Types are types which enable the analysis of multiple documents of the same type at the same time. All these documents need to be gathered in a directory or in subdirectories of this directory. The structure of individual documents is displayed as if each document is analyzed as a standalone document.

In Rhapsody Gateway, only one multi-type is delivered: **Multi Word**. Nevertheless you can create your own multi-types. The *Word Coupling Notes* document presents the **Multi Word** type.

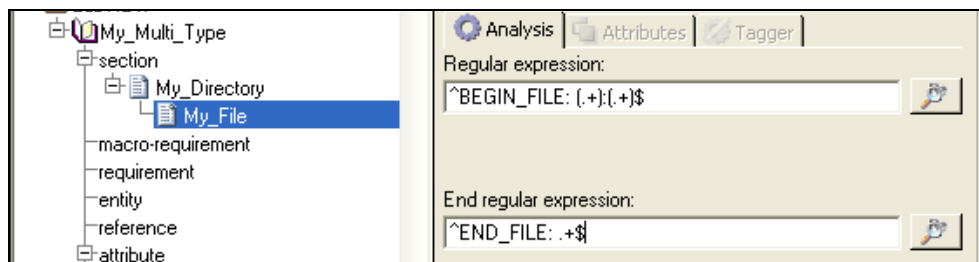
To create your own multi-type, follow these steps:

1. Create a new type by clicking on the **Add new type** button.

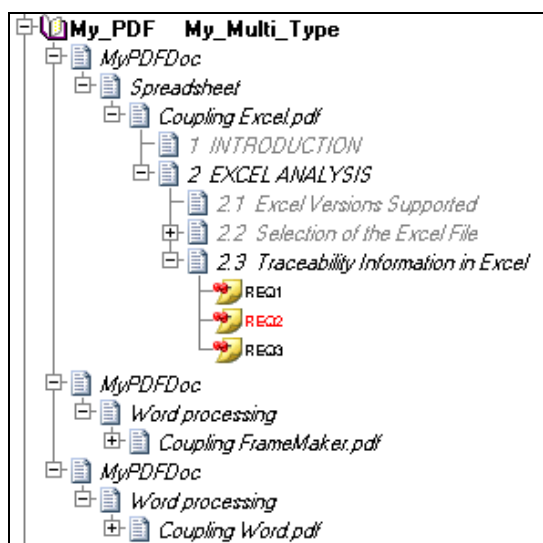
2. In the **Analysis** tab, select the **Convert** tool to use for individual files.
3. In the **Advanced Options** tab, fill in the extensions of files to be analyzed in the **Filters** field.
4. Create a **Directory** section. In this section, type **BEGIN_DIRECTORY** and **END_DIRECTORY** keywords into **Regular expression** and **End regular expression** fields:



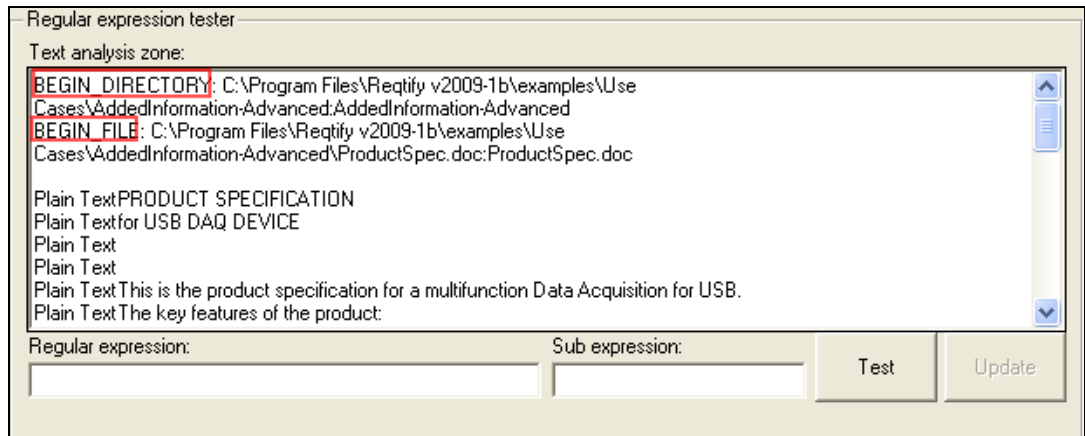
5. Now, create a **File** section. In this section, type **BEGIN_FILE** and **END_FILE** keywords into **Regular expression** and **End regular expression** fields:



6. Complete the section definition to capture headings, and then complete other elements of the type such as requirements and attributes.
7. A directory, with its subdirectories if any, and all documents in these directories are displayed in the Rhapsody Gateway main window.



During the multi documents analysis, **BEGIN_DIRECTORY**, **BEGIN_FILE**, **END_FILE** and **END_DIRECTORY** lines are added to each intermediate file for the multi-type creation:



These keywords are used to show the tree of directories and files in Rhapsody Gateway main window.

Capturing Coverage Information

The same approach can be applied to a document at a lower level, containing low-level requirements covering the high-level requirements. Coverage information is included in these low-level requirements and has to be captured as **References**.

Note

Coverage links can also be created from Rhapsody Gateway, as described in the *User Manual*.

The following is a simple Word document taken as example:

Heading 1	1. <u>My first section</u>
Normal	
Normal	LL-REQ_1
Requirement_Text	<i>Text of low level requirement 1</i>
Normal	<i>Covered Requirements: REQ_1</i>
Normal	
Normal	
Normal	LL-REQ_2
Requirement_Text	<i>Text of low level requirement 2</i>
Normal	<i>Covered Requirements: REQ_2</i>
Normal	
Heading 1	2. <u>My second section</u>
Normal	
Normal	LL-REQ_3
Requirement_Text	<i>Text of low level requirement 3</i>
Normal	<i>Covered Requirements: REQ_3, REQ_4</i>
Normal	
Normal	LL-REQ_4
Requirement_Text	<i>Text of low level requirement 4</i>
Normal	<i>Covered Requirements: REQ_2, REQ_4</i>

In this case, the **Types of Analysis** have to be customized to capture the requirement IDs and the coverage information:

- ◆ **Requirements** are defined using a syntax **LL-REQ_** followed by digits.
- ◆ **References** are defined using “**Covered Requirements:**” followed by the list of the higher level requirements.

Customization can be performed by extending the definition of the previously created type (**MyType**) or by creating a new type dedicated to the analysis of low level documents.

Note

This decision depends on your process, however we usually recommend that you create several dedicated types instead of a general one in which you try to define all your needs for all steps.

For the example purpose, we will create a new type called **LowLevel**.

1. Open the **Types Editor** and duplicate the default **Word** type. You can also duplicate a customized type. Name the type **LowLevel**.
2. Expand the tree for this new **Type of Analysis** and select the **Requirement** element.

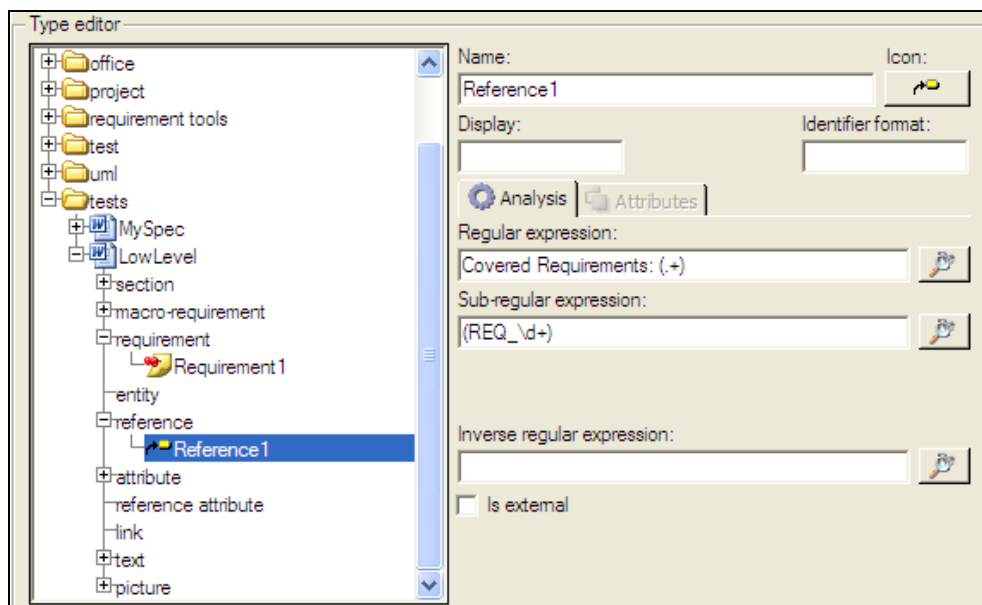
In our case the expression for requirements capture is still very simple: (LL-REQ_\d+)

- ◆ (...) Brackets define the limits of the captured string.
- ◆ LL-REQ_ is the fixed prefix for requirement IDs defined in our example.
- ◆ \d+ means “one or several digits”.

3. The **Reference** elements need to be defined to capture the coverage information.

Fill both **Regular expression** and **Sub regular expression** fields. We want to perform the analysis in two consecutive steps:

- ◆ Getting the full list of the referenced requirements after the “**Covered Requirements:**” tag.
- ◆ Within this list, identifying individually each requirement ID.



Note

The action of defining a **Reference** means you are defining Coverage. The captured information must not be the whole coverage link itself but only the target of the link, or the ID of the covered requirement(s).

4. The list of covered requirements can be captured by the expression: **Covered Requirements: (.+)**

- ◆ **Covered Requirements:** is the “tag” announcing the list of the referenced requirements. It is mentioned because of this key role; however the string itself does not have to be captured. Only the requirements list has to be captured.
- ◆ (...) Brackets define the boundaries of the string that is captured.

- ◆ `.+` means “one or several characters”. `.` means ‘any character’ and `+` means ‘one or several times’.

The previous expression gives the following captures:

Analyzed information	Result
Covered Requirements: REQ_1	REQ_1
Covered Requirements: REQ_2, REQ_4	REQ_2, REQ_4

The **Sub regular expression** field enables to define a second level of analysis by examining the result provided by the **Regular expression**. This second level is able to get all individual IDs in the list provided by the first analysis level.

The expression for requirements capture is: `(REQ_\d+)`

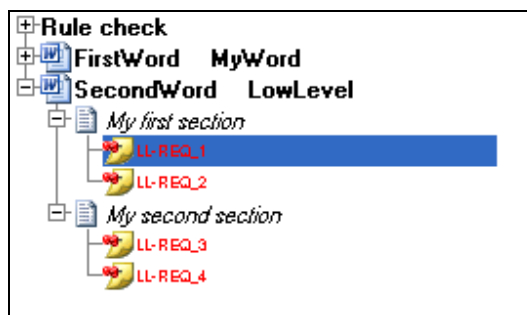
- ◆ `(...)` Brackets define the boundaries of the captured string.
- ◆ `REQ_` is the fixed prefix for requirement IDs defined in our example standard.
- ◆ `\d+` means “one or several digits”.

The previous expression gives the following captures:

Analyzed list	Result1	Result2	Result3
REQ_1	REQ_1		
REQ_2, REQ_4	REQ_2	REQ_4	

- Now, go back to the **Project Editor** and include the low level file in your project configuration.

Apply the new **Type of Analysis** and click **OK** to launch the analysis of the project files.



- In the main project workspace, Rhapsody Gateway displays the captured sections and low-level requirements with their text.

7. A **Rule check** section has been automatically created and contains all the elements that are violating rules. For example, low-level requirements are displayed in red because they activate the “**Traceability Graph Violation**” rule.

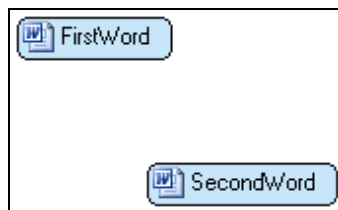
References have been correctly captured, but the project configuration is not defined to have the low-level document covering the high-level one. Captured coverage links are not consistent with the defined project process.

Project Definition

At this step, the information from individual documents: one at high level and another one at lower level have been captured.

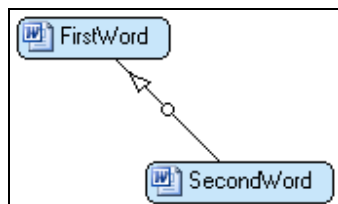
In addition, Rhapsody Gateway needs to know what kind of relationships are allowed and expected between individual documents in the project. This is done through coverage links created in the **Project Editor**.

- ◆ In such a configuration, traceability is not allowed between **SecondWord** and **FirstWord** documents.



References captured will activate the rule “**Traceability Graph Violation**”.

- ◆ Traceability between **SecondWord** and **FirstWord** documents is expected.



Requirements in the high-level document are supposed to be covered. If not, the rule “**Uncovered requirements**” will be activated.

See the *User Manual* for more information about the project configuration.

For additional information and better understanding, see also:

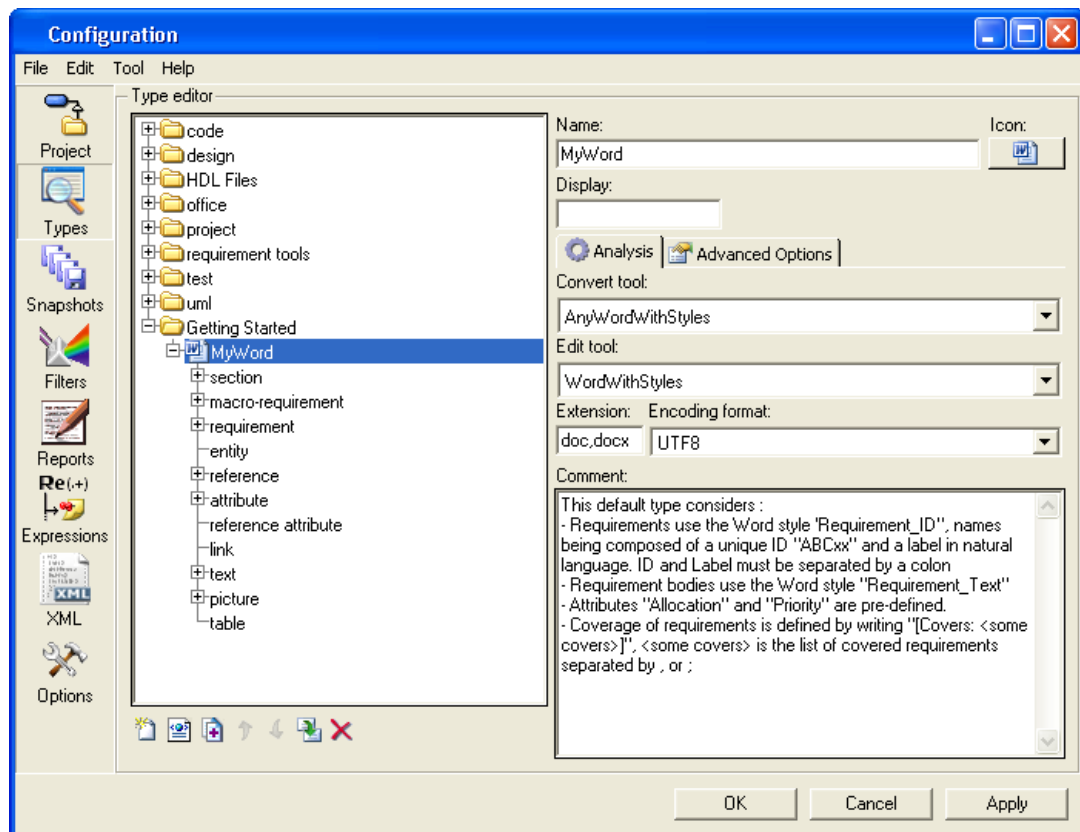
- ◆ The [FirstTraceability.rqtf](#) project, a ready to use example demonstrating these concepts.
- ◆ The [First Traceability animated demo](#), which presents the material that is explained in this section.

Types Editor

This chapter shows in detail the description of the Types Editor and all its functionalities.

General Aspects Definition

The general aspects of a type are displayed as follows when you select the root of the type tree:









The **Type editor** dialog box contains the type elements in the left area. According to the selected type the dialog box can contain two panes: **Analysis** and **Advanced** options in the right area. These panes are described thereafter.

The type definition has the following entry fields:

Field	Description
Name	Type name
Display	Used to define what you want to be displayed for a document of the selected type. For example, to only see the Document for all the files of the type selected, type <code>Document</code> in the Display field. To see Document [Name of document defined in project configuration editor], enter <code>Document \$i</code>
Comment	For free-form entry of comments on the selected type. (available for type for added elements)

The **Types Editor** dialog box contains several buttons:

Button	Description
	Click to associate one of the icons with the selected type (here the Word icon).
	Used to add a new customized file type.
	Used to add a new customized XML type.
	Used to add a new Type for an added element.
	Used to duplicate an already existing type.
	Deletes the selected customized type. You cannot delete a generic type.
OK	Applies modifications and closes the dialog box.
Cancel	Closes the dialog box without taking into account the changes made.
Apply	Applies modifications without closing the dialog box.

Contextual Menu

In this dialog box left side, items of the contextual menu are:

Function	Description
Add new type	If the selection is a customized type, it is used to add a new customized file type.
Add XML type	If the selection is a customized type, it is used to add a new customized XML type.
Add a type for added elements	If the selection is a customized type, it is used to add a new type for an added element.
Duplicate	If the selection is a type, it is used to copy all elements of the selected type into the specific directory for the project.
Copy	If the selection is a type or a category, it is used to copy all elements of the selected element onto the clipboard.
Paste	<p>If the copied element is a type, it is pasted as a new type at the selected position.</p> <p>Copy of categories is not possible.</p> <p>Paste of element types is not possible on library types.</p> <p>If the copied element is an element type:</p> <ul style="list-style-type: none"> • if the destination is a type: it is pasted in the right category • if the destination is a category: a new element type is created in the selected category using the information coming from the copied element • if the destination is an element type of the same category: it is pasted under the other element type in the category if the category is nestable or near the other element type. • if the destination is an element type of another category: a new element type is created in the category containing the destination using the information coming from the copied element
Delete	<p>If the selection is a type or a category, it deletes the selected type or category.</p> <p>Deletion is not possible on default types.</p>

Edit Menu

The configuration editor allows you to undo operations, and redo the previous operation.

The edit commands for undo and redo are contained in the **Edit** menu:

- ◆ **Undo**—Cancels actions on element editions or creations.
- ◆ **Redo**—Reverses the undo command.

Definition of a Type

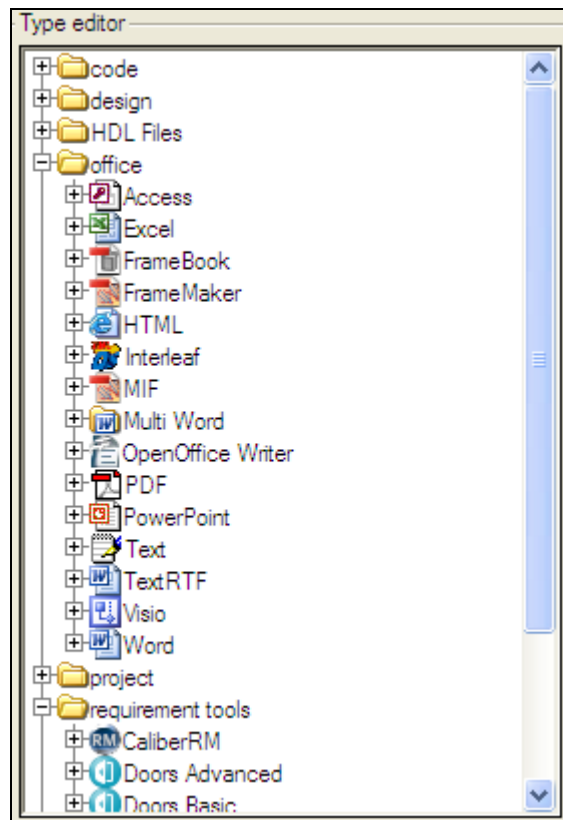
Three kinds of types can be created from the Types Editor:

- ◆ a file type
- ◆ an XML type
- ◆ a type for added elements

Types for added elements are presented in a specific chapter.

The types allow the user to define the following concepts. Please note that extensive definitions are not mandatory. Simple processes may request definition of requirements and references only.

Each type contains the same categories of elements as follows:










The **Types Editor** enables you to define the following traceability elements:

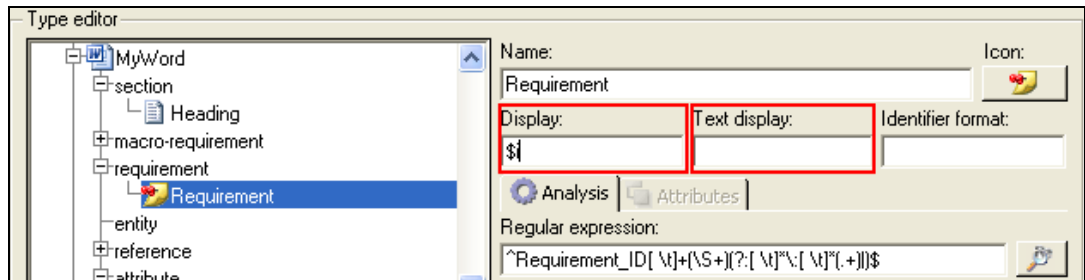
- ◆ **Section**—is used to capture the hierarchy and/or structure of the analyzed input file.
- ◆ **Macro-requirement**—is used to support hierarchy between requirements.
- ◆ **Requirement**—is used to capture the requirements identifiers.
- ◆ **Entity**—is an advanced concept used to force the analysis of defined elements in order to highlight elements without coverage (dead code modules).
- ◆ **Reference**—is used to capture the coverage links.
- ◆ **Attribute**—is used to capture requirement attributes.
- ◆ **Reference**—attribute is used to capture additional information attached to a coverage link.
- ◆ **Link**—is used to capture any link excluding a coverage link.
- ◆ **Text**—is used to capture requirement texts.
- ◆ **Picture**—is used to capture pictures and diagrams.

It is not mandatory to define ALL the elements for a Type. You are only supposed to define elements that are relevant for your needs and the support of your requirement standards.

To define and manipulate the type elements, select a branch of the tree structure for the customized type, and use the following buttons:

Button	Description
	Inserts a new type element.
	Inserts an element defined using XML.
	Inserts an element in a “Type for Added Element”.
	Moves up a selected element in a Type.
	Moves down a selected element in a Type.
	Duplicates the selected element.
	Deletes the selected element. It is not possible to delete an element used in a filter or in the project configuration.

Depending on the selection in the **Type** tree, some properties can be added to define the selected type elements. Some properties can be added in the **Display** field or in the **Text display** field, as follows:





The available symbols are:

Symbol	Property
\$i	element identifier
\$l	element label
\$t	element type
\$T	element text (line return will be automatically inserted before the text)
\${ Attribute }	attribute value

For example, if the following display format is specified for requirements: “\$t: \$i Priority: \${Priority}”, the elements tree will show: “Requirement: REQ001 Priority: High”.

If the following display text format is specified for requirements: “\$i - \$l Priority: \${Priority}\$t”, the resulting text of the selected element will be: “REQ001 - Call elevator Priority: High – Sample text”.

The **Analysis** pane contains fields dedicated to the capture of each element. The following buttons are available for each kind of elements:

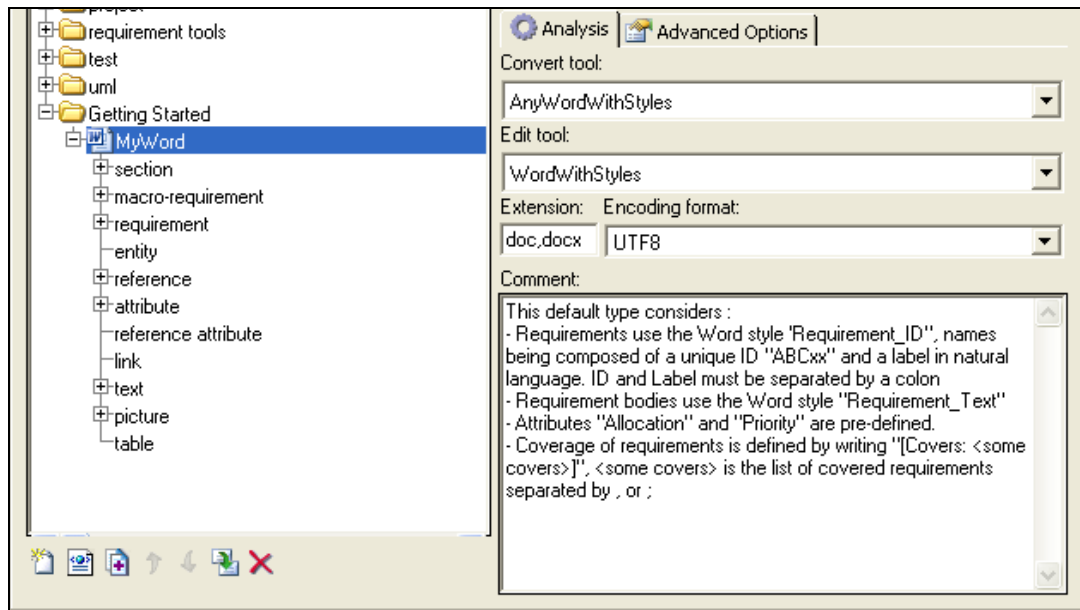
- ◆  —Click to change the icon displayed for the selected element. To restore the default icon, right click this **Icon** button then select **Default** in the context menu.
- ◆  —Displays the regular expression tester to validate your regular expressions.

See below the **Analysis** tab contents.

Definition of the Analysis Tab

General Aspect of Analysis Tab

The content of the Analysis tab depends on the selected type.



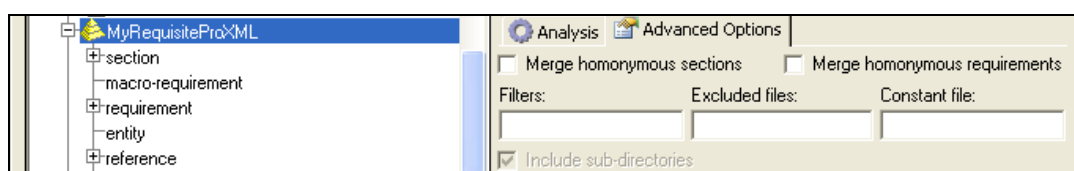
The fields of this pane are the following.

Function	Description
Convert tool	Indicates the converter used for the document analysis for which the type is applied.
Edit tool	Used to define the tool used for the document edition. It can be different from the Convert tool (you can use Word to edit a .txt file).
Extension	File extension to be used when selecting the file to be analyzed for the selected type. For example, enter doc to find *.doc type files only (simply enter the letters of the extension, not the dot).
Encoding format	Defines the encoding format of the input file, typically to support Asiatic fonts. Available encoding formats are: UTF8, UCS2, UCS2be and Shift_JIS. (This element is only available for file type.)
Comment	For free-form entry of comments on the selected type.

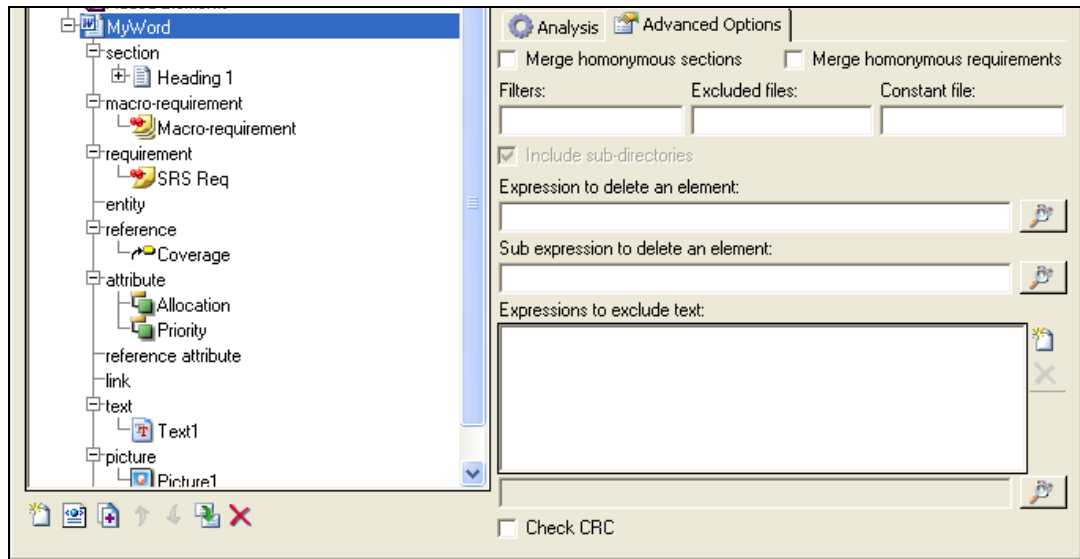
The screenshot shows the 'Type editor' window. On the left, a tree view displays the hierarchy of types: 'MyWord' (root), 'section', 'macro-requirement', 'requirement', 'entity', 'reference', 'attribute', 'reference attribute', 'link', 'text', 'picture', and 'table'. The 'Requirement' type under 'requirement' is selected and highlighted in blue. On the right, the configuration panel for the 'Requirement' type is visible. It includes fields for 'Name' (Requirement), 'Icon' (a red and yellow icon), 'Display' (empty), 'Text display' (empty), and 'Identifier format' (empty). Below these are tabs for 'Analysis' and 'Attributes'. The 'Analysis' tab is active, showing a 'Regular expression' field with the text '^Requirement_ID([\s]+\s+)?([\s]*\s+)' and a 'Sub-regular expression' field. There are also fields for 'End regular expression' and 'Prefix expression', each with a small icon to its right.

Definition of the Advanced Options of a Type

The **Advanced Options** pane for an XML type is shown in the following figure:




The **Advanced Options** pane for a text type is shown in the following figure:



This pane contains the following entry fields:


Field	Description
Merge homonymous sections	Used to merge sections with the same name.
Merge homonymous requirement	Used to merge requirements with the same name. Warning: if this option is set, requirement duplication is not found by Requirement defined several times rule.
Constant file	Definition of a constant file. Useful for interfaced tools which always produce the same model description file.
Filters	Allows processing of a document consisting of several separate files. Leave the Extension field blank and in the Filter field type <code>*.doc</code> , for example. If you include a document of this type in the project, Rhapsody Gateway allows you to select a directory (not a file) and scans this directory for all files with extension <code>doc</code> . You can define several extensions, separated by a comma: for example <code>*.c, *.h</code>
Excluded Files	This is a filter to exclude some files of the analysis, for example <code>*.svn</code> . This is helpful when you use Rhapsody Gateway to browse directories and subdirectories.

Field	Description
Expression to delete an element	Regular expression is used to avoid the analysis of elements (requirements, references, etc.). See detailed description below.
Sub-expression to delete an element	Regular expression allows a second level of processing. This is used to avoid the analysis of elements provided by the analysis of the expression "to delete an element". For instance, to avoid the capture of listed elements.
Expressions to exclude texts	Regular expression defines the text to be ignored during analysis. See detailed description below.
Check CRC	A CRC number is computed on the text between two elements. As usually, changes are indicated with  icon. Selecting Check CRC shows changes on non-captured information.

Exclusion of Text Areas


You can exclude parts of the analyzed document. For example, you might want to exclude a part because it contains information that could be captured as traceability information (requirements, references, etc.) when this is not appropriate.

To exclude part of the text, follow these steps:

1. Click  to define one or more expressions.
2. Define an expression in the **Expressions to exclude text** field.

For instance <<<[\S\s] *?>>> will exclude part of the document located between <<< and >>> tags. When the input documents with this type of analysis are analyzed, the text located between <<< and >>> will be ignored.

To delete an excluding expression, follow these steps:

1. Select an expression.
2. Click  to delete the selected expression.

Exclusion of an Element

For a given type you can exclude elements from the analysis. For instance you may want to keep “obsolete” requirements in your documents; but you do not want these requirements to be captured by Rhapsody Gateway even if they use the same syntax as valid requirements.

To exclude an element:

- ◆ Define an expression in the **Expression to delete an element field**.

For instance `(REQ_\d+)\s*\ (Obsolete\)`. The expression must capture the identifier of the element you want to ignore. In the input documents, all the requirements with an identifier `REQ_nn` followed by `(Obsolete)` mentioned after the identifier will be ignored.

Note 1

If elements have been removed from the analysis, this information is displayed in the **Messages** pane of the main window.

Note 2

If a requirement is ignored using this feature, all its elements attached for instance attributes or texts are also ignored.

Note 3

If your expression is not correctly defined and captures an element not corresponding to a project element, the rule "Impossible to delete" an element is activated.

For additional information and better understanding, see also:

- ◆ The [AdvancedTypes.rqtf](#) project, a ready to use example demonstrating these concepts.

Definition of Type Elements from Text Files

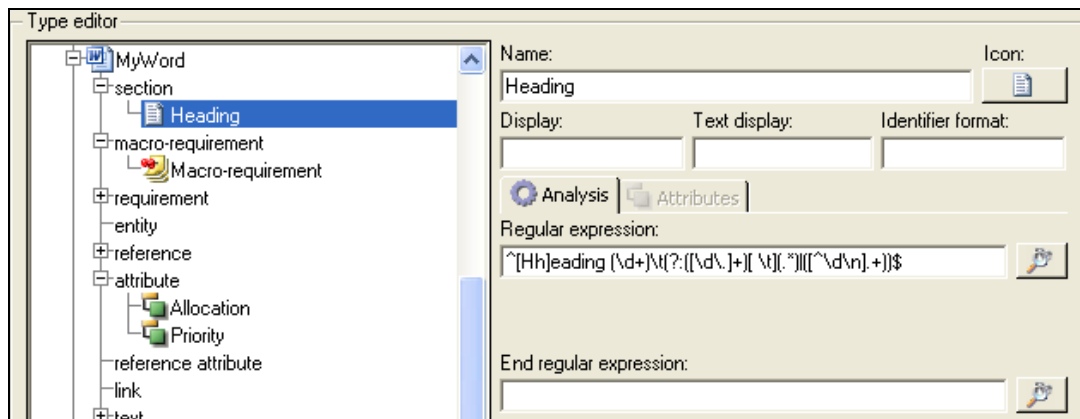
The text types are used when the intermediate file created by Rhapsody Gateway is a text file.

Following sections present all the elements of a type.

Defining Sections

Sections allow you to capture the hierarchical structure of the analyzed input file. The sections types depend on the type of file to be analyzed, such as headings for Microsoft Word files or worksheets for Microsoft Excel files. For details regarding the Sections concepts, see the *User Manual*.

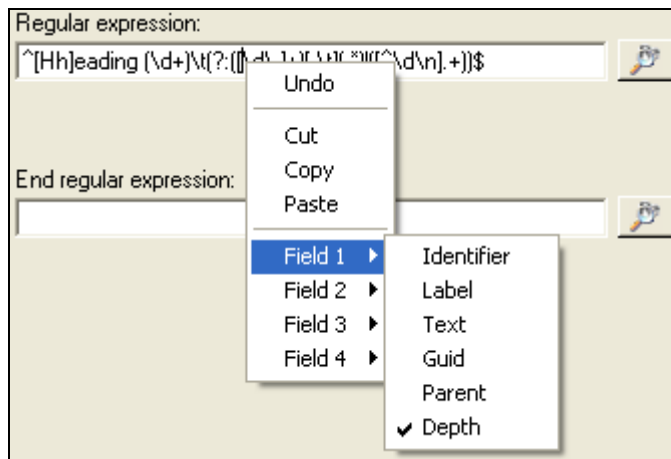
The **Types editor** pane from the Configuration window enables you to define **Sections** that capture expression, as shown in the following figure.



A **Section** has the following features:

- ◆ **Name**—Specifies the name of the Section. The name of a section is displayed in a tooltip when the cursor is over that element in the main window of the tool.
- ◆ **Display**—Defines a display name for the Section in the project workspace. If this field is blank, the Section names of the source document are used as display names.
- ◆ **Text display**—Defines a text which is displayed in the Text area when a Section is selected.
- ◆ **Identifier format**—Defines a new identifier for the Section. This identifier can use information provided by Regular expression. For example \\1 and \\2 represent first and second fields of the regular expression. In the same way, \\d can be used to prefix the identifier with the document name.
- ◆ **Regular expression**—Specifies the regular expression used to capture the Section in the source files (more precisely the intermediate file).
 - If the **End regular expression** field is empty, the **Regular expression** field defines the capture of the Section.
 - If the **End Regular expression** field is filled, the **Regular expression** field defines the expression which represents the beginning tag in order to identify a Section.

A right click in the **Regular expression** field opens a context menu which contains a list of fields automatically generated according to the number of sub-expressions. Each field can be assigned as identifier, label, text, guid, parent or depth.



- **Identifier**—is the section identifier.
- **Label**—is a field for labeling.
- **Text**—is used to capture the element text.
- **Guid**—is used to capture an internal identifier and is always unique.

Note

The GUID is the best way to identify precisely an element. It is possible to capture both GUID and identifier for any element where the GUID is the unique identifier from the original tool and identifier is the logical name that can be used for requirements traceability between other documents.

- **Parent**—allows you to capture an identifier which will be the current element parent. It can be useful when the structure of the intermediate file is flat.

Package	
Class1	parent=Package
Class2	parent=Package
Operation1	parent=Class1
Attribute1	parent=Class1

- **Depth**—allows you to capture the depth of the Heading.
- ♦ **End regular expression**—Specifies the end tag to identify a Section. This field allows recursive analysis. Thus it allows the capture of sections from files such as the following example:




```
[Section]
Info...
  [Section]
  Other info...
  [End of Section]
[End of Section]
```

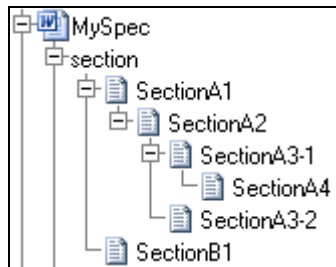
Note

Usually this field is used for dedicated interfaces and is pre-defined. If you need to analyze recursive structures in XML, use XML type definitions.

Creating a section expression

Sections can be created like a tree with several branches at each level. You can have several 'Level1' sections, several 'Level2' sections and so on. Follow these steps to create sections:

1. Select the type in the **Type editor**.
2. Select section item then click . A new section is created under the original section. Name this section SectionA1.
3. To create a second branch, click the parent section then click  to create the branch.
4. Name this section SectionB1.
5. To create SectionA2, select SectionA1 then click .
6. Select SectionA2 then create two subsections; SectionA3-1 and SectionA3-2.
7. Select SectionA3-1, and then create SectionA3-2 section.
8. You will obtain a section like the following figure:



Creating a concise section expression

Previous definitions of sections are often used. When the contents of the sections to be captured are practically the same, a concise capturing expression for sections is possible.

Here is an example of one capturing expression used for several headings:

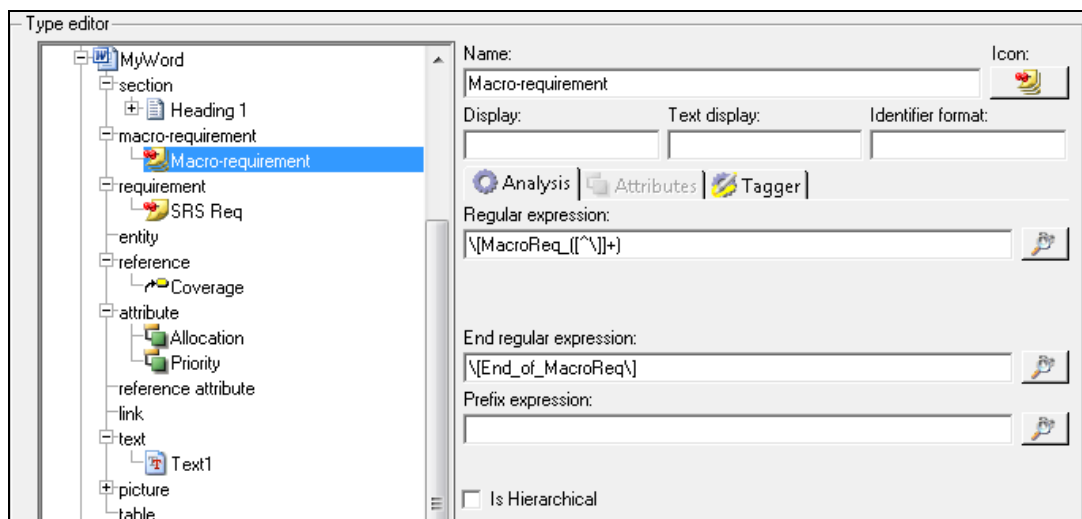
1. Select the type in the Type editor.
2. Select section then click Add new type. A new section is created beneath the Section element.

3. Name this section then enter `^Heading (\d+)\s(.+)\$` expression for the Regular expression.
4. Click right in Regular expression field.
5. Select Field1 then select Depth in the new submenu to assign Field1 to Depth.
6. Select Field2 then select Identifier in the new submenu to assign Field2 to Identifier.

Defining Macro-Requirements

Macro-Requirements allow you to support a hierarchy between requirements. For details regarding the Macro-requirements concepts, see the *User Manual*.

The **Types editor** pane from the Configuration window enables you to define the Macro-Requirements capturing expression, as shown in the following figure.

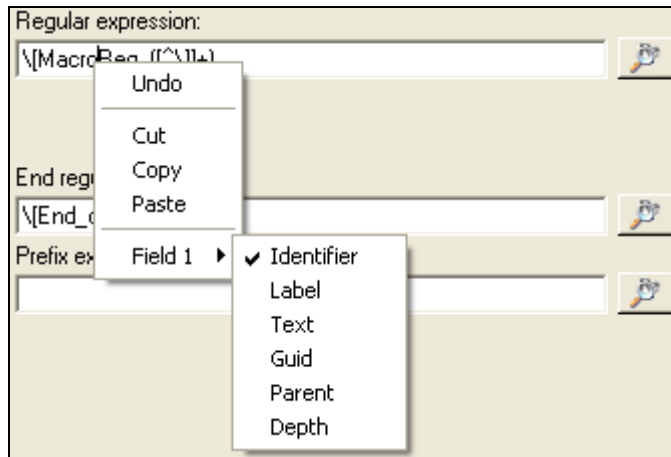


A **Macro-requirement** has the following features:

- ◆ **Name**—Specifies the name of the Macro-requirement. The name of a Macro-requirement is displayed in a tooltip when the cursor is over that element in the main window of the tool.
- ◆ **Display**—Defines a display name for the Macro-requirement in the project workspace. If this field is blank, the Macro-requirement names of the source document are used as display names.
- ◆ **Text display**—Defines a text which is displayed in the Text area when a Macro-requirement is selected.
- ◆ **Identifier format**—Defines a new identifier for the Macro-requirement. This identifier can use information provided by Regular expression. For example \1 and \2 represent first and second fields of the Regular expression. In the same way, \d can be used to prefix the identifier with the document name.

- ◆ **Regular expression**—Specifies the regular expression used to capture the Macro-requirement in the source files (more precisely the intermediate file). This expression represents the beginning tag that identifies a Macro-requirement.

A right click in the **Regular expression** field opens a context menu which contains a list of fields automatically generated according to the number of sub-expressions. Each field can be assigned as **identifier**, **label**, **text**, **guid**, **parent** or **depth**.



- **Identifier**—is the requirement identification that is referenced for coverage.
- **Label**—is a field for labeling.
- **Text**—is used to define the capture of the element text.
- **Guid**—is used to capture an internal identifier and is always unique.

Note

The GUID is the best way to identify precisely an element. It is possible to capture both GUID and identifier for any element where the GUID is the unique identifier from the original tool and identifier is the logical name that can be used for requirements traceability between other documents.

- **Parent**—allows you to capture an identifier which will be the current element parent.
- **Depth**—allows you to capture the depth of the Macro-requirement.
- ◆ **End regular expression**—Specifies the end tag that identifies a Macro-requirement. If this field is blank, the Regular Expression defines the starting position of the Macro-requirement which finishes at the end of the source document.
- ◆ **Prefix expression**—Specifies a Prefix expression for the Macro-requirement. It allows you to concatenate the prefix with elements that are captured by the Regular expression to get the whole Macro-requirement ID.

For example, requirements in a document are defined as follows:

REQ_Product1_

F1: Max time response shall be 5 ms

F2: The system shall be active in less than 10 clock periods

Definition of REQ_Product1 as a prefix will allow the capture of requirements as follows:

REQ_Product1_F1: Max time response shall be 5 ms

REQ_Product1_F2: The system shall be active in less than 10 clock periods


- ◆ **Is Hierarchical**—Allows you to nest requirements and macro-requirements in macro-requirements. If this option is not selected all macro-requirements are flattened. By default, this option is not selected.

Creating a macro-requirement

The following figure shows a source file to be captured using Macro-requirements:

```
Level1Requirement_1
Text of my macro_requirement
EndOfLevel1Requirement_1
```

To define a Macro-requirement to be captured, such as the Macro-requirement above, follow these steps:

1. Select **File > Edit Types**. The Type editor opens.
2. Select the type to modify then select item Macro-requirements.
3. Click the **Add new type** icon  on the toolbar.
4. Name the Macro-requirement. For Regular expression and End regular expression enter:
 \s+(Level\d+Requirement_\S*)
 and
 \s+EndOf(Level\d+Requirement_\S*)
5. Click **Apply** to apply your changes and reload the project.

Creating a hierarchic macro-requirement

If you select **Is Hierarchical** in Macro-requirement type definition for a source file which uses Macro-requirements as follows:

```

[MacroReq_MACRO_001]
REQ001
[MacroReq_MACRO_002]
REQ002
[End_of_MacroReq]
[End_of_MacroReq]

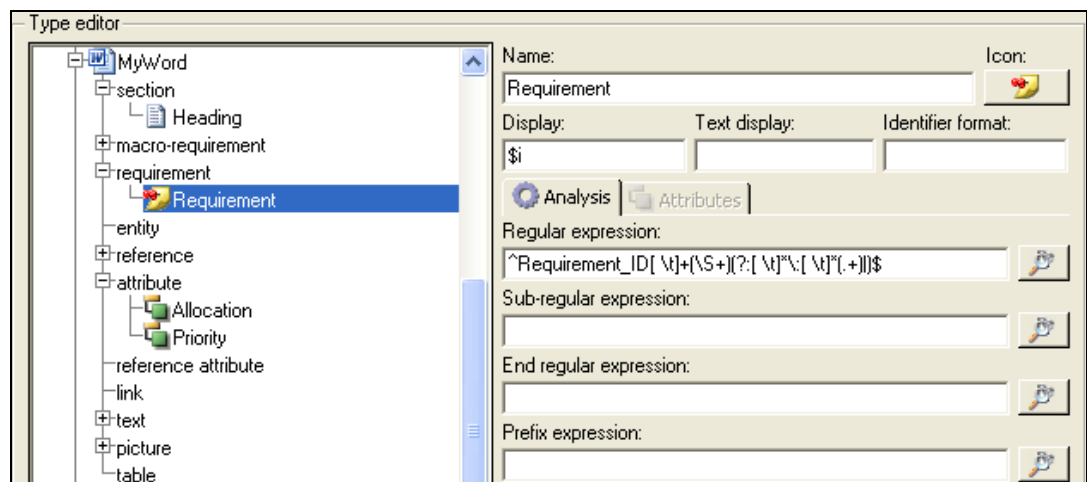
```

You will get an analyze such as follows.



Defining Requirements

The **Types editor** pane from the Configuration window enables you to define the Requirements capturing expression, as shown in the following figure.



A **Requirement** has the following features:

- ◆ **Name**—Specifies the name of the Requirement. The name of a Requirement is displayed in a tooltip when the cursor is over that element in the main window of the tool.
- ◆ **Display**—Defines a display name for the Requirement in the project workspace. If this field is blank, the Requirement names of the source document are used as display names.
- ◆ **Text display**—Defines a text which is displayed in the Text area when a Requirement is selected.

- ◆ **Identifier format**—Defines a new identifier for the Requirement. This identifier can use information provided by Regular expression. For example \1 and \2 represent first and second fields of the Regular expression. In the same way, \ d can be used to prefix the identifier with the document name.
- ◆ **Regular expression**—Specifies the regular expression used to capture the Requirement in the source files (more precisely the intermediate file). This expression represents the beginning tag that identifies a Requirement.
- ◆ **Sub-regular expression**—Specifies the sub-regular expression used to capture a second level of processing. This process analysis the result provided by the preceding regular expression.
- ◆ **End regular expression**—Specifies the end tag that identifies a Requirement. If this field is blank, the Regular Expression defines the starting position of the Requirement which finishes at the beginning of a next Requirement capture.
- ◆ **Prefix expression**—Specifies a Prefix expression for the Requirement. It allows you to concatenate the prefix with elements which are captured by the Regular expression to get the whole Requirement ID.

The following is an abstract of a table to show the capture of requirements and texts:

REQ1	This is the requirement text
-------------	------------------------------

The capture of the requirement and of the text can be performed with the following definition:

Requirement: (REQ\d+)

Text: \ | 2 ([^ \ |] +) = second column, everything except the end of the column.

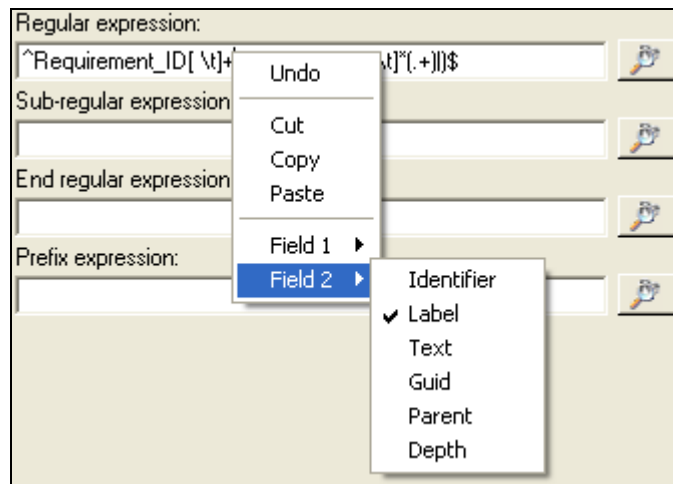
It is also possible to use a single expression like the following (REQ\d+) \ | 2 ([^ \ |] +)

and to use the advanced capture definition for regular expressions.

This expression defines two **fields**, between brackets: (REQ\d+) \ | 2 ([^ \ |] +). The advanced capture definition allows you to select the assignment of these captured fields.

Enter your regular expression then right click in the regular expression field to open the context menu.

A right click in the **Regular expression** field opens a context menu which contains a list of fields automatically generated according to the number of sub-expressions. Each field can be assigned as identifier, label, text, guid, parent or depth.



- **Identifier**—is the requirement identification that is referenced for coverage.
- **Label**—is a field for labeling.
- **Text**—is used to define the capture of the element text.
- **Guid**—is used to capture an internal identifier and is always unique.

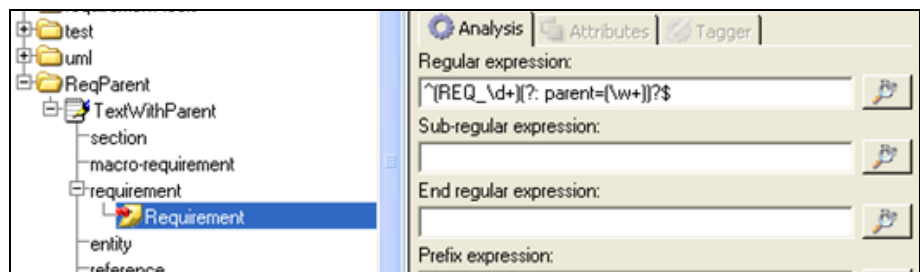
Note

The GUID is the best way to identify precisely an element. It is possible to capture both GUID and identifier for any element where the GUID is the unique identifier from the original tool and identifier is the logical name that can be used for requirements traceability between other documents.

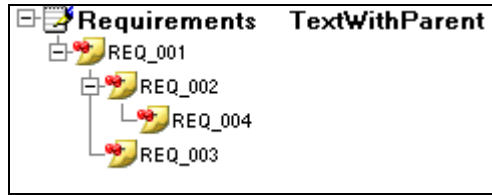
- **Parent**—allows you to capture an identifier which will be the current element parent, as shown on the example below:

```
REQ_001
REQ_002 parent=REQ_001
REQ_003 parent=REQ_001
REQ_004 parent=REQ_002
```

The syntax to capture the current element parent is the following:



The corresponding analysis result is the following:



- **Depth**—allows you to capture the depth of the Requirement.

In the example above, a classical capture using the **Text** definition of the type tree could also be used.

In the case below, however, it would not be possible. Default configuration attaches the text captured to the element, such as the requirement or section captured just before it, and this default configuration is not compliant for cases where the requirement ID is located after the text.

This is the requirement text	REQ1
------------------------------	------

Thanks to advanced allocation capabilities, you can analyze this configuration with a regular expression such as the following:

```
\|1 ([^\|]+)\|2 (REQ\d+)
```



You can analyze this expression by defining the following values for the fields' elements:


- **Field1** as **Text**
- **Field2** as **Identifier**

Very accurate captures of the information are allowed by this advanced allocation feature.

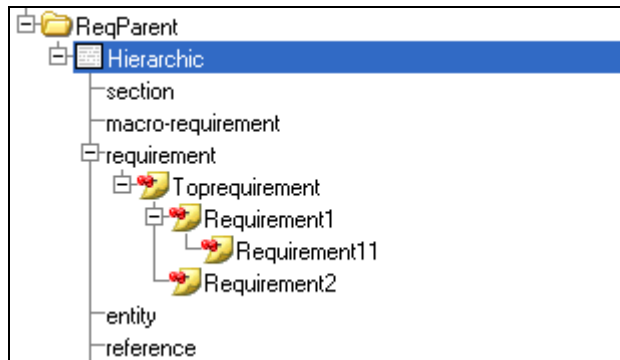
Creating hierarchic requirements

Hierarchic requirements can be created. They are displayed like a tree with several branches at each level. Follow these steps to create hierarchic requirements:

1. Select the type in the Type editor.
2. Select requirement item then click . A new requirement is created under the original requirement. Name this section Toprequirement.
3. To create a second branch, click the parent requirement then click  to create the branch.
4. Name this section Requirement1.

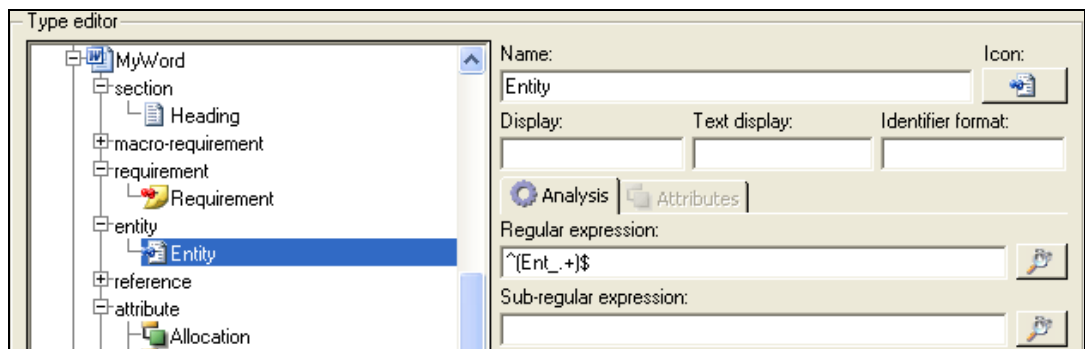
5. To create Requirement11, select Requirement1 then click .
6. Select Toprequirement then create Requirement2.

You will obtain a section like the following figure:



Defining Entities

By defining an **Entity**, the user defines an element that must cover (contain a reference to) a requirement.

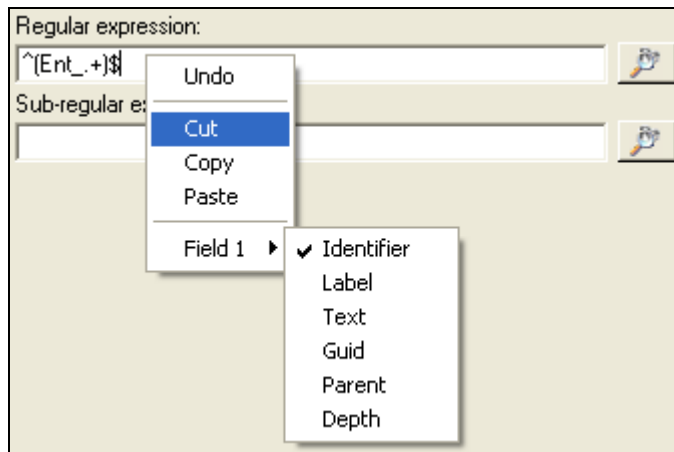


An **Entity** has the following features:

- ◆ **Name**—Specifies the name of the Entity. The name of an Entity is displayed in a tooltip when the cursor is over that element in the main window of the tool.
- ◆ **Display**—Defines a display name for the Entity in the project workspace. If this field is blank, the Entity names of the source document are used as display names.
- ◆ **Text display**—Defines a text which is displayed in the Text area when an Entity is selected.
- ◆ **Identifier format**—Defines a new identifier for the Entity. This identifier can use information provided by Regular expression. For example \1 and \2 represent first and second fields of the Regular expression. In the same way, \d can be used to prefix the identifier with the document name.

- ◆ **Regular expression**—Specifies the regular expression used to capture the Entity in the source files (more precisely the intermediate file). This expression represents the beginning tag that identifies an Entity.

A right click in the **Regular expression** field opens a context menu which contains a list of fields automatically generated according to the number of sub-expressions. Each field can be assigned as identifier, label, text, guid, parent or depth.



- **Identifier** is the entity identifier.
- **Label** is a field for labeling.
- **Text** is used to define the capture of the element text.
- **Guid**—is used to capture an internal identifier and is always unique.

Note

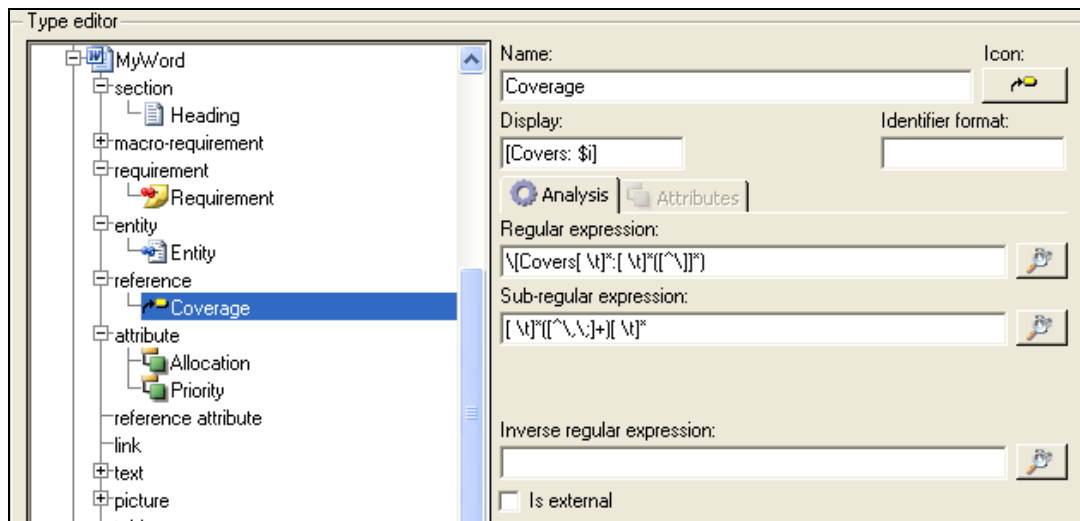
The GUID is the best way to identify precisely an element. It is possible to capture both GUID and identifier for any element where the GUID is the unique identifier from the original tool and identifier is the logical name that can be used for requirements traceability between other documents.

- **Parent** allows you to capture an identifier which will be the current element parent.
- **Depth**—allows you to capture the depth of the Entity.

Defining References

The definition of **References** is very important because **References** represent the way you declare coverage of requirements in your covering documents.

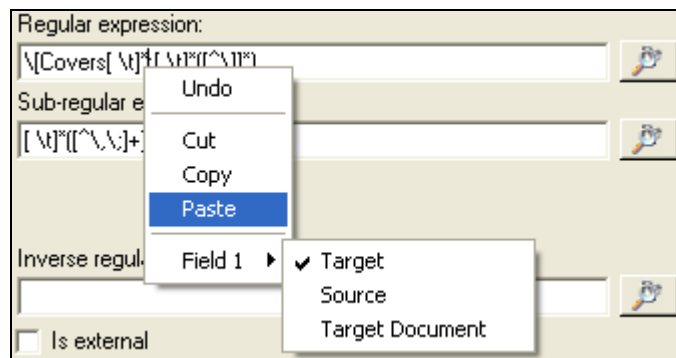
The target of a **Reference** must be a requirement, a macro-requirement or a derived requirement.



A **Reference** has the following features:

- ◆ **Name**—Specifies the name of the Reference. The name of the Coverage is displayed in a tooltip when the cursor is over that element in the main window of the tool.
- ◆ **Display**—For References, the **Display** field is not used to define what is displayed in the main window, but what is copied in addition to the requirement ID in the Copy-Paste action. See details in the Copy-Paste action section below.
- ◆ **Identifier format**—Defines a new identifier for the Reference. This identifier can use information provided by Regular expression. For example \\1 and \\2 represent first and second fields of the Regular expression. In the same way, \\d can be used to prefix the identifier with the document name.
- ◆ **Regular expression**—Specifies the regular expression used to capture the coverage information in the source files (more precisely the intermediate file). This expression represents the beginning tag that identifies a Reference. For instance, define an expression like `Covers_(REQ\\d+)` which only captures the referenced ID.

Right click in the **Regular expression** field opens a context menu which contains a list of fields automatically generated according to the number of sub-expressions. Each field can be assigned as **target**, **source** or **target document**.



- **Target** is used to capture the target requirement identifier or GUID.

- **Source** is used to capture the source requirement identifier or GUID.
- **Target Document** is used to capture the target document identifier.

- ◆ **Sub-regular expression**—Specifies the sub-regular expression used to capture a second level of analysis. This process analyzes the result provided by the preceding regular expression. This is helpful when a reference information contains reference to several requirements.

The following is an example of reference definition:

CoveredRequirements:REQ1, REQ2, REQ3

Regular expression: CoveredRequirements:(.+)

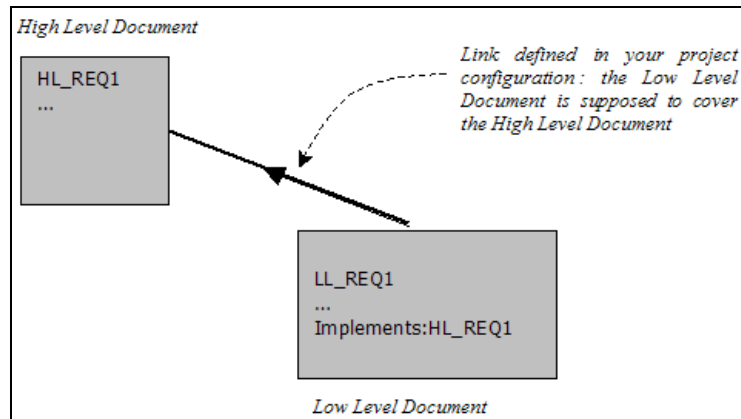
SubRegular Expression: (REQ\d)

- ◆ **Inverse regular expression**—With References, this tool allows you to handle the traceability information of requirements inserted in a top-down direction, using a “IsCoveredBy” approach instead of the default bottom-up approach “Covers”. See the detailed explanations below concerning the definition and the use of **Inverse Regular Expression**.
- ◆ **Is External**—Defines a link to be external. An external link is a link to an external requirement (example: http://...). External requirements are requirements defined externally such as an HTTP link or a requirement defined in a document which not belonging to the user. External requirements appear as undefined requirements in the upstream tree of the main Rhapsody Gateway window when they are covered by the current central selection.

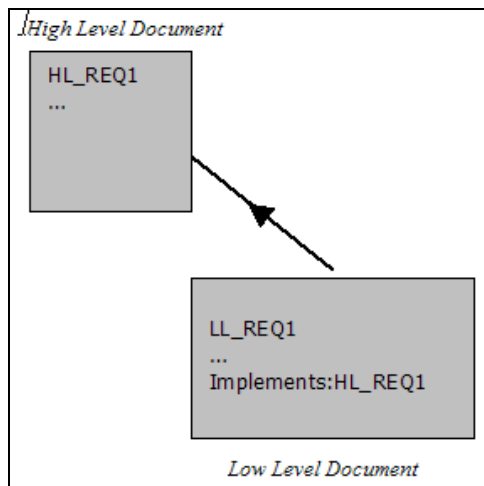
Using inverse regular expression command

Rhapsody Gateway manages the captured information as a coverage link because it is defined as a **Reference** element. The captured information must not be the whole coverage link itself but it must be the target of the link (i.e. the ID of the covered requirement(s)).

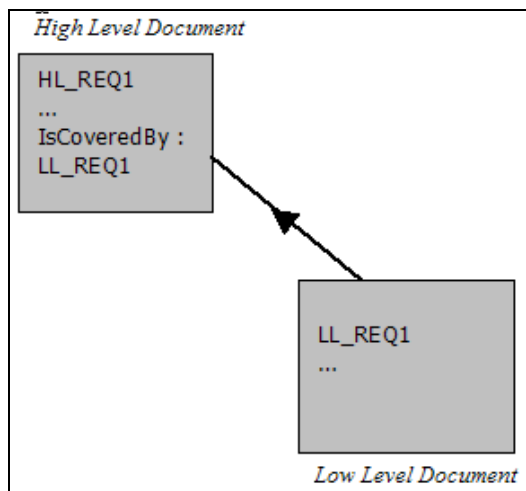
For a tag `Implements:HL_REQ1`, defining an expression based on `Implements:...` as a **Reference**, indicates to Rhapsody Gateway that it is a coverage information, but only `HL_REQ1` needs to be captured.



- ◆ HL_REQ1 is captured as a high level requirement it is defined as a **Requirement** for the type applied to the high level document.
- ◆ LL_REQ1 is captured as a low level requirement it is defined as a **Requirement** for the type applied to the low level document.
- ◆ `Implements:...` is captured as a reference, attached to LL_REQ1.
- ◆ HL_REQ1 is captured as the requirement referenced in the `Implements:HL_REQ1` definition.
- ◆ If HL_REQ1 exists, Rhapsody Gateway establishes the traceability between HL_REQ1 and LL_REQ1.
- ◆ If HL_REQ1 does not exist, the rule **Undefined requirement** is activated.
- ◆ HL_REQ1 exists but it is contained in a document for which you did not define a coverage link in your project configuration. To express that the document containing HL_REQ1 can be referenced by the one containing LL_REQ1, the rule **Traceability graph violation** is activated.

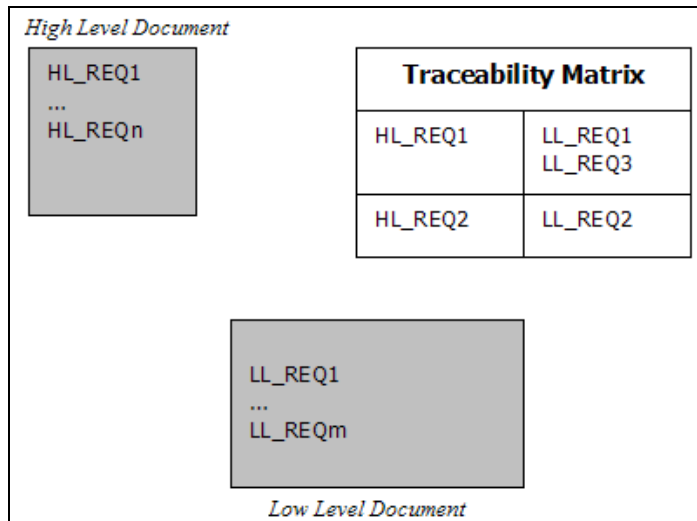


- ◆ HL_REQ1 is defined as a Requirement in the type applied to the high level document.
- ◆ LL_REQ1 is defined as a Requirement in the type applied to the low level document.
- ◆ Implements... is defined as a Reference in the type applied to the low level document, using the Regular Expression field and the SubRegular expression field if necessary.



- ◆ HL_REQ1 is defined as a Requirement in the type applied to the high level document.
- ◆ LL_REQ1 is defined as a Requirement in the type applied to the low level document.
- ◆ IsCoveredBy... is defined as a Reference in the type applied to the high level document, using the Inverse Regular Expression field.

In practice, the **Inverse Regular Expression** option is also very helpful in managing projects that have already been started with previously created traceability and for which the existing information looks like the following:



When applied to such project configurations, **Inverse Regular Expression** are useful to capture the information in the traceability matrix, showing that HL_REQ1 is covered by LL_REQ1 and LL_REQ3.

For additional information and better understanding, see also:

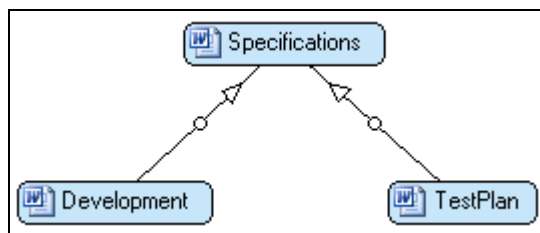
- ◆ The [IsCoveredBy.rqtf](#) project, a ready to use example demonstrating these concepts.
- ◆ The [IsCoveredBy demo](#), which presents the material explained in this section.

Copy For action

A **Specifications** document is supposed to be implemented and verified.

In the downstream document for the implementation phase, the references are defined using the syntax: `Refers to : (.+)?.`

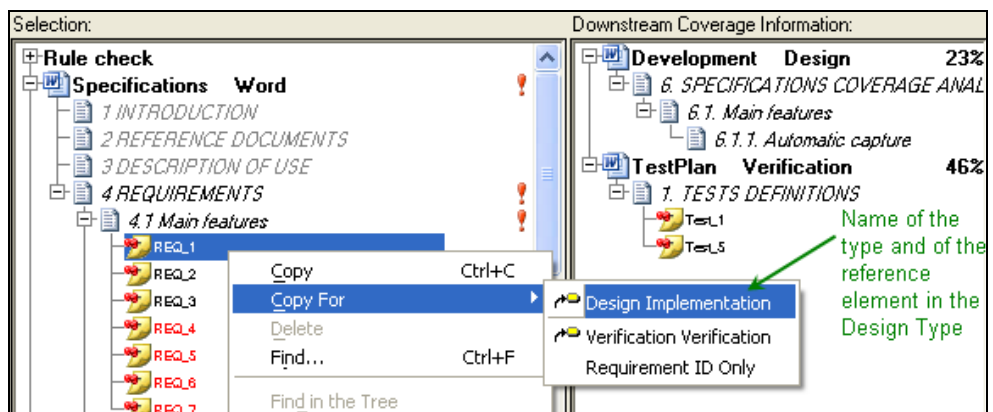
In the downstream document for the verification phase, the references are defined using the syntax: `\|3 Normal\t(.+?)\|` from which `(REQ_\d+)` is extracted.



As described in the Features for *Inserting Coverage Information in Project Files* of the *User Manual*, the copy-paste feature of Rhapsody Gateway allows the management of these multiple coverage cases.

Select a requirement to reference then select the target in the **Copy For** submenu. This copy-paste feature uses the definition of the **Reference** element for the type at low level:

- ◆ In the **Copy For** submenu appears the name defined for the **Reference** element for each type covering the document containing requirements. In the screenshot below, **Copy For** will show the **Reference** names defined for the types applied to Development and TestPlan.



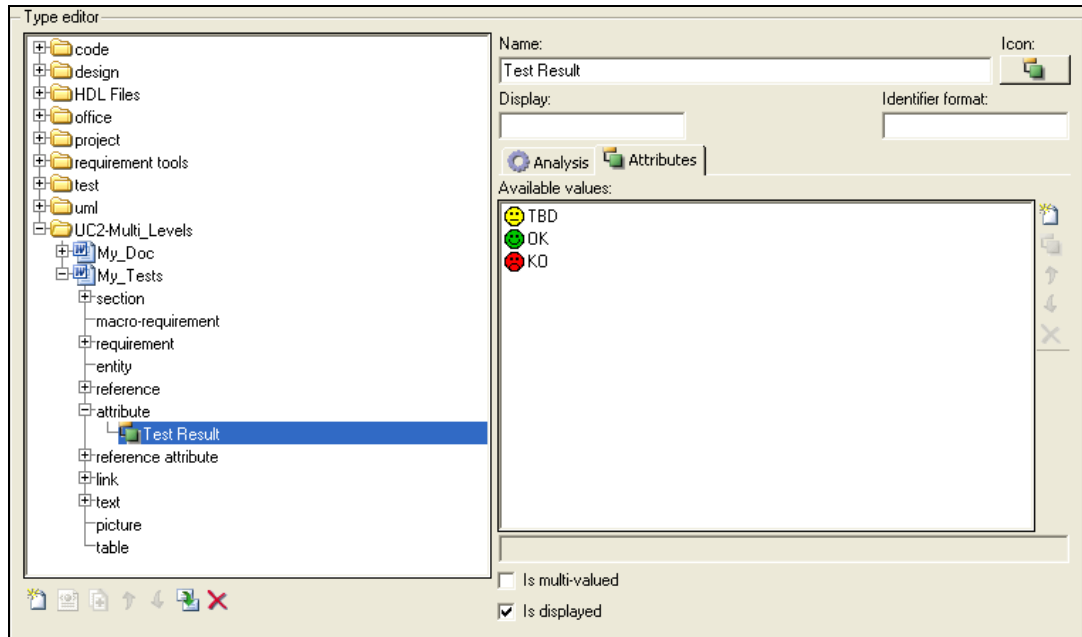
- ◆ The element to be pasted is defined in the **Display** field of the **Reference** element. If *Refers to : \$i* is the **Display** field value of the Implementation Reference element, then selecting the REQ_1 requirement and choosing **Copy For > Design Implementation**, will paste *Refers to : REQ_1* in the document.

For additional information and better understanding, see also:

- ◆ The [CopyFor.rqtf](#) project, a ready to use example demonstrating these concepts.

Defining Attributes

Attributes are used to give additional information containing requirements such as Priority, Safety criteria, Allocation, etc. These attributes can be Boolean (i.e. IsSafetyRequirement), or valued (i.e. Priority=high), or even contain several values (i.e. Allocation to several teams). All these concepts are supported by Rhapsody Gateway and are defined as **Attributes**.

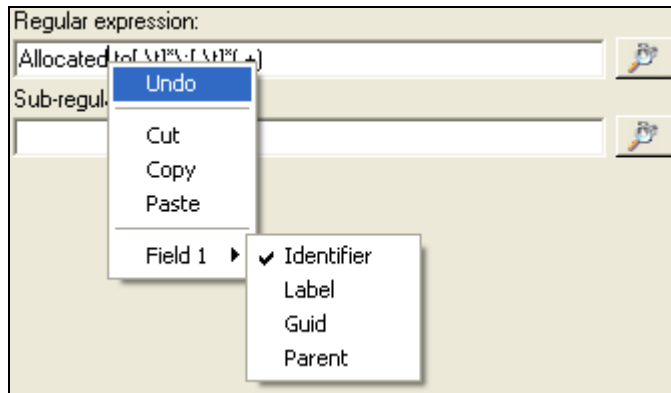


When you select an Attribute, the right side of the Types Editor contains an **Analysis** pane and an **Attributes** pane.

The **Analysis** pane has the following features:

- ◆ **Name**—Specifies the name of the Attribute. The name of the Attribute is displayed in a tooltip when the cursor is over that element in the main window of the tool.
- ◆ **Display**—Defines a display name for the Attribute in the project workspace. If this field is blank, the Attribute names of the source document are used as display names.
- ◆ **Identifier format**—Defines a new identifier for the Attribute. This identifier can use information provided by Regular expression. For example \1 and \2 represent first and second fields of the Regular expression. In the same way, \d can be used to prefix the identifier with the document name.
- ◆ **Regular expression**—Specifies the regular expression used to capture the attribute information in the source files (more precisely the intermediate file). This expression represents the beginning tag that identifies an Attribute.

A right click in the **Regular expression** field opens a context menu which contains a list of fields automatically generated according to the number of sub-expressions. Each field can be assigned as **identifier**, **label**, **guid** or **parent**.



- **Identifier**—is the attribute identifier.
- **Label**—is a field for labeling.
- **Guid**—is used to capture an internal identifier and is always unique.

Note

The GUID is the best way to identify precisely an element. It is possible to capture both GUID and identifier for any element where the GUID is the unique identifier from the original tool and identifier is the logical name that can be used for requirements traceability between other documents.






- **Parent** allows you to capture an identifier which will be the current element parent.

REQ1...		
REQ2...		
(...)		
Requirements	Attributes	...
REQ1	V1	...
REQ2	V2	...

Define **parent V1=REQ1** and **parent V2=REQ2** to avoid several definitions of requirements.

- ◆ **Sub-regular expression**—Specifies the sub-regular expression used to capture a second level of analysis. This process analyzes the result provided by the preceding regular expression.

Additional features for Attributes are contained in the **Attributes** pane:

- ◆ **Available values**—Lists the values allowed for the attribute. Use the  and  buttons to create your list of values. Use the  button to modify the icon associated with the attribute value. If Rhapsody Gateway captures an attribute value not declared in this list, the rule “Unauthorized value” is activated. To reorder the available values you can use the  and  icons.
- ◆ **Is multi-valued**—Allows the attribute to have several values defined in the list of Available values.
- ◆ **Is displayed**—Allows you to filter non visible attributes.

Example:

Your requirements standards define an attribute `Priority` which value can be `Low`, `Medium`, `High`.

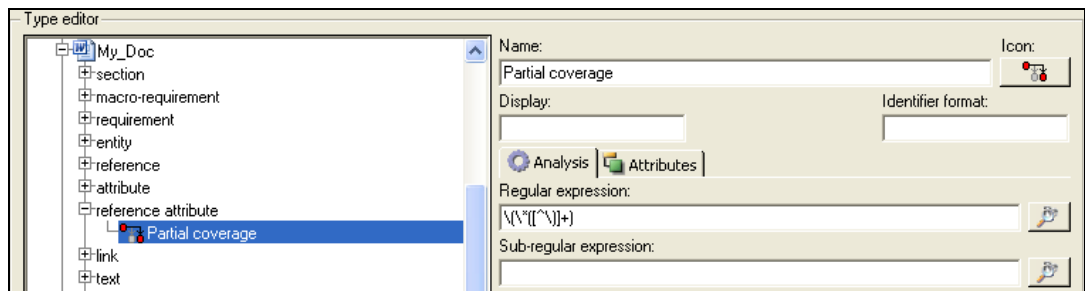
Several customization cases must be considered:

- It is possible to capture the attribute value using the regular expression: `Priority=(.+) .` In this case, anything inserted after `Priority=` will be captured, therefore typing errors or erroneous values may be captured without warning.
- It is possible to use the regular expression: `Priority=(Low|Medium|High) .` In this case, the priority value is captured ONLY if it is **Low**, **Medium** or **High**. If you have typing errors or other values, you will have no capture.
- It is possible to capture the attribute value using the regular expression: `Priority=(.+) .` and define **Low**, **Medium**, **High** as acceptable values using the list items. In this case, whatever is inserted after `Priority=` is captured, and you will see a warning message is the captured string is not **Low**, **Medium** or **High**. For this reason, this approach is recommended in the case of enumerated attributes.

Defining Reference Attributes

Reference attributes are used to give additional information concerning traceability links (references).

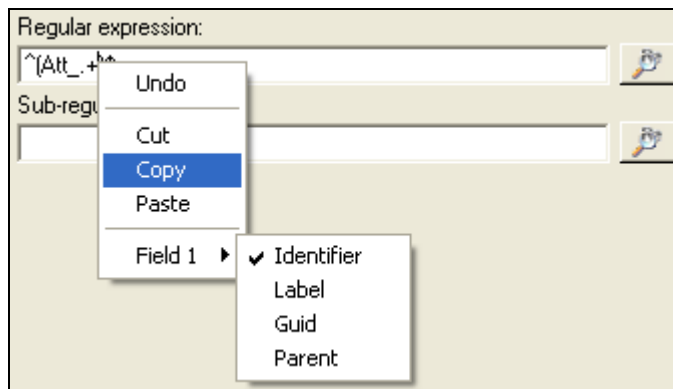
The *User Manual* describes how to use **reference attributes** and how they are displayed in the main window.



A **Reference attribute** has the following features:

- ◆ **Name**—Specifies the name of the Reference Attribute.
- ◆ **Display**—Defines a display name for the Reference Attribute in the project workspace. If this field is blank, the Reference Attribute names from the source document are used as display names.
- ◆ **Identifier format**—Defines a new identifier for the Reference Attribute. This identifier can use information provided by Regular expression. For example \1 and \2 represent first and second fields of the Regular expression. In the same way, \d can be used to prefix the identifier with the document name.
- ◆ **Regular expression**—Specifies the regular expression used to capture the reference attribute information in the source files (more precisely the intermediate file). This expression represents the beginning tag that identifies a Reference Attribute.

A right click in the **Regular expression** field opens a context menu which contains a list of fields automatically generated according to the number of sub-expressions. Each field can be assigned as **identifier**, **label**, **guid** or **parent**.



- **Identifier**—is the reference attribute identifier.
- **Label**—is a field for labeling.
- **Guid**—is used to capture an internal identifier and is always unique.




Note

The GUID is the best way to identify precisely an element. It is possible to capture both GUID and identifier for any element where the GUID is the

unique identifier from the original tool and identifier is the logical name that can be used for requirements traceability between other documents.

- **Parent** allows you to capture an identifier which will be the current element parent.
- ♦ **Sub-regular expression**—Specifies the sub-regular expression used to capture a second level of analysis. This process analyzes the result provided by the preceding regular expression.

Additional features for **Reference Attributes** are contained in the **Attributes** pane:

- ♦ **Available values**—Lists the values allowed for the reference attribute. Use the  and  buttons to create your list of values. Use the  button to modify the icon associated with the reference attribute value.
- ♦ **Is multi-valued**—Allows the reference attribute to have several values defined in the list of Available values.
- ♦ **Is displayed**—Allows you to filter non visible reference attributes.

For additional information and better understanding, see also:

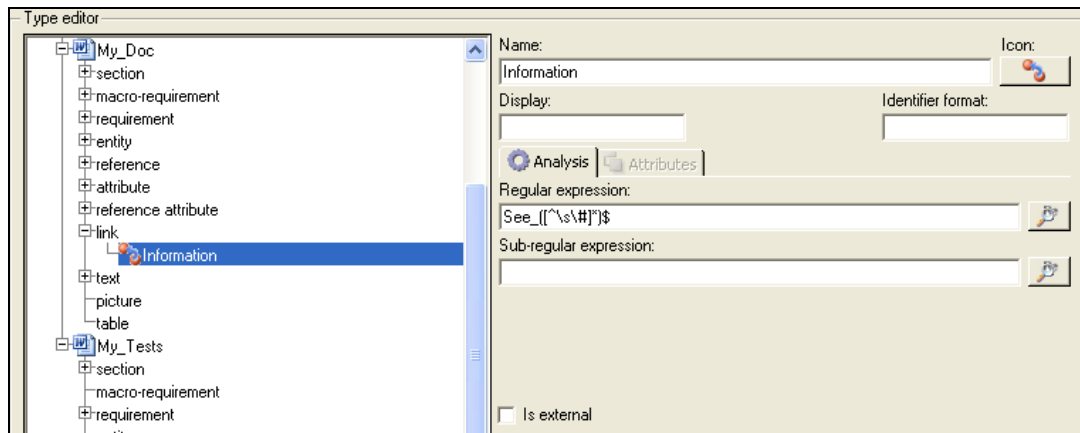
- ♦ The [Attributes.rqtf](#) project, an example about Attributes and Reference Attributes definition.
- ♦ The [Attributes demo](#), which presents the definition of Attributes and Reference Attributes.

Defining Links

Links elements are used to capture any relationship which does not represent traceability (coverage) between elements.

The *User Manual* describes how to use **links** and how they are displayed in the main window.

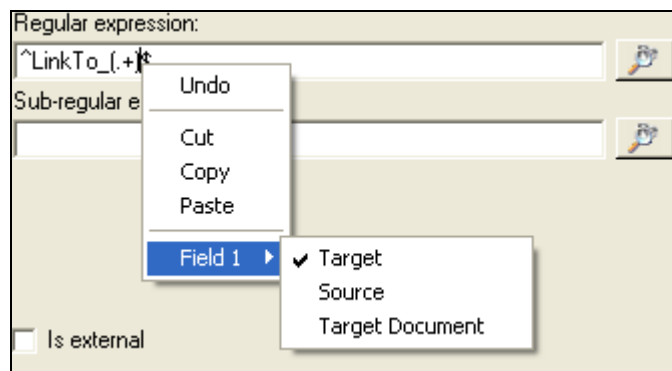
The target of a **Link** can be any kind of element: a requirement, but also a document, a section, etc.



A **Link** has the following features:

- ◆ **Name**—Specifies the name of the Link. The name of a Link is displayed in a tooltip when the cursor is over that element in the main window of the tool.
- ◆ **Display**—Defines a display name for the Link in the project workspace. If this field is blank, the Link names of the source document are used as display names.
- ◆ **Identifier format**—Defines a new identifier for the Link. This identifier can use information provided by Regular expression. For example \1 and \2 represent first and second fields of the Regular expression. In the same way, \d can be used to prefix the identifier with the document name.
- ◆ **Regular expression**—Specifies the regular expression used to capture the link information in the source files (more precisely the intermediate file).

A right click in the **Regular expression** field opens a context menu which contains a list of fields automatically generated according to the number of sub-expressions. Each field can be assigned as **target**, **source** or **target document**.



- **Target** is used to capture the target requirement identifier or GUID.
- **Source** is used to capture the source requirement identifier or GUID.
- **Target document** is used to capture the target document identifier.


- ◆ **Sub-regular expression**—Specifies the sub-regular expression used to capture a second level of analysis. This process analyzes the result provided by the preceding regular expression.
- ◆ **Is External**—Defines a link to be external. An external link is a link to an external requirement (example: http://...). External requirements are requirements defined externally such as an HTTP link or a requirement defined in a document which not belonging to the user. External requirements appear as undefined requirements in the upstream tree of the main Rhapsody Gateway window when they are covered by the current central selection.

Creating a link

The following figure shows an instruction to be captured using Links in the *Design* document:

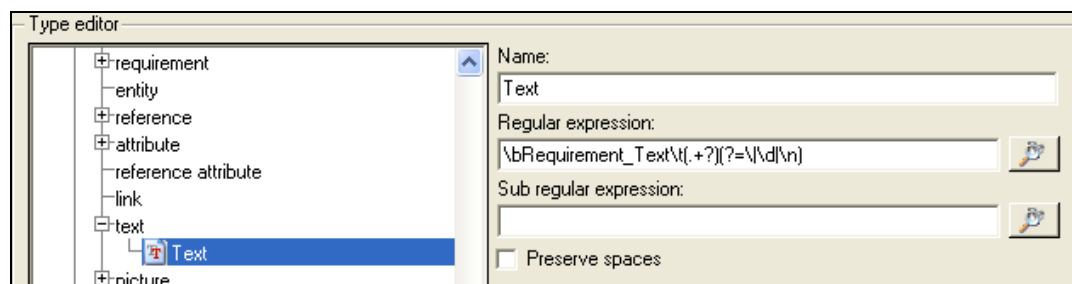
See_REQ_Specs1_(Specifications)

To define a Link to be captured, such as the Link above, follow these steps:

1. Select **File > Edit Types**. The Type editor opens.
2. Select the type to modify then select item Link.
3. Click the **Add new type** icon  on the toolbar.
4. Name the Link. For Regular expression enter:
See_(REQ_\S*)_\(([\^s\#]*)\)\\$
For the field1 choose **Target** and for the field2 choose **Target Document**.
5. Click Apply to apply your changes and reload the project.

Defining Texts

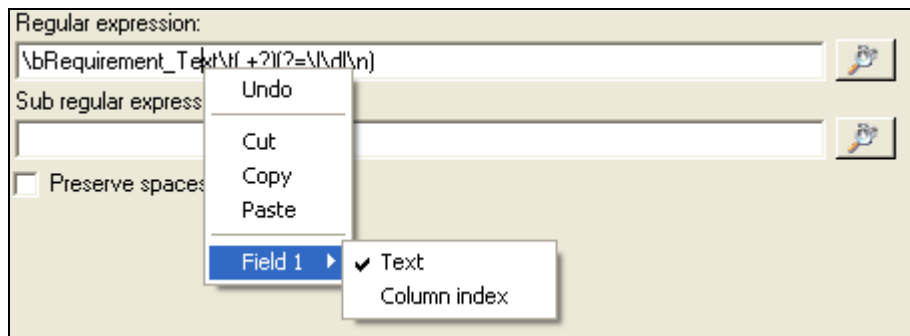
Texts elements are used to capture the wording of a traceability element. Rhapsody Gateway links a text to the elements such as a section, requirement, etc. detected immediately above it.



A **Text** has the following features:

- ◆ **Name**—Specifies the name of the Text.
- ◆ **Regular expression**—Specifies the regular expression used to capture the text information in the source files (more precisely the intermediate file).

A right click in the **Regular expression** field opens a context menu which contains a list of fields automatically generated according to the number of sub-expressions. Each field can be assigned as **text** or **column index**.



- **Text** is used to capture a text. If this text is placed in a table it becomes a cell.
- **Column index** is used to capture the column index text. Refer to *Defining Tables* section to capture the table.

Note

For a table cell capture, Column index is not mandatory.

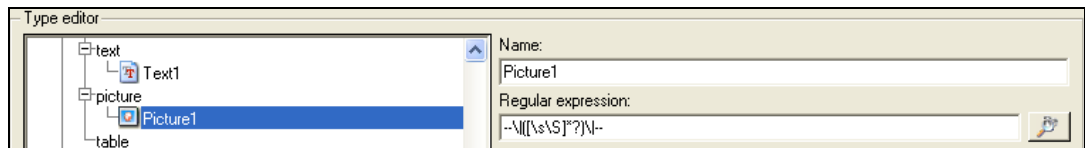
- ◆ **Sub-regular expression**—Specifies the sub-regular expression used to capture a second level of analysis. This process analyzes the result provided by the preceding regular expression.
- ◆ **Preserves spaces**—keeps separators at the beginning and at the end of the text.

Note

Also see the section about Definition of requirements. The text can be defined in a very effective way by using the allocation of fields.

Defining Pictures

Pictures are used to capture the image of a traceability element.



This element is displayed, but needs coding / decoding capabilities for each interface and/or image format. It is not supposed to be edited by the user.

A **Picture** has the following features:

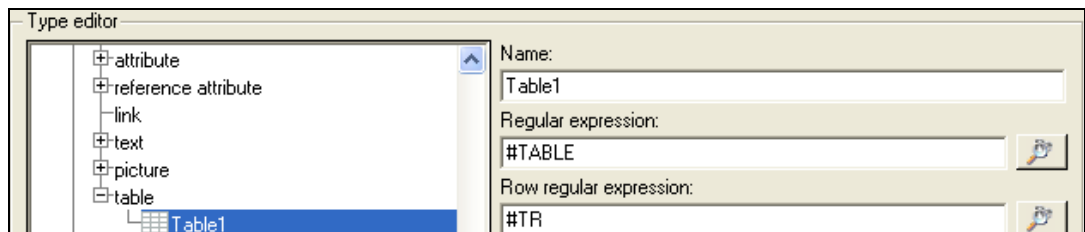
- ◆ **Name**—Specifies the name of the Picture.
- ◆ **Regular expression**—Specifies the regular expression used to capture the text information in the source files (more precisely the intermediate file).

Note

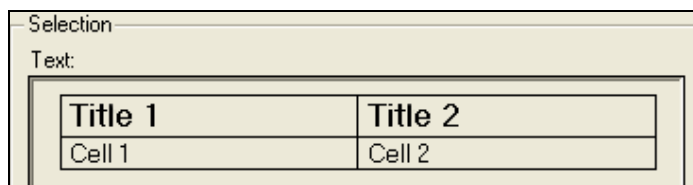
Rhapsody Gateway is able to handle diagrams and images from BMP, PNG, JPEG, WMF, EMF and SVG files format.

Defining Tables

Tables are used to capture the tables of a traceability element.



This element is displayed in the Text area as follows.



A **Table** has the following features:

- ◆ **Name**—Specifies the name of the Table type.
- ◆ **Regular expression**—Specifies the regular expression used to capture the beginning of a table in the source files (more precisely the intermediate file).

- ◆ **Row regular expression**—Specifies the regular expression used to identify each line beginning in the source files (more precisely the intermediate file). Refer to the *Defining Texts* section to capture cells.

Analyzing a table

Table can be analyzed and displayed for word processors. Follow these steps to analyze a table:

1. Define a table element in the type of analysis.
2. To define the table beginning, enter #TABLE value for **Regular expression** field. To define the line beginning, enter #TR value for **Row regular expression** field.
3. To capture the cell content, define a text element in the type of analysis and name it Cell.
4. Enter the following regular expression value `\| (\d+) \s [^\t]+ \t ([^\| \n] +)` for the **Regular expression** field of Cell.
5. Select the first couple of parentheses, check **Column index** in the contextual menu to define the group as the cell index. Select the second couple of parentheses, check **Text** in the contextual menu to define the cell content.
6. The analyzed table is displayed in the Text area.

Definition of Type Elements from XML Files

The XML types are used when the intermediate file created by Rhapsody Gateway is an XML file.

As reminder, the basic terms for XML are:

XML Tag

A **tag** is a generic name for an <element>. An opening **tag** looks like <element>, while a closing **tag** has a slash that is placed before the element's name: </element>.

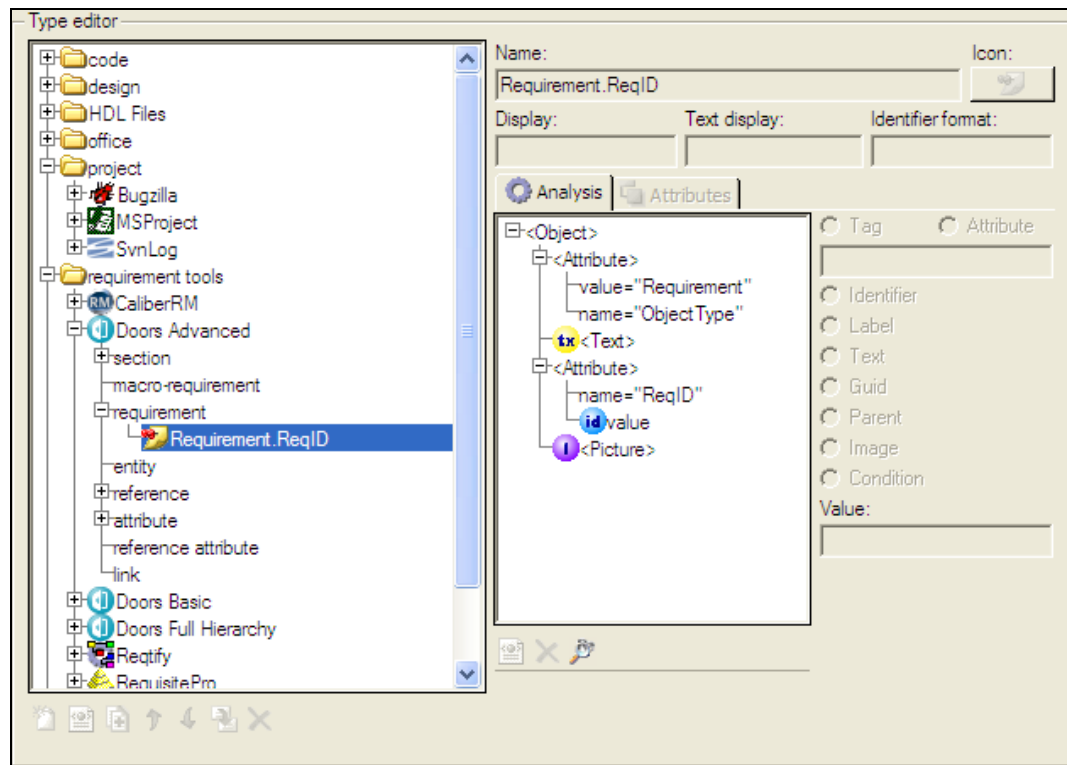
From now on, the term **tag** will refer to the opening of an element.

XML Attribute

Attributes are used to specify additional information about the element. An **attribute** for an element appears within the opening **tag**. The syntax for including an **attribute** in an element is:



```
<tag attributeName="value">
```

Each element of the type (Requirement, Attribute) has to be defined using the XML syntax, allowing you to capture the relevant information in the intermediate file.

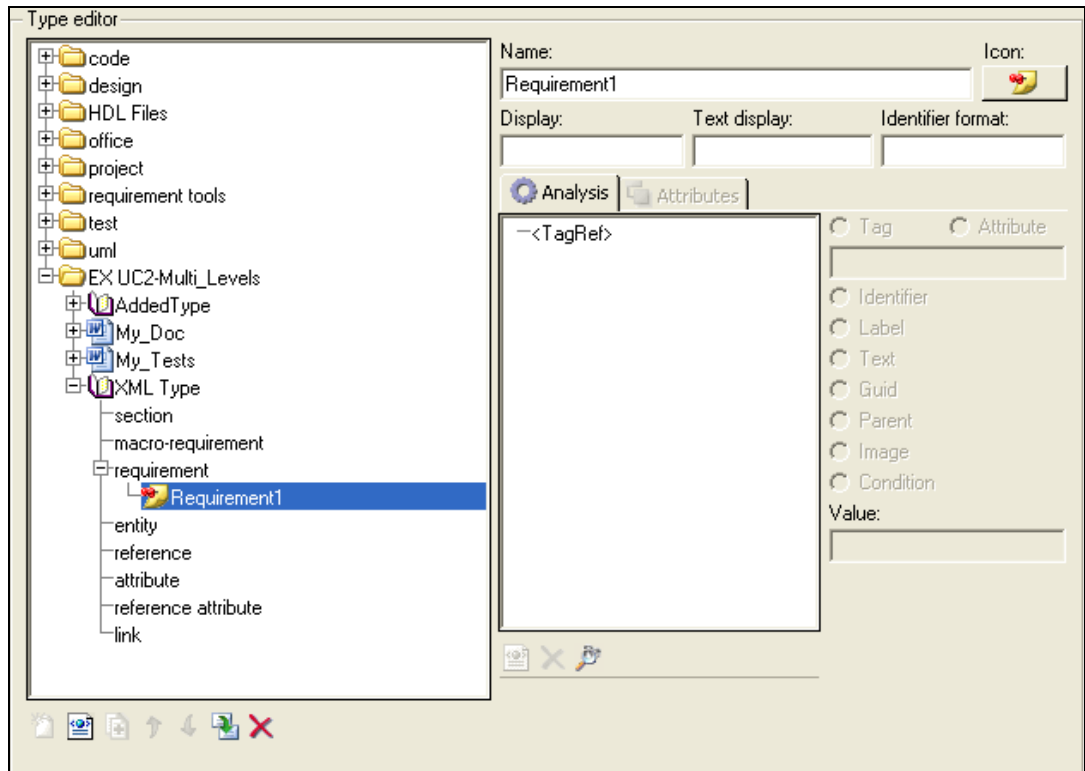


Creating a New XML Type

To create a new XML type of analysis, follow these steps:

1. In the Configuration Types, click . (You can also duplicate an existing XML type and edit it.)
2. To create an element for this new XML type, select a section of the tree then click . The Analysis Tab is filled with another element.

The new XML type and element are created as follows:

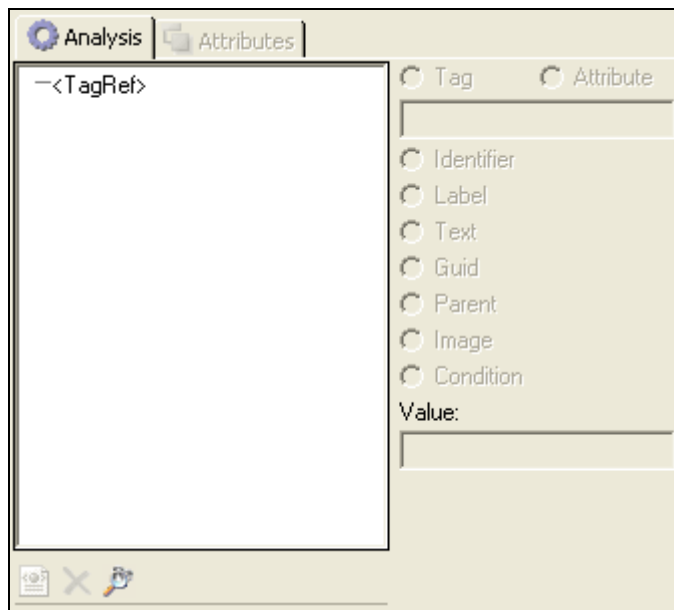


Creating the XML Structure

Each element of the XML type has the following features:

Name and **Display** fields are equivalent to the ones defined for the non-XML elements.

- ◆ **Name**—Specifies the name of the element. The name of the element is displayed in a tooltip when the cursor is over that element in the main window of the tool.
- ◆ **Display**—Defines a display name for the element in the project workspace. If this field is blank, the element names of the source document are used as display names.
- ◆ **Text display**—Defines a text which is displayed in the Text area when an element is selected.
- ◆ **Identifier format**—Defines a new identifier for the element.
- ◆ **Analysis area**—Defines the XML syntax you want to analyze.

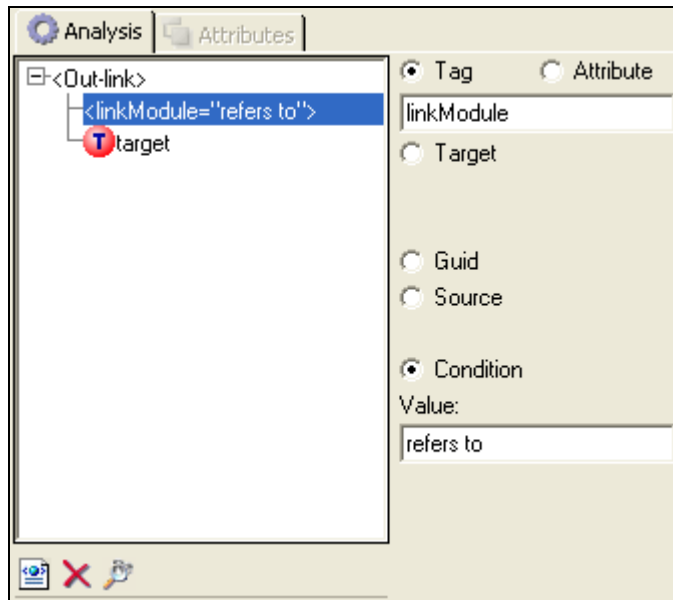


Options of the **Analysis** area have the following behavior:

- —Adds a new XML element underneath the one selected in the hierarchy.
- —Deletes the selected element.
- **Tag / Attribute**—This checkbox allows you to choose between a Tag or an Attribute. The root can only be a Tag. Attribute is active only for lower-level elements.
- **Tag_name /Attribute_name**—Specifies the name of the selected Tag / Attribute.
- **Value**—This field is active if one of the following XML element is checked. You can define a regular expression to extract a subset of the XML capture as an advanced definition of your condition.
- **Identifier (id)**—The result of the capture is used as the **identifier (ID)** of the element.
- **Label (lb)**—The result of the capture is used as the **label** of the element.
- **Text (T)**—The result of the capture is used as the text of the element.
- **Guid (G)**—The result of the capture is used as the **GUID** of the element. The GUID (Global Universal Identifier) uniquely identifies the element.
- **Parent (P)**—The result of the capture is used as the **parent** of the element.
- **Image (i)**—The result of the capture is used as the **picture** of the element.
- **Condition**—Used to detect the match of an XML element or attribute with the specified Value.

As an example, if ID is Bug_344200, to only capture the bug number you have to fill the **Value field** with Bug_(\d+). Parentheses are used to extract a subset of an identifier in Identifier, Label, Text, Guid or Parent fields.

According to the type element to define the proposed XML elements in the Analysis area are different. Thus, for links and references elements only **Target**, **Guid**, **Source** and **Condition** are available.



- **Target (T)**—The result of the capture is used as the **target element** to which the link is created.
- **Guid (G)**—The result of the capture is used as the **GUID** of the element. The GUID (Global Universal Identifier) uniquely identifies the element.
- **Source (S)**—The result of the capture is used as the **source** of the element.
- **Condition**—Used to detect the match of an XML element or attribute with the specified Value.

By default the source element is the analyzed element. Checking the **Is inverse** checkbox allows you to go up or down in the traceability graph. This option enables you to switch between Source and Target elements.

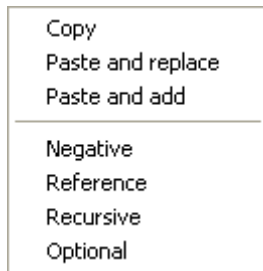
Building an ident

You can build an ident from multiple identifier fields of the XML tree.

In the **Identifier format**, you can construct this ident by combining separators and identifier field called using \n. One or several identifier fields can be captured when Identifier was selected for the XML attribute or node. The field numbering comes from the depth-first search in the XML tree. An identifier field number is assigned for the whole value when the condition is empty; otherwise an identifier field number is associated to each parenthese group of the value condition.

XML Structure Options

A contextual menu is available on the Analysis area:



- ◆ **Copy**—copies the subtree into the clipboard.
- ◆ **Paste and replace**—replaces the selected node by the clipboard tree.
- ◆ **Paste and add**—adds the clipboard tree as a sibling for the selected node.
- ◆ **Negative**—allows you to match, this subtree must be absent.
- ◆ **Reference**—defines this tag as a search start. Useful when you need to match several tags with the same name.

For example, to match TNC under a specific tag name TNP, when we have the following trees:

```
XML tree:
TNP
+--- TNC
+--- TNC
```

```
XML analysis tree:
TNP
+--- TNC
```

If no **reference** is defined, it will only match one TNC.

If a **reference** is defined on TNC node, all TNC will be matched.

- ◆ **Recursive**—when this flag is defined, the extracted string is composed with flagged tag text contents and all other children tags text contents.

For example, in the sentence `<Text>The black <i>fat</i>cat</Text>`

If **recursive** flag is not set, only 'The cat' will be returned.

If **recursive** flag is set, 'The black fat cat' will be returned.

- ◆ **Optional**—when this flag is defined on a branch, the corresponding branch is defined as optional.

Creating Types for Added Elements


You can define a Type for an added element if you want to provide project users with advanced capabilities of information creation from Rhapsody Gateway.

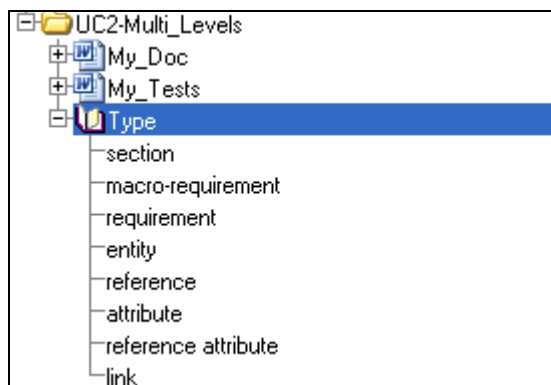
- ◆ Advanced definition of Boolean, Value and Enumerated attributes.
- ◆ Several kinds of Coverage links.
- ◆ Non-coverage links.

Once **Types for added elements** have been defined such as below, they can be re-used for all projects (like for the other types files).

You can use the `Added-Information-Advanced` example installed with Rhapsody Gateway to train yourself with these principles.

To create a new **Type for added elements**, follow these steps:

1. Click the  button in the **Types Editor**.
2. A new type is added at the end of your own types list.




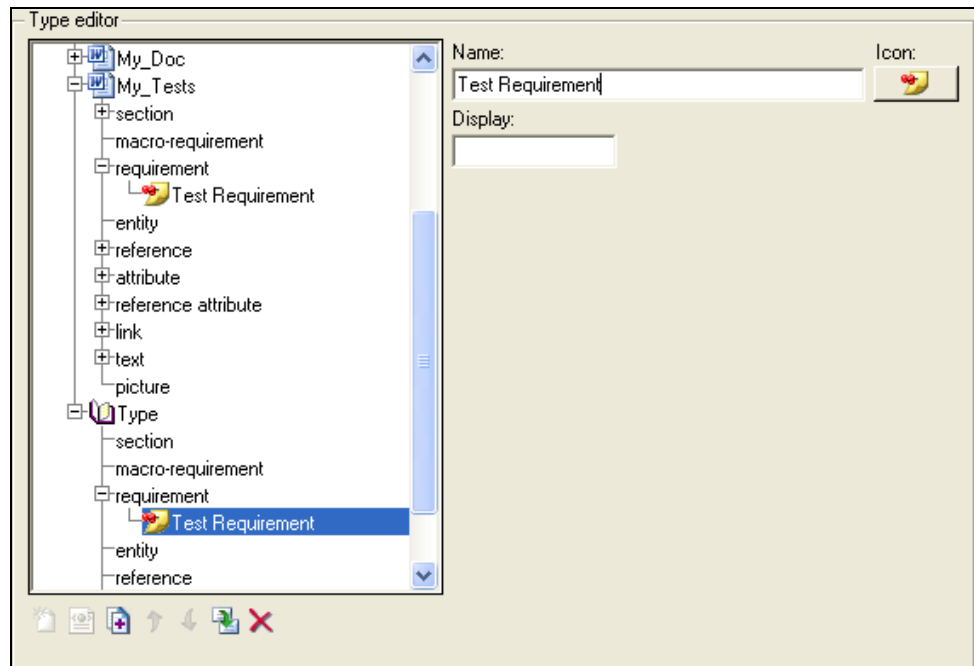
3. Name your type.
4. Now you have to complete your type declaration before affecting it to a modification file.
 - ◆ The first step is to declare the **Requirements** on which you want to authorize added elements.
 - ◆ Then define at least one type between **References**, **Links** or **Attributes** for this new added type.

Declaration of Requirements to Be Modified

First, you need to declare the requirements for which the user can add elements.

In your **Type for added element**, create a new type of requirement as follows:

1. Select requirement in the type then click .
2. A new type of requirement is added.
3. Name this type of requirement with **the same name** used for the requirement in the type applied to the project document.



For instance, if you defined a customized type “My_Tests” containing a requirement “Test Requirement”, you need to create a requirement also named “Test Requirement” in the **Type for the added element** as well.

Rhapsody Gateway uses this definition to activate / deactivate the additional features when a requirement is selected in the project workspace.

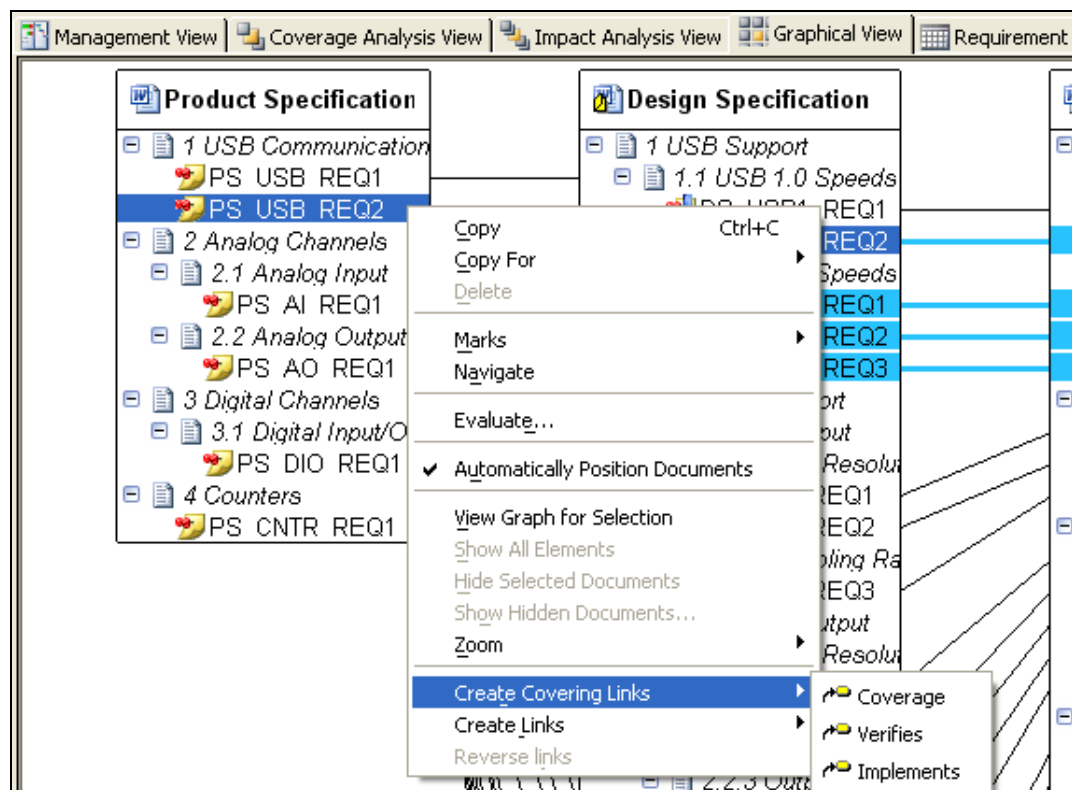
Creating References

In your **Type for added elements**, create a **Reference** and name it using the **Name** field. The **Display** Text field is a provision, not used yet.

The **Name** value is displayed in the context menu when the project user creates covering links from the graphical view.

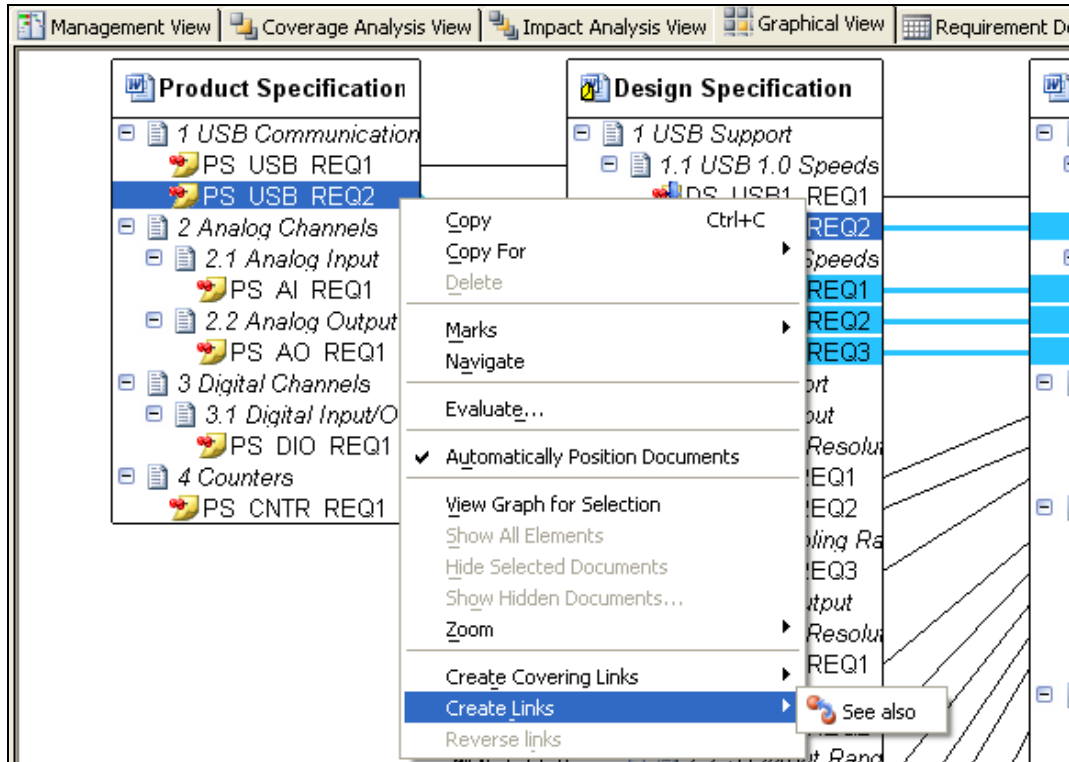
Note

You can change the **Name** of the reference only if the information has not been created yet for projects from Rhapsody Gateway. Rhapsody Gateway creates and analyzes created information using this name and added information can become inconsistent if you rename your reference element.



Creating Links

In your Type for added element, create a **Link** and name it using the **Name** field. The **Name** value is displayed in the context menu when the project user creates links from the graphical view.

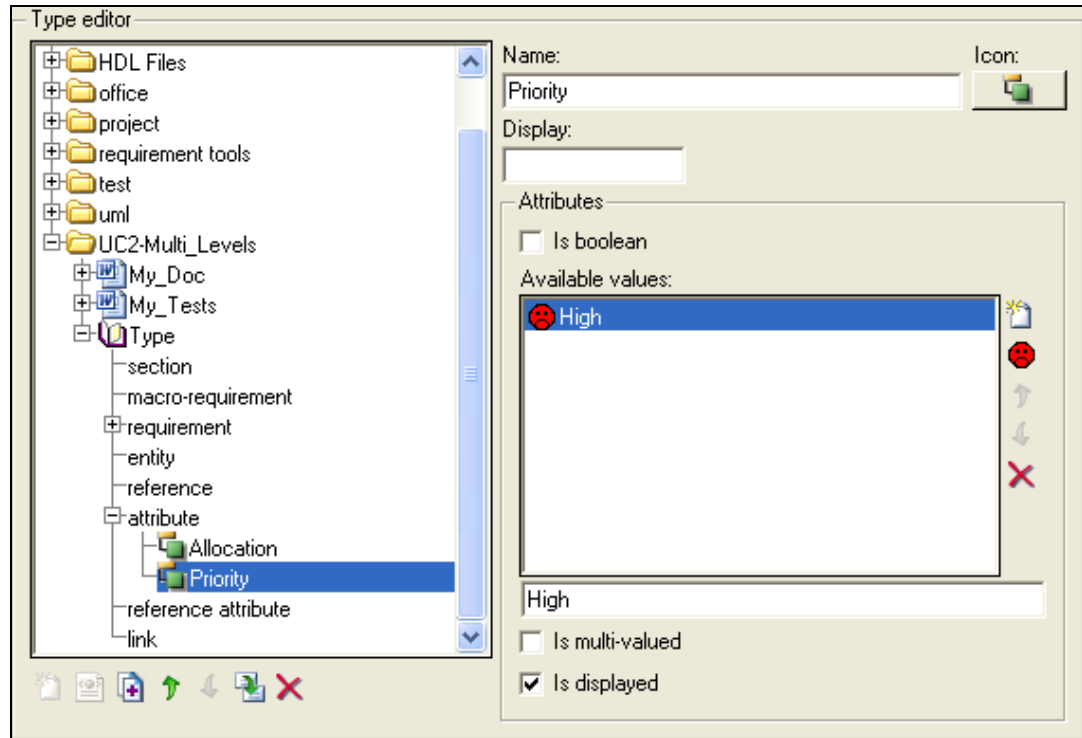


Note

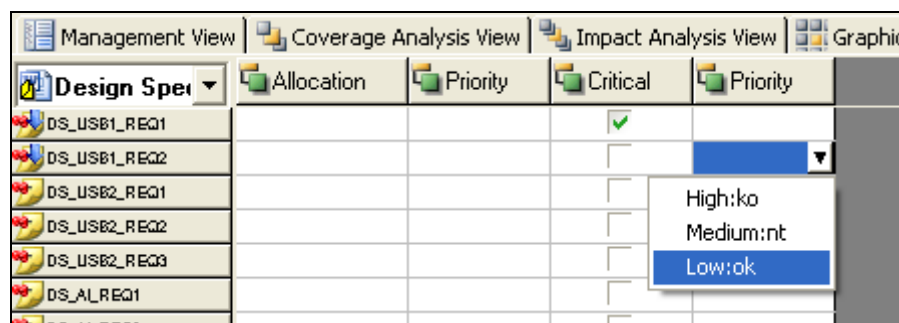
You can change the Name of the Link only if the information has not been created yet for projects from Rhapsody Gateway. Rhapsody Gateway creates and analyzed created information using this name and added information can become inconsistent if you rename your Link element.

Creating Attributes

In your **Type for added elements**, create an **Attribute** and define it as explained for the other Attributes.



Attributes will be available from the **Requirement Details view** as below, and need to be defined in this view.



Note

You can change the Name of the Attribute only if the information has not yet been created for projects from Rhapsody Gateway. Rhapsody Gateway creates and analyzes created information using this name and added information can become inconsistent if you rename your Attribute element.

For additional information and better understanding, see also:

- ◆ The [AddedElements.rqt](#) example project.

Frequently Used Regular Expressions

Rhapsody Gateway uses *Perl Regular Expressions* to catch the relevant information in the intermediate textual files. These Regular Expressions enable Rhapsody Gateway to manage a wide set of requirements standards and define multiple ways to identify the requirements and other elements.

Modeling and simulation tools usually do not need such a level of customization and default types in order to support most of the cases involving these tools.

Expressions for customization may look complex or difficult to read. However, if you express your capture request using natural language before trying to write the expression itself, the expressions will be simple to understand. Once you define your capture condition using natural language, transpose the keywords into expressions using this library of frequently used expressions. The **Regular expression Tester** in the next chapter is helpful in writing regular expressions.

Wildcard Meaning

The following key points will allow you to match most expressions. For advanced cases, contact the Support Team who can give you the expression you are looking for.

Matching Explicit Strings

The easiest way to match a fixed string is to write the string itself.

For example, to analyze REQ_1, REQ_2, the corresponding fixed string is REQ_.

Matching Character Sets

The following character escapes allow you to match variable characters:

- ◆ \d corresponds to a digit character.
- ◆ \w corresponds to word characters (letters, digits and alphanumeric _).
- ◆ . (dot) corresponds to any single character except line break characters.

Evaluating Occurrences

The following standard quantifiers recognize the different occurrences of characters or sub-expressions:

- ◆ `*` indicates 0 or more occurrences.
- ◆ `+` indicates 1 or more occurrences.
- ◆ `?` indicates 1 or 0 occurrences.

Specifying Location

The following boundary indicators identify the location of a matched sub-expression:

- ◆ `^` (caret) indicates the beginning of the line.
- ◆ `$` (dollar) indicates the end of the line.

Capturing Sub-expressions

To be captured as results, subparts of expressions have to be grouped between brackets:

- ◆ `(...)` allows you to delimit a group. Captured groups are numbered by counting their opening parenthesis from left to right.

In Rhapsody Gateway, a group is also called a **field**. You can define expressions containing several **fields**. See Defining Sections for more details on fields. Sub-expressions of groups can also be analyzed. See Defining Requirements to have more details on sub-expressions.

Specifying Alternatives

The following alternative character matches one of a choice of regular expressions:

- ◆ `|` allows you to create an alternation.

The alternation operator has the lowest precedence of any regular expression operator. Thus, the only way you can define its arguments is to use grouping.

For example, if `TestResult=(OK|KO)`, then `(OK|KO)` would match either OK or KO.

Identifying Separators

The following separator characters are recognized:

- ◆ `\s` matches a white space character. It is helpful for Word and more general than simply typing the space character because Word suggests, and often applies automatically, several types of “spaces”.

- ◆ `\s` matches a non-white space character. It means “no separator” is attempted.
- ◆ `\t` matches a tab character. This is often helpful because it separates the Word style, as the Word converter inserts a tab character between the style name and the text.
- ◆ `\n` matches a newline character.

Assertions

Another trick of advanced regular expression tools is ‘**look ahead**’ assertions and ‘**look behind**’ assertions. These are similar to regular grouped subexpression, except they do not actually grab what they match. Assertions purpose is to test the preceding and the following characters.

There are two kinds of ‘look ahead’ assertions: positive and negative. As you would expect, a positive assertion specifies that something *does* come next. A negative assertion specifies that something *does not* come next.

- ◆ `(?=pattern)` corresponds to a zero-width **positive** ‘look ahead’ assertion.
For example `/w+(?=t)/` matches a word followed by a tab, without including the tab. (i.e. a right match is found)
- ◆ `(?!pattern)` corresponds to a zero-width **negative** ‘look ahead’ assertion.
For example `/foo(?!bar)/` matches any occurrence of 'foo' that is not followed by 'bar'. (i.e. a right match is not found)

‘Look behind’ assertions may only look backwards by a fixed number of character positions. There are also two kinds of ‘look behind’ assertions: positive and negative. The former assures that a pattern *does not* precede the match. The latter assures that the pattern *does* precede the match.

- ◆ `(?<=pattern)` corresponds to a zero-width **positive** ‘look behind’ assertion.
For example `/(?<=t)w+/` matches a word following a tab, without including the tab. (i.e. a left match is found)
- ◆ `(?<!pattern)` corresponds to a zero-width **negative** ‘look behind’ assertion.
For example `/(?<!bar)foo/` matches any occurrence of 'foo' that is not following 'bar'. (i.e. a left match is not found)

Avoid matching some words

A way not to match lines containing specific keywords is to use an assertion.

If we do not want to match lines followed by the word `AWORD`, type this expression `(?!AWORD)`. The exclusion does not work correctly without using `\b`, so it is better to type `\b(?!s+AWORD)`.

Note

`\b` is useful to specify a word end. It is a mute character so it has no effect except indicating the characters group ending.

Matching Special Characters

Some characters have special meanings. Thus, in order for `() ^ | + * []` to be recognized, as inputs instead of special characters they need to be escaped.

- ◆ `\` (backslash) before special character escapes special characters to suppress their special meaning.

For example, to match an opening bracket, type `\(`.

Excluding Matching of Some Characters

In order to match any character, except some listed characters, you must do the following:

- ◆ `^` (caret) immediately after the opening `[:^characters]` negates the character set and matches all but the contained **characters**.

For example, to specify a formalism as “a fixed string `REQ_`, followed by 1 or several characters except a space or a comma”: `(REQ_[^\s\,]+)`.

Matching Information from a Table

The conversion of tables into intermediate files depends on source files.

- ◆ In a conversion from Word, the table conversion looks like:
`|<column number> <word style> <Column content>`
 where the character `|` indicates the beginning of a table description.
 An example of an intermediate file is
`|1 Normal REQ_1 |2...`
 an example of a Regular expression is
`\|1 Normal (REQ_\d+)`.
 This means that in first column the user wants to capture the fixed string `REQ_` followed by one or several digits.
 (See the Coupling Word note for more details).
- ◆ In a conversion from Excel, the table conversion looks like:
`|<column number> <Column content>`
 where the character `|` indicates the beginning of a table description.
 An example of an intermediate file is
`|1 REQ_1 |2...`
 an example of a Regular expression is
`\|1 (REQ_\d+)`
 This means that in first column the user wants to capture the fixed string

REQ_ followed by one or several digits.
(See the *Coupling Excel* note for more detailed description).

Capturing Multi-lines

One advanced point that is helpful is to match several lines:

- ◆ `[\s\S]*` allows you to match information on several lines.

In the default types of analysis, this expression is often used for Text capture expressions.

Examples of ? (question mark) Usage

The character `?` can be helpful in two defined ways.

Expressing an alternative without capturing expression as a result

Brackets are either used as capturing groups or as alternative groups. For most of the alternative groups, it is not useful to catch the alternative as a result.

Using `?:` allows you to avoid this capture. See the following examples:

- ◆ `TestDuration=(\d+) ms - Result=(?:Passed|OK)`
In this analysis of a test log file, your goal is to capture the test duration value if the test result value is **OK** or **Passed**.
This alternative condition could be expressed by `(OK|Passed)`. However, capturing this sub-expression is not very useful. To avoid this result use `?:` as in the following example: `(?:Passed|OK)`
- ◆ `(REQ_(?:FUNCT|PERF)_\d+)`
In this analysis of a requirement tag, your goal is to capture only Functional requirements REQ_FUNCT_xx and Performance requirements REQ_PERF_xx. Items REQ_FUNCT_xx or REQ_PERF_xx are of interest whereas FUNCT or PERF are not. Once again, the purpose is to express an alternative without capturing.

Note

This second example is just for tutorial. For such a need we recommend defining two kinds of Requirement items in your type of analysis; One for Functional requirements (REQ_FUNCT_\d+) and another for Performance requirements (REQ_PERF_\d+).

Managing the scope of an analysis

It is possible during analysis to find several occurrences of a given regular expression. A ? allows you to restrict the scope of analysis.

The figure below shows an Excel table that must analyze:

Requirement	Value	Priority Level	Comment
REQ: Max Response Time	10 ms	A	
REQ: Max Output level	6 V	A	5V +/- 20%

The formalism is REQ: followed by a label in natural language. The only formalism that can be specified is that the information is in the first column.

This analysis will apply on the table converted from Excel:

```
| 1 REQ: Max Response Time | 2 10 ms | 3 A
```

If we use `\| 1 (REQ: .+)` the analysis perimeter will be:

```
| 1 REQ: Max Response Time | 2 10 ms | 3 A
```

The scope is too large for our need. This is because the analyzed information contains several occurrences of the `|` character.

Using `\| 1 (REQ: .+?)` limits the analysis to the first occurrence of `|`:

```
| 1 REQ: Max Response Time | 2 10 ms | 3 A
```

Brackets specify that in this string you simply want:

```
| 1 REQ: Max Response Time |
```

Note

You could also use the expression `\| 1 (REQ: [^\|]+)` to show what you need. The meaning is “`\| 1` for the first column, string `REQ:` followed by any character except the `[^\|]` to be the mark for the next column `\|`, and then the `+` to mean several times.

Frequently Used Regular Expressions

This section presents a summary of the most commonly used regular expressions.

The first column lists text to be matched. The second one gives some proposals of regular expressions or sub expressions that can match the given text.

Input text	Matching regular expressions
REQ_1, REQ_2,... (fixed string then digits)	<code>(REQ_\d+)</code>
REQ_FUNCT_1 (fixed string then digits) REQ_PERF_1 (other fixed string then digits) The IDs integrate an idea of category, which may lead you to consider several kinds of requirements. Also the category becomes additional information.	Create two requirements elements in your type: <ul style="list-style-type: none"> • One with <code>(REQ_FUNCT_\d+)</code> • Another one with <code>(REQ_PERF_\d+)</code> Note that, in such a case, you can still perfectly define an attribute for requirements by capturing the information included in the ID. You can achieve this with an alternative: <ul style="list-style-type: none"> • Attribute Category using the expression <code>REQ_(FUNCT PERF)_\d+</code>, or • Attribute Functional using the expression <code>REQ_(FUNCT)_\d+</code> • Another attribute Performance using the expression <code>REQ_(PERF)_\d+</code>
SRS_LED-AA-Product_123 (fixed string SRS_ terminated by a _nn, but with undefined characters between SRS_ and _nn) If you know that there is no space or no separator in these characters.	<code>(SRS_\S+_\d+)</code>
If you do not know if there are spaces or separators in these characters.	<code>(SRS_.+?_\d+)</code>
You can also use a general expression, if you know that SRS_ is followed by digits, letters or “_” (but no space, no special character, etc.)	<code>(SRS_\w+)</code>

Attributes examples	Matching regular expressions
<ul style="list-style-type: none"> Extract of a Test log file Test: Scenario1 - Result = OK Test: Scenario2 - Result = KO 	<p>An attribute Result with <code>Result = (OK KO)</code></p> <p>or</p> <p>an attribute Passed with <code>Result = (OK)</code> and an attribute Failed with <code>Result = (KO)</code></p>
<ul style="list-style-type: none"> Extract of a Specifications document Allocation = xxx Priority = High 	<p>An attribute Allocation with <code>Allocation = (.+)</code></p> <p>An attribute Priority with <code>Priority = (High Medium Low)</code></p> <p>For attribute Priority you can also use multi-value capabilities.</p>


Coverage links examples	Matching regular expressions
[Covers <OneReqID>]	<code>\[Covers ([^\]]+)\]</code>
Covered Requirements: REQ1, REQ2,...	<p><code>Covered Requirements: (.+)</code></p> <p>+ Sub regular expression: <code>(REQ\d+)</code></p>
Covered Requirements: REQ1, REQ_A_2,...	<p><code>Covered Requirements: (.+)</code></p> <p>+ Sub regular expression: <code>(REQ\w+)</code></p>

Testing Regular Expressions

The **Regular expression tester** is provided to help you to test and validate the regular expressions you define for the types.

Accessing the Regular Expression Tester

To access the Regular expression tester directly from the Configuration dialog box click the Expressions button on the left side of the pane.


The tester can also be reached by clicking the  button close to the expression fields. In this case, the tester opens with the fields already filled with the defined regular expression.

The Regular expression tester dialog box is displayed as follows:



The Regular expression tester is composed of the following areas and buttons:

- ◆ **Text analysis area**—Specifies the text to analyze. The tests are going to be executed based on the contents of this text. This field can be filled using an import or manually.
- ◆ **Regular expression**—Specifies the regular expression to be tested on the text contents.
- ◆ **Sub expression**—Specifies a sub regular expression to be evaluated with the result provided by the regular expression.

- ◆ **Test**—Launches the test i.e. tries to match the regular expressions in the text to be analyzed.
- ◆ **Update**—Updates the field from which the Tester has been launched with the regular expression value. This is only available if the Tester has been launched using the  button.
- ◆ ¶—Displays or hides the hidden characters such as space ., end of line ¶ or tabulation ↵.
- ◆ A displaying area—Will either displays a messages indicating there is no correspondence or display the table of matching results.

Using the Regular Expression Tester

Even if the **Text analysis area** can be filled manually, the general usage is to import the source information into the tester. Rhapsody Gateway makes it easy for you to import the intermediate file with which regular expressions can capture the traceability elements.

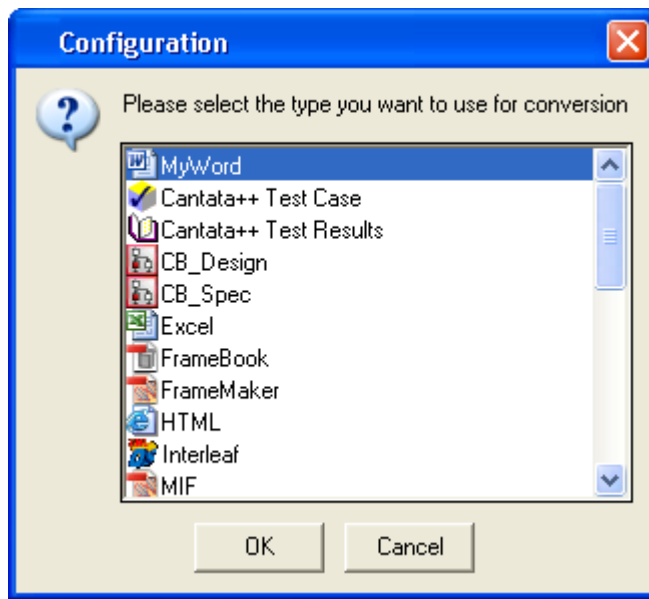
Use the File option to access the following commands for making imports:

- ◆ **Import file**—Imports source information from a file such as Word, or PowerPoint.
- ◆ **Import directory**—Imports the analysis of a directory such as Code, or MultiWord.
- ◆ **Import intermediate file**— Imports the analysis of a file already converted.

Importing a File

To import a file into the tester, follow these steps:

1. Select **File > Import** file option. The Configuration dialog box opens, as shown in next item.
2. Use the drop-down list to choose the type to convert.



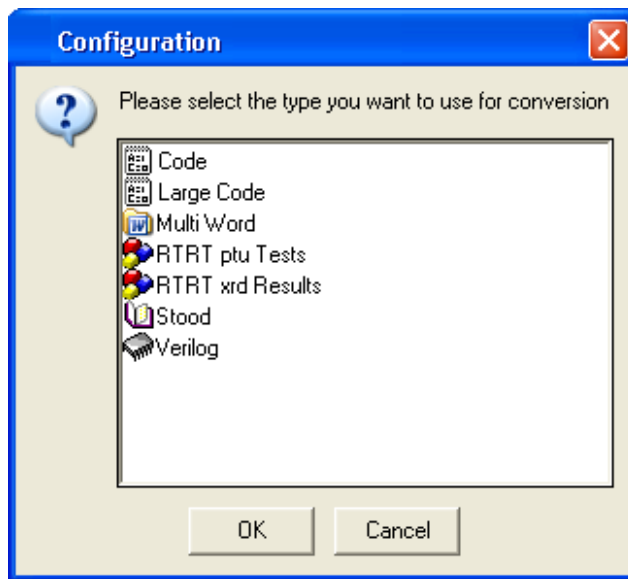
In this example, myWord type is based on the default Word type.

3. Click **OK** to apply the type you have chosen and open a new selecting dialog box.
4. Select the file to analyze and validate.
5. The **Text analysis area** is filled out with the intermediate file converted from the selected file.

Importing a Directory

To import directory contents into the tester, follow these steps:

1. Select **File > Import** directory option. The Configuration dialog box opens, as shown in next item.
2. Use the drop-down list to choose the type to convert.



3. Click **OK** to apply your choice and open a Browse for Folder window.
4. Select the directory which contains the files to be analyzed.
5. A new dialog box opens, listing the files located in the directory.
6. Select the file to analyze and validate.
7. The Text analysis area is filled with the intermediate file converted from the selected file.

Importing an Intermediate File

To import an intermediate file into the tester, follow these steps:

1. Select **File > Import** intermediate file option. The Open dialog box is displayed.
2. In the Look in field, browse to the location of the intermediate file.
3. Select the .txt file, or type the name of the intermediate file in the File name field.
4. Click Open. The intermediate file opens filling the Text analysis area of the tester.

Note


Non-file based types such as DOORS are not displayed by the **File > Import** option. For these types you need to work on the intermediate file and use the **File > Import intermediate file option**.

Testing Regular Expressions

Once a file has been added into the **Text analysis area**, regular expressions can be tested in the imported file contents.

1. Type your regular expression in the Regular expression box, and if necessary a sub regular expression.

Note

These fields are automatically filled if you launched the regular expression tester using the  button.

2. Click on the **Test** button.
3. The matching analysis between the regular expressions and the intermediate file is launched.
 - ◆ If your expression does not match anything in the analyzed text, the message "Text doesn't contain any correspondence with the regular expression" will appear.
 - ◆ If your expression matches elements of the analyzed text, the tester displays these results:

Regular expression tester


Text analysis zone:

```


BEGIN_DIRECTORY: C:\Users\F57\Documents\Reqify\2011-2a\examples\coupling\ControlBuild\Traceability:Traceability
BEGIN_FILE: C:\Users\F57\Documents\Reqify\2011-2a\examples\coupling\ControlBuild\Traceability
\requirements.doc:requirements.doc

Normal ControlBuildTM basic example
Normal Specifications document
Normal
Normal Note : The aim of this document is only to show the principles of a coupling with ControlBuild. It is a basic
specifications document built using the components "Clock_Tri" example provided by Geensys with ControlBuildTM.
Normal
Requirement_ID CDC_1 : Aims of the product
Requirement_Text The product must display the hour, and the air pressure
Normal
Requirement_ID CDC_2 : Time Measurement
Requirement_Text The time measurement is doing by a quartz system.
        
```

Regular expression: Sub expression:

Test Update 

	Line	Group 1	Group 2	Group 3	Group 4
1	9		CDC_1		: Aims of the product
2	12		CDC_2		: Time Measurement
3	15		CDC_3		: Command
4	18		CDC_4		: Display
5	21		CDC_5		: Date option
6	24		CDC_6		: Light option


4. The tester gives the following results:
 - ◆ **Line**—The line number where the text is detected.
 - ◆ **Group1**—The first captured field, corresponding to the first part of the regular expression you defined between brackets.
 - ◆ **Text**—The full text analyzed by the regular expression.
5. Click  button, if some characters are hidden they are displayed in red.

	Line	Group 1	Group 2	Group 3	Group 4	
1	9	-	CDC_1		: -Aims of the product	Re
2	13	-	CDC_2		: ...Time Measurement	Re
3	16	-	CDC_3		: Command	Re
4	19	-	CDC_4		: Display	Re

Click the number at the beginning of the row to directly access the corresponding line in the intermediate file in the **Text analysis area**.


Updating Regular Expressions

Tester allows you to try the input of regular expressions until you find the correct one.

Suppose you have accessed the regular expression tester using the  button from a specific field. Once you have verified the regular expression, you now want to update the corresponding field.

1. Click **Update**.

Note

This button is only available if Tester has been open using the  button.

2. The corresponding field is automatically updated using the expression from the regular expression tester.

For additional information and better understanding, see also:


- ◆ The [Regular expression tester demo](#), which shows how to use the tester.
- ◆ The [RegularExpressions.rqtf](#) example project, which contains a Word file you can use with the tester, like the one demonstrated in the viewlet.

Testing XML Syntaxes

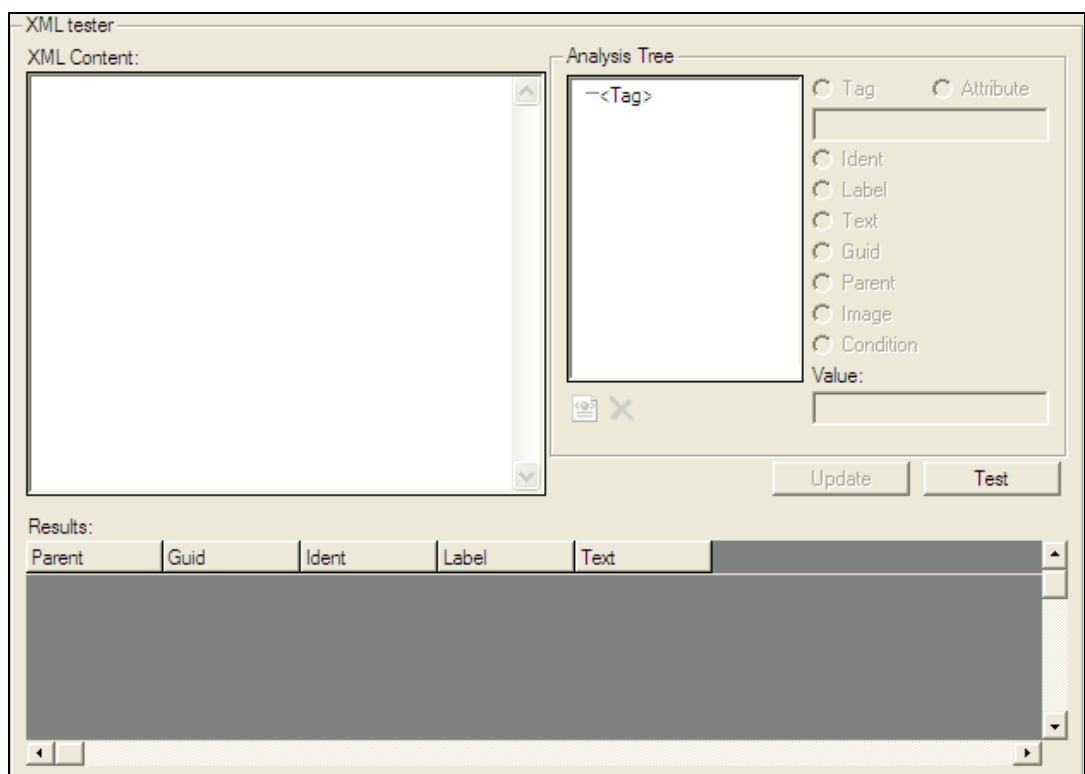
The **XML tester** is provided to help you to test and validate the XML expressions you define for the types.

Accessing the XML Tester


To access the XML tester directly from the Configuration dialog box, click the XML button on the left side of the pane.

The tester can also be reached by clicking the  button in the type editor. This button is available beneath the analysis tree when selecting an XML element. In this case, the tester opens with the fields already filled with the defined XML syntax.

The XML tester dialog box is displayed as follows:



The XML tester is composed of the following areas and buttons:

- ◆ **XML Content**—Specifies the text to analyze. The tests are going to be executed based on the contents of this text. This field can be filled using an import or manually.
- ◆ **Analysis Tree**—Specifies the XML syntax to be tested on the XML content.
- ◆ **Test**—Launches the test i.e. tries to match the XML syntax in the text to be analyzed.
- ◆ **Update**—Updates the field from which the Tester has been launched with the XML syntax. This is only available if the Tester has been launched using the  button.
- ◆ **Results**—a displaying area. Fill the table with matching results if some exist.

Using the XML Tester

Even if the **XML Content** can be filled manually, the general usage is to import the source information into the tester. Rhapsody Gateway makes it easy for you to import the intermediate file then you can test XML syntax to capture traceability elements.

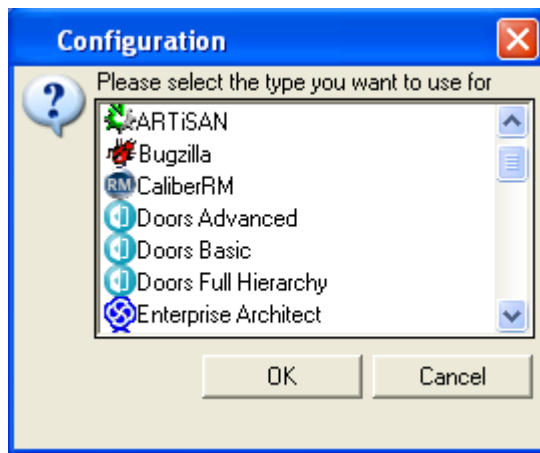
Use the File option to access the following commands for making imports:

- ◆ **Import file**—Imports source information from a file such as Access, or Rhapsody.
- ◆ **Import directory**— Imports the analysis of a directory such as Eclipse, or Code C.
- ◆ **Import intermediate file**—Imports the analysis of a file already converted, usually from databases such as DOORS or RequisitePro.

Importing a File

To import a file into the tester, follow these steps:

1. Select **File > Import** file option. The Configuration dialog box opens, as shown in next item.
2. Use the drop-down list to choose the type to convert.

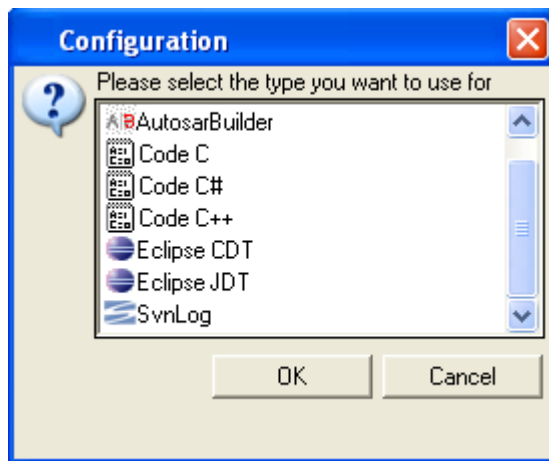


3. Click **OK** to apply the type you have chosen and open a new selecting dialog box.
4. Select the file to analyze and validate.
5. The **XMLContent** is filled out with the intermediate file converted from the selected file.

Importing a Directory

To import directory contents into the tester, follow these steps:

1. Select **File > Import** directory option. The Configuration dialog box opens, as shown in next item.
2. Use the drop-down list to choose the type to convert.



3. Click **OK** to apply your choice and open a Browse for Folder window.
4. Select the directory which contains the files to be analyzed.
5. A new dialog box opens, listing the files located in the directory.
6. Select the file to analyze and validate.

7. The **XML Content** area is filled with the intermediate file converted from the selected file.

Importing an Intermediate File

To import an intermediate file into the tester, follow these steps:


1. Select **File > Import intermediate file...** option. The Open dialog box is displayed.
2. In the Look in field, browse to the location of the intermediate file.
3. Select the `.xml` file, or type the name of the intermediate file in the File name field.
4. Click **Open**. The intermediate file opens filling the XML Content of the tester.

Testing XML Syntax

Once a file has been added into the **XML Content**, an XML syntax can be tested in the imported file contents.

1. Create your XML syntax to test.

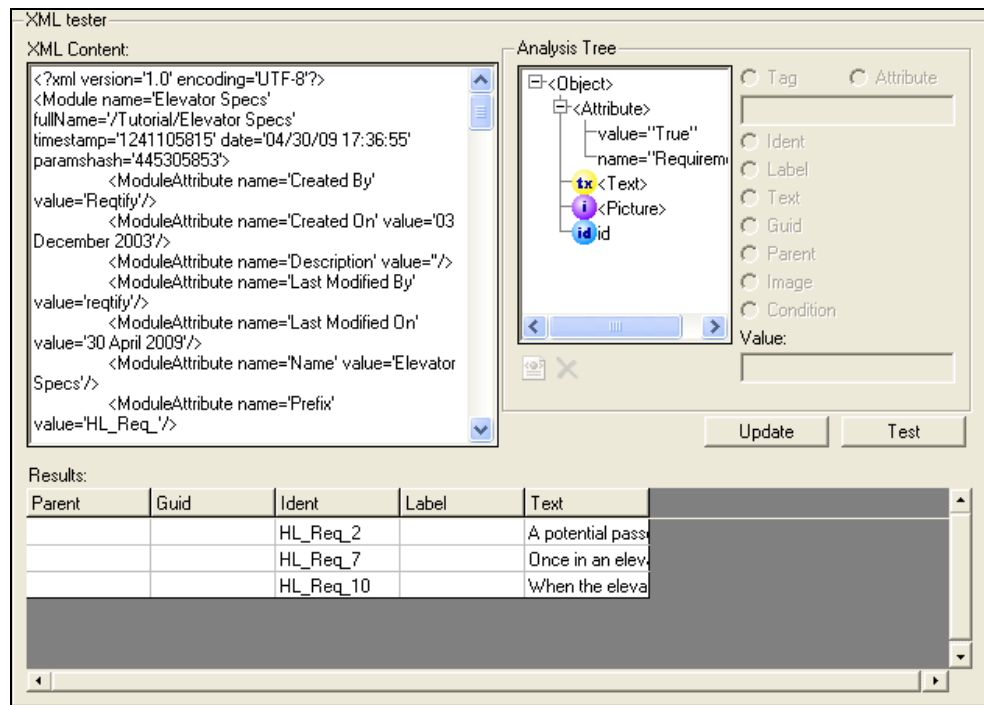
Note

These fields are automatically filled if you launched the XML tester using the  button.

2. Click on the **Test** button.

The matching analysis between the XML syntax and the intermediate file is launched.

3. If your syntax does not match anything in the analyzed text, the **Results** table is not filled in.
4. If your syntax matches elements of the analyzed text, the tester displays these results in the **Results** table:




The tester gives the following results:


- ◆ **Parent**—The parent value detected in the text.
- ◆ **Guid**—The GUID detected in the text.
- ◆ **Ident**—The Ident detected in the text.
- ◆ **Label**—The Label value detected in the text.
- ◆ **Text**—The text analyzed by the XML syntax.

Updating XML Syntax

Tester allows you to try the input of an XML syntax until you find the correct one.

Suppose you have accessed the XML tester using the  button from a specific field. Once you have verified your syntax, you now want to update the corresponding type element. In order to, click on the **Update** button. The corresponding XML element is automatically updated in the type using the syntax from the XML tester.

Note

The **Update** button is only available if the XML Tester has been opened using the  button.

Customizing Reports

Rhapsody Gateway can be used to generate traceability documentation in several formats such as RTF, HTML or PDF. The **Report Editor** enables you to construct customized model reports to be generated.

A **report** is a graphical representation of the report to be generated. The report structure is an assembly of formatting elements and Rhapsody Gateway **elements** such as attributes, requirements, or links.

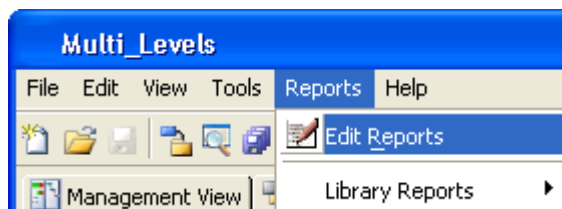
This section describes the uses of the Report Editor as follows:

- ◆ Accessing the Report Editor
- ◆ Report Editor Window
- ◆ Reports Creation
- ◆ Editing a Report
- ◆ Generating Reports
- ◆ Defining my First Report

Accessing the Report Editor

To open the **Report Editor**, follow these steps:

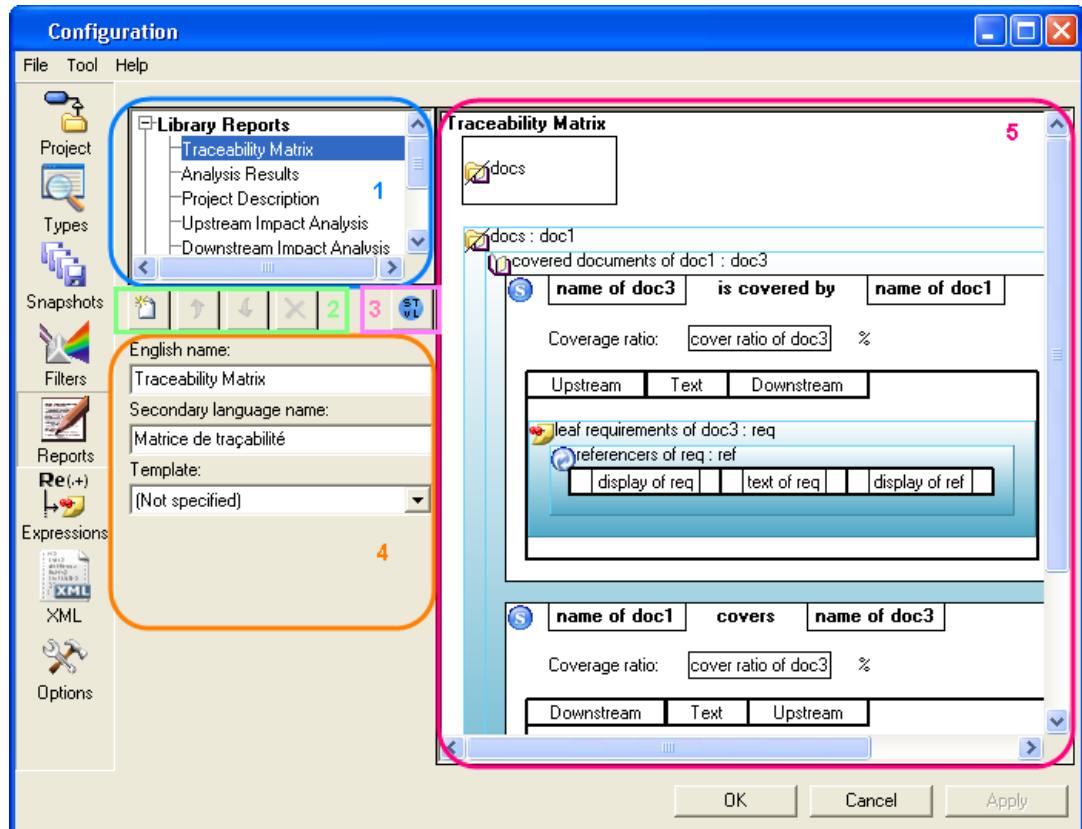
1. Select **Reports > Edit Reports** from the main window of Rhapsody Gateway, as below.



2. The **Report Editor** opens.

Report Editor Window

The following figure shows the **Report Editor** with a selected report sample.



The reports window contains the following areas and elements:

- ◆ **Report List Area (1)**—displays the list of reports.
- ◆ **Icon Bar (2)**—manipulates reports.
- ◆ **Reports elements (3)**—displays the elements that can be dropped into the **Graphical Area**.
- ◆ **Report Structure Configuration Area (4)**—allows you to type information or enter OTScript conditions.
- ◆ **Graphical Area to compose Report Structure (5)**—contains the graphical structure of the report.

Report List Area

The **Report List Area** displays the list of reports that can be generated.

The reports are sorted in the following categories:

- ◆ The **Library Reports** section lists the default available reports for all the projects. This list does not vary and all the documents are in read-only mode.

- **Traceability Matrix**—Lists the upstream to downstream covered links and the downstream to upstream covering links.
 - **Analysis Results**—Summarizes the coverage analysis for a project.
 - **Project Description**—Describes the project and its documents.
 - **Upstream Impact Analysis**—Lists the upstream traceability information for the selected elements of the project.
 - **Downstream Impact Analysis**—Lists the downstream traceability information for the selected elements of the project.
 - **Synthesis of Added Information**—Summarizes any added attributes, references, texts and covering links in the project.
 - **Rules Checking**—Contains a summary of any rules highlighted by the project.
- ◆ The **Reviewer Reports** section is only available if a corresponding coupling has been installed.
 - ◆ The **Project Reports** section is only available if some customized documents have been defined for the current project.

By default, the **Report Editor** opens without a selected report.

When a report is selected in the report list area the Graphical Area is automatically refreshed. (See **Graphical Area** for details about this area.)

Graphical Area

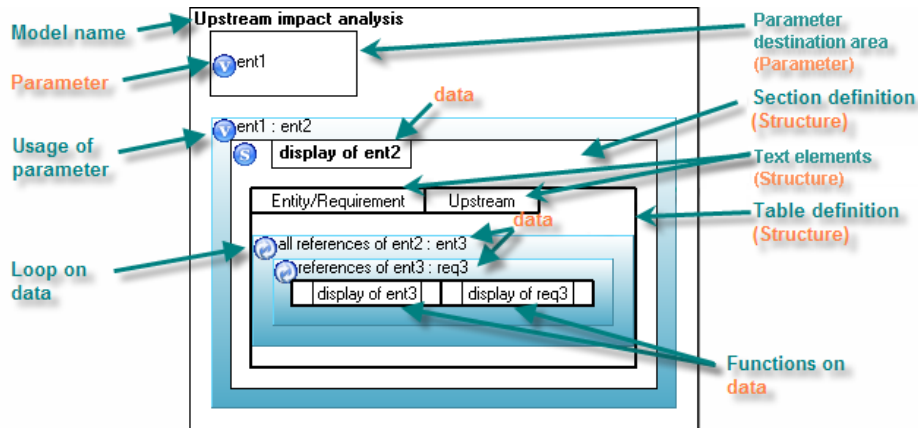
The **Graphical Area** displays the graphical representation of reports. Rhapsody Gateway enables you to graphically create a report structure from Rhapsody Gateway elements and from formatting elements.

A report structure looks like imbrications of rectangles. Each rectangle represents a Rhapsody Gateway element or a formatting element. All the elements are provided by the **Report Elements** (see **Report Elements**), and they need to be dragged and dropped into the graphical area to the desired position.

Note

It is not possible to drag and drop elements into read-only documents.

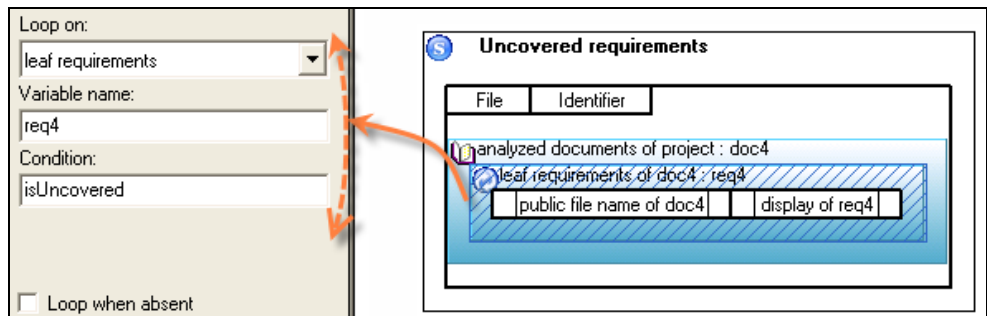
The following figure shows an example of a report representation as it appears in the **Graphical Area**.



Note

If you click an element in the **Graphical Area**, the relevant element is highlighted with hatching and Rhapsody Gateway will automatically refresh the associated form in the Report Structure Configuration Area. (See **Report Structure Configuration Area**.)

The following is an example of a highlighted element.



Report Structure Configuration Area

The **Report Structure Configuration Area** is a form. This form allows you to enter textual data, select displaying conditions, type OTScript conditions, etc.

This area is refreshed according to the selected element in the **Graphical Area** (see **Graphical Area**.)

Report Elements

Report Elements contain the formatting elements and the Rhapsody Gateway elements available for the report structure construction.






The **Report Elements** window shows the elements classified in three panes:

- ◆ **Structures**—lists the formatting elements that can be used to build your report structure. Structures is a static list.
- ◆ **Data**—lists the different project variables available for the report creation. The content of this data list depends on the choices made during the construction process. Each time a method providing element is added in the Graphical area, a corresponding new associated *variable*, also named *iterator*, is alphabetically inserted into the Data list. In addition, all the methods applicable to a variable are shared by categories. At the beginning of report creation, the pane only contains the *project* variable.
- ◆ **Parameters**—contains the project elements of traceability that the user will be asked to select before generation. Parameters is a static list. See *Inserting a Parameter into the Parameter Area* section for detailed information.

Report Editor Toolbar

The Report Editor Toolbar enables you to add, delete or move a report. It also provides access to the contents of Report elements.

The Report Editor Toolbar contains the following tools:

	New report —Adds a new blank report at the end of the report list.
	Up —Moves up a report within the report list.
	Down —Moves down a report within the report list.
	Removes an element —Deletes a selected report. The report is removed from the report list after confirming the action.
	<p>Report elements—With a single click, displays or hides the contents of the Report Elements.</p> <p>With a double-click, extracts or includes contents of the Report Elements in a stand-alone window or in the Report Structure Configuration Area.</p>

Creating Reports


There are two ways to create a new report:

- ◆ Create a report from scratch (see *Creating a New Report*)
- ◆ Create a new report from an already existing report (see *Duplicating an Existing Report*)

Creating a New Report

A new report can be created from scratch by users and then filled in.

To add a new report, follow these steps:

1. Select **New report**  from the icon bar.
2. A new blank report is added at the end of the report list. The Graphical Area is refreshed with the new report.
3. In the name field, type the desired name of the new report.
4. All occurrences of the report name are now replaced.

Duplicating an Existing Report

Sometimes existing reports will nearly correspond to the ones you wish to describe. You can modify their contents but in some instances you might also want to keep the report in its original form. Moreover, provided reports are in read-only mode and cannot be modified. In order to benefit from existing reports a duplicating feature allows the user to reproduce an existing report and modify it.

To duplicate a report follow these steps:

1. Select the report to duplicate in the Report List.
2. Click right then choose the **Duplicate** option from the context menu.
3. A report with the same name followed by a number is added at bottom of the **Report List**. The report has the same contents as the original report. This new report is modifiable.
4. Rename this report as desired. Now you can modify this report and use it as a basis for your own report construction.

Editing a Report

The following section describes how to use the **Report elements** to graphically construct a document structure. The topics are as follows:

- ◆ Parameters Usage
- ◆ Creating a Report Structure
- ◆ Inserting Data in a Report
- ◆ Customizing a Report

Parameters Usage

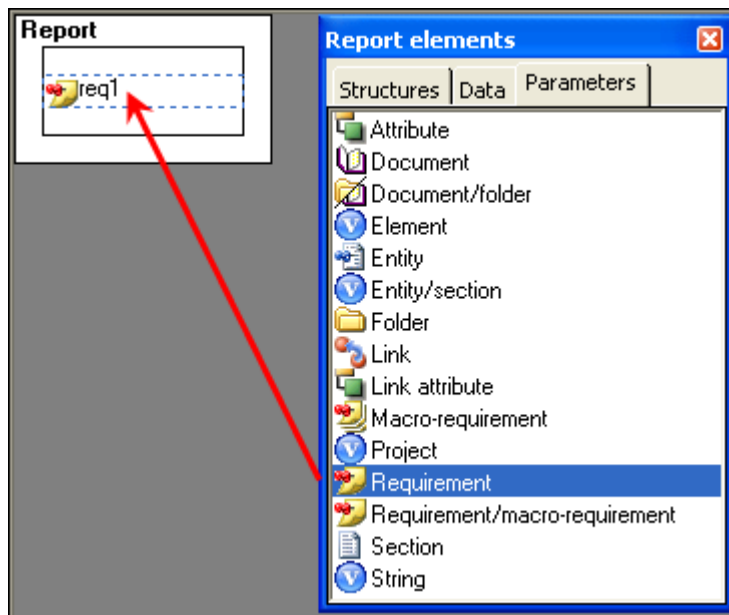
The concept of parameter is to allow the selection of specific documents for the generation of reports. The set of parameters is available from the Parameters tab of the Report elements window. This chart lists all the elements that can be used as a parameter.

Attribute	Entity/section	Project
Document	Folder	Requirement
Document/folder	Link	Requirement/macro-requirement
Element	Link attribute	Section
Entity	Macro-requirement	String

Inserting a parameter into the parameter area

To insert a Parameter in the Graphical Area, follow these steps:

1. Drag a parameter element from the Parameters tab into the Parameter area.



2. The "No parameter" text is replaced by the parameter name.
3. Enter a message if you would like to customize the message that appears when asking for parameters during generating:

Parameter name:	req1
English message:	Select a requirement
Secondary language message:	
Condition:	

Note

Only one Parameter element can be dropped into the Parameter area, whether it represents one or multiple elements.

Deleting a parameter from the parameter area

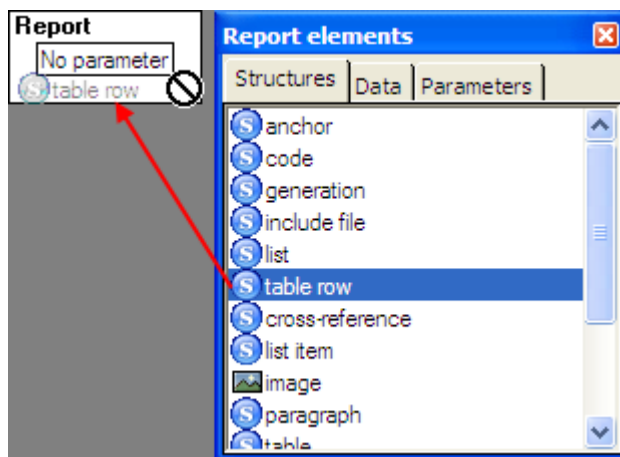
1. Click on the current parameter to select it.
2. Select **Delete** option from the context menu.
3. The element disappears and "No parameter" replaces the parameter.

Creating a Report Structure

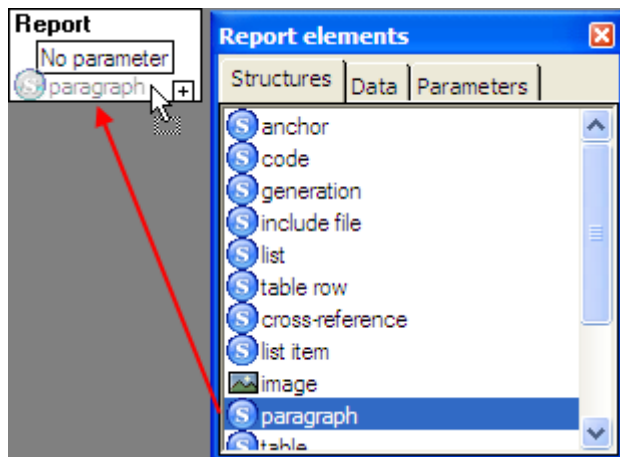
The insertion of formatting elements allows you to structure your reports; **Structures** elements are available from the Structures tab of the **Report elements** window.

To include a **Structure** element in a report, drag a **Structure** element from the Structures tab into the **Graphical Area** at the desired position, as shown below.

- ◆ If the chosen position is wrong for the kind of selected structure, a forbidden symbol is displayed and the **Structure** element cannot be inserted.



- ◆ If it is correct, a graphical representation of the **Structure** element is added in the graphical view of the report.



Note

When selecting an element in the Graphical area, it appears highlighted with hatching. In **Report elements**, elements available to drop into the selected element are written in bold.

This chart lists all the structure elements available in **Structure tab** of the **Report elements** window.

Anchor	Table row	Table
Code	Cross-reference	Section
Generation	List item	URL
Include file	Image	Text
List	Paragraph	Code OTScript

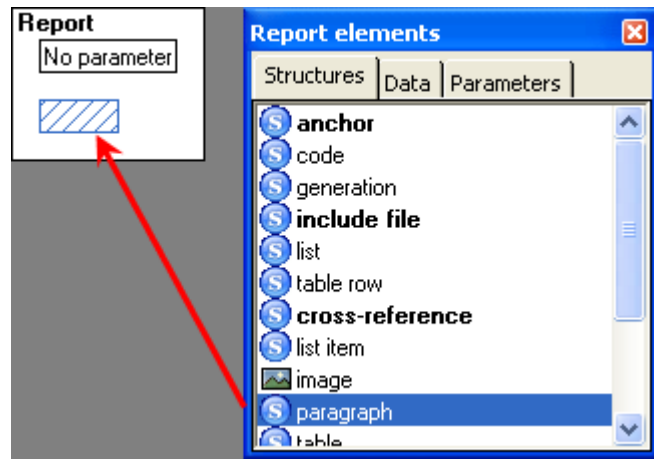
The following sections describe more precisely how to perform the insertion of these elements into the report.

Note

Go ahead to the **Destination Location for each kind of Structure Element** section to get specific information on a structure element position.

Inserting a paragraph

To insert a **Paragraph** into a report, drag the **Paragraph** element from the **Structures tab** to the desired position in the **Graphical area**.



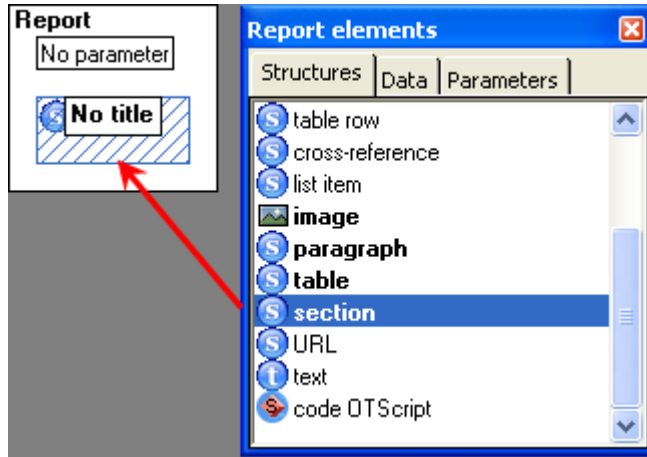
Note

A paragraph cannot contain another paragraph.

Creating a section

To insert a section in a report, follow these steps:

1. Drag a **Section** element from the **Structures** tab into the **Graphical area**.



Note

A section has to be added directly into the root report.

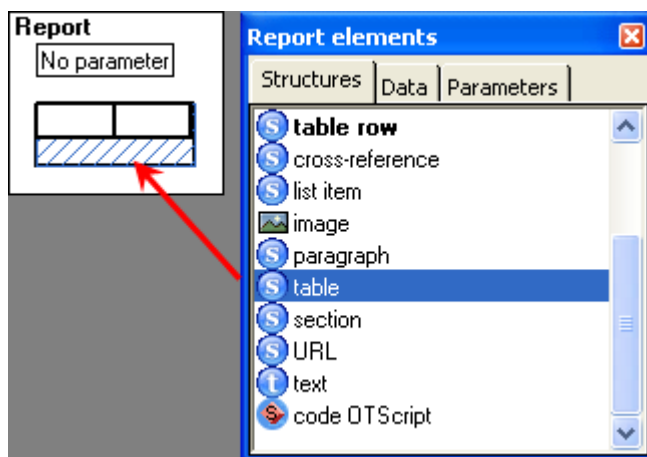
A section can contain another section, which corresponds to a sub-section.

2. To name the section, drag a text element into the "No title" area. Then complete the name field. See **Adding a text** section for more details.

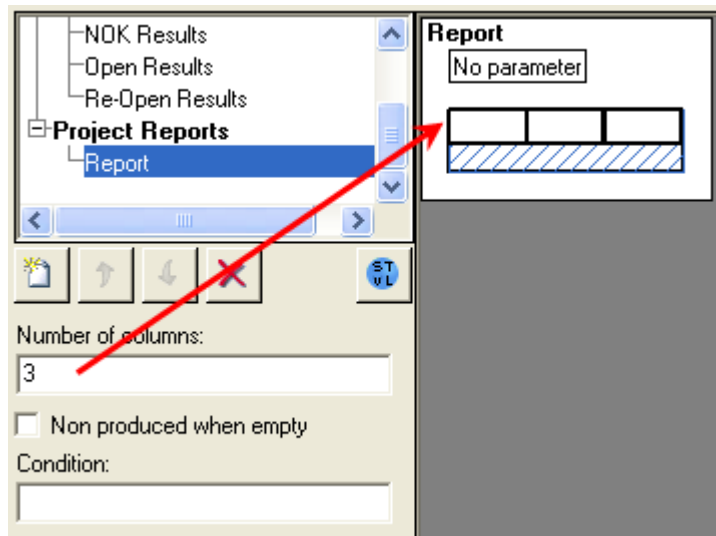
Creating a table

To insert a table into a report, follow these steps:

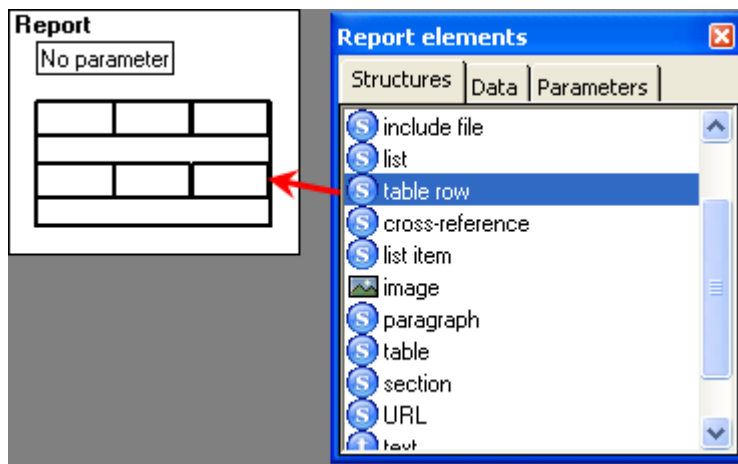
1. Drag a **Table** element from the Structures tab into the **Report** or a **Section**.



2. A **Table** contains two columns by default. To define the number of columns, change the value in the **Report Structure Configuration** area and validate it.



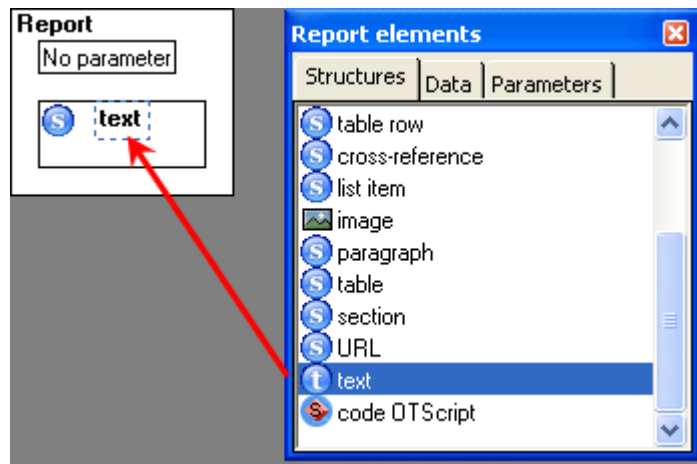
- Next, to add rows to the table, drag the **Table row** element into the inserted table. You can add as many rows as you want.



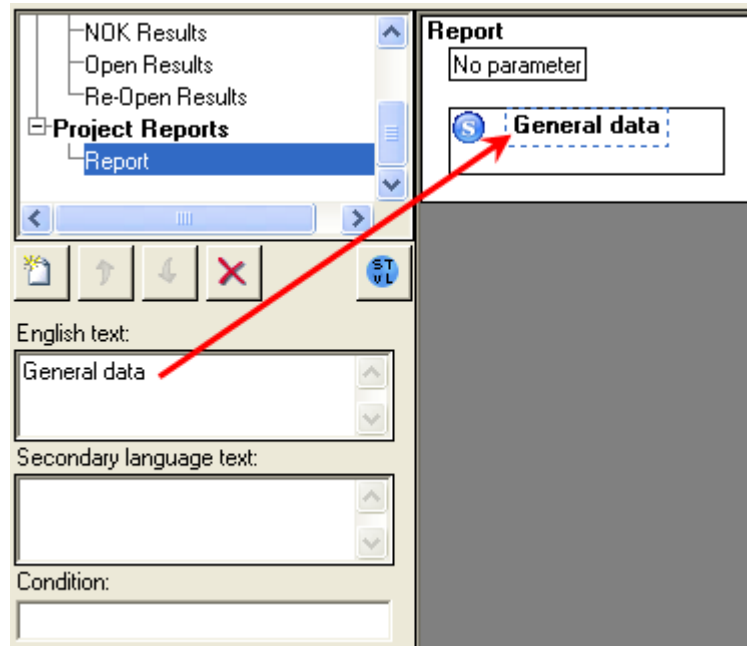
Adding text

Text elements can only be inserted in the following structures elements: Paragraph, Section title, Table title, and Include file. To insert a Text into a report, follow these steps:

- Drag a **Text** element from the **Structures** tab of the **Report elements**.
- Drop it at the desired position in a relevant structure element. Insert, for instance, a text in a section title.



3. The text element expects a typed text from the form or a data text.
4. Fill the section title.



Note

If the text is not well displayed in the structure element, click **F5** to expand the text display.

Adding an image

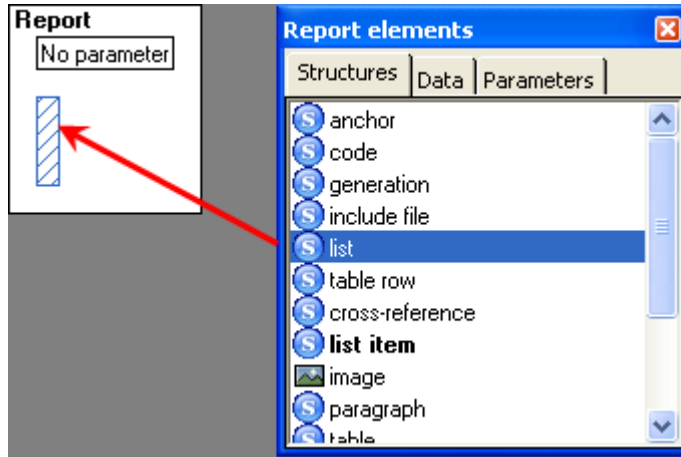
To insert an image into a report, follow these steps:

1. Drag an **Image** element from the **Structures** tab into the **Report** or a **Section**.
2. An image element expects a data image. Drop a data image into the structure element. See **Inserting Data** in a report.

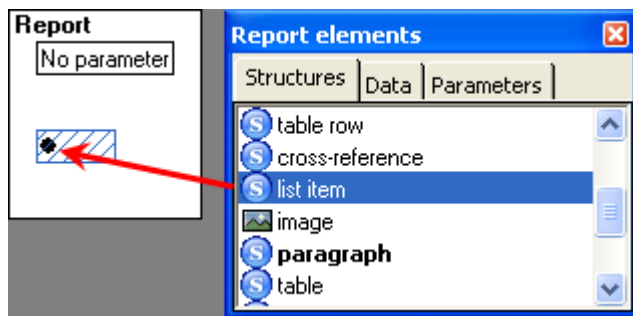
Creating a list of items

To insert a list of items into a report, follow these steps:

1. Drag a **List** element from the **Structures** tab into the **Report** or a **Section**.



2. Next, you have to insert items in the created list. Drag a List item element from the **Structures** tab into the inserted list. You can add as many items as you want.



Note

It is not possible to create lists with several levels of items.

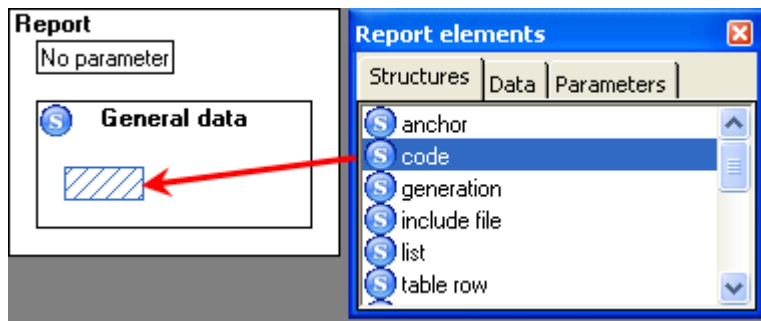
A list of items needs to contain a paragraph element before other information.

Inserting a code

This element is a paragraph, which will be generated written in `Courier New` font in the destination report.

To insert code into a report, follow these steps:

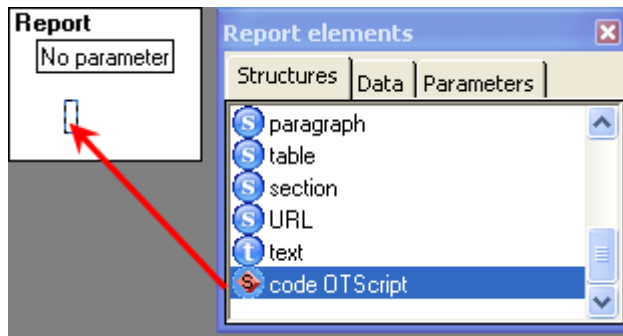
1. Drag a **Code** element from the **Structures** tab of the **Report** elements.
2. Drop it at the desired position in a **Section** structure.



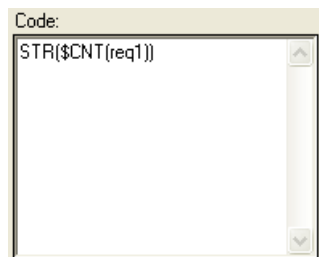
Inserting OTScript code

To insert an **OTScript code** into a report, follow these steps:

1. For instance, drag an OTScript code element from the **Structures** tab into a **Paragraph**.

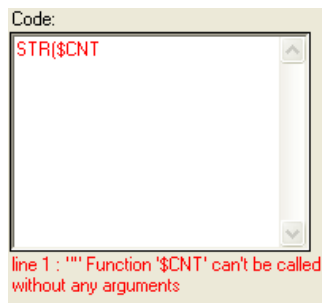


2. In the **Report Structure Configuration** area, enter some OTScript code to be evaluated during generation.



Note

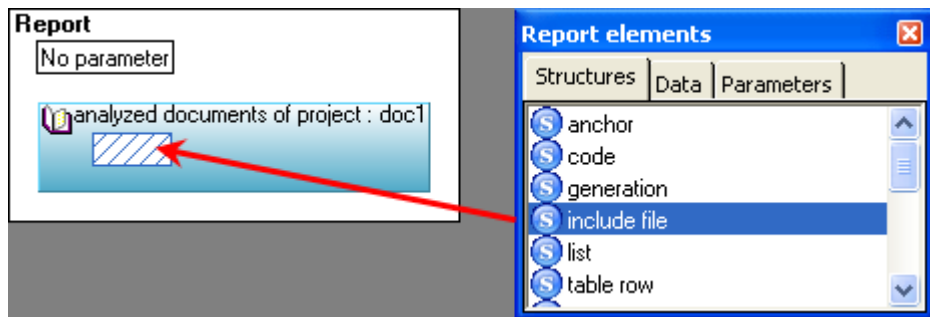
If OTScript code is filled with erroneous entry, the text will be displayed in red, and a message will be displayed. Below is an example of an erroneous code.



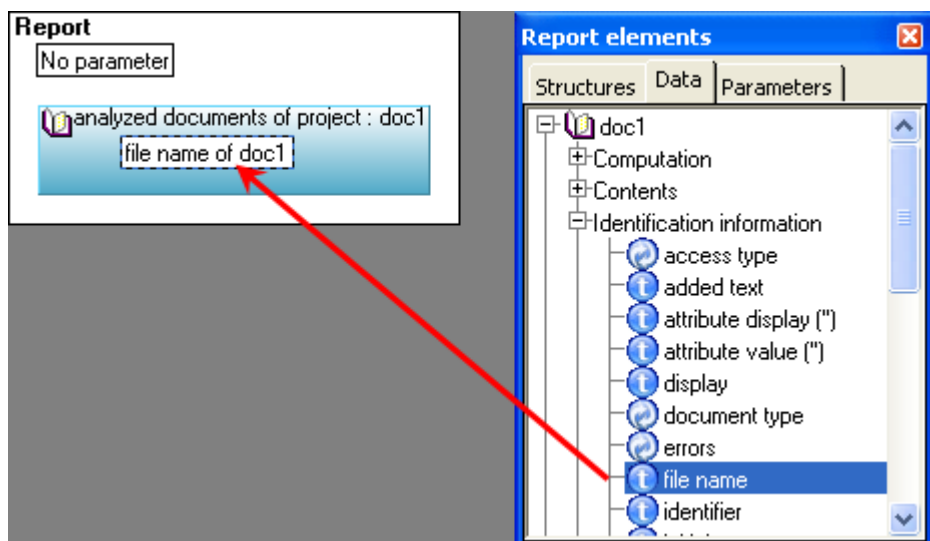
Including a file

To insert the contents of a file into a report, follow these steps:

1. Drag an **Include file** element from the **Structures** tab into a **Paragraph**. For instance a paragraph placed in 'analyzed documents of project'.



2. Drop a data element into this area, which corresponds to the path of a file that will be included.



3. In the generated report, the included file will appear as a hyperlink to be expanded. Thus the contents of a Microsoft Word document can be included into another Word document.

Note

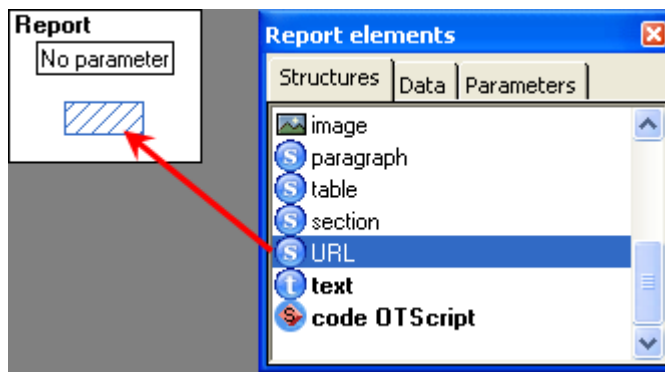
The generated document type must be able to insert the document to be included.

Inserting a URL

In the report, the **URL** element will be generated as a hyperlink. Therefore the relevant document needs to be expanded.

To insert a **URL** in a report, follow these steps:

1. Drag a **URL** element from the **Structures** tab into a **Paragraph**.



2. Enter the URL you want to insert into your report. The address has to be entered between quotation marks otherwise an error will occur. The error will be displayed in red.

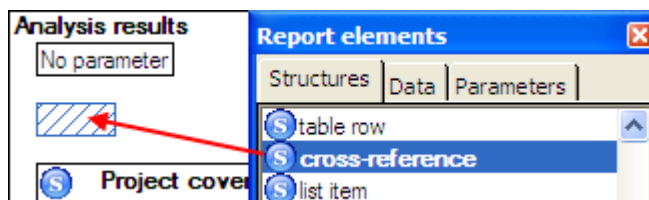


Adding a cross-reference

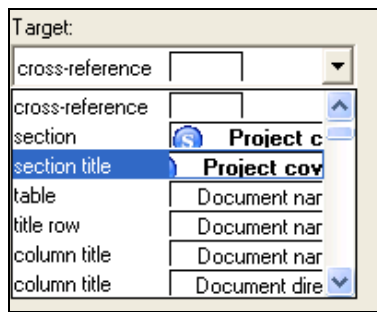
A **Cross reference** is a cross-reference such as in a Word document. The **Cross-reference** list is created from elements inserted in the report structure in process.

To insert a cross reference into a report, follow these steps:

1. Drag a **Cross reference** element from the **Structures** tab into a **Paragraph**.



2. Now choose the target of the cross-reference. The **Target** field lists all the elements that can be reached by the cross-reference.



Note

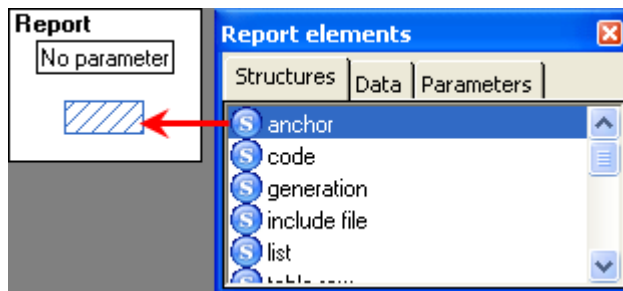
Such an element is only created for a Microsoft Word target result.

Inserting an anchor

The **Anchor** element will add a bookmark in the generated report.

To insert an **Anchor** in a report, follow these steps:

1. Drag an **Anchor** element from the **Structures** tab into a **Paragraph**.



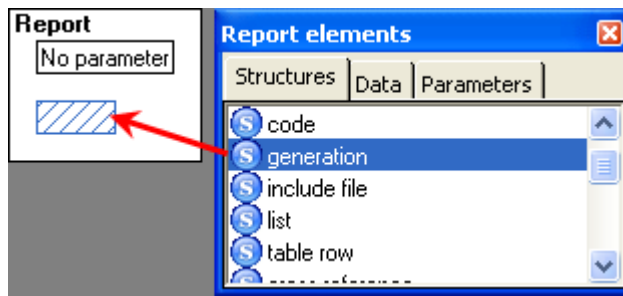
2. To name the anchor, drag a text element into the Anchor area. Then complete the name field. See **Adding a text** section for more details.

Inserting a Generation

The **Generation** element will generate another report within the current report.

To insert a **Generation** in a report, follow these steps:

1. Drag a **Generation** element from the **Structures** tab to the desired position in the **Graphical area**.



2. Enter English Report name for the report you want to insert into your report. If the report is unknown, a message will appear during the generation.

Report name:

☒ Non produced when empty
 Condition:

Destination location for each kind of structure element

The structure elements cannot be inserted anywhere. In fact, to insert a specific structure element in the report, sometimes you have to insert another kind of structure element first.

This chart summarizes the location where each structure element can be inserted.

Element	Can be inserted in
Anchor	Paragraph, table row, section title
Code	Section
Generation	Report, section
Include file	Paragraph
List	Report, section
Table row	Table
Cross-reference	Paragraph, table row
List item	List
Image	Report, section
Paragraph	Report, section, list item

Element	Can be inserted in
Table	Report, section, table row
Section	Report, section
URL	Paragraph
Text	Cross-reference, URL, paragraph, include file, section title, Title header of a table
Code OTScript	Cross-reference, URL, paragraph, include file, section title, Title header of a table

Note

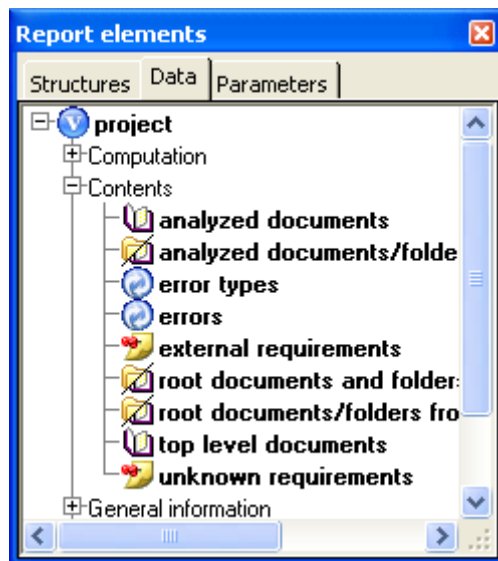
Remember that when you select an element in the Graphical area, the Report elements that are available to be dropped into the element will be highlighted in bold formatting.

Inserting Data in a Report

Structure elements allow you to build the structure of the report. However, some specific data needs to be inserted in the report in order to precisely define the contents. This data will be interpreted during the generation process of a report.

Data processing

The **Data** tab of the Report elements window looks like a tree. Each node gathers its authorized shared sub-data into sub-categories. Available sub-categories are: Computation, Contents, Identification information, Misc, Project links and Reviewer (if the Reviewer coupling has been installed). For instance, the content of the project contains **analyzed documents** data.







Two kinds of data exist, terminal data, also named leaves, and non-terminal data, also named nodes. During the building process when inserting a non-terminal data into the Graphical area, a variable, also named an iterator, is automatically created. This iterator name is generated from the data's name. I.e. the variable corresponding to the data Checklist Items is $item_n$, where n is greater than 0. This variable becomes a new node in the data tree and offers elements gathered in sub-categories.










Note:

Elements can be analyzed recursively or not. If you call:

- ◆ sections/entities/requirements: the child sections, entities and requirements of the section or document will be analyzed
- ◆ all sections/entities/requirements: all the sections, entities and requirements of the section or document and all their descendants will be analyzed

The following chart shows the different types of data.

Data representation	Data type	Characteristics
	Attributes	Non terminal element. It represents Rhapsody Gateway's attributes.
	Cover links	Non terminal element. It represents Rhapsody Gateway's cover links.
	Documents	Non terminal element. It represents Rhapsody Gateway's documents.
	Documents/Folders	Non terminal element. It represents Rhapsody Gateway's documents or folders.

Data representation	Data type	Characteristics
	Entities	Non terminal element. It represents Rhapsody Gateway's entities.
	Filters	Non terminal element. It represents Rhapsody Gateway's filters.
	Image	Terminal element. Needs to be dropped in an Image structure element. It allows the insertion of an image in the generated report.
	Links	Non terminal element. It represents Rhapsody Gateway's links.
	Loop	Non terminal element. It allows the insertion of elements in addition to the ones listed before, and allows you to loop them.
	Macro-Requirements	Non terminal element. It represents Rhapsody Gateway's macro-requirements.
	Requirements	Non terminal element. It represents Rhapsody Gateway's requirements.
	Sections	Non terminal element. It represents Rhapsody Gateway's section.
	Text	Terminal element. Needs to be dropped in an area which accepts a Text structure element: Paragraph, Section Title, Table Title, etc. It allows the insertion of text information into the generated report.

Data creation through an example

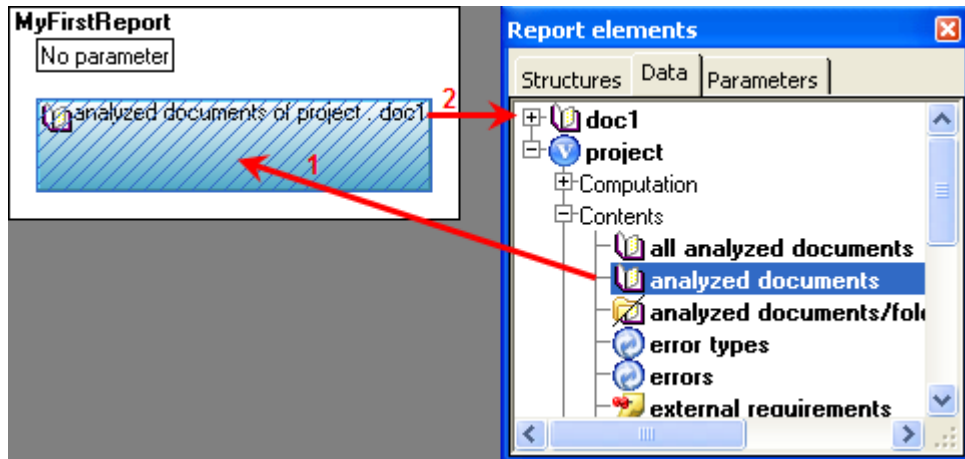
The purpose of this example is to build a report containing a list of all project requirements.

The creation of this short example allows the user to visualize the creation of new data.

1. First create a report from scratch as described in the section Adding a New Report. Name this report **MyFirstReport**. The **Graphical area** looks like the following example below.

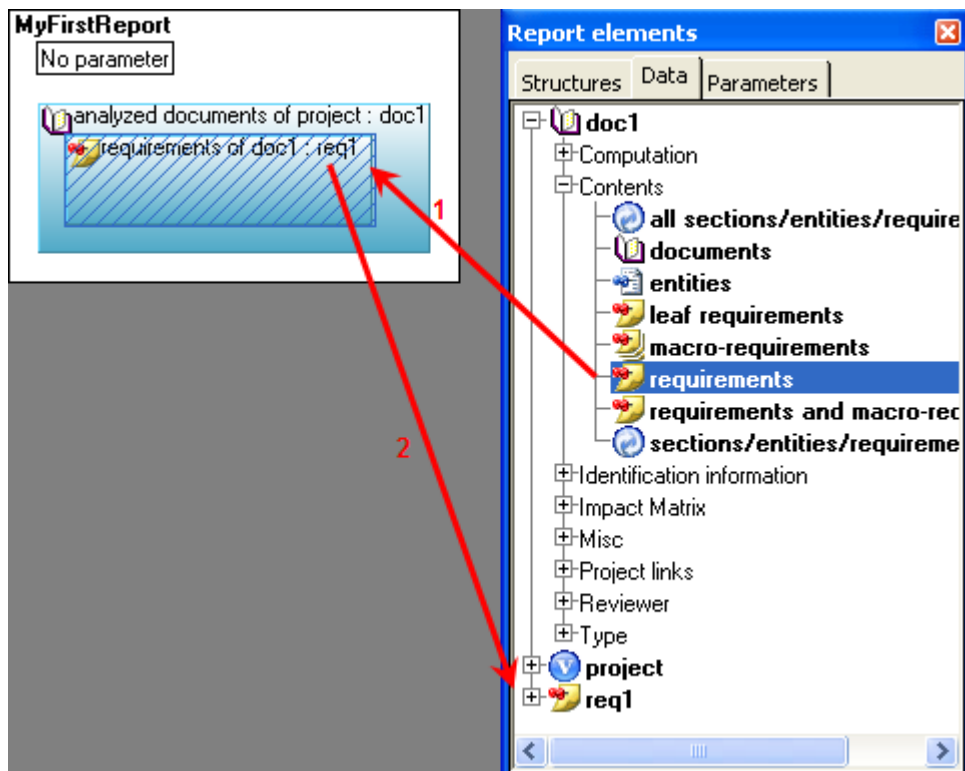


2. Add a loop on documents.



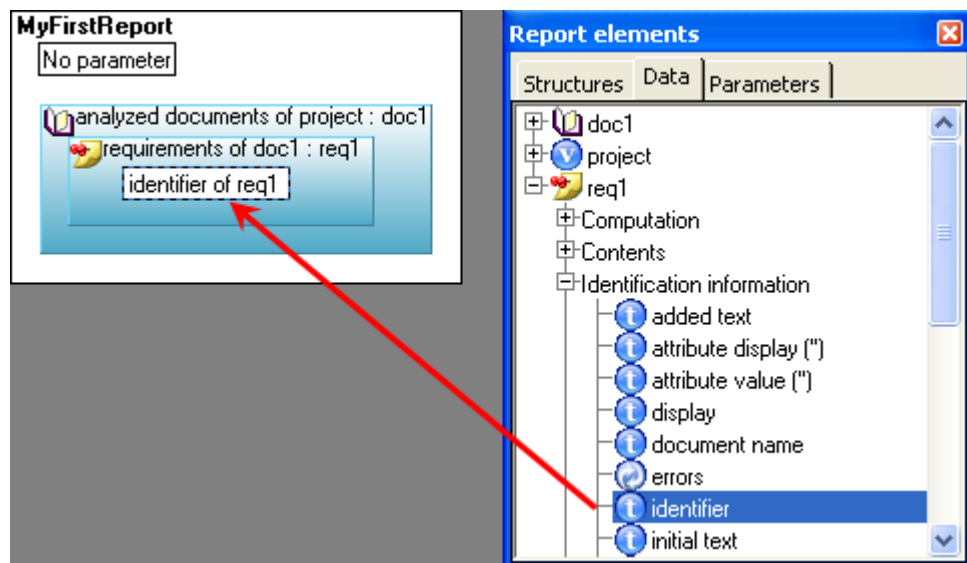
A new iterator **doc1** is created in Data tab.

3. Add a loop on requirements.



A new iterator **req1** is created in Data tab.

4. Insert a paragraph in the requirements loop.
5. Then add the requirement identifier.



Customizing a Report

Depending on the kind of selected elements, some additional fields become available in the Report Structure Configuration Area. (see **Report Structure Configuration Area**.) These fields let the user improve the appearance of his generated report.

Using options

Some options on structure elements or data elements are available to parameterize the contents of the generated report in specific conditions.

These choices are represented as check boxes in the Report Structure Configuration Area.

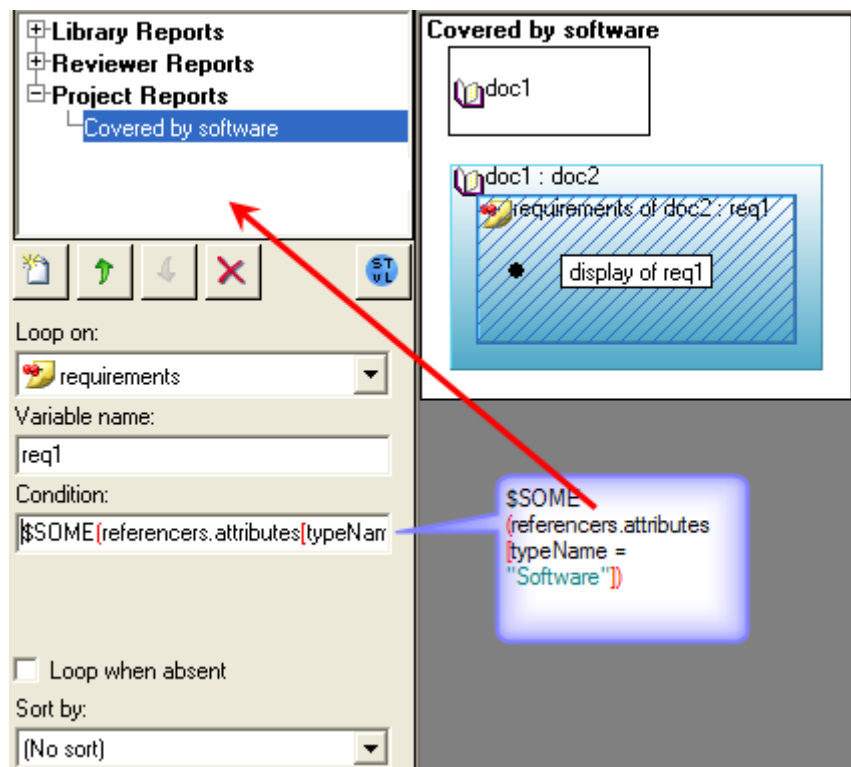
Options title	This option is available for	Purpose
With title	<ul style="list-style-type: none"> a Report 	Has the title of the report been displayed in the generated report?
Loop when absent	<ul style="list-style-type: none"> a Loop (data) 	Has the contents of the loop been displayed even if there is no iteration element for the loop?
Display blank when multiple	<ul style="list-style-type: none"> a Table row 	Is the cell of a table displayed blank if an element is repeated from the previous line?
Non produced when empty	<ul style="list-style-type: none"> a List a Table an Image 	Has empty structure been displayed in the generated report?

Options title	This option is available for	Purpose
	<ul style="list-style-type: none"> a Section 	
Style	<ul style="list-style-type: none"> a Table title a Table row a Paragraph 	Is a specific style applied to the corresponding report structure? The style is defined in the template for generation.
Condition	All elements.	Is the current element conditioned?
Nesting parent function	<ul style="list-style-type: none"> a Section 	Shall the section hierarchical organization appear in the generated report?

Using conditions

Most of the structure elements have got a **Condition** field, accessible from the Report Structure Configuration Area. This field allows you to enter a condition written in OTScript language. This condition allows you to restrict a set of elements when generating.

Below you will see a Loop on documents. The associated condition means that if referencers exist, attributes with typeName "Software" are the only interesting ones:



For details, see the *OTScript Language Essentials* chapter or contact the Support Team.

About Nesting parent function

To use the hierarchical nature of sections, you have to fill in the **Nesting parent function** field with a function name. Three values can be used:

Function name	Application data	Node type
parent	all sections/entities/requirements	document
reference	all referencers	requirement
referencer	all references	section/entity/requirement

If you, for instance, make a loop “all sections/requirements/entities” and if you add a section with the “parent” function for Nesting parent function in it, the report will contain a tree of sections with all sections, requirements and entities.

Templates

Templates enables the user to define the output report formatting, such as styles, header, footer, page setting, table of contents, indexes and appendices.

Template Description

`rtf`, `mif`, `txt` and `html` templates contain the two `%endheader` and `%begintrailer` keywords.

During the report generation, texts which are located before the `%endheader` keyword are pasted as they are in the report. The same applies for texts which are located after the `%begintrailer` keyword.

All elements located between the two keywords are replaced by the generation.

For `rtf` format, the style definition between these two tags is mandatory otherwise everything that is located between these tags will not be produced in the report. Styles can be customized.

`rtf` style names are listed in the table below.

French style name	Styles name provided by Word
Normal	Normal
Corps de texte	Body Text
Sous-titre	Subtitle
Titre 1	Heading 1

French style name	Styles name provided by Word
Titre 2	Heading 2
Titre 3	Heading 3
Texte brut	Plain Text
Liste à numéros	List Numbered
Liste à numéros 2	List Numbered 2
Liste à puces	List Bullet
Liste à puces 2	List Bullet 2
Lien hypertexte	Hyperlink

For `xls` templates, two cell formats can be used:

- ◆ ‘Text’ format: which displays data as text.
- ◆ ‘Standard’ format: which enables a best formatting of data, such as numbers interpreted as numbers, dates or for formulas, ...

The following default styles of the template are based on Excel ‘Text’ format.

Styles	Equivalence in the generation report
entry	Cells content
para	Paragraph of a report
sect, sect1, sect2, sect3, sect4, sect5, sect6, sect7, sect8, sect9	Sections
tableTitle	Table titles
title	General title of the report

The following styles are based on Excel ‘Standard’ format. These two styles are not used by default.

Styles	Equivalence in the generation report
entryStandard	is like entry but with "Standard" cell format.

Styles	Equivalence in the generation report
paraStandard	is like para but with "Standard" cell format.

If user wants to print information in ‘Standard’ cell format, he has to put one of this style in the style textfield of Report configuration window for the concerned parts of the report.

For your own `xls` templates, previous presented cell styles have to be defined.

Generation Options

Add-on information can be inserted in the `rg.ini`. For Excel, the section is named `ExcelReport`.

Defining maximum row height

To define a maximum row height, use `MaxRowHeight` such as follows:

```
[ExcelReport]
MaxRowHeight=xxx
```

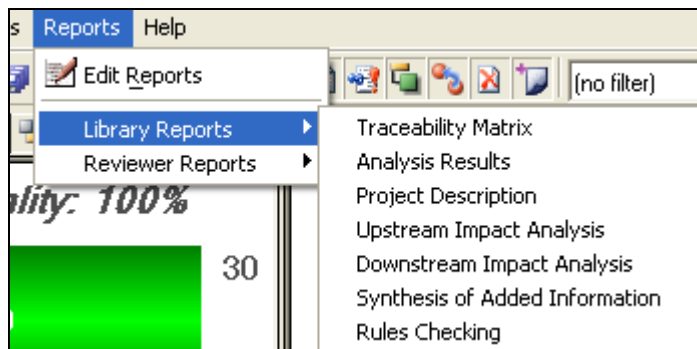
Naming sheet

To specify the name of a sheet, use `SheetNameIsSectionTitle` such as follows

```
[ExcelReport]
SheetNameIsSectionTitle=true
```

Generating Reports

The **Reports** menu from the main window shows the list of reports that can be generated.



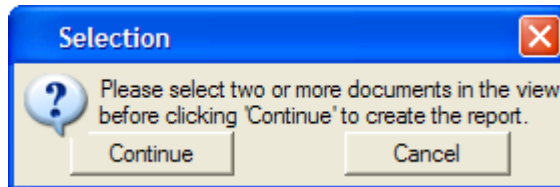
The Available Analysis Reports pane displays the list of reports that can be generated for the project. It contains a Library Reports section, listing the reports available for all

projects, and can contain an additional Project Reports section if templates have been defined for the current project only.

Double click on project report in the list to generate it.

Note

If some documents need to be selected to generate the chosen report, a dialog box like the following one opens:



The chosen model has been defined based on parameters (see *Parameters Usage*). Select the documents to analyze then validate.

In all cases, a dialog box opens to name the report and to choose the output format document. The generated report opens at the end of generation.

Traceability Matrix		
1. High Level Requirements is covered by Low Level Requirements		
Coverage ratio: 100%		
Upstream	Text	Downstream
REQ1 - Capture	The tool shall be able to capture semi-automatically the requirements included in a document and/or in a model. "Semi-automatic" means the text has to be formalized beforehand by the user or another dedicated tool.	LL-REQ1 - Capture
REQ1 - Capture	The tool shall be able to capture semi-automatically the requirements included in a document and/or in a model. "Semi-automatic" means the text has to be formalized beforehand by the user or another dedicated tool.	LL-REQ3 - Multiple Requirements
REQ2 - Update information when source changes	The tool shall take into account the successive versions of the documents and models, and update automatically the traceability information presented to the user.	LL-REQ2 - Automatic Update
REQ3 - Multiple requirements definition	The tool shall be able to consider several requirements definitions created by the user.	LL-REQ3 - Multiple Requirements
REQ4 - Traceability Reports	The tool shall generate traceability reports including all or part of the traceability information, using these user-defined templates.	LL-REQ4 - Reports Generation
REQ5 - Navigation to the Authoring tool	The tool shall allow navigation from the information displayed in the main window and the authoring tool from which the information has been captured	LL-REQ5 - Navigation
2. Low Level Requirements covers High Level Requirements		
Coverage ratio: 100%		
Downstream	Text	Upstream
LL-REQ1 - Capture	The tool captures requirements and traceability information from various sources using either API, dedicated converters or other interfaces provided by the authoring tool	REQ1 - Capture
LL-REQ2 - Automatic Update	Each time a document is updated, the update is detected and user has the opportunity to reload the modified information.	REQ2 - Update information when source changes
LL-REQ3 - Multiple Requirements	Several 'requirement' elements can be defined in the type of analysis.	REQ3 - Multiple requirements definition
LL-REQ3 - Multiple Requirements	Several 'requirement' elements can be defined in the type of analysis.	REQ1 - Capture
LL-REQ4 - Reports Generation	Any information item captured by the tool can be included in a report. User can define his own templates in addition to the templates provided by default.	REQ4 - Traceability Reports
LL-REQ5 - Navigation	A double-click on an element in the main window brings the user into the authoring tool, and the element is selected in this environment.	REQ5 - Navigation to the Authoring tool

Reports Creation

Short Report Example

An animated demo is accessible to show how to create a short report example. The purpose of this example is to list the requirements of a document, their text and their location in the document.

For additional information and better understanding, see also:

- ◆ The [Report Generation animated demo](#), which presents the material that is explained in this section.

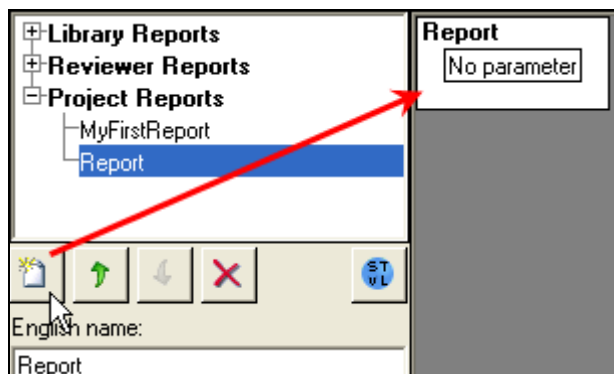
Defining my First Report

The purpose of this example is to create a restricted traceability matrix on selected documents. This matrix will display the relationships between requirements and coverages of the set of documents.

The associated project to validate this example is the Tutorial example available in Documents and Settings\<user>\My Documents\Rhapsody Gateway\<Rhapsody Gateway_version>\examples\Use Cases\GettingStarted (for Windows XP)

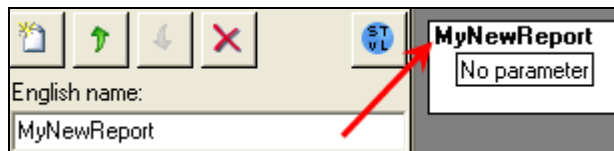
Users\<user>\My Documents\Rhapsody Gateway\<Rhapsody Gateway_version> \examples\Use Cases\GettingStarted (for Windows 7)

1. The first step in creating a new report is to click on **Add a report** icon.



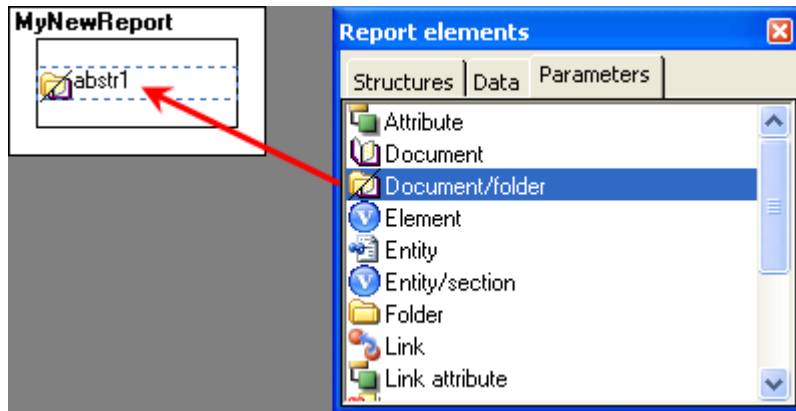
A new report appears in the right area.

2. Rename your report.



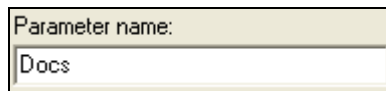
3. Open the **Report elements** window with a double-click.

Now, add a parameter to your report, because you only want to analyze specific documents in the generated report. In the Parameters tab, select Document/folder, drag it into the parameter area.



This parameter will allow you to select documents for the generation.

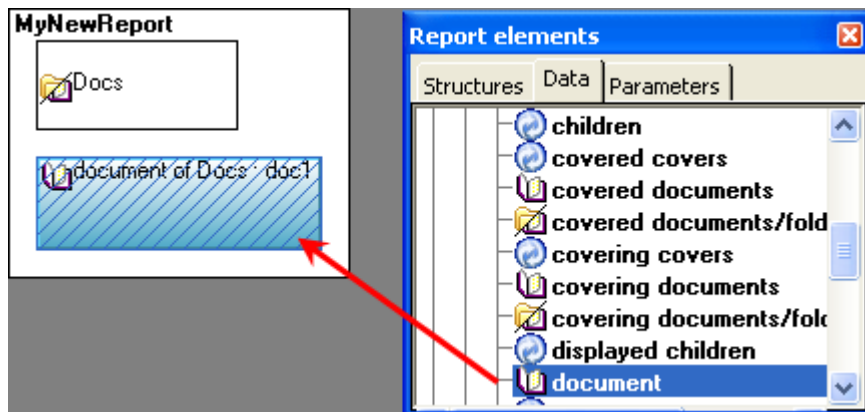
4. Rename the parameter iterator **abstr1** as **Docs**.



The right area is updated, and a new corresponding iterator **Docs** is added in the **Data** tab of Report elements. Expand the **Docs** contents to visualize all available elements of **Docs**.

5. Now, create a loop on the documents, in order to analyze them one by one.

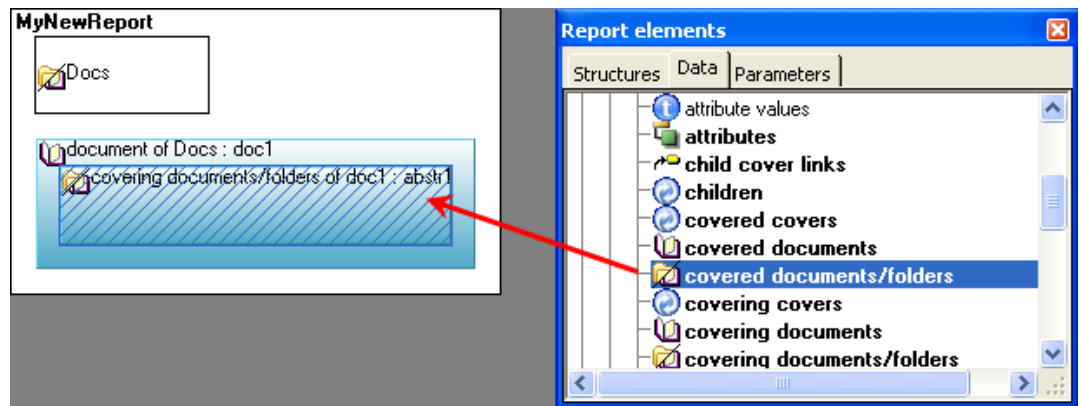
To do that expand the **Project links** of **Docs**, select document and drag and drop it into your report.



A new iterator **doc1**, as defined in documents, is added in the **Data** tab.

6. Only documents which cover your selected documents are of interest.

Expand the **doc1** in the **Data** tab, choose the **covering documents/folders** method in **Project links**, and then drag it into your report loop.

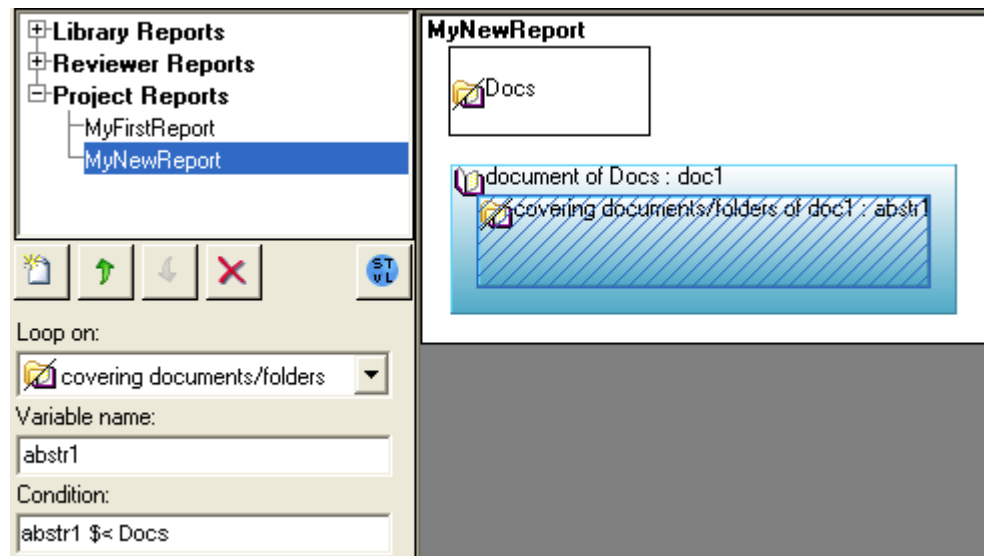


A new iterator **abstr1**, as defined on documents and folders, is added in the Data tab.

7. Covering documents which belong to your selection of documents are the only thing that is of interest. Therefore, you want to add a condition to covering documents that will filter only the relevant documents. This condition can be written in OTScript language and is defined in the covering documents/folders loop.

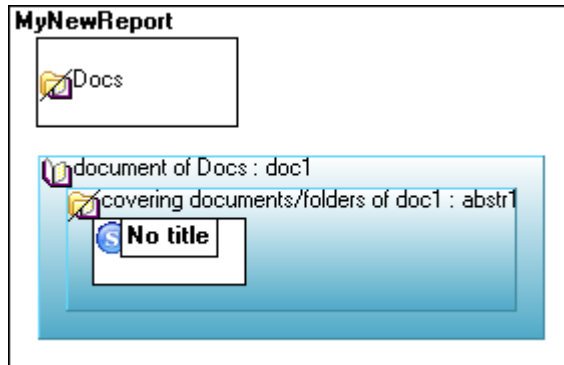
Next select the covering documents/folders loop. It is highlighted with hatching.

In the Condition field enter the code displayed in the figure between the iterator **abstr1** and the parameter **Docs** (\$< symbol means belonging to a list).



8. Now create a section to structure your report. Each created section will display each covering analysis of documents one by one.

Select the section element from the Structure tab and place it in the last added loop.

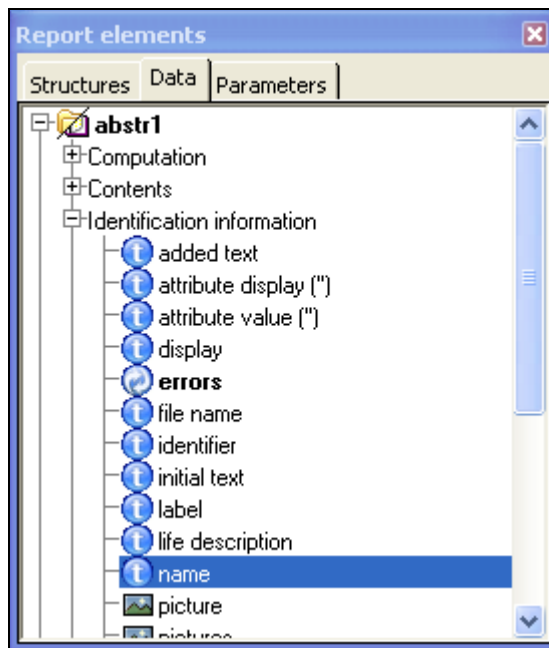


9. Name the section title using the textual information.

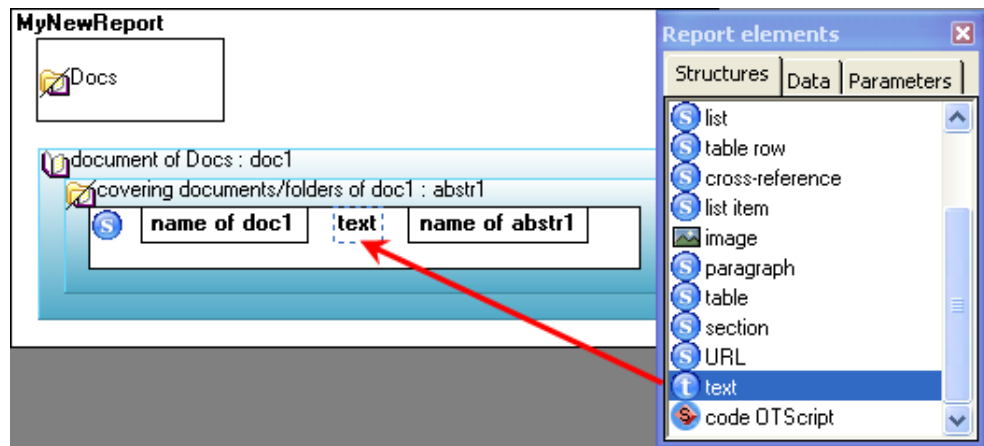
Expand **doc1** in the Report elements. Drag the **name** method from the **Identification information** of **doc1** into the title area.

Repeat the operation on covering documents and folders and drop **abstr1 name** into the title.

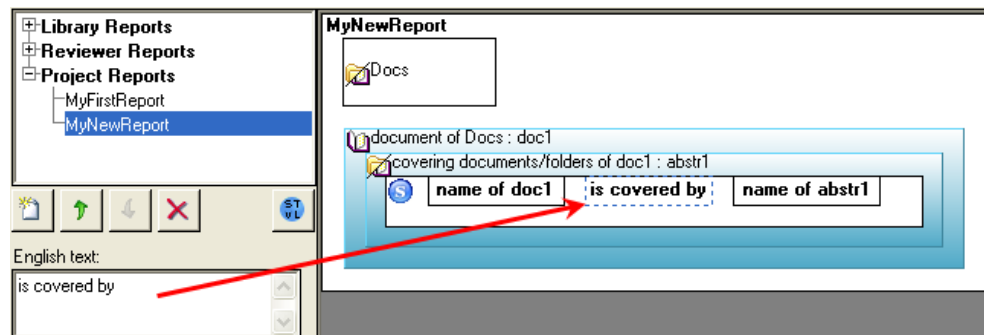
The following example shows the name method.



10. Now select a text element from the **Structure** tab and drop it in the title, as follows.



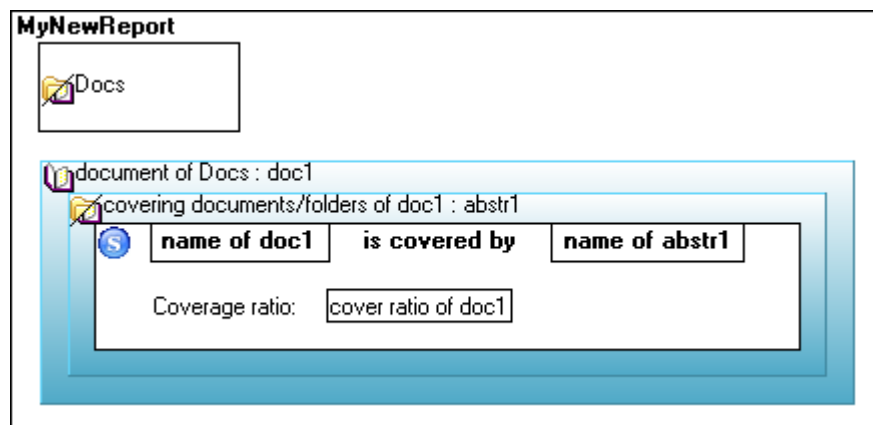
Fill the text field on the left area with an "is covered by" sentence.



11. It is important to know the coverage ratio of requirements for the selected documents.

Add a text to introduce the coverage ratio to display. For this, select paragraph from **Structure** tab and drop it in the section. Now, add a text element structure to the paragraph. In the corresponding field, name this text: "Coverage ratio".

In the same paragraph, drag and drop the **cover ratio (abstr1)** element picked up from **doc1/Computation**.



Consequently Parameter 1, takes the following value:

Parameter 1:
abstr1

12. Now, create a table of 4 columns to display detailed information concerning the analyzed documents. Select the **Table** element from Structures and drop it into the section. By default a new table has only 2 columns. You can add more by changing the value in the form.

Number of columns:
4

☐ Non produced when empty

--	--	--	--

13. Successively name your columns with: Upstream, Description, Downstream and Description values. To create your column title, drop a **text** element into the headers of the table and then complete the corresponding field forms.

English text:
Des

Secondary language text:

Coverage ratio: cover ratio of doc1

Upstream	Des	text	text
----------	-----	------	------

14. The purpose of this example is based on the analysis of the requirements' coverage of input documents. Drop **requirements and macro-requirements** from the **Contents** part of **doc1** into the table.

MyNewReport

Docs

document of Docs : doc1

covering documents/folders of doc1 : abstr1

name of doc1 is covered by name of abstr1

Coverage ratio: cover ratio of doc1

Upstream	Description	Downstream	Description
requirements and macro-requirements of doc1 : abstr2			

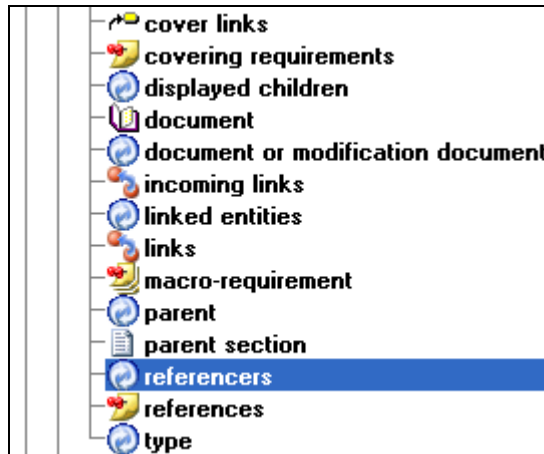
Report elements

Structures Data Parameters

- abstr1
- abstr2
- doc1
 - Computation
 - Contents
 - all sections/entities/requirements
 - documents
 - entities
 - leaf requirements
 - macro-requirements
 - requirements
 - requirements and macro-requirements
 - sections/entities/requirements
 - Identification information
 - Impact Matrix
 - Misc

15. To make it more explicit, it is better to rename **abstr2** in **Req**.

Now, we need to know which references cover the requirements of the selected documents. To get the referencers of the **Req** requirements, drop the **referencers** elements from the **Req/projects links** tree into the requirements loop.



Name the corresponding new iterator **Ref**.

16. To be sure to catch only referencers becoming from the set of selected documents create an OTScript condition:

```
(document = abstr1) OR (root = abstr1)
```

(In the OTScript section of this document, you will find such a condition creation explained with more details).

17. Now, insert information to fill the traceability matrix.

Drop a **table row** element from Structures tab into the references loop.

In the first cell, insert the **identifier** of the requirements **Req/Identification information**.

In the second cell, insert the **text** of the requirements **Req/Identification information**.

In the third cell, insert the **display** of the requirements **Ref/Identification information**.

In the fourth cell, insert the **text** of the requirements **Ref/Identification information**.

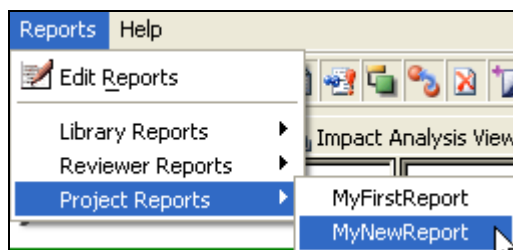
18. Finally, the report structure is finished.

The following screenshot is an example of what the document will look like.

19. Do not forget to click on **OK** to save it.

Now it is the time to generate your document and to create your first traceability matrix report.

This report needs input documents. Select documents: Product Specification and Design Specification from the main window of Rhapsody Gateway. Select your new project name from the Reports menu.



Enter a name for your report then choose a report format file, i.e. rtf.

You will obtain a report like the following one:

<i>MyNewReport</i>			
1. Product Specification is covered by Design Specification			
Coverage ratio: 100			
Upstream	Description	Downstream	Description
PS_USB_REQ1	The hardware shall support USB 1.0.	1.1 USB 1.0 Speeds	Hardware supports USB 1.0
PS_USB_REQ2	The hardware shall support USB 2.0	DS_USB2_REQ1	Low Speed: 1.5 Mbps
PS_USB_REQ2	The hardware shall support USB 2.0	DS_USB2_REQ2	Med Speed: 12 Mbps
PS_USB_REQ2	The hardware shall support USB 2.0	DS_USB2_REQ3	High Speed: 480 Mbp
PS_AI_REQ1	The hardware shall support 10 analog input channels	2.1 Analog Input	Hardware supports analog input : 10 channels
PS_AI_REQ1	The hardware shall support 10 analog input channels	DS_AI_REQ1	14 bits single-ended
PS_AI_REQ1	The hardware shall support 10 analog input channels	DS_AI_REQ2	13 bits differential
PS AI REQ1	The hardware shall	DS AI REQ3	48 kS/s

OTScript Language Essentials

This section provides an overview on how to use OTScript language. This language is especially used to write OTScript code in creating customized reports. The topics are as follows:

- ◆ OTScript Language Presentation
- ◆ OTScript General Syntax
- ◆ OTScript Writing in Rhapsody Gateway
- ◆ Example Analysis

Note

To use and benefit from this chapter, the user needs to have prior knowledge of the Report Editor contents. Report Editor allows the user to create his own report models in order to generate reports. See the Introduction to Report Editor Chapter for more information.

OTScript Language Presentation

OTScript is a high level language developed by Dassault Systèmes. It is a class-based object-oriented language. Thus when an object receives a message, a method associated with this message is executed. The result of the execution is returned as an answer of the object to the message.

OTScript is also a language with very concise syntax, list manipulation capabilities, native handling of binary associations, rule-checker and user interface description language.

OTScript is a strongly typed language. This means that the type of each expression must be known at compile-time. Some predefined types are provided by the language to permit type declaration.

OTScript is a rich language and only a sub-part of this language can be used in Rhapsody Gateway tool. Moreover, since OTScript is a proprietary language, this manual only provides an overview of its capabilities.

OTScript General Syntax

OTScript language is case sensitive. The code coloration and the completion are available for OTScript code writing.

OTScript supports the following primitive data types:

- ◆ Integer
- ◆ Real
- ◆ Boolean
- ◆ String

Reserved Words

OTScript regards the following words as reserved.

TRUE
FALSE

Predefined Variables

EACH

Current receiver.

Examples:

`EACH.requirements`

(If selection is a document, returns the list of all the requirements of current document.)

Operators

The following operations can be performed on Integer and Real elements:

- ◆ Arithmetic: addition, subtraction, multiplication, division (/)
- ◆ Comparison: <, >, <=, >=, =, /= (not equal)

The following operations can be performed on Boolean elements:

- ◆ Comparison: =
- ◆ Boolean: AND, OR, NOT(bool)

Examples:

- ◆ NOT(V)
- ◆ (if V=TRUE then NOT(V) => FALSE
- ◆ if V=FALSE then NOT(V) => TRUE)

The following operations can be performed on String elements:

- ◆ Concatenation: +

- ◆ Comparison: <, >, <=, >=, =, /= (not equal)

Testing Belonging to a Class

`obj1 ISA class1`

(Returns TRUE if the object belongs to class1 class)

Examples:

`doc1 ISA Folder`

(Tests if doc1 is from the Folder class. Result is TRUE if yes, FALSE if no.)

Some examples of classes are Folder, Section or Requirement.

Testing Belonging to a List

`obj1 $< list1`

(Returns TRUE if the object belongs to list1 list)

Examples:

`doc1 $< docs`

(Tests if the object obj1 belongs to list1 list. Result is TRUE if yes, FALSE if no.)

Testing Emptiness of a List

`$SOME(list1)`

(Returns TRUE if list1 is not empty)

Examples:

`$SOME(requirements)`

(Tests if the requirements list of a current document is not empty. Result is TRUE if yes, FALSE if no.)

`$NO(list1)`

(Returns TRUE if list1 is empty)

Examples:

`$NO(requirements)`

(Tests if the requirements list of the current document is empty. Result is TRUE if yes, FALSE if no.)

Testing String Matching

STRMATCH(string1, string2)

Test if string1 matches string2. Use * for 0-n characters and # for one character.

Examples:

```
STRMATCH(doc1.name, "TestCard*")
```

(Tests if doc1 name begins with "TestCard" string.)

Control Structures

<expr1>.<expr2>

Evaluate expression expr2 on expr1.

Examples:

```
coveredDocuments.requirements.addedText
```

(Returns the Add Texts elements which have been added to the requirements of the current selection covered documents.)

Note

Each expression is applied to its preceding element.

<expr>[<cond>]

Evaluate expr expression and select only elements verifying cond condition.

Examples:

```
requirements[STRMATCH(label, "DS_*")]
```

(Returns the requirements whose label matches the string "DS_".)

Frequently Used Statements

To express a concept of **existence** and **condition**, use the **\$SOME** function followed by [].

Examples:

```
$SOME(documents.leafRequirements[isUncovered])
```

(Returns TRUE if there are some documents which have uncovered leaf requirements.)

To express a concept of **belonging**, use a combination of **EACH** and **ISA**.

Examples:

```
EACH ISA Folder
```

(Returns TRUE only for elements which are a folder.)

Writing OTScript in Rhapsody Gateway

In Rhapsody Gateway, OTScript language is essentially used as condition writing for customized generation models of **Report Editor**.

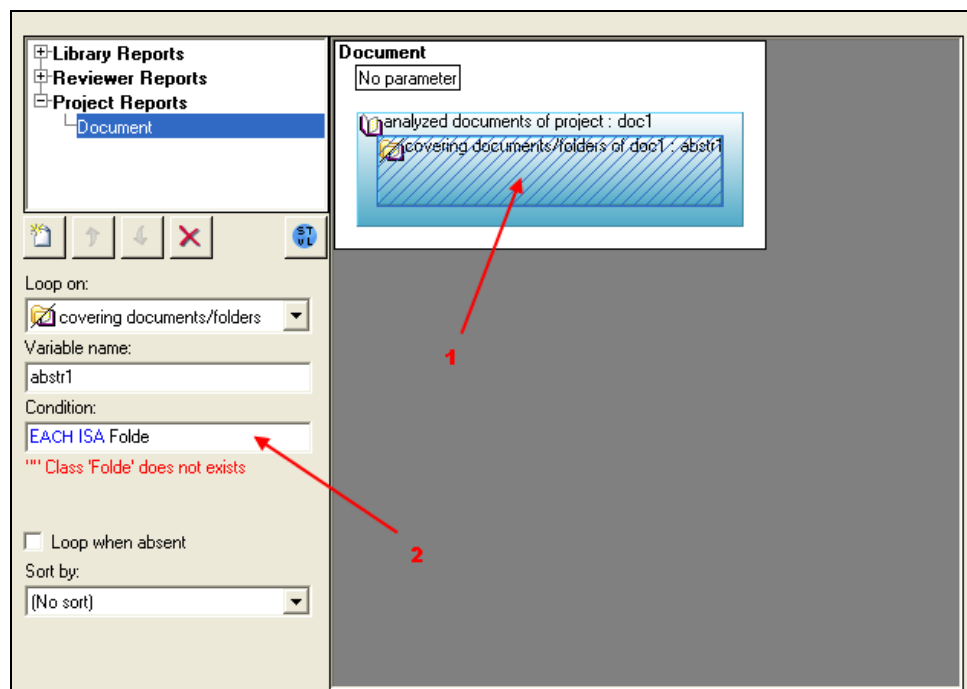
A condition is a Boolean, numeric or string expression that is evaluated as True or False. If the condition is Null, the condition is treated as False.

When writing a condition, its purpose and the entities implied in are to be identified. Entities should be handled using the OTScript methods.

Creating an OTScript Condition

Most of the Rhapsody Gateway elements, for instance loops, own a condition field. This field allows the user to enter his OTScript condition. Follow these steps to create a condition:

1. Select in the graphical view the element with which you want to associate a condition. This element appears with hashing.
2. Enter OTScript code, such as **EACH ISA Folder**, in the condition field:



Note

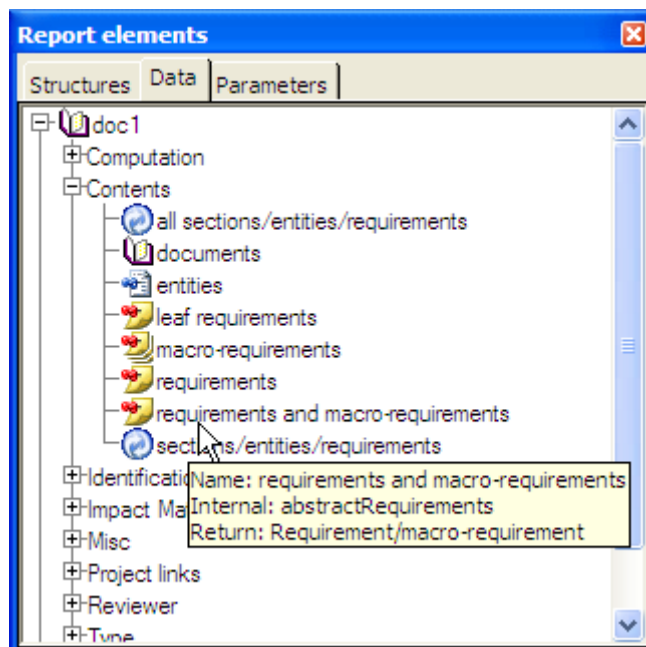
The OTScript syntax checking is made on-the-fly. The OTScript code stays in red while the code contains syntax errors.

Accessing the OTScript Methods

A dedicated accessing OTScript method is associated with each element that can be inserted in the graphical view. All these methods that depend on the context can be combined into conditions. To visualize this access method, follow these steps:

1. Open the **Report elements** window.
2. Expand the wished element methods.
3. Fly over methods to display information in a tooltip.
4. Tooltip displays at least three properties for the element:
 - ◆ **Name**—The displayed name of the method.
 - ◆ **Internal**—The internal name of the method, i.e. the OTScript method name.
 - ◆ **Return**—The return type of the method.

The following figure shows an example of information displayed in a tooltip.



Here for **requirements and macro-requirement** function, the OTScript method is **abstractRequirements**. This method returns requirements and macro-requirements.

Frequently Used Methods

Some OTScript methods are often used in condition writing. These methods are gathered in the following table:

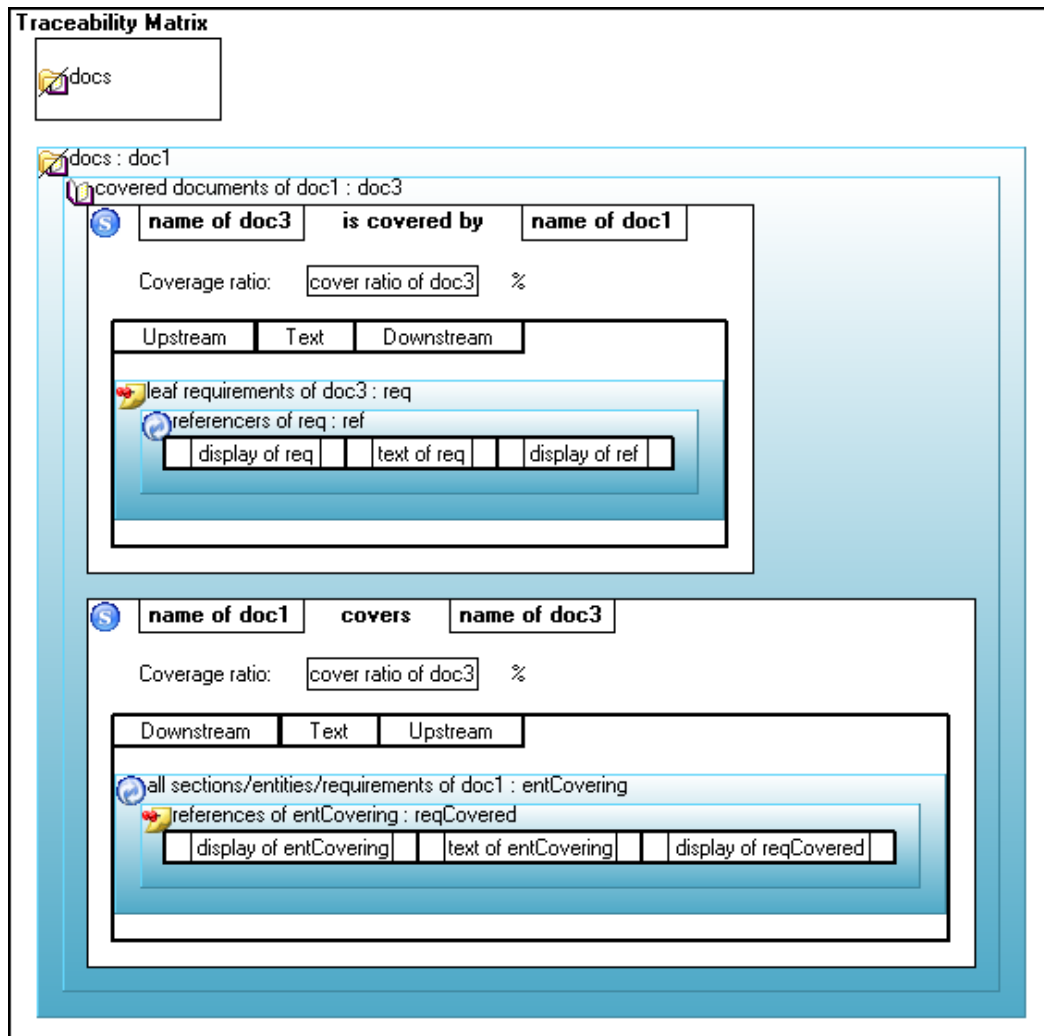
Method	Description
<code>document</code>	Document of current element.
<code>isDerived</code>	Requirement is derived.
<code>isCovered</code>	Requirement is covered.
<code>isUncovered</code>	Requirement is uncovered.
<code>isFiltered</code>	Requirement is visible.
<code>root</code>	Root is a folder which contains the document, if the document is in a folder. or Root is the document itself if it is not contained in a folder.
<code>\$SOME(attributes[typeName="Priority"])</code>	Requirement has an attribute of type "Priority".
<code>document ISA ModificationDocument</code>	Document is a modification document.
<code>STRMATCH(doc1.typeName, "*SRS*")</code>	Document's type contains the string "SRS".
<code>\$CNT(coverLinks) >= 2</code>	Requirement is covered at least twice.

Example Analysis

Rhapsody Gateway provides default reports which sometimes use OTScript conditions. In this next section we will analyze a provided report to learn why conditions are sometimes needed and how to write them.

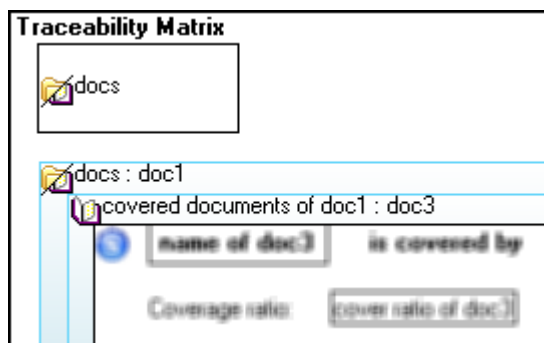
The following study is based on a **Traceability Matrix** report. The aim of this report is to obtain the coverage of upstream documents by downstream documents. Therefore, the objective is in the second document to catch all the requirements which are referenced in the first document.

The following figure shows the **Traceability Matrix** report structure as it is provided in the **Report Editor**:



Sub-part Analysis

For the analysis, we will only focus on following the model sub-part:



docs is the set of documents and folders selected by the user to generate their traceability matrix.

doc1 is an iterator to loop on docs elements.

doc3 is an iterator to loop on the set of documents which are covered by doc1.

For a given document **doc1**, the **covered documents** function returns all the documents which are covered by **doc1**. Thus only covered documents which belong to selected documents **docs** are relevant. Resulting documents have to be filtered to restrict the set of relevant documents.

In conclusion, a condition is needed to reduce the scope of analyzed documents.

Writing Condition Process

The user needs to accurately write a filter named condition in Rhapsody Gateway Report Editor. This condition has to compare the documents resulting from **covered document** function execution with his selected documents.

The user can implement such a condition thanks to the Rhapsody Gateway OTScript language subset.

User's condition must verify if a **covered document doc3** belongs to **docs**.

The corresponding OTScript expression is:

```
(doc3 $< docs)
```

Expressed with words: **doc3** document should belong to **docs**.

However, **docs** can contain both documents and folders. Add-ons need to be added to this condition.

User's condition must also verify if a **covered document doc3** should be inserted into a folder which belongs to the folders of **docs**.

Corresponding OTScript expression is:

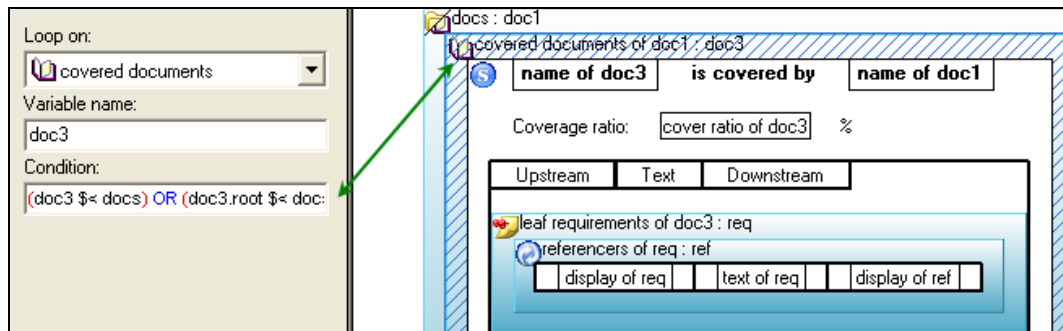
```
(doc3.root $< docs)
```

Expressed with words: **doc3** folder should belong to **docs**.

Now, the previous two expressions should be concatenated thanks to the **OR** operator to get the final condition:

```
(doc3 $< docs) OR (doc3.root $< docs)
```

The following figure shows the concerned loop and its associated condition:



Go Further with OTScript

OTScript is a complex language. In this document we have seen a summary of helpful syntaxes for writing with the OTScript language. An advanced guide is also available and is provided at the end of the training courses. One of the training courses deals with customized reports and treats condition writing.

Sometimes conditions are hard to express. The Support Team and Application Engineers can also help you for advanced definitions.

Management of Product Configuration

When the user customizes some templates or models for his own project, most of the time he would find it very helpful to re-use his work. In effect, customized information can be stored in several configurations.

Re-use of Customization Work from one Project to another

As described in the section concerning Rhapsody Gateway files, your customization work performed for one given project will lead to the creation of the directory of this project:

- ◆ Some **.types** files, containing all your types of analysis.
- ◆ A **<project name>** file, containing the filters you created.
- ◆ A **doc_templates** directory, containing your customized templates for formats such as rtf or csv, templates that will be used for documentation generation.
- ◆ A **doc_models** sub-directory, containing all the reports you created from the Documents Editor.

To re-use this work, you have several solutions:

- ◆ If you want to make the customized work available for another project, you can copy all these specific elements from your first project to all other concerned project directories.
- ◆ If you want to include these elements in your default configuration, making them available automatically for ALL projects, include these elements in the `<tool installdir>\config` directory:
 - The **.types** files have to be included in the `<tool installdir>\config\types` directory. Create your own directory (it will be presented as a folder in the Types Editor) and drop your **.types** file in.
 - The files you created for documentation generation formats (**rtf**, **csv**, etc.) must be located in the `config\doc_templates` directory.
 - The reports you created (XML files) must be located in the `config\doc_models` directory.

Additional Configuration Directories

Several configuration directories can be specified in Rhapsody Gateway. These configurations allow the user to save his customized files in one or several directories of his network environment in order to re-use and share them between his different projects.

All the information such as types, doc_templates or doc_models can be shared except for the filters information.

Defining Configuration Directories

Some additional configuration directories can be defined at Rhapsody Gateway startup. Every time several paths can be specified using a coma (,) to separate these paths. Three methods are available to define the location of shared configuration directories. Note that in all cases, the path to the config directory of the Rhapsody Gateway installation must be listed first, preceding the user shared config directory path.

To specify a user shared config directory, define the environment variable RG_CONFIG as follows:

```
RG_CONFIG =  
$TOOL\config,<AdditionalDir1>,<AdditionalDir2>
```

Another way is to modify the rg.ini file.

So as to specify either in the global (<path_to_Rhapsody Gateway>\bin.w32) or in the user (%USERPROFILE%\Application Data) rg.ini a "Config" parameter in the [Files] section as follows:

```
[Files]  
Config=$TOOL\config,<AdditionalDir1>,<AdditionalDir2>
```

The third way is to specify the configuration directories directly in the Rhapsody Gateway launching command.

So specify either on the command line or in a shortcut the following command:

```
<path_to_Rhapsody Gateway>\bin.win32\rg.exe -l eng  
Config="$TOOL\config,<AdditionalDir1>,<AdditionalDir2>"
```

Structuring Configuration Directory

A configuration directory must have a specific arrangement. It shall contain sub-directories and files shall be located in these sub-directories. More precisely, the files need

to be directly underneath the directory indicated in the path, else the file will not be considered.

For instance, AdditionalDir1 directory should include directories as below:

```
types
  | My Office
                                | word.types
  | My Code
                                | code.types
doc_models
  | My reports.xml
doc_templates
  | portrait.rtf
  | portrait_official.rtf
otscript
  | custom.br
```

Categories Creation

Document Categories

Document and folder categories can be set in the [DocumentCategory] section from the rg.ini file of the config directory. This section contains two kinds of category definitions.

The first category definition corresponds to the type folders association of categories. The syntax is Folder_<type folders>=<Category_name>.





















```
[DocumentCategory]
Folder_design=Design
Folder_requirement tools=Requirements
Folder_test=Tests
Folder_UML=Design
```

The second category definition corresponds to the definition of categories for documents and folders. A specific color can be assigned with a category so with a document or a folder.

```
[DocumentCategory]
Count=4
1=Requirements, Exigences
2=Specification, Spécifications
3=Design
```

```
4=Tests, Validation
1Color=red
2Color=orange
```

Count indicates the number of categories. The order of categories is the order that they will appear in the Management View. The association syntax is
`<category_order>=english category name,french category name,other language category name`. The color syntax
`<category_order>Color=<color_value>` assigns a color to a category.
`<color_value>` can be a code or a name from the following list:

Name	Value	Color	Name	Value	Color
aqua	#00FFFF		navy	#000080	
black	#000000		olive	#808000	
blue	#0000FF		orange	#FFA000	
brown	#A02820		purple	#800080	
chartreuse	#80FF00		red	#FF0000	
fushia	#FF00FF		silver	#C0C0C0	
gray	#808080		teal	#008080	
green	#008000		violet	#F080F0	
lime	#00FF00		white	#FFFFFF	
maroon	#800000		yellow	#FFFF00	

Cover Categories

Cover categories can be set in the [CoverCategory] section from the rg.ini file, in the config directory.

The following definition block corresponds to the definition of categories for covers.

Count indicates the number of categories. . The syntax is
`<category_order>=english category name,french category name,other language category name`

```
[CoverCategory]
Count=3
1=Satisfies, Satisfait
2=Verifies, Vérifier
3=Implements, Implémente
```

Upgrades and New Releases

Of course, when you upgrade your version of Rhapsody Gateway (from version N to N+1) the installer creates a default configuration that does not take into account the configuration you performed as described in the previous section.

If you want to keep using your customized configuration of Rhapsody Gateway with the new release:

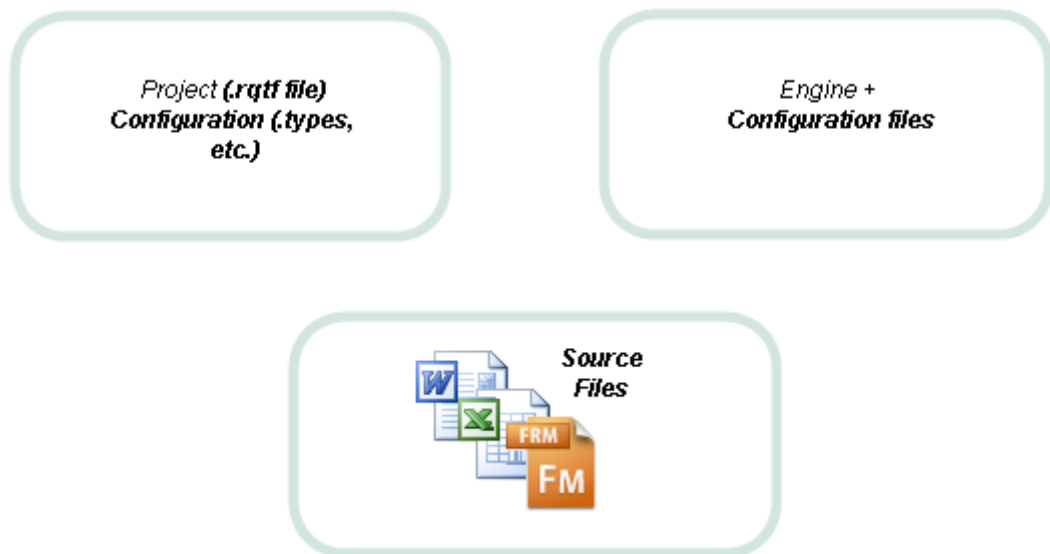
- ◆ Carefully read the *Release Notes* of the next version to know if changes occurred in the configuration definition. Direct transition is guaranteed from one Rhapsody Gateway version to the next one. Even if changes occur in the management of configuration files, Rhapsody Gateway will open the old configuration files and take charge of their upgrade.
- ◆ Customize your new version, as well as the previous one and as described in the previous section.
- ◆ Do not hesitate to contact the Support Team if you need help.

Installation: Local or Network?

Because Rhapsody Gateway does not use any centralized database but analyzes directly the project files, several configurations have to be considered:

- ◆ The Rhapsody Gateway program itself,
- ◆ The Rhapsody Gateway project,
- ◆ The project files or elements to be analyzed.

Each of these items can be available locally (on the local machine) or from the network.



In theory, all the configurations below can be used:

	Rhapsody Gateway Program	Project	Source Information
1	Local	Local	Local
2	Local	Local	Network
3	Local	Network	Local
4	Local	Network	Network
5	Network	Local	Local
6	Network	Local	Network
7	Network	Network	Local
8	Network	Network	Network

In practice:

- ◆ Configurations 1 to 4 are based on a local installation of Rhapsody Gateway. Each machine has the software installed. If a Corporate customization of Rhapsody Gateway has been developed, it has to be installed on each machine
- ◆ Configuration 1 is often used by single users
- ◆ Configuration 2 is often used
- ◆ Configuration 3 is rarely used: when the Rhapsody Gateway project is on the network, that means project files are also on the network
- ◆ Configuration 4 is often used
- ◆ Configuration 5 is rarely used: When the Rhapsody Gateway project and the source files are local, that means the user need is local. In such a case, Rhapsody Gateway is installed locally
- ◆ Configuration 6 is used
- ◆ Configuration 7 is rarely used: when the Rhapsody Gateway project is on the network, that means project files are also on the network
- ◆ Configuration 8 is often used

Note

If several users need to use a customized configuration for a given project, two options are possible:

- The project is saved on the network and all the users work with the same project.
- Each user has a local project definition and analyzes the project files on the network.

The first solution is usually preferred, but both are possible. The second solution is

often used when users need to work on different subsets of project files, according to their role in the project (development, verification, etc.)

Note also that Rhapsody Gateway can combine analysis of local and network files, and even Windows and Unix files.

Index

A

Add a type for added elements	29
Add new type	29
Add XML type	29
Attribute	31, 56
Available values	58, 60

C

Code	
inserting	108
Column index	63
Comment	34
Constant file	35
Convert tool	33
Copy	29
Copy For	55

D

Delete	29
Deletion of an element	36
Depth	45, 49
Display	28, 38, 41, 48, 50, 56, 59, 61, 67
Duplicate	17, 29, 31

E

Edit	
menu	30
Edit tool	33
Editor	
Regular expression tester	84

Report	95
Types	23
XML tester	90
Encoding Format	33
End regular expression	39, 42, 45
Entity	31, 48
Excluding files	36
Exclusion of an element	37
Exclusion of text	36
Expressions to exclude	36
Extension	33

F

Field	45
Filters	35

G

GUID	45, 49, 57, 59
------------	----------------

I

Identifier	45, 49, 57, 59
Identifier format	42, 45, 48, 50, 56, 59, 61
Image	
adding	107
Inverse regular expression	51, 54
Is displayed	58, 60
Is multi-valued	58, 60
Item	
adding in list	108

L

Label.....45, 49, 57, 59
 Link 31, 60
 List
 adding108

M

Macro-requirement.....31, 41
 Merge homonymous requirement35
 Merge homonymous sections35
 Multi-Types20

N

Name 28, 38, 41, 44, 48, 50, 56, 59, 61, 63, 64,
 65, 67

O

OTScript134
 accessing methods139
 conditions.....119, 127, 131, 138, 143
 example.....141
 inserting code.....109
 methods.....140
 syntax.....135

P

Paragraph
 adding104
 Parameters
 inserting101
 Parent.....45, 49, 57, 59
 Paste29
 Picture.....31, 64
 Prefix expression43, 45

R

Redo30
 Reference.....22, 31, 49
 Reference attribute31, 58

Regular expression... 19, 38, 45, 51, 57, 59, 62,
 63, 64, 65
 examples.....82
 Tester.....84
 testing.....88
 Report100
 adding.....100
 creating.....100
 duplicating.....100
 editing.....101
 Report Editor95, 96
 accessing95
 example125
 generating report122, 132
 options.....118
 template.....120
 windows96

Report elements
 structures113

Reports elements
 data.....114

Requirement.....31, 44, 72

Row
 adding in table105
 Row regular expression65

S

Section31, 38
 adding.....105

Source50, 61

Structures
 Anchor.....112
 code108
 cross-reference111
 Generation.....112
 image107
 include file.....110
 list 108
 options.....118
 OTScript code127, 131
 paragraph.....104

section	105
table.....	105
text	106
URL	111
Sub-expression to delete an element	36
Sub-regular expression ...	25, 45, 51, 57, 60, 62, 64

T

Table	
adding	105
Target	50, 61
Target document.....	50, 61
Tester.....	84, 85, 90, 91
importing directory	86, 92
importing file	85, 91
importing intermediatefile	87, 93
window	84, 90
Text 31, 45, 49, 63	
adding	106
Text display	38, 41, 48, 67
Traceability graph violation	52
Type	

adding.....	31
adding for added element	31
buttons.....	28
deletion.....	31
duplicating.....	17
Type of analysis.....	15
Types editor	23, 27

U

Unauthorized value.....	58
Uncovered requirement	26
Undefined requirement	52
Undo	30

X

XML	
Tester.....	90
XML Syntax	
testing	93
XML type	
adding.....	31
definition	65