

Eclipse ProR User Guide

by the RMF Community

May 20, 2016

Contents

1	Introduction	7
1.1	RMF, ProR, Essentials and formalmind Studio	7
1.2	Compatibility	8
1.3	Conventions	8
1.4	Improve this Document	9
1.5	Acknowledgements	9
1.6	License	10
2	Overview	11
2.1	Requirements Engineering & Management	11
2.2	Tools	11
2.3	Requirements Interchange Format (ReqIF)	12
2.3.1	ReqIF History	12
2.3.2	Previous Versions of ReqIF	13
2.3.3	Internal Attributes	13
2.4	Terminology	13
2.4.1	The Most Important ReqIF Elements	14
2.4.2	SpecElements	15
2.4.3	Comparing Excel and ReqIF	16
3	Tutorial	19
3.1	Basic Concepts	19
3.2	Tutorial 1: Creating a Basic ReqIF Model	19
3.2.1	Install ProR	19
3.2.2	Create the Model	19
3.2.3	Customizing the SpecType	21
3.2.4	Showing the New Attributes	22
3.2.5	Adding new SpecObjects	23
3.2.6	Rearranging Elements	24
3.2.7	Export Specification as HTML	24
3.2.8	Conclusion	25
3.3	Tutorial 2: Use Presentations	25
3.3.1	ID Presentation	25
3.4	Tutorial 3: Advanced SpecTypes	27

3.4.1	Multiple SpecTypes	27
3.4.2	Other SpecTypes	29
3.5	Tutorial 4: Links Between Elements	30
3.5.1	Creating SpecRelations	30
3.5.2	Linking Via Context Menu	30
3.5.3	Linking Via Drag and Drop	31
3.5.4	SpecRelations in the User Interface	31
3.5.5	SpecRelations in the Outline View	32
4	Reference	33
4.1	Eclipse	33
4.1.1	Prerequisites	33
4.1.2	Installation	33
4.1.3	Updates	34
4.1.4	Installing New Features	34
4.1.5	Workspaces	35
4.1.6	Committer License Agreement (CLA)	35
4.2	ProR User Interface	35
4.2.1	Editors and Views	36
4.2.2	RMF Specification Editor	37
4.2.3	Project Explorer View	40
4.2.4	ReqIF Validation	41
4.3	Configuration and Preferences	42
4.3.1	Global Preferences	42
4.3.2	General Configuration	42
4.3.3	Datatype Configuration	43
4.3.4	Presentation Configuration	44
4.3.5	Column Configuration	45
4.4	Access Control	45
4.5	Import and Export	45
4.5.1	Import	45
4.5.2	Export	47
4.6	Searching and Filtering	47
4.6.1	Quicksearch	48
4.6.2	ReqIF Search	48
4.6.3	Raw ReqIF Search	52
5	Presentations	53
5.1	Working with Presentations	53
5.2	Default Datatype Handlers	54
5.3	Built-in ProR Presentations	56
5.3.1	ID Generator Presentation	56
5.3.2	Headline Presentation	56
5.3.3	Linewrap	57

<i>CONTENTS</i>	5
6 Style Guide	59
6.1 Alphabetization	59
6.2 Punctuation	59
6.3 Contributing	60
6.3.1 Gerrit for Contributions	60

Chapter 1

Introduction

The importance of requirements has been recognized for a long time. And with the advent of computer-aided engineering tools, a number of proprietary solutions have popped up all over the place. While this has helped organizations to manage their requirements more efficiently, interoperability became a major issue.

The development of the ReqIF standard for requirements exchange finally provided a standard, feature-rich way of accessing requirements data. Eclipse was the obvious choice for a reference implementation of this open standard. The result is the Eclipse Requirements Modeling Framework, a complete, open source, user-friendly implementation of ReqIF.

This handbook is a comprehensive documentation of the ProR tool, which is based on Eclipse RMF. All answers with respect to tool use should be answered here. Furthermore, it contains a small tutorial (Chapter 3) to get you started quickly.

Keep in mind that tools are meant to support processes, not the other way around. ProR is a flexible tool, and it can be tailored to support your processes. But development processes are explicitly outside the scope of this handbook.

Information. If you are interested in adopting a lightweight development process that can be used with ProR, visit our initiative re-teaching.org.

1.1 RMF, ProR, Essentials and formalmind Studio

There are a few derivatives of the RMF project that may be confusing. The following will help you to understand the ecosystem and how the pieces fit together:

RMF. The Requirements Modeling Framework (RMF) is an open source project that is managed by the Eclipse Foundation. It consists of software code, documentation, mailing lists, online forum, etc.

ProR. ProR is the name of the user interface that allows users to work with ReqIF-based requirements. ProR is typically installed into existing Eclipse installations. There used to be a standalone build of ProR, which has been discontinued.

RMF Core. While ProR is the frontend of RMF, the Core is the back end. This distinction is primarily intended for developers.

ProR Essentials. The company Formal Mind created this collection of add-ons that make ProR much more usable. For instance, Essentials allows for the editing and rendering of formatted text.

formalmind Studio. As there is no standalone version of ProR available, Formal Mind created one, which comes with Essentials preinstalled. For users who “just want to edit requirements”, this is the most convenient way of getting started.

1.2 Compatibility

ProR and RMF require at least Java 6 and Eclipse 3.8. Some features of ProR Essentials require Java 7 from Oracle (not openJDK).

formalmind Studio is based on Eclipse Luna (4.4).

1.3 Conventions

Throughout this book you’ll see the following conventions:

Information. Tips are marked with this icon. They usually summarize how to work more effectively. Details about the subject matter are found in the text.

Warning. Warnings are marked with this icon. They point out things that can go wrong, or important steps that are easily missed.

Example

Examples often demonstrate how a concept is applied in practice. They will be marked with a black square at the beginning and at the end, as shown with this text.

When referring to **menus** or **user interface elements**, they are shown like here on gray background in small capitals.

When we introduce a new *term* or want it to stand out, it will be marked like *this* in the text. Often this is done only the first time that it appears.

Terms that represent ReqIF model elements (e.g. SpecObject) are capitalized. Sometimes, we abbreviate them, for instance “SpecElement” instead of the full “SpecElementWithAttributes”. They will still be capitalized to indicate that these are model elements.



Figure 1.1: The RMF team during a Sprint in April 2012 in Düsseldorf, Germany: Lukas Ladenberger, Mark Brörkens, Ingo Weigelt, Said Salem and Michael Jastram (left to right)

Definition. Definitions appear throughout the text and concisely define new terms. They appear in the text wherever they are introduced first. Please take advantage of the index at the end of the book to quickly find definitions.

1.4 Improve this Document

Documentation is one of those things that gets easily neglected in open source projects. It is also one of the easiest for outsiders to contribute to. Please refer to Chapter 6 if you would like to help improving this user guide.

1.5 Acknowledgements

Many parties were involved in the creation of RMF. We would like to thank the core team that made it possible.

The roots of this project were created by Andreas Graf, Michael Jastram and Nirmal Sasidharan, who joined together individual projects to create RMF. Their efforts were financed by the research projects itea Verde and FP7 Deploy. RMF was assembled at the Eclipse Foundation, where it has been active ever since. Figure 1.1 shows four of the five RMF Committers at a joint coding

session (missing is Andreas Graf).

1.6 License

This work, or parts thereof, are licensed under the Eclipse Public License Version 1.0 as part of the Eclipse Requirements Modeling Framework (RMF) project at eclipse.org/rmf.

Chapter 2

Overview

This chapter provides a high-level overview of requirements engineering, requirements tooling, ReqIF and the terminology.

2.1 Requirements Engineering & Management

This book is concerned with ProR a tool for requirements engineering.

Requirements Engineering. “Requirements engineering (RE) refers to the process of formulating, documenting and maintaining software requirements.” (Wikipedia).

We’d argue that RE also includes *system* requirements. Further, requirements are typically unstructured natural language. Of high interest these days is model-driven requirements engineering.

But engineering the requirements is not enough: they need to be *managed*.

Requirements Management. “Requirements management is the process of documenting, analyzing, tracing, prioritizing and agreeing on requirements and then controlling change and communicating to relevant stakeholders. It is a continuous process throughout a project.” (Wikipedia).

2.2 Tools

There are many tools available for requirements engineering. These include free or cheap ones, like Microsoft Word and Excel, Wikis and issue trackers. There are expensive, professional ones available, like IBM® Rational® DOORS®, PTC Integrity or Visure IRQA. Lately, there are also web-based tools, like Polarion.

ProR falls into the category of free tools. But compared to the ones mentioned, it contains important features from professional tools, including traceability and typed attributes. Further, by taking advantage of the Eclipse ecosystem, the tool can be augmented by plug-ins for version support, model integration and much more.

Information. Professional support, commercial components and integration services are available from Formal Mind and other service providers.

2.3 Requirements Interchange Format (ReqIF)

ReqIF stands for Requirements Interchange Format. It is an exchange format for requirements and a data model. ProR is an editor that can directly view and modify ReqIF data.

ReqIF was created to support the exchange of requirements across organizations. For instance, it allows a manufacturer to send requirements to suppliers. The suppliers can then comment and review the requirements, or they can create a system specification that is linked to the requirements.

ReqIF is an official OMG standard.

Warning. ReqIF uses its own terminology. Section 2.4 defines the ReqIF vocabulary and how it relates to the terms used in classical requirements engineering.

2.3.1 ReqIF History

For technical and organizational reasons, two companies in the manufacturing industry are rarely able to work on the same requirements repository and sometimes do not work with the same requirements authoring tools. A generic, non-proprietary format for requirements information is required to cross the chasm and to satisfy the urgent industry need for exchanging requirement information between different companies without losing the advantage of requirements management at the organizations' borders.

The Requirements Interchange Format (ReqIF) described in this RFC defines such a tool-independent exchange format. Requirement information is exchanged by transferring XML documents that comply to the ReqIF format.

In 2004, the HIS (Hersteller Initiative Software), a panel of Germany's automotive manufacturers (Daimler, VW, Porsche, Audi and BMW Group) developed the idea of creating the "Requirements Interchange Format". In 2005, the first version of that format was presented at the REConf, a conference about requirements engineering and management, in Munich. In 2008, the HIS Steering Committee decided that the internationalization and maintenance of the Requirements Interchange Format should be proceeded with the ProSTEP iViP Association. A project was set up and a team was built that includes members of the ProSTEP iViP Association, representatives of manufacturing companies

(Audi, BMW Group, Daimler, VW, Bosch and Continental), tool vendors (At-ego, IBM, MKS) and development partners (HOOD GmbH, PROSTEP AG).

Information. Further reading: The HIS Exchange Process for Requirements—all you ever wanted to know.

The ReqIF team expects that making the Requirements Interchange Format an OMG standard increases the number of interoperable exchange tool implementations on the market, fosters the trust of companies exchanging requirement information in the exchange format and provides safety of investments to tool vendors.

2.3.2 Previous Versions of ReqIF

This document is submitted as RFC of the Requirements Interchange Format (ReqIF) to the OMG. Before the submission, the Requirements Interchange Format has been a specification proposed by the HIS and in its latest version, a recommendation of ProSTEP iViP. For these versions, the abbreviation “RIF” has been applied. The HIS released the Requirements Interchange Format as RIF 1.0, RIF 1.0a, RIF 1.1; RIF 1.1a and the ProSTEP iViP released the recommendation RIF 1.2.

As the acronym RIF has an ambiguous meaning within the OMG, the acronym ReqIF has been introduced to separate it from the W3C’s Rule Interchange Format. ReqIF 1.0 is the direct successor of the ProSTEP iViP recommendation RIF 1.2.

Warning. The ProR GUI does not currently support RIF.

2.3.3 Internal Attributes

ReqIF allows users to define the attributes that SpecObjects may carry. In addition to these, there are a number of internal attributes that are defined by the ReqIF standard. Examples include an internal ID or the last change timestamp.

These internal attributes are rarely of interest to users who just want to work with requirements. However, they may be of interest to tool experts, or may be inspected for troubleshooting.

Internal attributes can be accessed from the Properties View, using the **All Attributes** tab.

2.4 Terminology

Working with ProR can be confusing as it uses the terminology from ReqIF. For instance, ReqIF uses *SpecObjects*, rather than *requirements*. In the following, we define the more important terms. More are defined throughout the document. You can use the index to find the definition of terms.

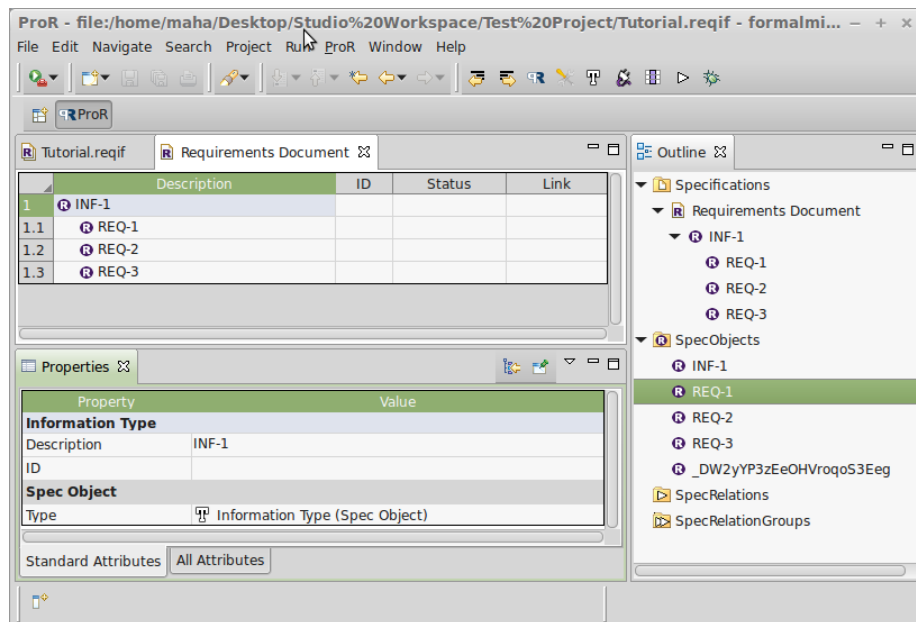


Figure 2.1: Specification example

Information. This book uses ReqIF terminology throughout. Please refer to this chapter to understand the meaning of these terms.

A *ReqIF model* is the data structure that holds all the information together. In practical terms, it's just a file, that usually ends in `.reqif` or `.reqifz`. It contains not just the requirements, but also the data types of those requirements and a lot of other stuff. It has been described in detail in Section 2.3.

ReqIF. ReqIF is an XML-based format for requirements, intended to be used as an exchange format. It is an open OMG standard and described in detail in Section 2.3. The XML data is typically stored in a file with the extension `.reqif`. It is also possible to store the ReqIF file together with associated data (embedded objects) or other ReqIF files. These are then stored in a ZIP archive with the extension `.reqifz`.

Before defining the most important elements, we will provide a brief overview with a concrete example.

2.4.1 The Most Important ReqIF Elements

Figure 2.1 shows a simple ReqIF model, open in ProR.

The **Specification Editor** (the central table) shows the first four *SpecObjects*, as visualized in a specification. The tree-like structure is recognizable:

INF-1 is a node with three children, REQ-1, REQ-2 and REQ-3 (this can be seen by the indentation). Let's look at INF-1 and REQ-1. When one is selected in the main pain, its attributes appear in the **Properties View**, the pane at the bottom.

INF-1 has two *Attributes*, "Description" and "ID". The *SpecType* is "Information Type" (shown as the header in the **Properties View**).

REQ-1, REQ-2, and REQ-3 have three Attributes, "Description", "ID" and "Status" (this is not obvious from the figure). To accommodate this, a column called "Status" has been created. As the "Information Type" has no "Status" attribute, it is not shown in the **Properties View**.

2.4.2 SpecElements

A requirement is called a *SpecObject*. This is arguably the most important element in ReqIF, the actual requirements that you are working with. The SpecObjects of a ReqIF model can be directly accessed in ProR via the outline. It is more common to access them via a *Specification*. When a SpecObject is selected, its details (attributes and internal information) are shown in the **Properties View**.

SpecObject. A SpecObject is a data structure for storing requirements information. It has a number of *attributes*. The most typical attributes include the requirements text and a human-readable ID. A *SpecType* determines the attributes of the SpecObject. It is a *SpecElement*.

There are other elements in ReqIF that have a type and attributes. We call these *SpecElements*, although in the official ReqIF specification, they are called *SpecElementsWithAttributes*.

SpecElement. A SpecElement is an abstract ReqIF element that has a *SpecType* and *Attributes*. Concrete manifestations include *SpecObjects*, *Specifications*, *SpecRelations* and *SpecRelationGroups*.

Creating links between SpecObjects is a central functionality of requirements tools. In ReqIF terminology, links are called *SpecRelations*.

SpecRelation. A SpecRelation is a data structure for connecting two SpecObjects. It contains a *source* and a *target* reference to the SpecObjects that are connected. As a SpecRelation is a SpecElement, it has a type and attributes.

SpecObjects do not have any particular order. SpecObjects can be organized into a tree-like structure by using a *Specification*. A Specification is a root element for a tree of SpecObjects. The SpecObjects are referenced. This means that the same SpecObject can be referenced multiple times, both within one Specification or in different Specifications.

Specification. A Specification is a data structure for organizing SpecObjects into a tree structure. This tree consists of references. As a Specification is a SpecElement, it has a type and attributes.

All SpecElements have a *SpecType*. A SpecType defines *AttributeDefinitions*, which defines the attributes for the SpecElement of that type.

Example

An AttributeDefinition is just a data type with a label. Slightly simplified, examples would be:

- Attribute *ID* of type *String*
- Attribute *Status* of type *Enumeration* with the values *accepted* and *rejected*
- Attribute *ReqIF.Text* of type *XHTML* (rich text).

SpecType. The SpecType defines a set of *AttributeDefinitions*. A SpecElement with the given type has the attributes defined by the AttributeDefinitions.

AttributeDefinition. An AttributeDefinition belongs to a SpecType. It consists of a label and a DatatypeDefinition, which provides the type. Some AttributeDefinitions can be configured further. AttributeDefinitions can also have a *default value*.

Lastly, there are seven *DatatypeDefinitions*, some of which can be customized further.

DatatypeDefinition. DatatypeDefinitions are the fundamental types in ReqIF and include:

-
- Boolean – true or false
- Integer – the range can be customized
- Real – range and precision can be customized
- Date – also includes the time
- String – the maximum length can be customized
- Enumeration – both single and multiple choice are supported
- XHTML – allow embedding objects of any type

2.4.3 Comparing Excel and ReqIF

With the basic terminology in place, we have a quick look at the ProR user interface and compare it with Excel. We do this, as most readers will be familiar with Excel, and it is sometimes used for simple requirements engineering.

Specification. (Excel-equivalent: Sheet) A ReqIF model can have an arbitrary number of Specifications. In the GUI, it is represented as an Excel-like grid view (see Figure 2.1).

The *Specification* is the “container” for the requirements. Think of an Excel document that allows you to create an arbitrary number of sheets. Each sheet can be compared to a single Specification. Most notable differences are: (1) the Specifications are *references* rather than independent entities (which means that the same requirement can be referenced and can appear in multiple places); (2) A Specification manages a hierarchy of requirements, while an Excel sheet is a flat list. This is shown in the Figure, where INF-1 is the parent to three requirements. The hierarchy is visible both in the main editor, as well as in the outline.

SpecObject. (Excel-equivalent: Row) A SpecObject represents the actual requirement, and is typically organized in a Specification.

Each row in the Excel spread sheet is the equivalent of a *SpecObject*. A requirement typically has a number of attributes. Again compared to Excel, each row in a sheet represents a requirement and each cell represents an attribute. However, in Excel, all rows have the same columns (all requirements have the same attributes), while ReqIF allows mixing SpecObjects of different SpecTypes. Also, not all attributes need to be shown in a Specification.

Figure 2.1 shows in the **Outline View** a flat list of all SpecObjects. For instance, the SpecObject `_DW2y...` is not referenced in the Specification at all. Selecting a SpecObject shows its SpecType (in the figure, it is “Information Type”) and all attributes (in the figure, “Description” and “ID”).

Attribute. (Excel-equivalent: Cell) An attribute holds the actual content of a SpecObject.

In Excel, a new attribute is simply created by putting a column header on a column. In ReqIF, columns are created via **ProR | Column Configuration**, or by clicking on . But content will only be shown if the SpecObject of that row has an attribute of that name.

Besides the actual text of the requirement, typical attributes include ID, status, etc. Note that there are no “standard” attributes. However, the ProSTEP Implementor Forum defined a recommendation for a set of standard attributes.

Chapter 3

Tutorial

3.1 Basic Concepts

In this tutorial, we will use the ReqIF terminology, which can be confusing. Therefore, please familiarize yourself with the terminology first (Section 2.4).

Warning. In ReqIF terminology, a requirement is called *SpecObject*, a link is a *SpecRelation*, and the document view consists of *SpecHierarchies*. Confused? Then please have a look at Section 2.4, where the terminology is described.

3.2 Tutorial 1: Creating a Basic ReqIF Model

In this section, we will build a ReqIF model from scratch, step by step.

3.2.1 Install ProR

The easiest way for installing ProR is downloading formalmind Studio. This is a standalone-application that is based on Eclipse ProR, combined with some enhancements.

Alternatively, you can install ProR in any Eclipse-Installation via its update site (listed on the RMF Download page). This is recommended for advanced users only who need to integrate RMF with other Eclipse-based components.

Information. The installation is described in detail in Section 4.1.2.

3.2.2 Create the Model

- If you do not already have one, create a new *project*: Select **File | New | Project**;
- Create the ReqIF model by selecting **File | New | Reqif10 Model**;

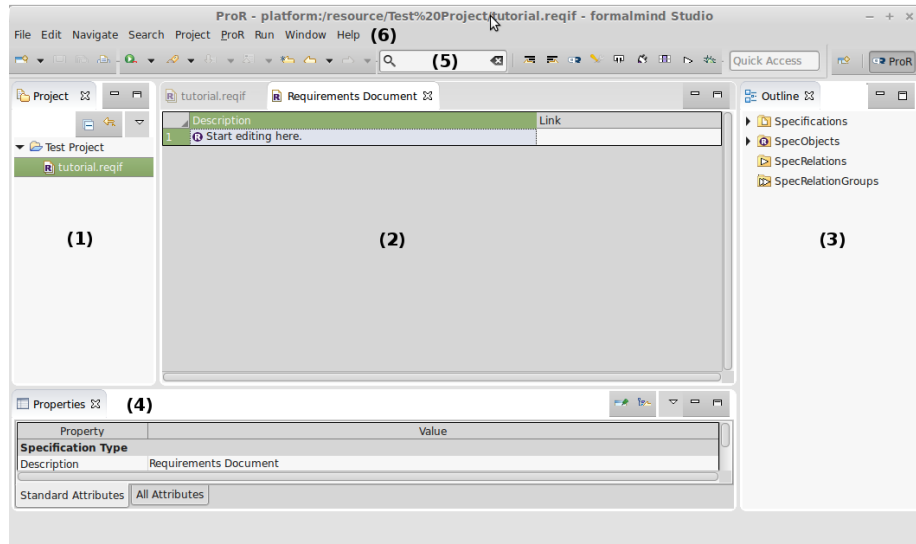


Figure 3.1: The ProR user interface

- Select the project and name the file “tutorial.reqif”. Click **Finish**;
- Upon completion, the model will be opened, as well as the one and only **Specification** contained in this model.

After this, your window should look more or less as shown in Figure 3.1.

You will see your ReqIF file in the **Project Explorer** window (1).

The **Specification Editor** (2) shows your *Specifications*.

In the Specification Editor, you see the *SpecObjects* that exist in this Specification. There is currently only one, with the description “Start editing here”.

The **Outline** (3) has four folders:

Specifications. Shows the Specifications in the ReqIF. You can expand the tree to expose the hierarchy of SpecObjects in the ReqIF model.

SpecObjects. Shows all SpecObjects in the ReqIF model as a flat list. Keep in mind that SpecObjects in Specifications are references. In contrast, this folder shows all SpecObjects created for the ReqIF model, whether or not they are referenced.

SpecRelations. Shows all SpecRelations in the ReqIF as a flat list. For now, we will ignore SpecRelations.

SpecRelationsGroups. These are special constructs that can be used for grouping SpecRelations with the same source and target.

The properties of a selected *SpecElement* are shown in the **Properties View** (4). As the only SpecObject in the model is selected, we see its *SpecType*

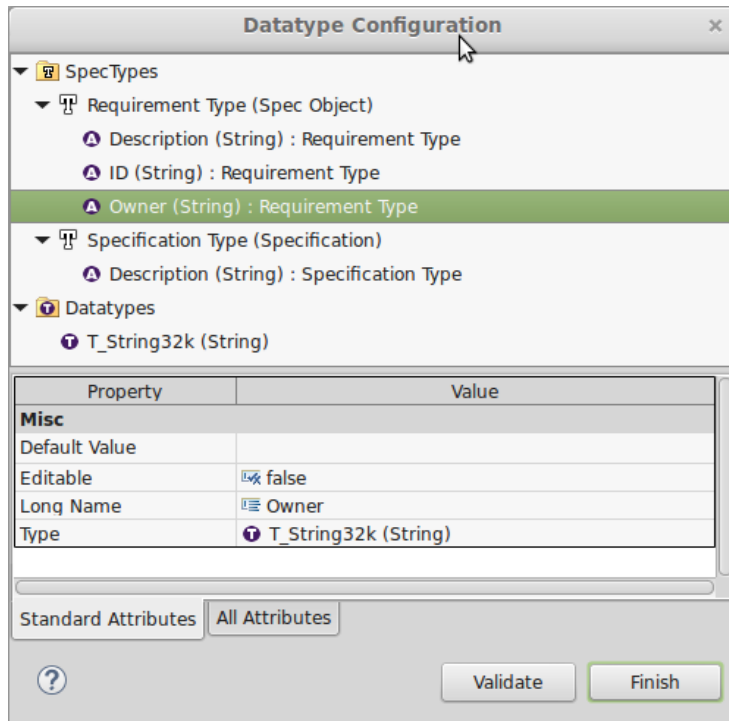


Figure 3.2: Datatype Configuration Dialog

(“Requirements Type”) and its only *Attribute* (“Description”) with the value “Start editing here.” There are two tabs **Standard Attributes** and **All Attributes** at the bottom of the **Properties View**. The **Standard Attributes** tab shows you all standard attributes of the selected element. **All Attributes** shows all existing ReqIF attributes of the selected element.

Above the main working windows is the tool bar (5) and, at the very top, the menu bar (6).

3.2.3 Customizing the SpecType

To get familiar with the tool, we will:

- Add two more attributes to the SpecObjectType called “ID” and “Owner” and
- We will show those *Attributes* in the *Specification*

The results of this are shown in Figure 3.2.

To add new attributes, we open the **Datatype Configuration** dialog with **ProR | Datatype Configuration**. Alternatively you can also click on **T** in the Tool Bar.

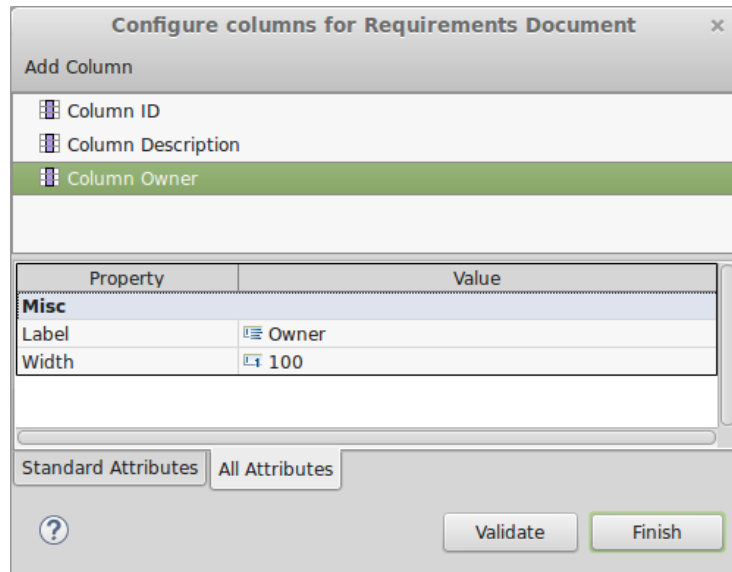


Figure 3.3: Column Configuration

The resulting dialog box has two folders in the upper pane: one for *SpecTypes* and one for *Datatypes*. Currently, there is only one Datatype “T_String32k” and two SpecTypes, one called “Requirements Type” with the attribute “Description” and one called “Specification Type” with the attribute “Description”.

In the lower pane are the details in regards to each attribute.

We add more Attributes to “Requirements Type” by right-clicking it and selecting **New Child | Attribute Definition String**. This will create a new element. Upon selecting it, we can rename it and tailor the details. Double-click on the “Long Name” variable and type in “ID”. Change the Type by double-clicking the field and choosing “T_String32k” from the dropdown menu. Repeat the process but this time change the “Long Name” to “Owner”. In the end, the dialog should look as shown in Figure 3.2.

Upon closing the dialog, little will have changed - the Specification still shows just two columns, Description and Link. However, if you select the requirement, you will see the new Properties (ID and Owner) in the Property view.

3.2.4 Showing the New Attributes

To show the new Attributes in the Specification, we have to configure the columns shown in the **Specification Editor**. We do this by selecting **ProR | Column Configuration**. You can also click on  in the Tool Bar.

The resulting Dialog shows one entry, “Description” for the one and only Column of the Specification. In the “Value” column double click on “Description” to choose it and replace it with “ID”.

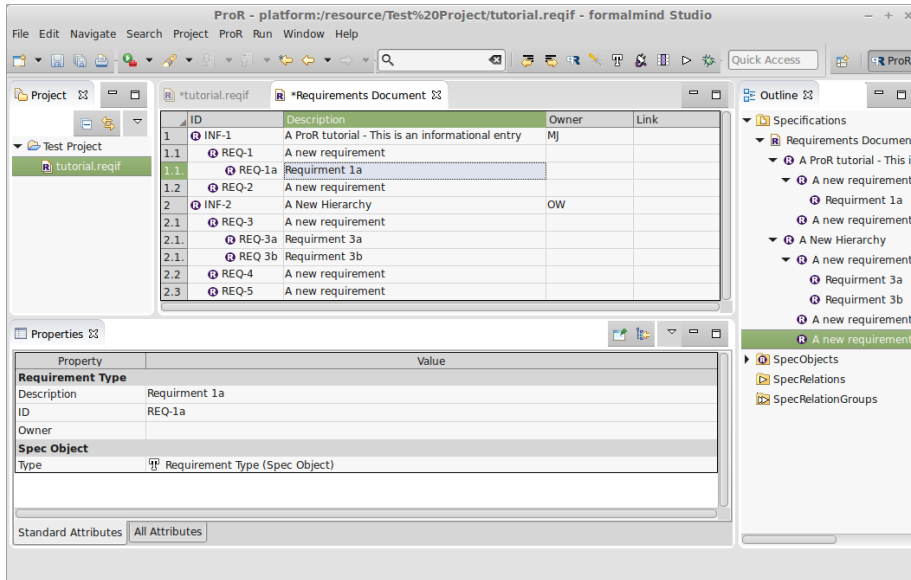


Figure 3.4: Adding SpecObjects

By clicking on the “Add Column” icon at the top of the dialog, create a new column and name it “Description”. In this view, the columns can be dragged and dropped to change their order as desired.

The resulting window is shown in Figure 3.3.

Information. Note that you have to provide free text for the columns for the same reason that we used free text for the “Labels” earlier: This way we can easily match multiple SpecObjects of different types.

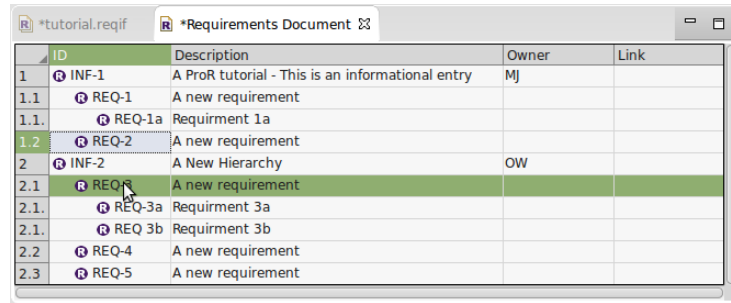
You can actually adjust the width of the columns simply by dragging the column headers.

3.2.5 Adding new SpecObjects

Now we can finally add *SpecObjects* by right-clicking on a row in the Specification View. In the context-menu, there are two submenus: **New Child** and **New Sibling**.

In both menus, there are three entries **Spec Hierarchy**, **SpecObject** and **SpecObject (Requirement Type)**. Some background is needed here:

We said before that Specifications contain references to SpecObjects. A **SpecHierarchy** is the wrapper that allows the hierarchical structure and that points to the referenced SpecObject. Usually, we don’t have to be concerned with them. Therefore the second option: If selected, a new SpecHierarchy is created and associated with a new SpecObject, which in turn is set immediately



ID	Description	Owner	Link
1	INF-1 A ProR tutorial - This is an informational entry	MJ	
1.1	REQ-1 A new requirement		
1.1.1	REQ-1a Requirment 1a		
1.2	REQ-2 A new requirement		
2	INF-2 A New Hierarchy	OW	
2.1	REQ-3 A new requirement		
2.1.1	REQ-3a Requirment 3a		
2.1.1	REQ-3b Requirment 3b		
2.2	REQ-4 A new requirement		
2.3	REQ-5 A new requirement		

Figure 3.5: Drag and Drop

to the given *SpecType*. If we had more than just one *SpecType* (besides “Requirements Type”), there would be an entry for each *SpecType* in the context menu.

To continue the exercise, select the **New Child | SpecObject (Requirement Type)**. Now we have two *SpecObjects*. The original is numbered on the far left-hand side of the pane with a “1”. The second one, the child, is numbered “1.1”. Now we should change the ID’s of each entry. Click in the cell of the column “ID” (in row 1) and type in INF-1. Under Description, type ”A ProR tutorial.” For the second, change the ID to REQ-1 and ”Learn how to create a new requirement” in the “Description” column.

Feel free to add a few more rows and or even new structures. Yours should look something similar to Figure 3.4.

3.2.6 Rearranging Elements

SpecObjects can be reordered by using drag and drop. This works both in the **Specification View** and the **Outline View**.

You can drag and drop a *SpecObject* as a sibling or a child. The highlighting feedback will enable you to see what you’re moving and where to. For instance, if you are dragging a *SpecObject* over another one, the entire cell will be highlighted. This means, that the *SpecObject* will be assigned as a child to the dropped *SpecObject*.

If you are dragging a *SpecObject* between two rows, you also get visual feedback on whether the *SpecObject* will be assigned as a sibling.

Figure 3.5 shows drag and drop in action.

You can also move elements by using cut and paste. Pasting will insert an element as a child.

3.2.7 Export Specification as HTML

If you want to export your Specification as HTML, open the Specification you would like to export and initiate printing **File | Print...**. The HTML representation is generated and opened in your system’s web browser.

3.2.8 Conclusion

Quite an achievement—but there’s still a bit of a way to go. One improvement we can make is simplifying data entry. Another is improving the visibility of the descriptions. In the next part of the tutorial, we will address these issues.

3.3 Tutorial 2: Use Presentations

We will continue where we left off at the end of Tutorial 1 and we will assume that you have ProR open with a model identical or similar to the one we created earlier.

In this tutorial we will introduce *Presentations*. Presentations are Eclipse Plug-Ins that extend the functionality of ProR. Specifically:

- Presentations can change the way Attributes are rendered in the Specification,
- Presentations can change the way Attributes are edited in the Specification and
- Presentations can perform tasks in the background.

ProR comes with a number of standard presentations that we will introduce in this tutorial.

3.3.1 ID Presentation

It would be nice if every SpecObject had its own unique ID. Actually, they do: The unique ID is shown in the **All Attributes** tab of the **Property View**, if a SpecObject is selected. But that ID is meant for machines and is not practical.

The ID Presentation allows the automatic creation of more user-friendly IDs. Let’s create one.

Remember that Presentations are associated with *Datatypes*, not *Attributes*. Thus, we first have to create a new Datatype called “T.ID“. We then associate that Datatype with the Attribute “ID“. We described this process in the first tutorial. Figure 3.6 shows the configuration dialog, when all is done.

The next step is the association of the Datatype with the Presentation.

We open the Presentation Configuration and create a new Presentation from the dropdown menu **Select Action...**, of type “Id” Presentation. We associate it with the newly created Datatype. After configuration, it would look as shown in Figure 3.7.

Note that you can adjust the prefix, count and the vertical alignment of the presentation.

Warning. At this point, the Presentation does not yet check for duplicates. It simply grabs a new value from count, increments it and uses it. Also, existing values are never overwritten.

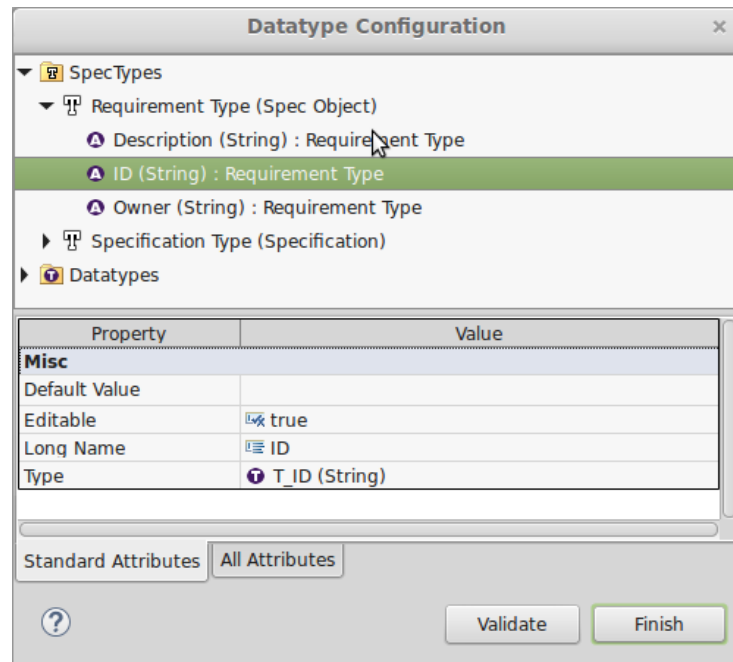


Figure 3.6: Datatype Configuration Dialog

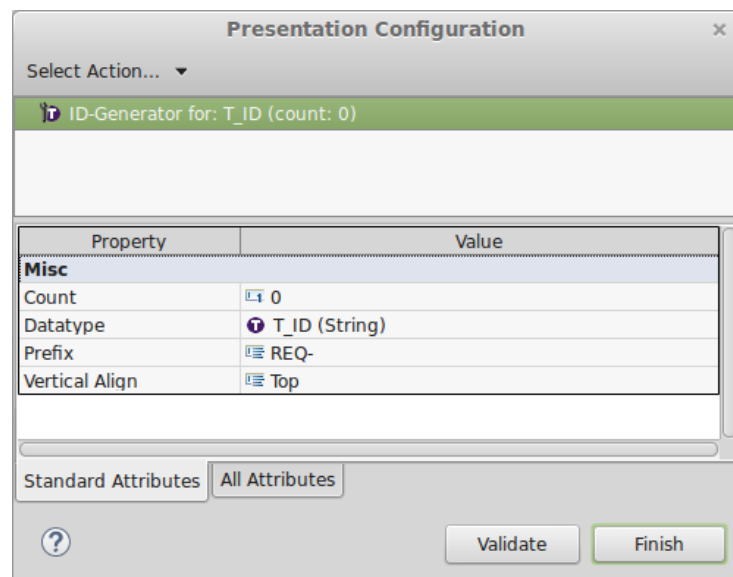


Figure 3.7: Configuration of the ID Presentation

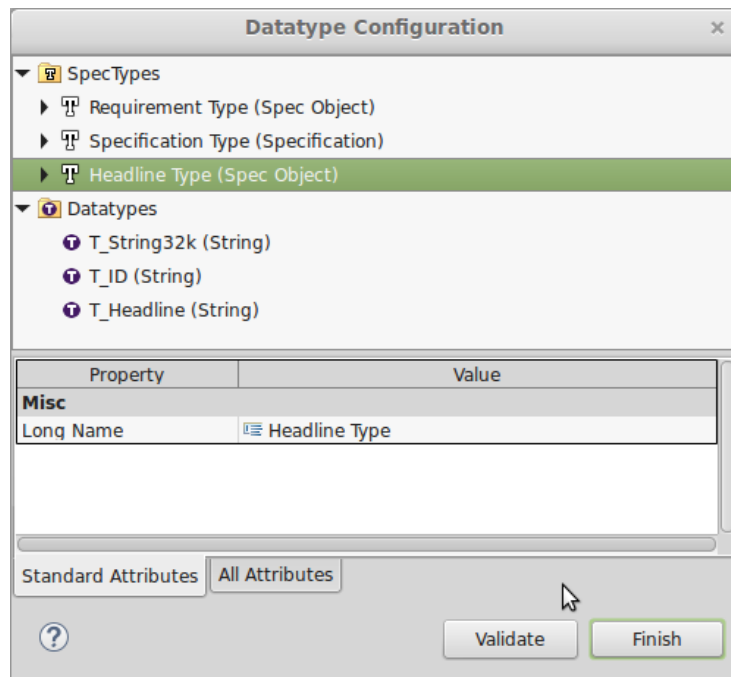


Figure 3.8: Datatype Configuration for the Headline Presentation

3.4 Tutorial 3: Advanced SpecTypes

So far, we have a model with only one `SpecObjectType`. In this tutorial, we will show how we can work with multiple `SpecTypes`, and we will introduce other `SpecTypes`.

3.4.1 Multiple SpecTypes

The first entry in our Specification (“A ProR Tutorial”) isn’t really a requirement. Thus, it doesn’t need an ID or an owner, and it would be nice to highlight it somehow. For highlighting, we have the **Headline Presentation**. We will:

- Create a new *SpecType* for headlines.
- Create a new *Datatype* that will be used for the headline content.
- Associate that *Datatype* with the **Headline Presentation**.

By selecting **ProR | Datatype Configuration**, we open the dialog where we can create new `SpecTypes` and `Datatypes`. For the first time, we create a new `SpecType` by right-clicking on **SpecTypes**. One of the entries in the child menu is **SpecObject Type**.

ID	Description	Owner	Link
1	A ProR Tutorial		
1.1	REQ-1 A new requirement		
1.1.1	REQ-3a Requirement 3a		0 ▷ ◀ ▷ 1
1.1.2	REQ-1a Requirement 1a		1 ▷ ◀ ▷ 0
1.2	REQ-5 A new requirement		
2	INF-2 A New Hierarchy	OW	
2.1	REQ-3 A new requirement		
2.1.1	REQ-3b Requirement 3b		
2.2	REQ-4 A new requirement		
2.2.1	REQ-2 A new requirement		
2.3	REQ-3a Requirement 3a		0 ▷ ◀ ▷ 1

Figure 3.9: Activated Headline Configuration

Once created, we select it and rename as “Headline Type” in the **Property Pane**.

Then we give it a new Attribute called “Description” by right-clicking it and selecting **String**.

Warning. It is important that we call it “Description”. This matches the Attribute name from “Requirement Type”. By using the same name, we ensure that the Attributes **appear in the same column**, even though they are different Attributes from different SpecTypes.

We do not set the type yet, as we need to create a new Datatype. We do this, as before, by right-clicking on the **Datatypes** folder in the upper pane. Create a child of type **Definition String**. Call it “T_Headline” and set the length to 50. Now we can go back to the “Description” Attribute (in the upper pane) and set the type to T_Headline, which is now available from the dropdown.

When all this is done, the resulting configuration should look as shown in Figure 3.8.

You can change the type of a SpecObject by selecting it and changing it in the **Properties View**. Please note that currently all existing values are lost when changing the type.

Note the following:

- For SpecObjects of this SpecType, the columns “ID” and “Owner” are now empty and cannot be edited.
- Note how the **Property View** changes as you select SpecObjects of different SpecTypes.
- Right-clicking on a row now shows one more option for child/sibling creation: A new entry of type “Headline Type”.
- At this point, the description of the headline is not yet formatted, as we have not activated the Presentation yet.

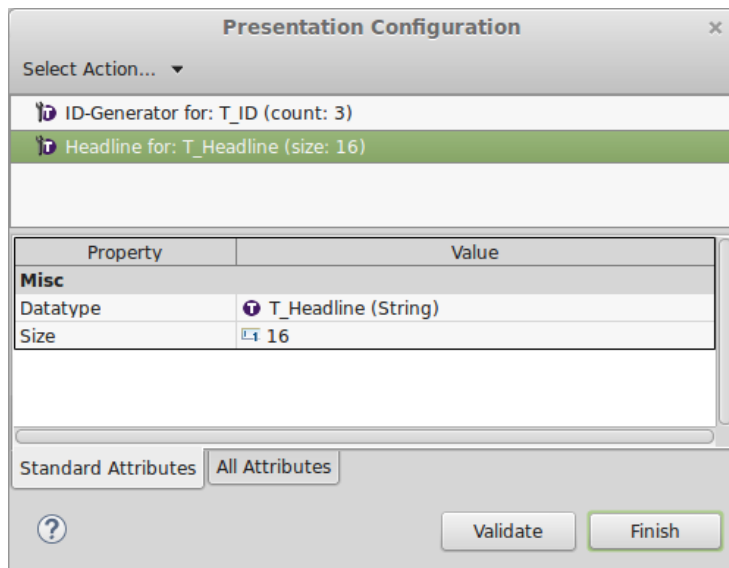


Figure 3.10: Presentation Configuration for Headline

Last, we will create a **Headline Presentation** and activate it for the type “T.Headline”. This is done via the **Presentation Configuration**, as before. The configuration allows you to change the font size of the headline in the “Size” property. The presentation configuration is shown in Figure 3.10.

After all the changes, the Specification should look as shown in Figure 3.9. Note that the formatting is also applied in the **Properties View**.

3.4.2 Other SpecTypes

You may have noticed in the **Datatype Configuration Dialog**, that right-clicking on **SpecTypes** offered more options, besides **SpecObject Type**. A number of Elements can have Attributes, not just SpecObjects.

We will now create a *SpecificationType* and assign it to our Specification.

Try to create a *SpecificationType* and configure it as shown in Figure 3.11:

Next, we will assign this type to the one Specification that we have. To do this we select the Specification in the **Outline View** (first element in the Folder **Specifications**). That will show the Specification’s properties in the **Properties View**. The **Type** property is empty. We select “Specification Type” from the dropdown.

As soon as it is selected, the Attribute “Description” will appear in the Properties View, as now the Specification has an attribute of that name. If we set it to a meaningful name, the label in the Outline View will change as well.

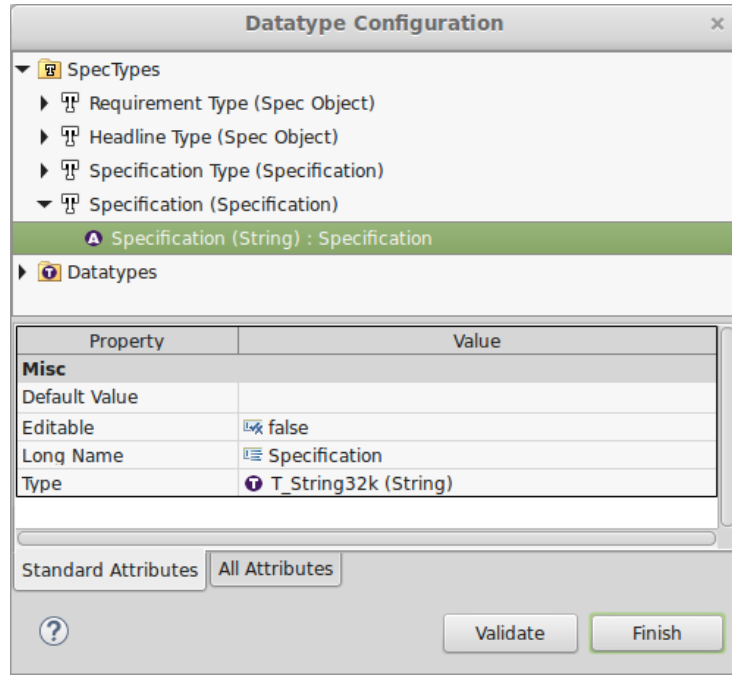


Figure 3.11: Creating a New SpecType

3.5 Tutorial 4: Links Between Elements

A central feature of professional requirements tools is the ability to connect requirements with each other. These are typically called links, traces or relations. In ReqIF, these are *SpecRelations*.

3.5.1 Creating SpecRelations

There are two ways of creating SpecRelations: (1) via the context menu and (2) via drag and drop.

Information. We recommend to use the context menu for linking. It works even if source and target are not visible, it allows n:m linking, and it allows for the assignment of a SpecType to the newly created SpecRelations in one step.

3.5.2 Linking Via Context Menu

Linking via context menu happens in two steps: initiating linking and completing linking.

ID	Description	Owner	Link
1	A ProR Tutorial		
1.1	REQ-1 A new requirement		
1.1.	REQ-1a Requirement 1a		1 ▷ 0
1.1.	REQ-3a Requirement 3a		0 ▷ 1
	▷		Requirement 1a
1.2	REQ-5 A new requirement		
2	INF-2 A New Hierarchy	OW	
2.1	REQ-3 A new requirement		
2.1	REQ-3b Requirement 3b		

Figure 3.12: Showing Links in the GUI

Linking is initiated by right-clicking on one SpecObject, or a selection of SpecObjects. The context menu contains the option **Initiate Linking**, with a number in parentheses. The number indicates the count of affected SpecObjects.

Linking can only be completed if it has been initiated. In that case, right-clicking on a single SpecObject or a selection of SpecObjects will show two more entries in the context menu. These are **Complete Linking To Selection** and **Complete Linking From Selection**. It will also indicate in parentheses with how many SpecObjects the linking had been initialized.

Each of these entries has a submenu, listing all SpecTypes for SpecRelations. By selecting the appropriate option, SpecRelations will be created between the two sets of SpecObjects. The SpecRelations will have the selected SpecType.

3.5.3 Linking Via Drag and Drop

SpecRelations are also be created by “Link-Dragging”. This is platform specific:

Linux. Dragging with Ctrl-Shift.

Mac. Hold down OPTION and APPLE/COMMAND keys while dragging.

Windows. Dragging with Alt.

3.5.4 SpecRelations in the User Interface

After creating SpecRelations, the right-most column (**Link**) in the **Specification Editor**, has content. It summarizes the number of incoming and outgoing SpecRelations. The number to the left of the triangle is incoming links, the other, outgoing links (see Figure 3.12).

Showing SpecRelations **for a single SpecObject** can be switched on or off by double-clicking on the Link cell.

Showing SpecRelations **for all SpecObjects** can be switched on or off with the little triangle icon in the toolbar ▷.

As SpecRelations can have Attributes, these are shown in the corresponding column, if the column name matches the Attribute name. Selecting the row of a SpecRelation will show all its Attributes in the **Properties View**.

The **Link** cell of a SpecRelation shows the label of the link target of that SpecRelation. Selecting it will show the link target's Attributes in the **Properties View** (instead of the SpecRelation's Attributes, as for all other cells).

Information. Selecting the **Link** column of a SpecRelation allows the link target to be quickly inspected.

Figure 3.12 shows a Specification where REQ-1a and REQ-3a are linked.

3.5.5 SpecRelations in the Outline View

The **Outline View** has a folder showing all SpecRelations.

Chapter 4

Reference

ProR is a desktop application that is based on Eclipse for systems engineering in general, and requirements engineering in particular.

4.1 Eclipse

ProR is an extension of the generic Eclipse Platform. The following is concerned with Eclipse in general.

Information. Please consult the Eclipse platform overview for further information.

4.1.1 Prerequisites

Eclipse is a Java-based application. You need a Java Runtime Environment (JRE) on your computer in order to run ProR.

ProR requires JRE 1.6 or better. However, some of the features from ProR Essentials require JRE 1.7 or better. Further, we recommend the version from Oracle, and not OpenJDK.

Information. You can download Java at java.com.

4.1.2 Installation

This chapter explores the installation of *Eclipse Products*, i.e. software that you can download and run on your computer. This is in contrast to *features* or *plugins*, which can be added to an existing product.

When working with Eclipse, you have to start with a base installation. For working with ProR, we recommend using formalmind Studio, but you can start with any Eclipse product.

Once you have identified the product you would like to use, you need to download it, which is typically a .zip file. Create a folder and extract the content of the .zip file into that folder.

Information. We recommend to call the folder `studio` or `pror`, and to store it where your executables are located: On Windows in `Program Files`, on Linux in `/bin`. But any location will do.

We recommend to creating a shortcut for launching it.

You launch the product by double-clicking on the launcher in the folder you created. For formalmind Studio, this is called `studio.exe` or `studio`.

The first time you launch Eclipse, it will ask you for the *Workspace* location, see Section 4.1.5.

4.1.3 Updates

The Eclipse Update Manager regularly checks for new versions and alerts the user if one is found. It can also be started manually via `Help | Check for Updates`.

4.1.4 Installing New Features

Before you start installing new features, you typically need to connect with the update site that is hosting the feature to be installed.

Update Site. An update site is a machine-readable URL that allows ProR to install new functionality. Note that visiting the update site with a web browser rarely produces anything meaningful.

To install a new feature, follow these steps:

- Open the installation dialog via `Help | Install new Software...`.
- In the `Work with:` dropdown, either paste the Update Site URL, or select it from the drop down, if you have used it before. Note that some popular update site URLs may already be preinstalled.
- Upon selecting an update site, you will see a list of components available from that update site. Note the checkboxes below that may result in some entries being hidden. In particular, some update sites do not categorize. Unchecking `Group items by category` may unveil hidden entries.
- Click `Next >`. If all dependencies can be resolved, details about the installation are shown. Otherwise you have to troubleshoot dependencies (an unthankful job!).
- Click `Next >`, review and accept the license.
- Click `Finish`. If the component has not been digitally signed, you will receive a warning, which you can typically ignore.
- It is recommended to restart after the installation.

Information. Signatures on Content. In this day and age, security is obviously very important, particularly for content downloaded from the Internet. But note that signing is not enough: content must be signed with a trusted, non-expired signature.

Eclipse content should be signed by eclipse.org. Especially small project release plug-ins that are not signed, reasoning that to a user, a self-signed signature is as good (or bad) as a missing signature.

Ultimately, you need to decide for yourself whether you are willing to run unsigned and/or untrusted content.

4.1.5 Workspaces

The workspace is a folder on your computer, where all information related to ProR is stored. This includes all your ReqIF files, organized into projects, as well as various meta data.

Information. Read more about the The Workbench in the Eclipse documentation.

Also, it is possible to have more than one workspace, and to switch between them. This feature can be useful for advanced users.

4.1.6 Committer License Agreement (CLA)

The Committer License Agreement (CLA) needs to be signed by contributors to Eclipse projects. It essentially states that you hold all rights to your contribution and that you allow Eclipse to use them under the Eclipse Public License.

4.2 ProR User Interface

Figure 4.1 shows the user interface of ProR.

(1) The **Project Explorer** shows a hierarchical listing of the project and the associated models.

(2) The editor area shows two kinds of editors. First, each ReqIF file has a **ReqIF Editor** that shows basic information about the mode. In addition, **Specification Editors** can be opened for each Specification.

(3) The **Outline View** has four folders that show the content of the selected model:

Specifications shows the Specifications in the ReqIF model. You can expand the tree to expose the hierarchy of SpecObjects in each Specification.

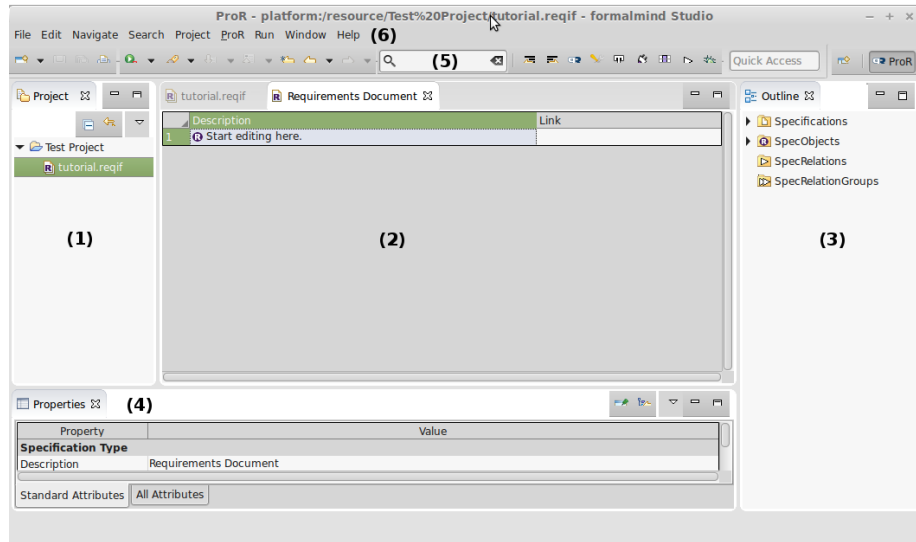


Figure 4.1: The ProR user interface

SpecObjects shows all SpecObjects in the ReqIF model as a flat list. Keep in mind that SpecObjects in Specifications are references. In contrast, this folder shows all SpecObjects created for the ReqIF model, whether or not they are referenced.

SpecRelations shows all SpecRelations in the ReqIF as a flat list.

SpecRelationsGroups represents an optional mechanism for grouping SpecRelations between two specific specifications.

(4) The properties of a selected Element are shown in the **Properties View**. It has two tabs, one for **Standard Attributes** and one for **All Attributes**.

(5) Above the main working windows is the tool bar, which may change according to which editor is active.

(6) The menu bar provides access to all Eclipse and ProR features.

4.2.1 Editors and Views

The Eclipse user interface consists of *Views* and *Editors*. Views change their content according to the current selection and are not savable. Editors are typically associated with a resource (e.g. a file) and can be saved. The editors' selection can determine what is shown in the Views. For instance, the **Properties View** typically shows details about the element selected in the **Specification Editor**.

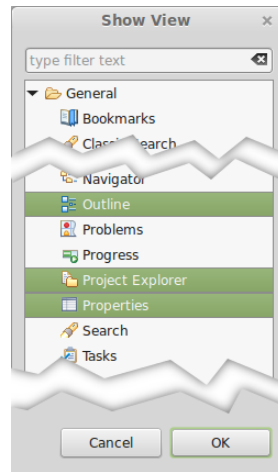


Figure 4.2: Views.

You can browse through the available Views and open them via **Window | Show Views...**, resulting in a menu similar to the one shown in Figure 4.2.

Upon opening a ReqIF model, the editor opens providing an overview of the model. In essence what you are seeing is the Eclipse Workbench, with several modifications. Here you will find a quick overview of each component. A more detailed description of the Workbench can be found in Eclipse's Workbench User Guide.

Information. There may be more than one editor for a given file type. To pick a specific one, right-click the file and select **Open With...**, which will give you the list of installed editors. In particular, the **fmStudio Main Editor** from Formal Mind is more powerful than the stock editor (see Section ??).

A model contains any number of Specifications, and the details of each Specification can be inspected individually. The windows in which all relevant information appears are called *views*. At your disposal are many views with productivity, debugging, help and team resources. We will be focusing only on the views relevant to ProR.

4.2.2 RMF Specification Editor

The Specification Editor shows the SpecObjects of a Specification, arranged in a grid view. The hierarchy is shown via the indent of the first column, as well as through the numbering in the row header.

The columns show the Attributes with the names that match the column names (as shown in the header). The columns can be resized by dragging. Other operations, in particular reordering, adding and deleting columns are done via

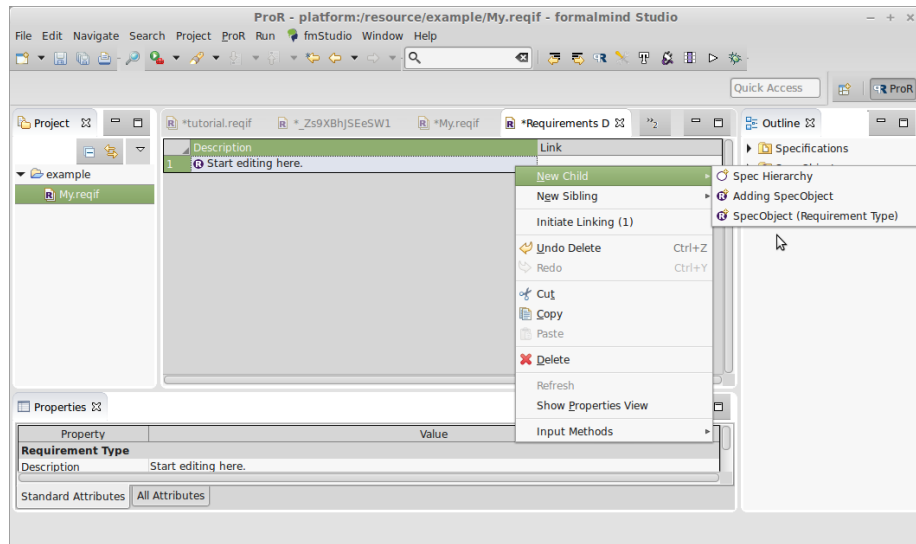


Figure 4.3: Specification Editor with the context menu open for creating a new child element

the **Column Dialog**, accessible via **ProR | Column Configuration** or the toolbar .

The leftmost column shows the hierarchy and carries an icon. The icon indicates whether it is a lone SpecHierarchy  or a SpecObject .

Information. Would you like to rearrange the columns?

In the top half of the **Column Configurator**  window, a list of the exiting columns appear. Simply drag and drop them into the desired order. The changes appear in real time in the **Specification Editor**. Close the window and the changes will be accepted.

Information can be entered either directly into the cell by double-clicking it or via the **Properties View**. While the Specification Editor only allows the editing of those attributes for which a column exists, the **Properties View** will always show all attributes.

The SpecObjects can be reordered via drag and drop. To move an existing SpecObject into the position of parent or child of another existing SpecObject, simply drag the child directly *onto the target SpecObject*, as shown in Figure 4.4. The result is shown in Figure 4.5.

Alternatively, as you drag the SpecObject *onto the line below or above* the level you would like to move it to, it will become a sibling rather than a child of the SpecObject. This is shown in Figure 4.6, with the resulting ordering shown in Figure 4.7.

To duplicate a SpecObject, simply copy and paste it into the required position. The copying, cutting and pasting functions are accessible through the

ID	Description	Owner	Link
1	A ProR Tutorial		
1.1	REQ-1 A new requirement		
1.1.1	REQ-3a Requirment 3a		0 > 1
1.1.2	REQ-1a Requirment 1a		1 > 0
1.2	REQ-5 A new requirement		
2	INF-2 A New Hierarchy	OW	
2.1	REQ-3 A new requirement		
2.1.1	REQ-3b Requirment 3b		
2.2	REQ-4 A new requirement		
2.2.1	REQ-2 A new requirement		
2.3	REQ-3a Requirment 3a		0 > 1

Figure 4.4: Click on a requirement and drag onto target parent.

ID	Description	Owner	Link
1	A ProR Tutorial		
1.1	REQ-1 A new requirement		
1.1.1	REQ-3a Requirment 3a		0 > 1
1.1.2	REQ-1a Requirment 1a		1 > 0
1.2	REQ-5 A new requirement		
2	INF-2 A New Hierarchy	OW	
2.1	REQ-3 A new requirement		
2.1.1	REQ-3b Requirment 3b		
2.1.2	REQ-2 A new requirement		
2.2	REQ-4 A new requirement		
2.3	REQ-3a Requirment 3a		0 > 1

Figure 4.5: Result: The requirement is now a child of the chosen parent.

ID	Description	Owner	Link
1	A ProR Tutorial		
1.1	REQ-1 A new requirement		
1.1.1	REQ-3a Requirment 3a		0 > 1
1.1.2	REQ-1a Requirment 1a		1 > 0
1.2	REQ-5 A new requirement		
2	INF-2 A New Hierarchy	OW	
2.1	REQ-3 A new requirement		
2.1.1	REQ-3b Requirment 3b		
2.1.2	REQ-2 A new requirement		
2.2	REQ-4 A new requirement		
2.3	REQ-3a Requirment 3a		0 > 1

Figure 4.6: Click on a requirement and drag onto the line (bolded) below or above target sibling.

ID	Description	Owner	Link
1	A ProR Tutorial		
1.1	REQ-1 A new requirement		
1.1.1	REQ-3a Requirment 3a		0 > 1
1.2	REQ-5 A new requirement		
2	REQ-1a Requirment 1a		1 > 0
3	INF-2 A New Hierarchy	OW	
3.1	REQ-3 A new requirement		
3.1.1	REQ-3b Requirment 3b		
3.1.2	REQ-2 A new requirement		
3.2	REQ-4 A new requirement		
3.3	REQ-3a Requirment 3a		0 > 1

Figure 4.7: Result: The requirement is now a sibling of the chosen requirement.

traditional dropdown menu, by right-clicking on a desired cell or the corresponding keyboard shortcut.

By right clicking on any cell, you have a few options at your disposal. Outside of the usual **Undo**, **Cut**, **Copy**, **Paste** and **Delete**, commands, the following are also available:

New Child. A new SpecObject will be created as a child element.

New Sibling. A new SpecObject will be created as a sibling element.

Initiate Linking. This is the option to create a link between requirements.

Once a link is initiated and then by right clicking a target selection, the options to complete the links either to or from a selection will appear. By default, the links are illustrated in the **Link** column to the right.

Show Properties View. Opens the **Properties View**, where the selected element can be inspected and edited.

4.2.3 Project Explorer View

The **Project Explorer View** is by default on the left side of the main window. Here you can inspect files and models associated with any project. If for some reason the Project Explorer View does not appear, Navigate to **Window | Show View | Other | Project Explorer View**.

In the main area of this viewer is a hierarchical listing of the project and it's components. Use the black arrow to the left to collapse or display the project's contents. Below the view's title and to the right are the options to collapse the project folder and link the project with the editor. To the right of these options is another dropdown menu.

This view is covered in more detail by the Eclipse documentation.

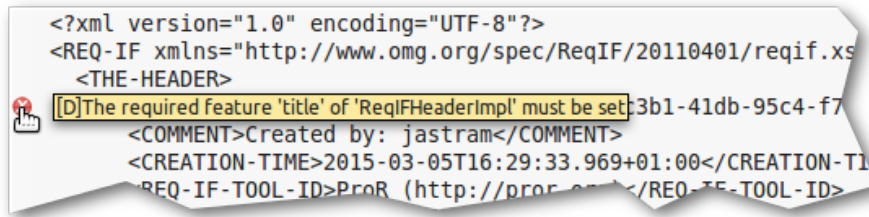


Figure 4.8: Error Markers generated by Consequent

4.2.4 ReqIF Validation

ReqIF is spreading, and partners start handing .reqif and .reqifz files back and forth. But chances are that the first import of such a file will fail. What is the problem? Consequent, the free ReqIF-Validator may provide the answer.

The validation results are shown in the Eclipse Problem View. In addition, it is possible to open the ReqIF file in question in a text editor and to generate error markers, as shown in Figure 4.8.

Information. In addition to the validator that is built into the tool, there is also a command line version available, which can be downloaded for free from <http://formalmind.com/consequent>.

You can validate a ReqIF file as follows:

- Right-Click the file to be validated in the Project Explorer (no need to open the file)
- Select **Validate | Consequent ReqIF Validation**
- A dialog indicates the completion of the validation, which is shown in the Problem View.
- If the Problem View is not visible, open it by hand via **Window | Show View | Other... | General | Problems**
- To see the error in the file, you must open it in a text editor as follows:
 - Right-Click the file in the Project Explorer
 - Select Open With — Text Editor
 - Errors are shown with an icon in the left margin (hovering over the error reveals the problem) – see screenshot below
 - For errors with line numbers, it is possible to double click on an error in the Problem View to jump to the corresponding position in the Text Editor

- Error markers do not reset automatically. Re-run the validation to update them. You can manually clear error marks (e.g. for files that do not exist any more) by right-clicking on an element in the Problem View and selecting Delete.

Validation Rules

You can access and individually disable the validation rules via **Window | Preferences | Model Validation | Constraints**.

Information. A printable version of the validation rules can be retrieved from the build server at <http://hudson.eclipse.org/rmf/job/rmf.develop.mars/ws/org.eclipse.rmf.reqif10.constraints/plugin.xml>

Consequent Acknowledgement

Consequent is part of the open source Eclipse project and was financed by the ProStep ReqIF Implementor Forum.

4.3 Configuration and Preferences

Both the current model, as well as ProR as a whole, can be configured and customized extensively.

4.3.1 Global Preferences

The application-wide settings of ProR are accessed via **Window | Preferences | ProR**. Configuration elements are:

Pror. In the top level menu, the warning message for encountering simplified XHTML can be disabled.

Default Presentations. ProR has basic cell editors for each ReqIF *Datatypes*. But it is possible to install new editors with better or different capabilities. With this setting, Presentations can be selected to handle certain Datatypes by default.

Information. Particularly popular is the free Presentation from formalmind Essentials for handling XHTML. The standard editor from ProR converts rich text to plain text. The rich text Presentation is preinstalled with formalmind Studio.

4.3.2 General Configuration

This configuration is accessed either via **ProR | General Configuration**, or via the  button on the toolbar.

Currently, there is only one configuration element: **Label Configuration**.

Label Configuration

The **Label Configuration** is used to determine what to use for the text labels of elements in places, where a shorthand is needed. Examples are elements in the **Outline View** or the link targets in the **Link** column.

ProR will determine the label by looking at the label configuration, which is a list of Strings. It will go through the list, top to bottom. If the element has an attribute with a matching name, that attribute value is used as the label.

If none is found, then the element's internal ID is displayed.

To configure, select **Label Configuration** in the top pane of the dialog. On the bottom pane, you see the **Default Label** property. Doubleclick the value (on the right), then click on the ellipses (...) to open the configuration dialog. Under **Feature**, you will see the list of attribute names that will be used for the label, if found.

Use the **Add** and **Remove** buttons to add more attribute names to be searched for. The search order can be adjusted with **Up** and **Down**.

Information. It is good practice to use the ID Presentation (5.3.1) to generate user-friendly IDs, and to use these as the first match for a label. As IDs are unique, you'll always have a unique identifier that is typically also used for communication.

4.3.3 Datatype Configuration

This configuration is opened via **ProR | Datatype Configuration...**

The dialog shows two folders, one for SpecTypes and one for Datatypes. SpecTypes are created for typing elements that have attributes (SpecObjects, Specifications, SpecRelations). New SpecTypes can be created by right-clicking on the folder and selecting **New Child**. Through the same mechanism, attribute definitions can be added to a SpecType. Attribute definitions are typed. Selecting an element shows its properties in the lower pane, where it can be configured.

Attributes must have a name and a Datatype. Some Attributes allow further customization. The Datatype is selected from a dropdown. New Datatypes can be created by right-clicking on the folder **Datatypes** and selecting **New Child**. Again, selecting a Datatype shows its properties in the lower pane, where it can be configured. A Datatype should have at least the **long name** property set.

As an example, consider the Datatype Configuration shown in Figure 4.9. The SpecType for "Requirements Type," which is applicable to SpecObjects, is expanded. The SpecType has two Attributes, "Description" (String) and "Status" (Enumeration). Status is selected, and in the pane below the mandatory values, **Long Name** and **Type** have been set. Further customization of the attribute is possible, e.g. by converting it to a multi-valued Attribute by setting the corresponding flag to **true**.

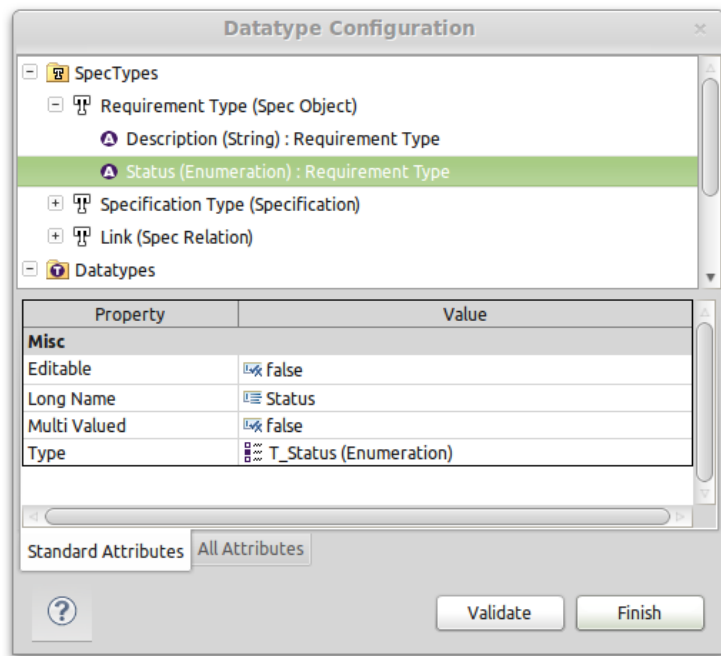


Figure 4.9: Datatype Configuration Dialog

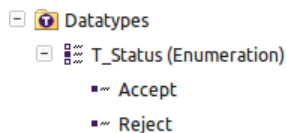


Figure 4.10: Enumerations

Enumeration Datatypes

An Enumeration Datatype must have enumeration values. These are created by right-clicking the Enumeration Datatype and selecting **New Child | Enum Value**. You may have to unfold the enum value to select it, so that you can provide it with a **Long Name**. Figure 4.10 shows a correctly configured Enumeration Datatype.

4.3.4 Presentation Configuration

Presentations are software components that extend the functionality of ProR. Chapter 5 is dedicated to Presentations.

4.3.5 Column Configuration

This configuration is specific to the Specification Editor.

The Column Configuration Dialog configures the Columns of a Specification. Columns are identified by name. The width of the column can be adjusted directly by dragging the column separator in the table header.

If the SpecObject has an attribute where the name of the attribute matches the name of the column, then that attribute is shown in that column.

4.4 Access Control

The ReqIF standard provides a flag for marking certain elements as read-only. This flag is accessible through the **Properties View** by selecting the tab **All Attributes**. However, this flag is not honored in the user interface: even if an element is marked as read-only, it is normally writable. We may implement this in the future.

Warning. Access Control is a ReqIF feature designed for data exchange. As ProR uses ReqIF as a data model (and not as an exchange format), access information is only stored, but not used for managing access.

4.5 Import and Export

Importers and Exporters are accessed via **File | Import...** and **File | Export...**. The corresponding dialogs will show generic Importers and Exporters, as well as specific ones. The specific ones are in a folder called **ProR (ReqIF)**.

This section also lists Importers and Exporters from third parties. Note that not all third-party Importers and Exporters may be listed here.

4.5.1 Import

The following importers exist:

ReqIFz Import. This standard importer takes ReqIF archive (.reqifz) and imports it as an Eclipse project.

CSV. This standard importer takes comma-separated data and imports it into an existing ReqIF model. It is described further below.

Axiom. This commercial importer supports the selective merging of exchange data with an existing ReqIF model. It is described in detail in Section ??.

More information at the Formal Mind Product Site.

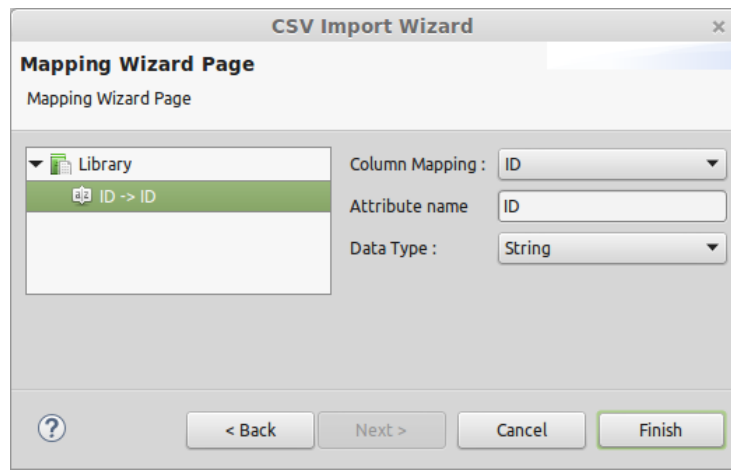


Figure 4.11: A complete mapping for a CSV import

CSV Import

The CSV import adds the content from a comma-separated file into an existing specification. Therefore, you first need to create a ReqIF model. Once this is done, the importer is used as follows:

Warning. The importer will create new SpecTypes, Attributes and Datatypes. Your existing types will not be reused! We recommend to clean up the datatypes after the import.

- Initiate the importer via **File | Import... | ReqIF | CSV**.
- On the first dialog page, you need to select the source file containing the CSV data. The dialog defaults to comma-separated data, but other separators can be chosen as well.
- A checkbox allows to indicated whether the file contains a header row. If it is checked, the first row will be interpreted as the labels for the columns in the file.
- Last, the destination model has to be chosen. After this, the **next** button is enabled.
- On the next page, the columns have to be mapped. Initially, there is just a single element, **Library**.
- Right-click **Library** to open the context menu that allows you to add a new mapping. There are four options. They all do the same, but the first three prepopulate the attribute name with a sensible value.

- You need to create a mapping for each column that you would like to import. A mapping consists of three elements:

Column Mapping. Select the column from the CSV file. The column headings are shown if the checkbox was selected on the previous page, otherwise just the column numbers.

Attribute Name. The name for the Attribute in the resulting ReqIF model.

Data Type. The datatype of the resulting attribute. The safest option is **String**, for other datatypes, a conversation attempt is made.

- You do not need to create a mapping for all columns. Figure 4.11 shows the importer dialog with one complete mapping.
- Once all mappings are complete, hitting finish will complete the import.

You can read more in the Formal Mind Blog.

4.5.2 Export

The following exporters exist:

ReqIFz Export. This standard exporter takes an Eclipse project and produces a ReqIF archive (.reqifz).

Axiom. This commercial exporter supports the selective exporting of exchange data for supplier communication. More information at the Formal Mind Product Site.

HTML. The HTML export is not a “real” export, as it is accessed differently. It produces an HTML view from an open Specification. To use it, you need to have a Specification Editor open. Then select **File | Print...**

4.6 Searching and Filtering

ProR has three search interfaces. Each has a different focus:

Quicksearch (Section 4.6.1). This interface allows search-as-you-type in the open editor. It is useful for quickly finding the right row in a specification, but just performs a simple text search on all attributes.

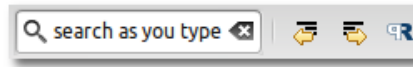
ReqIF Search (Section 4.6.2). This interface allow the user-friendly construction of attribute-specific searches within the current model.

Raw Search (Section 4.6.3). This interface is powerful, but requires the queries to be constructed by hand. It allows to fine-tune the search scope, including the search of the whole workspace.

Except the quicksearch, the results are shown in the Eclipse Search Result View.

4.6.1 Quicksearch

This feature allows you to search-as-you-type within the open Specification. The search box is embedded in the toolbar and only visible if a Specification Editor is open.



You can just begin typing in the box. With every keystroke, the view will update and collapse those rows that do not match. All attributes are searched.

4.6.2 ReqIF Search

Searching is initiated via **Search | Search... | ReqIF Search**, or via the search icon  on the toolbar. This will present you the wizard shown in Figure 4.12.

Information. The search dialog shows several tabs on the top. This handbook will cover ReqIF Search and ReqIF Search (Raw)—our ReqIF search tools.

Depending on your Eclipse configuration, other search option may be shown and may still be useful. For instance, the File Search is useful for searching within all files, including .reqif files.

Search Criteria

The ReqIF Search conditions consist of *search criteria*, that are added by clicking on **Add new**. This creates a new criteria panel. An arbitrary number of criteria can be defined. Criteria can be removed by clicking on **×** on the left side of the panel, or by clicking **clear** to remove all.

The radio button on top allows to either **Match all** or **match any** of the criteria.

Each criteria consists of three parts, *attribute*, *operator* and *value*. One such criteria is shown in Figure 4.12, with the attribute dropdown open.

Attribute. Choose an attribute (datatype) from the list. The first four (IDENTIFIER, DESC, LONG_NAME, and LAST_CHANGE) are ReqIF-internal attributes and are rarely of interest to normal users (see Section 2.3.3). The others attributes are taken directly from the document. The more attributes you create, the longer the list will grow. After you have chosen an attribute from the list, the rest of your choices (which are determined by the datatype of the attribute) are displayed.

Operator. The operator dropdown is specific to the selected attribute and is described for each type below.

Value. The value or values are specific to the operator and are also described below.

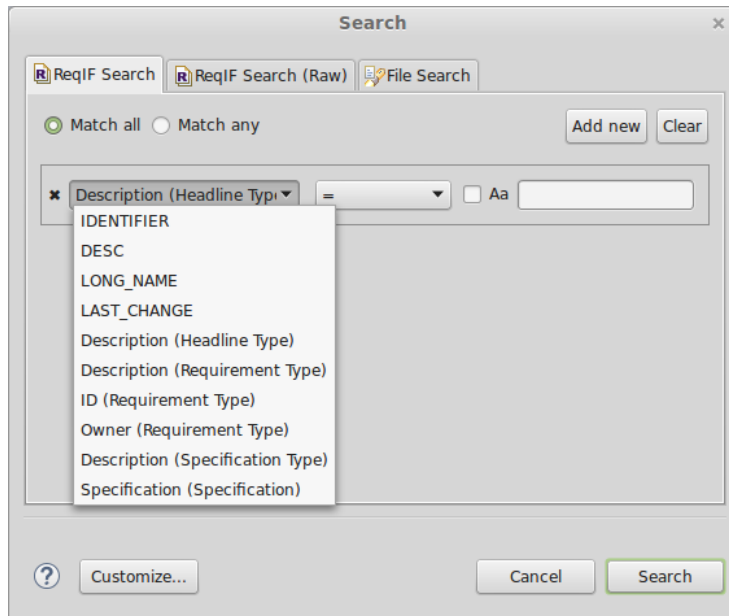


Figure 4.12: ReqIF Search function with attribute dropdown menu

Operators for All Types

The following operators are available for all attributes, no matter what their type:

Equals (=). For a match, the value must match *exactly* as provided. This implies that the attribute has actually been *set*. For instance, an empty string matches an empty string. But it does not match if the value has not been set (i.e. if it is *null*, in technical terms).

Set. For a match, the value must be set to any value. **Note:** A *default value* does not represent a match.

Not set. For a match, the value must not be set. **Note:** If the attribute in question does not exist, then there will not be a match either. See example below.

Example

Not Set Operator. Assume you receive a ReqIF model for review, with two SpecTypes, one called *Information Type* which contains an attribute *Description* and one called *Requirement Type* which contains two attributes, *Description* and *Status*. You are supposed to fill out the status for all requirements.

To find those that are still missing, you search for **Status not set**. This will deliver all requirements for which no status has been set, even if there is a default value.

Operators for Plain and Rich Text

All text searches can be made case-sensitive by checking the corresponding checkbox **Aa**.

The following operators are available for **plain text** attributes:

Not equals ($\neq$). For a match, the value must not *exactly* as provided. If the attribute is not set, this constitutes a match.

The following operators are available for **plain text and rich text** (XHTML) attributes.

Contains. For a match, the value must be a substring of the attribute.

Contains not. For a match, the value must not be a substring of the attribute. If the attribute is not set, this constitutes a match.

Regex. The provided value will be interpreted as a regular expression. Search will take place across line breaks.

Information. When searching rich text (XHTML), the invisible formatting will be included in the search, except for the *regex (plain)* operator described below.

Search will take place across line breaks. But this is only relevant for Regex search, where linebreaks can be matched explicitly (**\n**) or as part of whitespace (**\s**).

The following operators are available for **rich text** (XHTML) attributes.

Regex (plain). The provided value will be interpreted as a regular expression. Search will take place against a version of the attribute value where the tags have been stripped out and been replaced by whitespace.

Example

Searching XHTML. As XHTML contains invisible formatting tags, this should be taken into account when searching. For instance, the search **contains formalmind.com** will find direct textual references to the domain, as well as hyperlinks. e.g. `<a href="http://formalmind.com">click</a>`.

Operators for Numbers (Integer and Real)

The interfaces for integer attributes and real attributes are identical, but the text boxes will only accept numbers of the appropriate type.

Not equal ($\neq$). The provided value is not equal to the given number. This operator matches if the attribute has no value set.

Between. A match exists if the attribute value is between the given numbers, **including them**.

Greater than ($>$). A match exists if the attribute value is greater than the value, **excluding it**.

Less than ($<$). A match exists if the attribute value is less than the value, **excluding it**.

Operators for Dates

The granularity of the date criteria are one day, while ProR date stamps also include the time and have timezones.

Warning. Timezones. Dates in ProR have timezones. The dates entered in the search interface assume the local time zone. This can have implications if the values in the ReqIF model have been created by a user in a different time zone. For example, consider a date has been written on “Tuesday 23:00” in the time zone of User 1. But for User 2, it is already “Wednesday 01:00”. If User 1 would search for “Tuesday”, there would be a match. But for User 2, in a different time zone, not.

Equal ($=$). The day matches, no matter what the time (but timezones apply).

Not equal ($\neq$). Any day except the given day, or not set.

Between. A match exists if the attribute value is between the given numbers, **including them**.

Before. A match exists if the attribute value is before the date, **excluding it** (i.e. before the time 00:00:00 on that date).

After. A match exists if the attribute value is after the date, **including it** (i.e. after the time 00:00:00 on that date).

Operators for Enumeration

While ReqIF enumerations may be single value or multi value, this distinction is immaterial for the search functionality.

Not equal ($\neq$). Anything except an identical list will match. **Note:** An empty list matches an unset attribute value.

All. All selected value must also be selected on the attribute (but the attribute may have more).

Any. For a match, at least one of the list values must also be set on the attribute.

Operators for Boolean

Only the standard operators (equals, set, not set) are available.

4.6.3 Raw ReqIF Search

The raw search feature has been described in the Formal Mind Blog.

Chapter 5

Presentations

ProR has an extension mechanism that allows custom rendering, custom editing and automatic modifications to the requirements model. These extensions are called *presentations*.

Presentations can be created and configured via **ProR | Presentation Configuration...** or the icon  in the toolbar. Launching it will show the dialog shown in Figure 5.1. Initially, the dialog will be empty. The dialog in the figure shows two presentation configurations.

Presentations are specific for a model, and its configuration is also stored in the requirements model.

ProR ships with a small set of standard presentations. Additional presentations can be installed into ProR.

Warning. Note that presentations are specific to ProR, not to ReqIF. This means that other tools will simply ignore presentations. Also note that the behavior of the tool is unpredictable if a presentation used by a model is not installed in ProR.

5.1 Working with Presentations

All presentations are configured in the same way. To configure a presentation for the current model, first open the presentation configuration dialog as described above.

Initially, the dialog is empty. You can add new presentations using the **Select Action...** dropdown. It will list all built-in and installed presentations. Upon selecting an entry, it will be added to the list below.

Information. It is possible (and often useful) to add the same presentation multiple times. For instance, one could add multiple ID generation presentations for different types of elements (with correspondingly different prefixes).

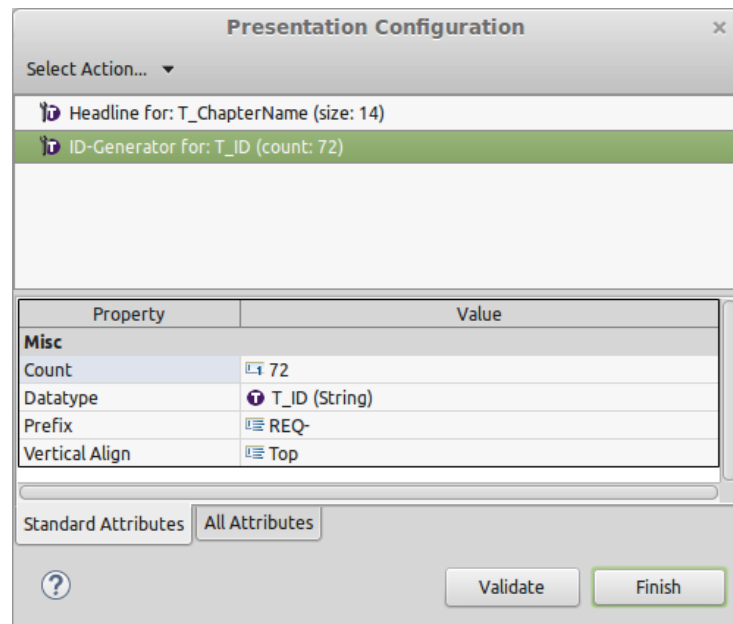


Figure 5.1: The presentation dialog with two configured presentations

Selecting a presentation in the upper pane will show its configuration parameters in the lower pane. In Figure 5.1, the ID-Generator is selected, and the lower pane shows its four configuration parameters.

All presentations have a configuration parameter **Datatype**. Typically, the datatype determines whether a presentation is applied to an element or not.

Information. If you want a presentation to be applied to a dedicated *Attribute* (rather than *Datatype*, then simply create a specific datatype for that attribute. And conversely, you can reuse a single presentation for multiple attributes by using the same datatype for all relevant Attributes.

The order of entries sometimes (but rarely) matters. Presentations can be reordered using drag and drop.

Last, to delete a presentation, select **Delete** from the context menu by right-clicking.

5.2 Default Datatype Handlers

Sometimes, it is desirable to use a presentation as the default for rendering (and editing) a specific datatype. For example, the open source ProR can not render formatted (XHTML) text and instead shows a simplification in plain text. But formalmind Studio offers the RTF presentation that allows rich rendering and

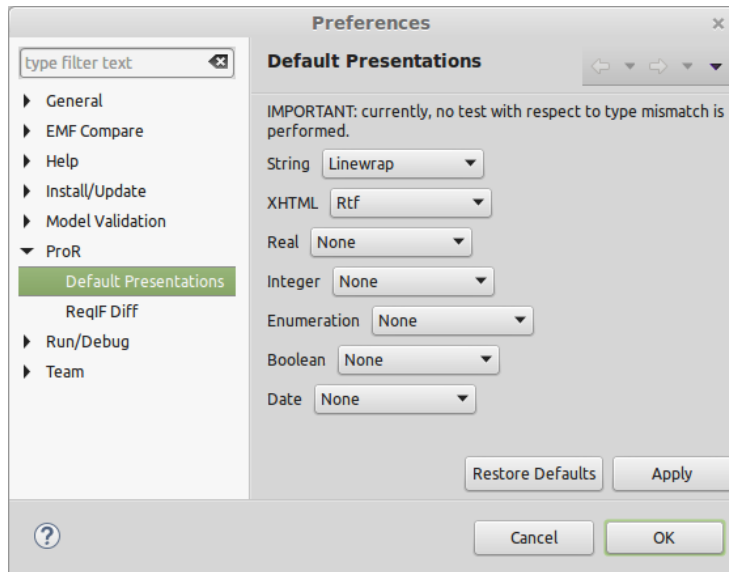


Figure 5.2: The preference dialog for handling default presentations

editing. Thus, it would be nice if XHTML datatypes would always be rendered using the RTF presentation, if installed.

This is possible. And in fact, presentations can request to be used as default handlers. If such a presentation is installed, it will automatically take over from the default.

Users can configure and override this. To do so, go to **Window | Preferences | ProR | Default Presentations**. The resulting dialog is shown in Figure 5.2.

For each of the standard ReqIF datatypes, a handler can be chosen. The default is **none**. The following entries are available:

None. This is the default and implies that the build-in renderer is used. Upon starting the tool, ProR will check whether a new presentation has been installed that requests to act as the default handler for that datatype. If such a presentation is found, it will be set as the handler.

Use Built-in. This forces the built-in handler to be used. Even if an installed presentation requests to be the handler, the built-in handler will continue to be used.

List of installed presentations. The remaining entries list all installed presentations. Note that the dialog does not filter the matching types. I.e. even though the RTF presentation is only applicable for XHTML, it is shown in the dropdown for all datatypes.

Warning. Please make sure that you only set datatype handlers of the correct type. Also note that there is no access to the configuration parameters of

default handlers. Therefore, this mechanism makes only sense for presentations that do not require any additional configuration.

5.3 Built-in ProR Presentations

The following presentations are part of the Eclipse RMF project and are therefore open source.

5.3.1 ID Generator Presentation

This presentation automatically generates user-friendly IDs, consisting of a prefix and a number, e.g. “REQ-12”. The configuration parameters are shown in Figure 5.2.

Information. Attributes to which this presentation applies are read-only and cannot be edited manually any more (as long as the presentation is active).

The parameters are:

Count. The counter position. When a new ID is created, it will be incremented by one. Changing it allows the user to start or continue with a different number.

Datatype. The presentation will be applied to all attributes with this datatype. Must be of type *String*.

Prefix. The prefix to the number. Note that changing this will only modify newly created IDs, not those already generated.

Vertical align. Allows to adjust how the text shall be rendered in the cell.

Warning. Currently, the presentation does not check whether duplicate IDs exist!

5.3.2 Headline Presentation

This presentation renders the content of a cell boldface in a large font. This can be useful if you want to format text to stand out (e.g. a headline), without having to resort to XHTML.

The parameters are:

Datatype. The presentation will be applied to all attributes with this datatype. Must be of type *String*.

Size. The size of the text in point.

5.3.3 Linewrap

This presentation wraps text at word boundaries, essentially what one would expect anyway. The built-in renderer breaks the text anywhere, including in the middle of words.

This presentation is usually set as the default presentation.

Information. Regular users can just ignore this presentation. It has been created for developers to understand the presentation mechanism.

Chapter 6

Style Guide

This simple guide is intended to explain the style used in this documentation. We request that you follow it as closely as possible when creating content in order to maintain a clear, concise and easy to read body of work.

The documentation was created with and is maintained with L^AT_EX. That stands to reason that much of the formatting is software-dependent, leaving us to worry about more important things. But that still leaves some decisions for us to make. For example, alphabetization, certain punctuation, spelling and hyphenation of certain words, and the like will be accounted for here.

6.1 Alphabetization

What should and should not be alphabetized:

- General terms such as “requirements” or “specifations” not used in a ProR or Eclipse specific manner are written in lower-case.
- ProR- and Eclipse-specific terms shall be capitalized. i.e., terms such as “Eclipse Workbench,” “SpecObject,” “Editor,” and “Views” are capitalized.

6.2 Punctuation

- **Mdashes** shall be used when needed—here is a perfect example—with no spaces.
- **Periods** will be followed by double-spaces.
- When **Colons** are used to introduce a series of numbered items in a list, do not capitalize the first item after the colon (unless it’s a proper noun) and separate the list items with semi-colons, e.g.: There are three differences: (1) the first difference; (2) the second difference; and (3) the third difference.

6.3 Contributing

Documentation is one of those things that gets easily neglected in open source projects. It is also one of the easiest for outsiders to contribute to. The documentation is managed as Latex, which may scare some people. But no worries, those who don't want to learn Latex don't have to.

There are broadly two ways for contributing to the documentation:

File a bug. Visit the RMF Bug Tracker. You can just point out a problem or request for improvement. You can also provide some text to be added to the documentation (unformatted). If you do, however, then you need to sign a Committer License Agreement (CLA)

Submit improved $\LaTeX$ via Gerrit. If you are technically inclined (meaning that you know what $\LaTeX$ and git are, and how to use them), then you can contribute via the Gerrit code review system, as described at eclipse.org.

6.3.1 Gerrit for Contributions

TODO - when done, update the parent section as well.