elements

Documentation Standards and Techniques

Doug Palmer[1]
TARMS Inc.

September 8, 2000

Typeset in LaTeX.

[1] doug@tarms.com

Contents

1 Introduction **1**

2 UML Documentation **1**

2.1 Package Layout 2

2.2 Documentation in Rose 2

2.3 Using LaTeX in Rose Documentation 3

2.3.1 Embedded LaTeX Use 4

3 Automatic Documentation Generation **4**

3.1 Diagram Translation 5

3.2 Large Documents 6

4 Citation **7**

1 Introduction

This document outlines documentation standards for the elements object model from a modeller's perspective. It is intended to provide a common standards for documentation patterns and layout, as well as guide to available facilities. This document does not address the production of additional documentation and essays; the focus here is on model documentation.

LaTeX[3] has been chosen as a vehicle for the automatic generation of documentation. Using Rational Rose[1] as the modelling tool, a Rose script traverses the model to generate documentation in LaTeX format. This documentation can then be processed to produce books or articles. The use of LaTeX is not expected to cause major difficulties for modellers. However, there needs to be some awareness of the package, to avoid falling foul of the special characters and commands that LaTeX uses.

2 UML Documentation

Rational Rose provides documentation screens for almost every UML element. In most cases, documentation can simply be added to whatever UML element that you are working with.

The documentation screens only accept plain ASCII text. Additional information in the form of mathematics, emphasis, accents, layout, etc. needs to be handled by embedded LaTeX commands.

2.1 Package Layout

This section properly belongs in a document on modelling standards. However, the modelling standards adopted make certain assumptions about the handling of packages. These assumptions need to be summarised.

`elements` is structured as a set of interdependent packages. Under the *Logical View* category in Rose. An individual package consists of a main sub-category of Logical View, which contains the bulk of the package, along with additional categories that represent other packages within `elements`.

In most cases, the classes in the additional categories will be *stubified;* replaced by empty name-holders. If the package *extends* an interface or class from another package, then additional attributes, methods, documentation, etc. will be added to the stubified class.

2.2 Documentation in Rose

Documentation in Rose should follow the following general conventions:

Main Category The main category in the package should contain documentation describing the package and its uses.

Use Cases Use cases should be named with a description of the use case and should hold documentation describing the use case or the example. Test cases and data should appear in use case documentation.

Interfaces, Classes, etc. Interfaces and classes should be documented with a description of the purpose of the interface and the class.

If a class is referred to in another piece of documentation, it is good practise to include a cross-reference to the class. In general, inheritance, realisation and other relationships are documented in special sections within the class and, therefore, need not have a cross-reference appended. The `\umlclassname{<name>}` command identifies a class, interface or exception and includes a suitable hyperlink or cross-reference to the class, if possible. The `\umlclassref{}` command allows cross-references to named classes, interfaces, exceptions, etc. The cross-reference is automatically removed if there is no class definition in this particular document.

Associations Associations should be documented with a brief description of the association. Roles should also, probably, be documented, if it adds anything to the documentation.

There are no standards for constraints, at present. Although any documentation in the constraints field is included in the automatically generated documentation.

Attributes Attributes should be documented with a brief description of what the attribute is and what it is for.

Operations In general, the documentation section for the operation should contain a brief description of what the operation is *for* in one paragraph, with subsequent paragraphs providing a more detailed discussion of the operation.

Alternately, the brief description can be placed in the documentation section and the detailed description relegated to the semantics section. This approach is useful when documenting class hierarchies where the basic definition of the operation remains constant, but the semantics change from class to class.

When classes implement the operations defined in interfaces, the documentation should reiterate the brief description of the operation. The more detailed description can then describe how the class is able to implement the operation.

Arguments to operations should be documented with a brief definition of the argument.

The preconditions and postconditions fields are not currently used in the automatically generated documentation. Until someone can find a clear use for them, I suggest that we leave them alone.

Extensions Extensions, in the form of additional documentation, operations and attributes, can be added to non-main package classes, interfaces, etc. These extensions will be placed in a special extension section in the package documentation.

2.3 Using LaTeX in Rose Documentation

For the most part, there are no special restrictions on documentation imposed by the use of LaTeX. Paragraphs should have a blank line between them. There is an informal convention to break a line after the end of a sentence, since TeX justifies the paragraph and it makes editing easier.

The main difficulty lies in handing certain special characters: $, #, %, _, { and } all need to be escaped by preceding them with a backslash (ie. `\$`, `\#`, `\%`, `\_`, `\{` and `\}`). Handling the `\`, `^` and `~` characters is a little more difficult, and they should be avoided.

2.3.1 Embedded LaTeX Use

LaTeX commands can be embedded into Rose documentation. On processing — usually as a result of automatic documentation generation, see section 3 — these embedded commands will be treated as ordinary LaTeX. The main uses of embedded LaTeX are:

- The addition of emphasis, by using the `\emph{}` command;
- The addition of mathematics, by using the `$...$` and `\[...\]` environments;
- The layout of tables, via the `tabular` or `longtable` environments;
- The processing of citations, references and index entries;
- Handling accented characters and
- The use of ligatures.[2]

Those intending to use embedded LaTeX are urged to examine [4]. If people want to get more sophisticated, they should read [3, 2].

Accented characters are generated by special commands. For example, `\'{e}` produces an é, `\c{c}` produces a ç or `\o` produces an ø.

Ligatures are generated by simply typing sequences of characters. Double quotes are handled by doubling single quotes (eg. ` ''Hello'' ` produces “Hello”). There are three lengths of dash, produced by doubling and tripling the dash characters (eg. `---` produces —, `--` produces –).

The `tabular` environment assumes that the generated table will stay on one page. For large tables, the `longtable` package allows multi-page tables; this package is included as part of the `umldoc` package used to process the UML documentation. Only single page tables can be floated to empty pages[3].

3 Automatic Documentation Generation

A single Rose package (ie., a single main category, plus stubified or extended ancillary categories) can be documented by using the ***latex.ebs*** Rose script.

To use this script, load the model that represents the package into Rose, load the ***latex.ebs*** script and run it. You will be presented with a dialog box asking for the main category[4] and then a file box asking for the output file.

[2] Squishing several characters into a single glyph

[3] Bugger!

[4] In future versions, I am considering allowing the selected category(s) to be the main category.

The file box does not actually use the file name given, it simply specifies the directory to write into.[5] The actual file names used are based on the main package name: the actual documentation is placed in a file called `*packagename*.tex`; a wrapper file, which can be used to generate a single article describing the package is placed in a file called `*packagename*-package.tex`; any class or use-case diagrams are placed in files called `cd*packagename*n.wmf`, these files will need to be converted to a suitable format (see section 3.1).

Once you have all the files you need, you can create package documentation by using the following steps:

1. Use LaTeX to process the generated documentation using the command **`latex *packagename*-package.tex`**.

2. If you have references (see section 4) run BibTeX using the command **`bibtex *packagename*-package`** to get references from the bibliography database.

3. Rerun LaTeX, possibly several times, if there are unresolved cross-references, or unstable long tables. Usually, three runs ensures stability; the transcript will inform you of any instabilities or unresolved references.

4. Run `dvips` to convert the output from LaTeX into Postscript, using the command
`dvips *packagename*-package`. Any diagrams are incorporated at this point.

5. Use Adobe Distiller to convert the resulting postscript file into an Portable Document Format (PDF) file, suitable for distribution to the great unwashed.

3.1 Diagram Translation

Rose scripts only know how to generate Windows metafiles (WMF) for diagrams. This is all very well, but it is not very useful for documentation generation. These files need to be converted into a more portable format — Encapsulated Postscript (EPS).

There are no readily available cross-platform filters for converting WMF to EPS which keep the scalability of both WMF and EPS[6]. The only thing that appears

[5] As soon as I figure out a way of just specifying directories, this will change!

[6] Both WMF and EPS are scalable formats, meaning that they can be compressed and expanded without losing information. This is pretty useful, as it allows large diagrams to be included in PDF files in a small format, with the user being able to zoom in on the diagrams. Most programs that can read and write both formats, such as PaintshopPro, convert the WMF into a bitmap and save the bitmap — not very useful.

to be available is the `wmf2eps` program[5].

The `wmf2eps` program uses the Adobe Postscript driver for Windows to convert the files. You *must* install the latest Postscript driver to successfully use this program and use the Adobe Distiller driver.

You can set up configuration files within `wmf2eps` to batch-process a series of WMF files. I would suggest that, once you have generated the documentation, you set up a configuration file — one per package – to handle the conversion, since you are likely to generate the same diagrams over and over again. In some cases, you will need to scale the WMF to a smaller size before converting to EPS, to avoid overflowing the EPS boundaries. These scale factors can also be stored in the configuration file.

If you have generated EPS diagrams already, and have not changed the diagrams since the last documentation generation, you can not bother with reconverting the diagrams.

3.2 Large Documents

The above documentation process creates one file of documentation per package, along with a "wrapper" file for producing per-package documentation. Instead of producing per-package documentation, you can link the documentation together to produce a documentation book, with one chapter per package.

To do this, you will need to write your own wrapper. This is not difficult, it is essentially a list of the files you want included, along with title, contents and bibliographic information. You will need, to use the following packages:

`times` Typesets in the Times-Roman font. This make the PDF files smaller, as this font is assumed to be resident in the reader.

`elements` A package containing stuff specific to the `elements` project, such as the logo, accessible via `\Elements`, etc.

`umldoc` A package that interprets the output of the Rose script and lays the documentation out.

`hyperref` A package that allows hyperlinks, both internally and externally, in a PDF document. Cross-references, table of contents entries, citations, etc. are automatically hyperlinked. The base `hyperref` package marks links in a fairly unattractive way; you will probably also want to use the `color` package and set up the `hyperref` options to provide more aesthetic links.

4 Citation

Citations can be added to documentation by using the `\cite{}` command. The arguments to this command consist of a list of keys into a bibliographic database. When processed by LaTeX, these citations are recorded and then filled out by a program called BibTeX. When the document is processed again, the bibliography is added to the document. When the document is processed yet again, the citations within the document are matched to the bibliography.

A common `elements` bibliography database (`elements.bib`) will be maintained, with all contributions inserted into the database. Since there are likely to be a number of citations in the form of URLs, the standard "plain" bibliography style has been extended, as `elements.bst` to allow URLs with hyperlinks in them as part of the reference[7]. Both `elements.bib` and `elements.bst` will be kept up to date on the `elements` web page.

Rose allows links to external files and URLs as part of its model. While these links are supported by the documentation generator, their use is deprecated, as they make the gathering of meaningful reference data difficult.

In many packages, there will be no references. Rather than create an empty references section in the package documentation, the generated documentation starts with the bibliography command commented out in the `*packagename*-package.tex` file. You will need to uncomment this to get a bibliography.

References

[1] Rational Corporation. *Rational Rose*. http://www.rational.com/products/rose/index.jtmpl.

[2] Michel Goosens, Frank Mittleback, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesely, 1994.

[3] Leslie Lamport. *LaTeX, a Document Preparation System*. Addison-Wesley, second edition, 1994.

[4] Tobias Oetiker, Hubert Partl, Irene Hyna, and Elisabeth Schlegl. *A not very Short Introduction to LaTeX2e*, 1994. (revision 1999). http://ctan.tug.org/tex-archive/info/lshort/english/lshort.pdf.

[7] I have yet to correctly allow PDF hyperlinks to appear and disappear as required by the document. For now, make sure that you have the `hyperref` package in the preamble of any document that you make.

[5] *A Windows Metafile to Encapsulated Postscript filter*.
http://tug.ctan.org/cgi-bin/CTANfilesearch.pl?FILESTRING=wmf2eps.