Rubber

Documentation for version 1.5.1

Sebastian Kapfer

Table of Contents

1 Introduction

The purpose of Rubber is to make the building of a document automated, from the source files to the final document file, replacing the work of a Makefile.

The basis is a routine that compiles a LaTeX source the right number of times to resolve all references and make all tables of contents, list of figures, and so on. On top of that, Rubber provides a modular system to handle various tasks needed apart from compilations. This, for instance, includes processing bibliographic references or indices, as well as compilation or conversion of figures. Additionally, modules can perform a post-processing of the document (for instance to convert a DVI to PostScript or PDF) or even a preprocessing (useful when the LaTeX source is generated by another program, like cweave).

Dependency analysis is performed by parsing the source files, so that modifying any source, user package, graphics file or other dependency leads to appropriate compilations. Modules are triggered either explicitly using command line options, or implicitly when the sources are parsed. For instance, BibTeX support is activated whenever the source contains commands that generate a bibliography, graphics support is activated by `\usepackage{graphics}` and similar commands, and so on. The modular approach allows any additional feature to be supported by simply writing a module to support it.

Some information cannot be extracted from the LaTeX sources. This is the case, for instance, with the search paths (which can be specified in environment variables like `TEXINPUTS`), or the style to be used with Makeindex. To address this problem, one can add information for Rubber in the comments of the LaTeX sources, see Chapter 3 [Directives], page 6.

The package comes with three different command line programs:

`rubber`
: Builds the specified documents completely. The source files may be either LaTeX sources or documents in a format Rubber knows how to translate into LaTeX.

`rubber-pipe`
: Does the same for one document but it reads the LaTeX source from standard input and dumps the compiled document on standard output.

`rubber-info`
: This is a utility for extracting various kinds of information from a LaTeX document, either from the source or from the compilation log files.

2 Command lines

The command line of each program is read using the GNU Getopt conventions. `rubber` and `rubber-pipe` mostly have the same syntax.

2.1 Syntax for `rubber` and `rubber-pipe`

The syntax of the command lines for `rubber` and `rubber-pipe` are:

```
rubber [options] files
rubber-pipe [options]
```

The source files may be either LaTeX sources (in which case the suffix `.tex` may be omitted) or documents in a format Rubber knows how to translate into LaTeX. This currently includes CWEB documents (filename extension `.w`, via `cweave`), Literate Haskell (`.lhs` via `lhs2tex`) and Knitr (`.Rtex`). If any compilation fails, the whole process stops, including the compilation of the next documents on the command line, and the program returns a non-zero exit code.

The options are the following:

`-b`
`--bzip2` Compress the final document (in `bzip2` format). This option is equivalent to saying `-o bzip2` after all other options. It is incompatible with the option `--gzip`.

`--clean` Remove all files produced by the compilation, instead of building the document. This option is present in rubber only. It applies to the compilation as it would be done with the other options of the command line, i.e. saying

```
rubber --clean foo
```

will not delete foo.ps, while saying

```
rubber --ps --clean foo
```

will.

`-c <command>`
`--command <command>`
Execute the specified command (or directive) *before* parsing the source files. See Chapter 3 [Directives], page 6.

`-e <command>`
`--epilogue <command>`
Execute the specified command (or directive) *after* parsing the source fiels. See Chapter 3 [Directives], page 6.

`-f`
`--force` Force at least one compilation of the source. This may be useful, for instance, if some unusual dependency was modified (e.g. a package in a system directory). This option is irrelevant in rubber-pipe.

`-z`
`--gzip` Compress the final document (in `gzip` format). This option is equivalent to saying `-o gz` after all other options. It is incompatible with the option `--bzip2`.

`-h`
`--help` Display the list of all available options and exit nicely.

`--inplace`
Go to the directory of the source files before compiling, so that compilation results are in the same place as their sources.

`--into <directory>`
Go to the specified directory before compiling, so that all files are produced there and not in the current directory.

`--jobname <name>`
Specify a job name different from the base file name. This changes the name of output files and only applies to the first target.

`-k`
`--keep` This option is for `rubber-pipe` only. With this option, the temporary files will not be removed after compiling the document and dumping the results on standard output. The temporary document is named `rubtmpX.tex`, where `X` is a number such that no file of that name exists initially.

`-n <num>`
`--maxerr <num>`
Set the maximum number of displayed errors. By default, up to 10 errors are reported, saying `-n -1` displays all errors.

`-m <module>[:<args>]`
`--module <module>[:<args>]`
Use the specified module in addition to the document's packages. Arguments can be passed to the package by adding them after a colon, they correspond to the package options in LaTeX. The module is loaded *before* parsing the document's sources.

`--only <sources>`
Compile the document partially, including only the specified sources. This works by inserting a call to `\includeonly` on the command line. The argument is a comma-separated list of file names.

`-o <module>[:<args>]`
`--post <module>[:<args>]`
Used the specified module as a post-processor. This is similar to the `-m` options except that the module is loaded *after* parsing the document.

`-d`
`--pdf` Produce PDF output. When this option comes after `--ps` (for instance in the form `-pd`) it is a synonym for `-o ps2pdf`, otherwise it acts as `-m pdftex`, in order to use pdfLaTeX instead of LaTeX.

`-p`
`--ps` Process the DVI produced by the process through `dvips` to produce a PostScript document. This option is a synonym for `-e module dvips`, it cannot come after `--pdf`.

`-q`
`--quiet` Suppress all messages during the process.

`-r <file>`
`--read <file>`
Read additional directives form the specified file (see also the directive `read`).

`-S`
`--src-specials`
Enable generation of source `\specials` if the compiler supports it. This is equivalent to setting the variable `src-specials` to '`yes`'.

`-s`
`--short` Display LaTeX's error messages in a compact form (one error per line).

`--synctex`
Enable SyncTeX support in the LaTeX run.

`-I <dir>`
`--texpath <dir>`
Add the specified directory to the search path of TeX files.

`--unsafe` Permit the document to invoke arbitrary external programs. This is potentially dangerous, only use this option for documents coming from a trusted source.

`-v`
`--verbose`
Increase the verbosity level. The default level is 0, levels up to 3 exist. Beware, saying `-vvv` makes Rubber speak a lot.

`--version`
Print the version number and exit nicely.

`-W <type>`
`--warn <type>`
Report warnings of the given type, if there was no compilation error. The available types are:

`boxes` overfull and underfull boxes,

`refs` undefined or multiply defined references,

`misc` other warnings,

`all` all of the above.

2.2 Syntax for `rubber-info`

The command-line syntax for `rubber-info` is the following:

```
rubber-info [options] [action] source
```

The options are all those accepted by `rubber` and `rubber-pipe`, as described in Section 2.1 [rubber command line], page 2. The `action` specified what kind of information

has to be extracted. At most one such argument must be present on the command line, `--check` is assumed if none is present. The possible actions are:

`--boxes` Extracts from the log file the places in the source where bad boxes appeared (these are the famous overfull and underfull `\hbox` and `\vbox`).

`--check` Report errors if there are any, otherwise report undefined references if there are any, otherwise list warnings and bad boxes. This is the default action.

`--deps` Analyse the source files and produce a space-separated list of all the files that the document depends on and that Rubber cannot rebuild.

`--errors` Extract from the log file the list of errors that occured during the last compilation.

`-h`
`--help` Display the list of all available options and exit nicely.

`--refs` Report the list of undefined or multiply defined references (i.e. the `\ref`'s that are not defined by one `\label`).

`--rules` Analyse the source files and produce a list of dependency rules. One rule is produced for each intermediate target that would be made when running `rubber`. Rules are formatted in the style of Makefiles.

`--version`
Print the version number and exit nicely.

`--warnings`
Stupidly enumerate all LaTeX warnings, i.e. all the lines in the log file that contain the string "Warning".

3 Directives

Some information cannot be extracted from the LaTeX sources. To address this problem, one can add information for Rubber in the comments of the LaTeX sources, in the form of directives. A directive is a line like

```
% rubber: cmd args
```

The line must begin with a '`%`', then any sequence of '`%`' signs and spaces, then the text '`rubber:`' followed by zero or more spaces and a directive name, possibly followed by spaces and arguments.

The argument in the directive line are separated by spaces, single and double quotes allow escaping of spaces (and quotes). The directive can contain variable references with the syntax '`$VAR`' or '`${VAR}`'. For details on the use of variables, see Section 3.2 [Variables], page 7.

If a directive name has the form '`foo.bar`', it is considered a command `bar` for the module `foo`. If this module is not registered when the directive is found, then the directive is silently ignored. See the individual documentation for modules for module-specific directives.

3.1 General directives

`alias <name1> <name2>`
: Pretend that the LaTeX macro '`name1`' is equivalent to '`name2`'. This can be useful when defining wrappers around supported macros, like:

```
% rubber: alias ig includegraphics
\newcommand\ig[1]{\includegraphics[scale=.5]{#1}}
```

`clean <file>`
: Indicates that the specified file should be removed when cleaning using `--clean`.

`depend <file>`
: Consider the specified file as a dependency, so that its modification time will be checked.

`make <file> [<options>]`
: Declare that the specified file has to be generated. Options can specify the way it should be produced, the available options are `from <file>` to specify the source and `with <rule>` to specify the conversion rule. For instance, saying

```
% rubber: make foo.pdf from foo.eps
```

indicates that `foo.pdf` should be produced from `foo.eps`, with any conversion rule that can do it.

`module <module> [<options>]`
: Loads the specified module, possibly with options. This is equivalent to the command-line option `--module`.

`onchange <file> <command>`
: Execute the specified shell command after compiling if the contents of the specified file have changed. In case the file or command contains spaces, they must be enclosed within double or single quotes.

`path <directory>`
: Adds the specified directory to the search path for TeX (and Rubber). The name of the directory is everything that follows the spaces after '`path`'.

`produce <file>`
: Declares that the LaTeX run will create or update the specified file(s). This implies that Rubber will check if the file contents changed and re-make other files derived from it; it also implies that rubber –clean will remove the indicated file. `produce` may be combined with `watch`. For example, Rubber's built-in behavior of recompiling the main LaTeX source until the auxiliary file's contents no longer changes may be achieved using

```
produce document.aux
watch document.aux
```

`read <file>`
: Read the specified file of directives. The file must contain one directive per line. Empty lines and lines that begin with '`%`' are ignored.

`rules <file>`
: Read extra conversion rules from the specified file. The format of this file is the same as that of `rules.ini`, see Section 5.3 [rules.ini], page 15.

`set <name> <value>`
: Set the value of a variable as a string. For details on the existing variables and their meaning, see Section 3.2 [Variables], page 7.

`setlist <name> <values>`
: Set the value of a variable as a list of strings. The list is space-separated, possibly empty. For details on the existing variables and their meaning, see Section 3.2 [Variables], page 7.

`shell_escape`
: Mark the document as requiring external programs (shell-escape or write18). Rubber does not actually enable this unless called with the option `--unsafe`.

`synctex` Enable SyncTeX support in the LaTeX run.

`watch <file>`
: Watch the specified file for changes. If the contents of this file has changed after a compilation, then another compilation is triggered. This is useful in the case of tables of contents, for instance.

3.2 Variables

The following variables are defined by Rubber (or its modules) and used by various modules to influence compilation. All variables are strings that should be defined by the `set` directive, unless explicitly specified.

`arguments (list)`
: Extra command-line arguments that are passed to the compiler. Note that this is potentially dangerous and has no reason to be portable across different compilers. This variable contains a list of strings, it should be set using the `setlist` directive.

`base` The base name of the main source file, including its path but without the extension.

`engine` The name of the TeX engine used. By default this is '`TeX`', this can be changed to '`VTeX`', '`pdfTeX`', '`Omega`' or others by the modules that change the compiler, see Section 4.3 [Compiler choice], page 12.

`ext` The extension of the main source file.

`file` The name of the current file (this is set during parsing).

`job` The job name of the document, with no path indication. Note that changing the value of this variable does not affect compilation; in order to actually change the job name, use the command line option `--jobname`.

`latex` Specify which program should be used for compiling. By default this is `latex`, various modules can change it in order to use a different compiler instead, see Section 4.3 [Compiler choice], page 12.

`logfile_limit`
Specify how much of the LaTeX logfile Rubber reads. The default is 1 MB, which should be ample for any real document.

`line` The current line number in the current file (this is set during parsing).

`path` The path name of the main output file.

`src-specials`
The kind of source `\special`s that should be generated. When empty (which is the case by default), no `\special`s are generated. When set to '`yes`', the default set is generated, otherwise the variable is passed as the argument of the `-src-specials` switch of the compiler.

4 Modules

4.1 Supported Packages

For every package that a document uses, Rubber looks for a module of the same name to perform the tasks that this package my require apart from the compilation by LaTeX. Modules can be added to the ones provided by default to include new features (this is the point of the module system). The standard modules are the following:

`asymptote`
: Process the `.asy` files generated by the LaTeX package, then triggers a recompilation.

`beamer`
: This module handles Beamer's extra files the same way as other tables of contents.

`biblatex`
: Takes care of processing the document's bibliographies with Biber or BibTeX when needed. For details, see Section 4.1.1 [BibLaTeX], page 10.

`bibtex`
`bibtopic`
`multibib`
: Takes care of processing the document's bibliographies with BibTeX when needed. The `bibtex` module is automatically loaded if the document contains the macro `\bibliography`. For details, see Section 4.1.2 [BibTeX], page 10.

`combine`
: The combine package is used to gather several LaTeX documents into a single one, and this module handles the dependencies in this case.

`epsfig`
: This modules handles graphics inclusion for the documents that use the old style `\psfig` macro. It is actually an interface for the graphics system, for details see Chapter 5 [Graphics], page 14.

`glossaries`
: Run `makeglossaries` and recompiles when the `.glo` file changes.

`graphics`
`graphicx`
: These modules identify the graphics included in the document and consider them as dependencies for compilation. They also use standard rules to build these files with external programs. For more details, see Chapter 5 [Graphics], page 14.

`hyperref`
: Handle the extra files that this package produces in some cases.

`index`
`makeidx`
`nomencl`
: Process the document's indexes with `makeindex` when needed. For details, see Section 4.1.3 [Indexing], page 10.

`minitoc`
`minitoc-hyper`
: On cleaning, remove additional files that produced to make partial tables of contents.

`moreverb`

`verbatim` Adds the files included with `\verbatiminput` and similar macros to the list of dependencies.

`ltxtable` Add dependencies for files inserted via the `ltxtable` LaTeX package.

`xr` Add additional `.aux` files used for external references to the list of dependencies, so recompiling is automatic when refer- enced document are changed.

4.1.1 BibLaTeX support

If the document loads the package `biblatex`, Rubber enables BibLaTeX support. Rubber will extract the appropriate external bibliography tool such as `biber` or `bibtex` from the `backend` option to the `biblatex` package and invoke it as needed.

`biblatex.path <directory>`
: Add the specified directory to the seach path for BibTeX database files (`.bib` files). The directory will be passed down to the bibliography tool in the environment variable `BIBINPUTS`.

4.1.2 BibTeX support

If the document contains a call to `\bibliography` or `\bibliographystyle`, then the BibTeX module is used. This triggers the execution of BibTeX between compilations when new references are made, bibliographies are changed, and in other appropriate cases. The following directives may be used to control BibTeX's behaviour:

`bibtex.crossrefs <number>`
: Set the minimum number of `crossref` required for automatic inclusion of the referenced entry in the citation list. This sets the option `-min-crossrefs` when calling `bibtex`.

`bibtex.path <directory>`
: Add the specified directory to the seach path for BibTeX database files (`.bib` files). The directory will be passed down to the bibliography tool in the environment variable `BIBINPUTS`.

`bibtex.stylepath <directory>`
: Add the specified directory to the search path for BibTeX style files (`.bst` files).

`bibtex.tool <command>`
: Call the specified command instead of `bibtex` to process the .aux file, for example `bibtex8`.

Multiple bibliographies can be handled by the `multibib` package. The directives provided by the `multibib` module are the same as those of the `bibtex` module, and they may be used with an optional first arument of the form '`(foo,bar,quux)`' in order to specify that the directive only applies to the bibliographies named '`foo`', '`bar`' and '`quux`'. By default, directives are applied to all bibliographies.

4.1.3 Index generation

The use of the packages `index`, `makeidx` and `nomencl` triggers the generation of an index (or several of them). Currently Rubber can use either the standard Makeindex or the

more sophisticated Xindy. The following directives may be used to control how indices are generated:

`index.tool <name>`
: Specifies which tool is to be used to process the index. The currently supported tools are `makeindex` (the default choice) and `xindy`.

`index.language <language>`
: Selects the language used for sorting the index. This only applies when using `xindy` as the indexing tool.

`index.modules <module>...`
: Specify which modules to use when processing an index with `xindy`.

`index.order <options>`
: Modifies the sorting options for the index. The argument must be a space-separated list of words among '`standard`', '`german`' and '`letter`'. his only applies when using `makeindex`.

`index.path <directory>`
: Adds the specified directory to the search path for index style files (`.ist` files).

`index.style <style>`
: Specifies the index style to be used.

Each of these directives may be used with an optional first arument of the form '`(foo,bar,quux)`' in order to specify that the directive only applies to the indexes named '`foo`', '`bar`' and '`quux`'. By default, directives are applied to all indices.

When using the package `makeidx` instead of `index`, the directives must of course be prefixed by '`makeidx.`' instead of '`index.`', and the optional first argument is not accepted.

4.2 Post-processors

The following modules are provided to support different kinds of post-processings. Note that the order matters when using these modules: if you want to use a processing chain like

```
foo.tex -> foo.dvi -> foo.ps -> foo.pdf -> foo.pdf.gz
```

you have to load the modules `dvips`, `ps2pdf` and `gz` in that order, for instance using the command line

```
rubber -p -o ps2pdf -z foo.tex
```

`bzip2` Produce a version of the final document compressed with `bzip2`.

`dvipdfm` Runs `dvipdfm` at the end of compilation to produce a PDF document.

`dvips` Runs `dvips` at the end of compilation to produce a PostScript document. This module is also loaded by the command line option `--ps`.

`expand` Produce an expanded LaTeX source by replacing `\input` macros by included files, bibliography macros by the bibliography produced by `bibtex`, and local classes and packages by their source. For details, see Section 4.2.2 [Expand], page 12.

`gz` Produce a version of the final document compressed with `gzip`.

`ps2pdf` Assuming that the compilation produces a PostScript document (for instance using module `dvips`), convert this document to PDF using `ps2pdf`.

4.2.1 Dvips and Dvipdfm

The `dvips` and `dvipdfm` modules can be used to call the associated DVI drivers to produce PostScript or PDF documents from the DVI output of LaTeX. The following directives may be used to change their behaviour:

`dvipdfm.options <options>`
`dvips.options <options>`
Pass the specified options to the driver. The argument is a space-separated list of options that are passed before the DVI file's name.

4.2.2 Source file expansion

The module `expand` produces a LaTeX source from the original one by expanding included files and packages. If the main file is `foo.tex` then then expanded file will be named `foo-final.tex`. This file will be self-contained, in particular it will not need BibTeXing nor user-defined packages, as may be required when preparing the final version of a document for publication.

As additional effect, this module removes all comments from the source file (including those that may contain Rubber directives). It also removes any text that may be present after `\end{document}`.

Please note that this module is rather experimental.

The following options control how the expansion is done:

`class` If the document class is user-defined (i.e. if it is in a local directory instead of a system directory), the `\documentclass` call will be replaced by the code of the `.cls` file. Note that this is dangerous in general.

`nobib` Do not expand the bibliography. When this option is *not* present, any call to `\bibiography` is discarded and the call to `\bibliographystyle` is replaced by the document's `.bbl` file.

`nopkg` Do not expand user-defined packages. When this option is *not* present, local packages (i.e. those that are in the current directory) are replaced by their contents.

4.3 Changing compilers

The following modules are used to change the LaTeX compiler:

`aleph` Use the Aleph compiler instead of TeX, i.e. compile with `lamed` instead of `latex`.

`omega` Use the Omega compiler instead of TeX, i.e. compiles the document using `lambda` instead of `latex`. If the module `dvips` is used too, it will use `odvips` to translate the DVI file. Note that this module is triggered automatically when the document uses the package `omega`.

`pdftex` Instructs Rubber to use `pdflatex` instead of `latex` to compile the document. By default, this produces a PDF file instead of a DVI, but when loading the module with the option `dvi` (for instance by saying `-m pdftex:dvi`) the document is compiled into DVI using `pdflatex`. This module is also loaded by the command line option `--pdf`.

`vtex` Instructs Rubber to use the VTeX compiler. By default this uses `vlatex` as the compiler to produce PDF output. With the option `ps` (e.g. when saying `rubber -m vtex:ps foo.tex`) the compiler used is `vlatexp` and the result is a PostScript file.

5 Graphics conversion

Rubber includes a system for automatic conversion of graphics (and other files) between various file formats. This is used when graphics are included using commands from the packages `graphicx`, `graphics` or `epsfig`.

When a graphics inclusion macro like `\includegraphics` (even with the parameters allowed by the `graphicx` package) or `\epsfig` is found in a LaTeX source, Rubber looks for the corresponding file or a way to generate it. If the call to the macro does not specify a file extension, then the list of possible suffixes is tried (according to the current compiler and the options passed to the graphics package). For each possible file name, Rubber tries to find a good way to convert this file from a source, and looks for the file itself. The precise method is described in Section 5.1 [Conversion algorithm], page 14.

5.1 Conversion algorithm

Files are converted using conversion rules, as described in Section 5.2 [Standard rules], page 15, and each rule can take a number of formats as input and produce a number of formats as output.

Assume you are using `pdflatex` to compile a document that contains a macro call `\includegraphics{foo}`. Since `pdftex` only accepts figures in `png`, `pdf` and `jpg` format, one of `foo.png`, `foo.pdf` and `foo.jpg` must be available. If the only available file around happens to be `foo.gif`, it has to be translated into one of these formats. There are tools to convert `gif` files into just about any other format, so we have to choose the proper conversion to perform.

Hence all rules have a *cost*, an integer that represents how unlikely a given conversion is. For instance the rule form `.fig` to `.eps` files (for XFig support) has cost zero because it is almost certain that this is the expected rule when the `.fig` file exists. On the other hand, the rule from `.eps` to `.jpeg` has cost 11 because this conversion is possible though usually not wanted.

Now assume that a directory contains a LaTeX source that contains `\includegraphics{foo}` and that both `foo.eps` and `foo.pdf` are present, because we want to compile the document both to PostScript and to PDF format. How can Rubber decide which figure is the source file and which is converted from the other? By default, in this situation, no conversion will ever be performed, since there is no way to decide. Generally, a file will never be overwritten if the source could have been produced by some conversion rule. This behaviour can be overridden using the `make` directive, see Section 3.1 [General directives], page 6.

To sum up, for each requested file, the detection algorithm works as follows. If relevant, take all possible suffixes (i.e. all accepted file formats, if no format is specified in the source), and take all possible directories where the file will be searched for by LaTeX. In each case, you get a file name `foo`. If there is an applicable rule with cost at most 0 to produce `foo`, use that rule. Otherwise, if the file `foo` exists, use it without conversion. Otherwise use the applicable rule with the least cost. If all fails, proceed to the next file name.

The list of standard rules is defined in the file `rules.ini` in the data directory. The syntax of this file is described in Section 5.3 [rules.ini], page 15, the standard rules are

described in Section 5.2 [Standard rules], page 15. Additional rules can be defined in a file with the same format and declared using the directive `rules`.

The path searched for graphics files is the same as that for LaTeX inputs by default. If the macro `\graphicspath` is used, the specified paths are also searched, the same way as LaTeX does. Limited support is also provided for the `\DeclareGraphicsExtensions` and `\DeclareGraphicsRule` macros.

5.2 Standard conversion rules

The following built-in rules are available:

`eps_gz` This rule is used to extract a bounding box from a gzipped EPS file, in order to be able to compile a document while keeping large figure files compressed.

`fig2dev` This is the exporting program that goes with XFig. It is used to convert files in `.fig` format into EPS, PDF or PNG. Rubber also supports the use of combined EPS/LaTeX of PDF/LaTeX, it is detected when an `\input` macro refers to a filename that ends in `.eps_t` or `.pstex_t`. When the suffix is `.eps_t` or `.pdf_t`, the same document will compile both in PostScript and PDF.

`mpost` MetaPost has extra support in Rubber. If a graphics is included with a filename that ends with a dot and a decimal number like `foo.42`, then it is considered to be generated from `foo.mp` if it exists. Dependency analysis is performed between MetaPost sources, so that recompilation always occurs when needed. If the compilation of a MetaPost source fails, the errors are reported as it would be done for LaTeX errors.

`shell` Other programs can be used using the `shell` rule. This rule takes an extra parameter `command`, specified in the rule file, that defines a shell command-line. This is used for simple conversion rules using command-line tools like `convert` (from ImageMagick), `epstopdf`.

5.3 Syntax of `rules.ini`

The rules file has a format in the style of Wind*ws INI files. The lines that start with a semicolon `;` are comments, empty lines are ignored. The file is decomposed in sections, each introduced by a line of the form

```
[rule-name]
```

where `rule-name` is a unique identifier for the rule. Following this header are lines of the form

```
key = value
```

where `key` is an attribute name and `value` is its value, without any quotes. The spaces around the value are stripped.

The following attributes are used:

`target` A regular expression that matches the target file name.

`source` A template that describes the source file name, with references to groups marked in the target expression as `\1`, `\2` etc. This template may also contain choices of the form `{foo,bar,quux}` to denote several possibilities for the source name.

`rule` The name of the conversion rule. This name currently refers to internal modules of Rubber, as described in Section 5.2 [Standard rules], page 15.

`cost` The cost of the rule, as an integer. In the default file, this ranges between 0 and 12.

6 Encoding issues

6.1 Design choices about input and output encoding.

Since version 1.5, rubber decodes `.tex` sources and log files with the UTF-8 encoding. Legacy 8-bits encodings are quite common, but can easily be converted, and should become rare now that the LaTeX community recommends UTF-8 for new documents.

For data exchanged with the underlying operating system, like arguments received from the command line, paths passed to subprocesses or messages displayed on the console, it makes sense that rubber uses the system default encoding: generally UTF-8, a code page depending on local settings on Windows.

In the rare cases where it converts auxiliary files by itself, rubber simply avoids to decode, and only deals with bytes between 0 and 127.

6.2 Conversion of legacy sources to UTF-8

Most text editors allow you to explicitly select this encoding for newly created files.

An existing source with any other encoding can be converted, either by selecting the appropriate option when saving from your favorite editor, or with a dedicated tool. For example, the following commands translate `doc.tex` from the `latin1` encoding.

```
cp doc.tex doc.tex.bak
iconv doc.tex.bak -f latin1 -o doc.tex -t utf-8
```

You should then update the `inputenc` line, check that the compilation goes well, and finally remove the `doc.tex.bak` backup.

Legacy US-ASCII sources, only containing bytes betwen 0 and 127 and relying on TeX constructs like `\'e` to encode local characters, are valid UTF-8 sources, and will be decoded correctly. No conversion is needed.

More generally, conversion from usual (8-bits, US-ASCII-compatible) encodings will only replace bytes from 128 to 255 with new UTF-8 byte sequences, but US-ASCII bytes will remain unchanged.

6.3 Handling of unconverted legacy sources

For various reasons, some sources cannot be converted. Rubber attempts to deal with them, but the result will never be perfect.

Accurate detection of the source encoding is not as easy as it seems. An `inputenc` command anywhere in an included `.tex` source can affect remaining bytes in all including files. Its arguments can be generated, or depend on the result of a test. In other words, rubber would need a full TeX interpreter, including features that the author now consider as bugs, like tolerating encoding errors inside comments.

Instead, rubber translates bytes from 0 to 127 to the right ASCII character, most bytes from 128 to 255 to the Unicode replacement character (often displayed as an empty square), and rare sequences of such bytes as random non-ASCII characters.

All encodings encountered in the TeX world share the property that a byte between 0 and 127 anywhere in the file always represents a single US-ASCII character, and that such

characters cannot be represented by another byte sequence. Since rubber searches for TeX commands expressed with such characters, it will most of the time be able to work even if it ignores all other bytes.

Non-ASCII characters will look ugly in log messages displayed on the screen, and this is acceptable.

Real problems may occur when commands intended for cooperation with the underlying operating system, like `\includegraphics` or `\input`, receive arguments, like paths, containing non-ASCII characters.

These characters will be decoded incorrectly, resulting in various problems later (no file found with the given path, for example). This problem is not specific to rubber, and such path arguments should always be formed of ASCII characters, ideally roman letters, decimal digits and a dot for the extension.

7 Index

7.1 Index of directives

7.2 Index of modules