The rpgtex Package

A package for generating beautiful RPG documents

Jack Fraser-Govil
October 24, 2025

Welcome to the **RPGTEX** package. This LaTeX package is designed to allow users to flexibly typeset documents associated with Role Playing Games such as *Dungeons & Dragons* – and many more besides. This packages defines a central engine: `rpgcore` which define a number of useful functions and classes, and a flexible set of `themes` which control how those commands are rendered in the final document.

Attribution & License

This package would not have been possible without the team who developed its predecessor, the 'DND 5e LateX Template' . That code was released under an MIT license, the text of which can be found in the LICENSE file. `rpgtex` is released under the same license.

Contents

Part I: rpgtex Core

1: Installation & Usage 2
Getting `rpgtex` 2
Configuring `rpgtex` 2
Package & Class Usage 3
Compiling 3

2: Main Engine 4
"Theme Commands" 4
Title Pages 4
Part Pages 6
Dice Commands 6
Utility Commands 8
Theme Commands 9

3: Variables 11
Colo(u)rs 11

4: Fonts 12
Font Names 12
Contours 12

Part II: rpgtex Classes

5: rpgbook Class 14

6: rpghandout Class 15

7: rpgcard Class 16

Part III: Themes

8: `default` Theme 18

9: `dnd` Theme 19

10: `scifi` Theme 20

PART I
rpgtex Core

Chapter 1: Installation & Usage

Getting `rpgtex`

There are a number of different ways to acquire `rpgtex`. Once you have installed it, it is vital to ensure that it is properly configured (see below).

texmf Installation

The simplest way to use `rpgtex` is to install it on the `texmf` path, where the compiler can automatically find it:

```
git clone
    https://github.com/DrFraserGovil/rpgtex.git
    "$(kpsewhich -var-value
    TEXMFHOME)/tex/latex/rpgtex"
```

This will clone the repository into yourLaTeX path.

Indirect Installation

If you want to tinker with `rpgtex` – such as by creating a new theme – it is helpful to have it in a more accessible location. Clone the repository into a location of your choice:

```
git clone
    https://github.com/DrFraserGovil/rpgtex.git
    ~/your/rpgtex/directory
```

You then have two options to make the package visible to the compiler:

Use TEXINPUTS

Setting the environment variable `TEXINPUTS` allows the compiler access:

```
TEXINPUTS=~/your/rpgtex/directory/::
```

(Or similar commands, depending on your shell – in `fish` you would call `set TEXINPUTS dir`).

Use Symlinks

You can symlink the install location to the texmf directory, allowing the compiler to act as if you had performed the texmf installation:

```
ln -sf ~/your/rpgtex/directory "$(kpsewhich
    -var-value TEXMFHOME)/tex/latex/rpgtex"
```

Overleaf (Not recommended!)

We do not recommend using Overleaf since the free-tier subscription has reduced compilation times drastically, making compiling documents using complex packages such as this one extremely difficult. Nevertheless:

1. Download this GitHub repository as a ZIP archive using the Clone or download link above.
2. On Overleaf, click the New Project button and select Upload Project. Upload the ZIP archive you downloaded from this repository.
3. Manually create the file `rpg-config.cfg` with the contents "`\edef\RpgPackagePath{../}`". This replaces the configuration step described below.

Configuring `rpgtex`

Wherever one installs `rpgtex` from, it is vital that it is properly configured. From within the `rpgtex`-root directory, call:

```
./configure
```

Or – if one is (reasonably!) wary about running arbitrary executables – manually create the relevant file:

```
cd <rpgtex root directory>
cmd="\edef\RpgPackagePath{$(pwd)}"
echo $cmd >> core/rpg-config.cfg
```

Why is configuration necessary?

TeX is generally set up so that when a file calls `include` or `input` it is possible to use filepaths relative to the package itself. `rpg.sty` can call `\input{core/font.sty}` and it will know to first check for the file relative to rpg.sty; even if the package resides within the texmf path and the user has no idea where `rpgroot/rpg.sty`, or `rpgroot/core/font.sty`, are.

An annoying exception to this is fonts and typefaces. `xelatex` searches for fonts based on *filepaths relative to the current working directory* – or from those installed in as system fonts.

Since `rpgtex` includes several (license free) typefaces as part of the provided themes, this poses a problem. We must either require that:

1. `rpgtex` documents can only be prepared in restricted locations relative to the install location of `rpgtex`.
2. Users must identify and specify the `rpgtex` root path when preparing a document
3. Users must install the provided fonts to the system path
4. `rpgtex` must be configured to know 'where it is', and so provide an absolute filepath to the internal fonts.

The Configuration step is the most portable and easiest-to-use of these options.

Without a `core/rpg-config.cfg` file, any document which includes `rpgtex` will fail to compile.

Package & Class Usage

`rpgtex` can be used either as a standalone package, or as part of a number of classes

Standalone Package

The standalone package can be used directly by including the `rpgtex` package:

```
\documentclass{arbitrary-class}

\usepackage[options]{rpgtex}

\begin{document}
....
```

This will load only the core commands into the document, and (unless called explicitly) no themes will be imported. Using the package in this way does not activate any of the commands which change the overall geometry, background or headers of the document.

Classes

`rpgtex` can also be loaded through a number of classes which drastically alter the appearance of the document, defining new geometries backgrounds and adding headers.

The provided classes are:

1. `rpgbook` (page 14). Based on the standard book class, this is designed for larger RPG documents.
2. `rpghandout` (page 15). Based on the article class, this is designed for shorter documents
3. `rpgcard` (page 16). A small-document class designed for creating modular 'handout' cards for items, spells or abilities.

Compiling

`rpgtex` uses the `fontspec` package to allow custom fonts, and therefore requires compiling with `xelatex` or `luatex`:

```
xelatex main.tex  #works
luatex main.tex   #works
pdflatex main.tex #fails
```

Chapter 2: Main Engine

"Theme Commands"

Several commands in this documentation are described as **Theme Commands**. These are commands that the user is *not expected to call*, but which are executed by the internal engine in the process of rendering the page, or as a result of other commands that the user has called.

A user who wishes to simply write documents using an unmodified `rpgtex` need only concern themselves with the User-Facing Commands.

On the other hand, these Theme Commands have been designed to provide a convenient interface for creating custom Themes – and so their documentation allows for designers to create powerful and flexible themes from within `rpgtex`.

Theme Commands can be split into two groups:

1. **Backend Commands** These are commands which are executed within a theme (or a class) to modify internal values, such as fonts and colors. A designer interacts with these commands by calling them.
2. **Placeholder Commands** These are virtual commands which are designed to be overwritten with completely custom code, which is executed when the core engine runs the command. A user interacts with these commands by redefining them (usually with `RenewDocumentCommand`).

A 'theme' is therefore a collection of Backend Commands (to configure the 'core engine') and redefinitions of Placeholder Commands to provide their own unique functionality.

Title Pages

User-Facing Commands

`\maketitle {}`

When called, creates theme-defined title pages using a custom format.

Syntax

```
\title {A title}
\subtitle {The subtitle} (optional)
\cover {path/to/image} (optional)
\author {Dr. W. Riter} (optional)
\begin {document}
 \maketitle
 ....
\end {document}
```

Details

Calls either `\RpgDrawCover` or `\RpgSimpleTitle` depending on the value passed to `RpgUseCoverPage`.

If `RpgUseCoverPage` has been set to true (usually by a class such as `rpgbook.cls`), then the image stored in `\@cover` (if there is one) is automatically used as a full-page background image. This is independent of the theme definition of `RpgDrawCover`, and occurs before that function is called – all subsequent drawing occurs over the top of the cover image.

\cover {m}
\@cover

Saves an image path to the variable \@cover, automatically used by \maketitle as the background image.

Syntax

```
\cover {path/to/cover_image}
```

Details

If RpgUseCoverPage has been set to true, then the image at this path will be used as a full-page image in the background of the page created by maketitle.
The default value is empty (\cover {}).

\subtitle {m}
\@subtitle

Saves a string to the variable \@subtitle. Themes may use this when defining their RpgDrawCover and RpgSimpleTitle.

Syntax

```
\subtitle {<string>}
```

Details

This command has no effect on its own (unlike cover which is automatically included in the background).
The default value is empty (\subtitle {}).

Theme Commands

\RpgUseCoverPage {m}

If true, maketitle creates a title page to populate, else the title is rendered as a heading.

Syntax

```
\RpgUseCoverPage {true/false}
```

Details

This is a Backend Command. When true, maketitle attempts to use \@cover and then calls RpgDrawCover. If false, it calls RpgSimpleTitle.

\RpgDrawCover {}

Executes over the top of the \@cover image to render a front cover.

Details

This is a Placeholder Command, used by themes to customise the appearance of the title page which appears in rpgbook class. The default value renders a single node at the centre of the page containing \@title, \@subtitle, \@author and \@date variables in the centre. More advanced themes (such as dnd or scifi) add decorative embellishments and place the text at custom locations.
This command is executed by maketitle if \RpgUseCoverPage {true} has been set by the theme, class or directly by the user. The command is called from within an existing tikz environment with the remember,overlay options active, allowing for page coordinates (i.e. current page.north) to be used.
If a \@cover has been defined, this command is executed after the image is placed, drawing on top of it.

`\RpgSimpleTitle {}`

Renders a 'header' title - a simple text-only title at the top of the page.

Details

This is a Placeholder Command, used by themes to customise the appearance of the title header which appears in `rpghandout` class. The default value places the title, subtitle and author at the top of the page. More advanced themes (such as dnd or scifi) add decorative embellishments and place the text at custom locations.
The Simple Title is configured so that, in a twocolumn document, it occupies the full page width; calling `centering` with the simple title therefore centers the text above both columns.

Part Pages

`\part`
`\part * {o m}`

Defines a wrapper around the standard `part` command that allows for tikz-based custom page formatting

Syntax

```
\part (*)[<image>]{<part-name>}
```

Details

There are three distinct behaviours that can be exhibited, depending on the presence or absence of the `*`, and the presence and value of `<image>`.

Command	Behaviour

`\part *{partname}`
`\part *[<any text>]{partname}`
`\part [none]{partname}`
	Uses original `part` command defined by underlying class.

`\part {partname}`	Calls `RpgDrawPartPage` on a blank background.
`\part [path/to/image]{partname}`	Places the corresponding image as a full-page background, and then calls `RpgDrawPartPage`.

`RpgDrawPartPage` (page 6) is a Theme Function, which executes a series of tikz functions to place the part title according to the theme specifications.

`\RpgDrawPartPage {m}`

Uses Tikz to draw a custom part page when activated by `\part` (page 6).

Syntax

```
\RpgDrawPartPage {<part title>}
```

Details

This is a Placeholder Command, allowing the designed to determine where to place the part name on the page, and what embellishments accompany it. The command is called from within an existing tikz environment with the `remember,overlay` options active, allowing for page coordinates (i.e. current page.north) to be used.
The default `part` command allows a user to specify a background image for their part page – it is not necessary to provide one within the drawing command.

Dice Commands

Dice are a mainstay of RPGs, and so it is important to have a standard way to report and simplify their expressions. We provide an interface for a standard 'dice + modifier' expression.

`\RpgDice {m}` Evaluates expressions of the form $ndx \pm m$, and outputs using a theme-dependent layout.

Syntax

```
\RpgDice {<dice-expression>}
```

Details

Uses regular expressions to extract and simplify the `dice-expression`, which must follow the following format:

Dice format

1. It must contain either 'd' or 'D' (the 'dice symbol')
2. The dice symbol must be immediately followed by a single number (the 'dice size')
3. The dice symbol may optionally be prefixed by a single number (the 'dice count')
4. The first (non-whitespace) character must be either the dice count (if present) or the dice symbol
5. The dice size must be followed by either a '+', '-', or the end of the expression.
6. After this, any number of standard numeric expressions may follow. This expression will be evaluated into a single 'modifier'.

The dice ignores any whitespace before the beginning of the expression, and arbitrary whitespace within the 'modifier' part of the exprssion.

Example	Output
`\RpgDice { 1d6-2}`	1d6-2
`\RpgDice {2D6 + 3*2^2}`	2d6+12
`\RpgDice {1d16}`	1d16
`\RpgDice {d8-3}`	d8-3
`\RpgDice {2*1d6}`, `\RpgDice {1 d6}`, `\RpgDice {3d 6 +3}`	(Fails to compile)

`RpgDice` is neat, but not necessarily impressive by itself. The true power of the expression is that it calls `RpgDiceFormat` to perform the output formatting (after performing the regular expression parsing), allowing designers to customise their dice formatting.
`RpgDice` is loaded in both `layout` and non-`layout` calls.

`\RpgDiceFormat {m m m}` Prints the values computed by `RpgDice`

Syntax

```
\RpgDiceFormat {<dice-count>}{<dice-size>}{<added bonus>}
```

Details

This is a Placeholder Command, used by theme designers to determine how `RpgDice` is rendered. The default option is: `\RpgDiceFormat {m m m} { #1d#2 #3}`, such that `RpgDice{ndx + a + b}` gives "ndx + c", where c is the numerical value of a+b, with an additional check to see if `#3` is equal to 0, in which case it is not printed (so as not to '1d6 + 0').
The dnd implementation performs a more advanced operation, computing the average value of the roll, and formatting that first, to replicate the format used by monster stat blocks.

Example (with `\RpgSetTheme {dnd}`)	Output
`\RpgDice {1d6-2}`	1 (1d6-2)
`\RpgDice {2D6 + 3*2^2}`	19 (2d6+12)
`\RpgDice {1d12}`	6 (1d12)
`\RpgDice {d83-3}`	39 (d83-3)

Utility Commands

`\RpgOrdinal {o m}`

Converts a numeric value to the corresponding ordinal.

Syntax

```
\RpgOrdinal [<command>]{<count>}
```

Details

The command outputs the `count` followed by the english abbreviations for the corresponding ordinal. The optional `command` argument is inserted between the numeral and the suffix, allowing for the customisation of appearances.

Example	Output
`\RpgOrdinal {1}`	1st
`\RpgOrdinal {2}`	2nd
`\RpgOrdinal {13}`	13th
`\RpgOrdinal [\textsuperscript ]{7}`	7^{th}
`\RpgOrdinal [\textbf ]{133}`	133**rd**
`\RpgOrdinal [<arbitrary text>]{133}`	133<arbitrary text>rd

Note that due to a lack of brace-capturing, it is not possible to chain multiple commands..

`\RpgPage {O{t} m}`

Outputs the current page reference for a label, with an option to enclose it in specific brackets or parentheses.

Syntax

```
\RpgPage [t/p/b/c]{<label-reference>}
```

Details

The optional arguments wrapping of the main reference. The options are:

t (default) No wrapping

p (parentheses)

b [square brackets]

c {curly braces}

An invalid input resolves to `?page~\pageref {<ref}?`.

Example	Output
`\RpgPage {example:current page}`	page 8
`\RpgPage [p]{example:current page}`	(page 8)
`\RpgPage [b]{example:current page}`	[page 8]
`\RpgPage [c]{example:current page}`	{page 8}
`\RpgPage [(error)]{example:current page}`	?page 8?

`\RpgPlural {o m m}`

Generates grammatically correct plural forms of a word based on a given count.

Syntax

```
\RpgPlural [<custom-plural>]{count}{<text>}
```

Details

The command outputs the count followed by the value of `<text>`. For a count of 1, the command then finishes. For any other count, it appends an "s", pluralizing the text.
The optional argument `[<custom-plural>]` overrides the default logic, allowing for irregular plurals.

Example	Output
`\RpgPlural {1}{hat}`	1 hat
`\RpgPlural {2}{hat}`	2 hats
`\RpgPlural [octopodes]{1}{octopus}`	1 octopus
`\RpgPlural [octopodes]{359}{octopus}`	359 octopodes

Theme Commands

`\RpgLayoutOnly {m}`

Executes the contents of the command if `layout` mode is active.

Syntax

```
\RpgLayoutOnly {<content-to-execute>}
```

Details

If the internal value `\l__rpg_layout_bool` is True, then `content-to-execute` is run, otherwise it is ignored.
This command is primarily used by theme developers and document class files to conditionally load or activate modules based on whether the package was loaded via a document class (layout mode active) or directly via `\usepackage {rpgtex}`.

`\RpgSetTheme {m}`

Activates a chosen theme.

Syntax

```
\RpgSetTheme {<theme-name>}
```

Details

Searches for the file `<theme-path>/<theme-name>/<theme-name>.cfg`, and inputs it. If this is a properly configured theme file, then it activates the chosen theme given the current global parameters. If the file does not exist, throws an error.
If `\l__rpg_layout_bool` is True, the command automatically inserts `\clearpage`, as required to ensure the old headers are not overwritten by the new theme.
`<theme-path>` is modified via `RpgSetThemePath`.

`\RpgSetThemeColor {m}`

Sets the `themecolor`, and simultaneously updates the co-varying colors (page 11).

Syntax

```
\RpgSetThemeColor {color-name}
```

Details

If `color-name` specifies a valid color, then the value of `themecolor` is updated, as well as a number of other colors (`tipcolor`, `sidebarcolor` and `tablecolor`) which are set to be equal to the themecolor by default.
Of the rpg-provided colors, only `narrationcolor` is unaffected by this command.

`\RpgSetThemePath {m}` Changes the value of the theme path searched for by `RpgSetTheme`

Syntax

```
\RpgSetTheme {<path-name>}
```

Details

Updates an internal variable to be equal to the input value; does not check if the theme path is valid or not. Useful if you wish to create a new theme outside of the `rpgtex` file structure.

Chapter 3: Variables

Colo(u)rs

`rpgtex` by default defines a number of colors[1] which are used for different elements:

themecolor A 'basic color' which is (by default) equal to the following three colors:

1. **sidebarcolor** The background color of the `RpgSidebar` environment
2. **tablecolor** The background color of every other row in an `RpgTable`
3. **tipcolor** The background color of the `RpgTip` environment

narrationcolor The background color of the `RpgTip` environment

contourinnercolor The default color of the inner text within a `RpgContour` command

contouroutercolor The default color of the external contour drawn around text within a `RpgContour` command.

Calling `\RpgSetThemeColor` (page 9) updates the value of `themecolor`, as well as the three 'co-varying' colors (i.e. everything except `narrationcolor`). When `printmode` is active `\RpgSetThemeColor{white}` is called, making environments transparent.

[1] Yes, I hate myself, but we're going with the code-based spelling.

Chapter 4: Fonts

Family vs Style

When defining the Font for an element, the interface allows one to specify both a `family` and a `style`. Formally speaking, `family` defines the **typeface** used by the associated element, whilst the `style` determines the options passed to that typeface (bold, italics, size etc.).

The distinction is largely irrelevant, as the construction of the final font object is often simply the concatenation of the two:

```
\def\RpgFontX
{
 \l__rpg_x_family \l__rpg_x_style
}
```

The separate definitions is therefore largely a matter of clarity and readability. It is generally fine to place commands should be 'family' in the 'style' element.

Font Names

Contours

`\RpgContour {O{} m}`

Renders text with a contour effect. The color and style are set through key/value pairs.

Syntax

```
\RpgContour [inner=<color>,outer=<color>,style=<code>]{<text>}
```

Details

The `style` command is applied to the text, whilst the optional `inner` and `outer` commands set the base text colour and the external contour color respectively. If the colors are not set, the default values are the `contourinnercolor` and `contouroutercolor` values defined by the theme (page 11). The contour does not automatically linebreak, but can be controlled manually with a `\newline` command (not \\ or \par)

Example	Output
`\RpgContour [inner=red,outer=black]{example}`	example
`\RpgContour [style=\Huge \it ]{example}`	*example*
`\RpgContour []{multi\newline line\newline example}`	multi line example

Quirks

Due to the tokenisation required for the line-splitting and space-preservation, the text inside the contour can exhibit some quirks if stylisation is applied within the `<text>` argument.
Unbraced commands (such as `\it` or `\footnotesize`) will only apply to the first word in the text.
Braced commands *can* work, but will cause a compilation error if a `\newline` is included.

`\RpgContour []{\Huge \it only first word changes}`	*only* first word changes
`\RpgContour []{\textit {all words change}}`	*all words change*
`\RpgContour []{\textit {all word \newline change}}`	(fails to compile)

For robustness, we therefore recommend that all stylisation be applied through the `style` command, which is applied to each tokenised element, and therefore guaranteed to work as expected.

PART II
rpgtex Classes

Chapter 5: rpgbook Class

Chapter 6: rpghandout Class

Chapter 7: rpgcard Class

PART III
Themes

Chapter 8: `default` Theme

Chapter 9: dnd Theme

Chapter 10: scifi Theme