The rpgtex Package

A package for generating beautiful RPG documents

Jack Fraser-Govil
October 28, 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

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 24). Based on the standard book class, this is designed for larger RPG documents.
2. `rpghandout` (page 25). Based on the article class, this is designed for shorter documents
3. `rpgcard` (page 26). 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)

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}`.

`\RpgSetFont` See page 16

`\RpgSetPaper` Sets a background image to be used as the `paper' image.

Syntax

```
\RpgSetPaper {path/to/image}
```

Details

If `layout` mode is active, then this configures `rpgtex` to use the image as the `background image' of every page with `fancy, plain` or `clear` pagestyle. This allows for custom `paper textures' to be loaded in in the background.
The pagestyle `clear` is equal to `empty`, with the exception of the page texture.

`\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 15).

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.

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 9
`\RpgPage [p]{example:current page}`	(page 9)
`\RpgPage [b]{example:current page}`	[page 9]
`\RpgPage [c]{example:current page}`	{page 9}
`\RpgPage [(error)]{example:current page}`	?page 9?

`\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

Chapter 3: Environments

Rpg Boxes

`rpgtex` defines three `colorbox' environments, which inherit from tcolorbox .

`RpgNarration {o}`

A tcolorbox wrapper designed for text that is read aloud to players

Syntax

```
\begin {RpgNarration}[color=<color>,<tcbox-options>]
  <text>
\end {RpgNarration}
```

Details

RpgNarration does not (by default) set a title, using only `body text', which is typeset using the `RpgFontNarration` font. The optional `<tcbox-options>` argument can be a list of all the basic tcolorbox options (see that documentation). The `color` argument is an alias for `colback` (`colbacktitle` is also set, but is ignored as the title is empty). Due to the order of processing, if both `color` and `colback` are set, the value of `colback` is used.
Themes may alter the appearance of the narration block using the tcb interface, calling `\tcbset {rpgnarration /.append~style={...}}` to overwrite the existing instructions.

`RpgSidebar {o m}`

A decorated tcolorbox wrapper designed for information which is set outside the main text.

Syntax

```
\begin {RpgSidebar}[color=<color>,<tcbox-options>]{<title>}
  <text>
\end {RpgSidebar}
```

Details

RpgSidebar requires a title (using `RpgFontSidebarTitle`) as well as the body text (`RpgFontSidebarBody`). RpgSidebar is typically more highly decorated than RpgTip, and does not have the `breakable` flag set. It is usually best to use one of the `float' options.
The optional `<tcbox-options>` argument can be a list of all the basic tcolorbox options (see that documentation). The `color=x` argument is equivalent to calling both `colback=x` and `colbacktitle=x`. Due to the order of processing, if both `color` and `colback` are set, the value of `colback` is used.
Themes may alter the appearance of the sidebar using the tcb interface, calling `\tcbset {rpgsidebar /.append~style={...}}` to overwrite the existing instructions.

`RpgTip {o m}` A simple tcolorbox wrapper designed for information which is set outside the main text.

Syntax

```
\begin {RpgTip}[color=<color>,<tcbox-options>]{<title>}
  <text>
\end {RpgTip}
```

Details

RpgTip is similar to RpgSidebar, requiring a title (`RpgFontTipTitle`) in addition to the body text (`RpgFontTipBody`). However, it is generally simpler, enabling it to safely break over page boundaries. The optional `<tcbox-options>` argument can be a list of all the basic tcolorbox options (see that documentation). The `color=x` argument is equivalent to calling both `colback=x` and `colbacktitle=x`. Due to the order of processing, if both `color` and `colback` are set, the value of `colback` is used.
Themes may alter the appearance of the narration block using the tcb interface, calling `\tcbset {rpgnarration /.append~style={...}}` to overwrite the existing instructions.

Which Colorbox To Use?

The choice between RpgSidebar and RpgTip is somewhat arbitrary -- although they have a mechanical difference by default (one being breakable, the other not) -- this can be overridden by themes. Instead, the intention is that they serve slightly different purposes:

RpgSidebar is used for `important information' -- key rules or summaries which readers *should* pay attention to.

RpgTip is for `helpful additions' -- tips, tricks and trivia that are not necessary, but which might be useful, and are too big to fit into a footnote or parenthetical.

Colorbox Examples

```
\begin{RpgNarration}[color=blue!30!white]
 This is text that you would read out loud to
     players, describing a scene.
\end{RpgNarration}

\begin{RpgSidebar}{A Sidebar}
 This is an important block of text, that you
     should pay attention to.
\end{RpgSidebar}

\begin{RpgTip}{A Tip}
 This is some helpful - but not vital - text.
\end{RpgTip}
```

This is text that you would read out loud to players, describing a scene.

A Sidebar
This is an important block of text, that you should pay attention to.

A Tip
This is some helpful - but not vital - text.

RpgTable

`RpgTable {o m}` Begins an environment for creating visually appealing and consistent tables.

Syntax

```
\begin {RpgTable}[<options>]{<column-specifications}
  <table-contents>
\end {RpgTable}
```

Details

RpgTable is a wrapper for the tabularx (or xltabular -- see `breakable`) environment, and so accepts the standard set of column specifications:{c,l,r,pwidth,,...} and the extended set (i.e. X). It therefore acts almost identically to the standard tabular environment with a few stylistic differences.

Stylistic Changes

The RpgTable environment makes the following changes:

1. **Title.** If the `title` option is set, a title-heading is rendered above the tablular in the `RpgFontTableTitle` font.
2. **Auto-headings.** The first row of the tabular environment is automatically rendered in the `RpgFontTableHeader`, allowing for trivial header labels.
3. **Font Integration.** The main body of the table is rendered in `RpgFontTableBody` font.
4. **Auto-colouring.** The rows alternate between being transparent and being set to the `tablecolor` variable (page 15). This is powered by `rowcolors`.

Optional Arguments

width=<dimexpr> Fixes the width of the tabular environment to the value of this argument. Default value is the current `\linewidth`.

color=<color-name> If set, uses this value instead of `tablecolor` for the alternating coloration.

title=<text> Sets the text to be rendered as the title of the table.

breakable If flag is present, renders using `xltabular`, enabling the table to break over pages. **only available in 1-column mode (a fundamental limitation of xltabular).**

noheader If flag is present, suppresses the autoformatting of the title. The first row is instead rendered in the body formatting.

RpgTable Example

This is the standard usage of the table, showing automatic formatting of the header rows and the word-wrapping abilities of the X-column:

```
\begin{RpgTable}[width=0.75\linewidth,
    color=green!30!white]{lX}
    Header 1 & Header 2
    \\
    Text & Some text which fills up the space to
        75\% of the line width then breaks
 \\
 Alternating & This row is transparent
 \\
 Colour & but this one is the colour we set in
     the header
\end{RpgTable}
```

Header 1	Header 2
Text	Some text which fills up the space to 75% of the line width then breaks
Alternating	This row is transparent
Colour	but this one is the colour we set in the header

This example adds a title, but suppresses the header formatting:

```
\begin{RpgTable}[title={Test
    Table},noheader]{XlX}
    Plain & Header & Text
    \\
 The middle column & is & small, but the first
     and third are big!
\end{RpgTable}
```

Test Table

Plain	Header	Text
The middle column	is	small, but the first and third are big!

Chapter 4: Variables

rpgtex defines many dozens to hundreds of variables, most with the (expl3) syntax `\l__rpg_[x]`. Most of these are used in the internal functioning of the macros, however a number of them are useful for a designer to understand.

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 8) updates the value of `themecolor`, as well as the three `co-varying' colors (i.e. everything except `narrationcolor`). Other colors are modified simply using the xcolors interface:
`\colorlet{narrationcolor}{html}{FFFFF}`

Command Line Interface

By default, LaTeXdoes not have a `command line interface' which allows a user to modify the document from within the command line: changes to the document have to be placed inside the file, and then compiled. However, we found that -- particularly with the *print* option (which suppresses background images on the paper, reducing ink requirements for printing), it was convenient to be able to compile the same document in either `normal' mode, or `print mode', without modifying the text.

To this end, we have provided a method for pseudo-`command line variables' to be inserted into the RpgOptions module. To do this, we exploit the fact that TeX can read documents from an input stream, not just files.

`\RpgCMD` Holds key-value pairs to be inserted into RptOptions after the standard parsing is run, ideal for command line modification.

Syntax

```
xelatex "\def \RpgCMD {<rpg-options>} \input {<document}"
```

Details

This will compile the `<document>`, with the contents of `RpgCMD` parsed as if they had been placed into `\documentclass [<rpg-options>]{rpgclass}` or when invoking the package: `\usepackage [rpg-options]{rpgtex}`.
Values passed to `RpgCMD` will override values passed to the package the standard way.

The `rpgtex` compiler which we have provided (page 22) performs this insertion by default for several predefined variables:

```
rpgtex document.tex -p (alias for xelatex "\def\RpgCMD{bg=print} \input document.tex")
```

Thereby allowing the user to switch between `print` and `full` mode with a compiler switch.

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

Chapter 5: Fonts

rpgtex allows for a high degree of customisation of the fonts and typefaces used for the elements within a document. The key interface for the designer is the \RpgSetFont command, which accepts a wide variety of key-value inputs, detailed below.

Font Interface

\RpgSetFont {m}

Saves new font values and styles to the internal RpgFont[X] variables, which are then available for themes to use.

Syntax

```
\RpgSetFont {<key-value-pairs>}
```

Details

Note that this interface **does not automatically change all fonts**. The SetFont-interface saves values to internal variables which populate the corresponding `RpgFont[X]` macros. It does not invoke `fontspec` and does not automatically assign fonts to the designated elements. *However*, a theme designer may then use the `RpgFont[X]` macros within their commands, thereby assigning fonts to the relevant elements. In this case (or if a user manually invokes a font), this command will act to update the font.
To summarise: if a writer uses a theme which does not make use of the `RpgFontSection` font, then calling `RpgSetFont{section-style=\it }` will have no effect. They would need to set `\titleformat {\section }{ \RpgFontSection }{}{}` (titlesec is loaded by default), in which case calling `RpgSetFont` would change the font for all subsequent calls to `\section`.
The exception to this is the main-body font, which is achieved by updating `\normalfont`.

Defining Fonts

The arguments passed to the `style' can be any form of latex formatting (i.e. `\slshape\scriptsize\bfseries`, and so on). To update the typeface, however, you must define a font family:

```
\newfontfamily{\myfont}{arial}
\RpgSetFont{main-body-family=\myfont}
```

For custom typefaces - or where you wish to `mix and match' typefaces in different modes, you can use the full power of the fontspec package :

```
\newfontfamily{\myfont}{custom-font}[
 Path=/path/to/local/font.otf,
 ItalicFont=*-Bold,
 BoldFont=Arial,
]
\RpgSetFont{main-body-family=\myfont}
```

This would define a font which used a local .otf file for the main font, but the bold typeface when `\textit{}` were called, and used arial as the mock `boldface'.

Font Elements

rpgtex provides 28 Font Commands, each comprised of a *family* and a *style*.

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 safe to place commands that should be in family into the style key, as long as it doesn't conflict with other styling.

Font vs Implementation

We generally encourage designers to place all text visualisation within the relevant Font rather than elsewhere -- if all subsections are going to be in red, then define `subsection-style=\color{red}`, rather than setting it within the titlesec specification (`\titleformat {\subection}{\RpgFontSubsection\color{red}}...`).

There will naturally be some exceptions to this: we found that the `RpgTitleFont` colour we wanted within `RpgDrawCover` diverged so strongly from that in `RpgSimpleTitle` that it made sense to define a special colour when rendering over a background image.

These fonts are assigned to typesetting elements by the theme designer -- what we have intended to be the section font may, within a different theme, be used for a different element. This document outlines how we have used these elements in the provided themes, though this is not prescriptive.

Font Element	Values	Usage
`\RpgFontBody`	`main-body-family` `main-body-style`	The main body text of the document, which `RpgSetFont` sets equal to `\normalfont`. Updating the fontsize here (i.e. using `\large`) can cause some counterintuitive results since it will *only* update the body text, and not adjust the other elements relatively. Adjusting the font size for the entire document should be done in the documentclass declaration.
`\RpgFontTitle`	`title-family` `title-style`	The font used for `\@title` when `\maketitle` is called.
`\RpgFontSubtitle`	`subtitle-family` `subtitle-style`	The font used for the value of `\@subtitle` (page 5), `\@author` and `\@date` when `\maketitle` is called.
`\RpgFontPart`	`part-family` `part-style`	The font used when `\part` is called.
`\RpgFontTocPart`	`toc-part-family` `toc-part-style`	The font used for a part in the table of contents
`\RpgFontChapter`	`chapter-family` `chapter-style`	The font used when `\chapter` is called.

`\RpgFontTocChapter`	`toc-chapter-family` `toc-chapter-style`	The font used for a chapter in the table of contents
`\RpgFontSection`	`section-family` `section-style`	The font used when `\section` is called.
`\RpgFontTocSection`	`toc-section-family` `toc-section-style`	The font used for a section in the table of contents
`\RpgFontSubsection`	`subsection-family` `subsection-style`	The font used when `\subsection` is called.
`\RpgFontSubsubsection`	`subsubsection-family` `subsubsection-style`	The font used when `\subsubsection` is called.
`\RpgFontParagraph`	`paragraph-family` `paragraph-style`	The font used when `\paragraph` is called.
`\RpgFontSubparagraph`	`subparagraph-family` `subparagraph-style`	The font used when `\subparagraph` is called.
`\RpgFontTableTitle`	`table-title-family` `table-title-style`	The font used for `<text>` if `\RpgTable` (page 13) is called with the `title=<text>` option.
`\RpgFontTableHeader`	`table-header-family` `table-header-style`	The font used for the first row of a `\RpgTable`.
`\RpgFontTableBody`	`table-body-family` `table-body-style`	The font used for the text within an `\RpgTable` after the first row.
`\RpgFontTipTitle`	`tip-title-family` `tip-title-style`	The font used for the title of an `RpgTip` environment (page 12).
`\RpgFontTipBody`	`tip-body-family` `tip-body-style`	The font used for the body of an `RpgTip` environment (page 12).
`\RpgFontSidebarTitle`	`siderbar-title-family` `siderbar-title-style`	The font used for the title of an `RpgSidebar` environment (page 11).

`\RpgFontSidebarBody`	`sidebar-body-family` `sidebar-body-style`	The font used for the body of an `RpgSidebar` environment (page 11).
`\RpgFontNarration`	`narration-family` `narration-style`	The font used for all (since they have no title) of an `RpgNarration` environment (page 11).
`\RpgStatBlockTitle`	`stat-block-title-family` `stat-block-title-style`	The font used for the title of a `statblock' environment - in the dnd theme this corresponds to the `monster` environment.
`\RpgStatBlockSection`	`stat-block-section-family` `stat-block-section-style`	The font used for sections within a `statblock' environment (should one be defined).
`\RpgStatBlockBody`	`stat-block-body-family` `stat-block-body-style`	The font used for text within a `statblock' environment (should one be defined).
`\RpgFontFooter`	`footer-family` `footer-style`	The font used for the footer text
`\RpgFontPageNumber`	`page-number-family` `page-number-style`	The font used for the page number within the footer
`\RpgFontDropCap`	`drop-cap-family` `drop-cap-style`	The font used for the large drop-cap letter created by a `RpgDropCap` (see below).
`\RpgFontDropCapInternal`	`drop-cap-internal-family` `drop-cap-internal-style`	The font used for the first line of text following the drop cap.

Decorative Text

In addition to the fundamental typeface alterations `rpgtex` includes a number of commands to turn text into decorative elements.

`\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 15). 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.

`\RpgDropCap {O{}, m m}` Creates a decorative `drop cap' letter to begin a new chapter with, and modifies the following text.

Syntax

```
\RpgDropCap [<lettrine-args>]{<letter>}{<text>}
```

Details

This command uses the lettrine package and the magaz package to create an easy-to-use environment in which the first letter is enlarged (and stylised in the `RpgFontDropCap` font). The second argument formats *up to the first line* of text in the `RpgFontDropCapInternal` font (usually a simple `scshape` command).

This command can be a little fragile -- lettrine does not usually play well with the `FirstLine' command provided by magaz -- and we've used a few workarounds to allow both linebreaking, and the formatting of only the first line of text. There may need to be a small amount of manual calibration, but it is better than the default.

Example

```
\RpgDropCap {A}{n example: \blindtext }
```

Output

An example: Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Chapter 6: rpgtex Compiler

`rpgtex` is shipped with a special compiler, `rpglatex`. This is simply a python3 script which acts as a wrapper around either xelatex or luatex, but includes several quality-of-life changes to the interface to make it easier to use with `rpgtex`.

`rpglatex` Compiles latex documents using either xelatex or luatex

Syntax

```
rpglatex [options] <file>
```

Details

`rpglatex` has the following features:

Feature	Description	Options
Compiler Selection	The `xelatex` compiler is selected by default, but the `-l, -luatex` flags set it to use luatex instead.	`-l, -luatex`
Build Directory	Compilation files (.aux, .log etc.) are stored in a build directory. The default is `.build` in the calling location, but can be changed with the `-b` flag	`-b <build dir>`
Output Naming	The name of the output file can be changed from the default (equal to the input tex name)	`-o <output name>`
Multi-pass Compiling	By default, the compiler runs twice in a row to enable references and `tikz[remember]` commands to function. A full three-compilation suite (necessary for very complex or reference-heavy documents) can be activated with the `-f, -full` flag	`-f, -full`
Volume Control	latex is notoriously noisy, producing copius output. By default, this is suppressed and only a summary is printed. The summary can be removed (rendering it completely silent) with the `-q` command, or the original output recovered in verbose mode; `-v`. These outputs are always overriden if a compilation error occurs, in which case the full trace is output to the console.	`-q, -v`
Auto-bibtex	If the `-r` or `-ref` flag is set, `bibtex` is automatically called in between the multi-compilation steps	`-r, -ref`
Auto-visualisation	If the `-show 1` option is set (which it is by default), the compiler will call `xdg-open <output-file>` upon completion of the compilation; automatically opening or context-switching to the document. This can be turned off by calling `-show 0`	`-show`
Print Mode	A special interface for `rpgtex`, this uses the `\RpgCMD` interface (page 15) to inject code into the latex document, setting the `bg=print` mode and suppressing the background output.	`-p, -print`

PART II
rpgtex Classes

Chapter 7: rpgbook Class

Chapter 8: rpghandout Class

Chapter 9: rpgcard Class

PART III
Themes

Chapter 10: `default` Theme

Chapter 11: dnd Theme

Chapter 12: `scifi` Theme

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Index