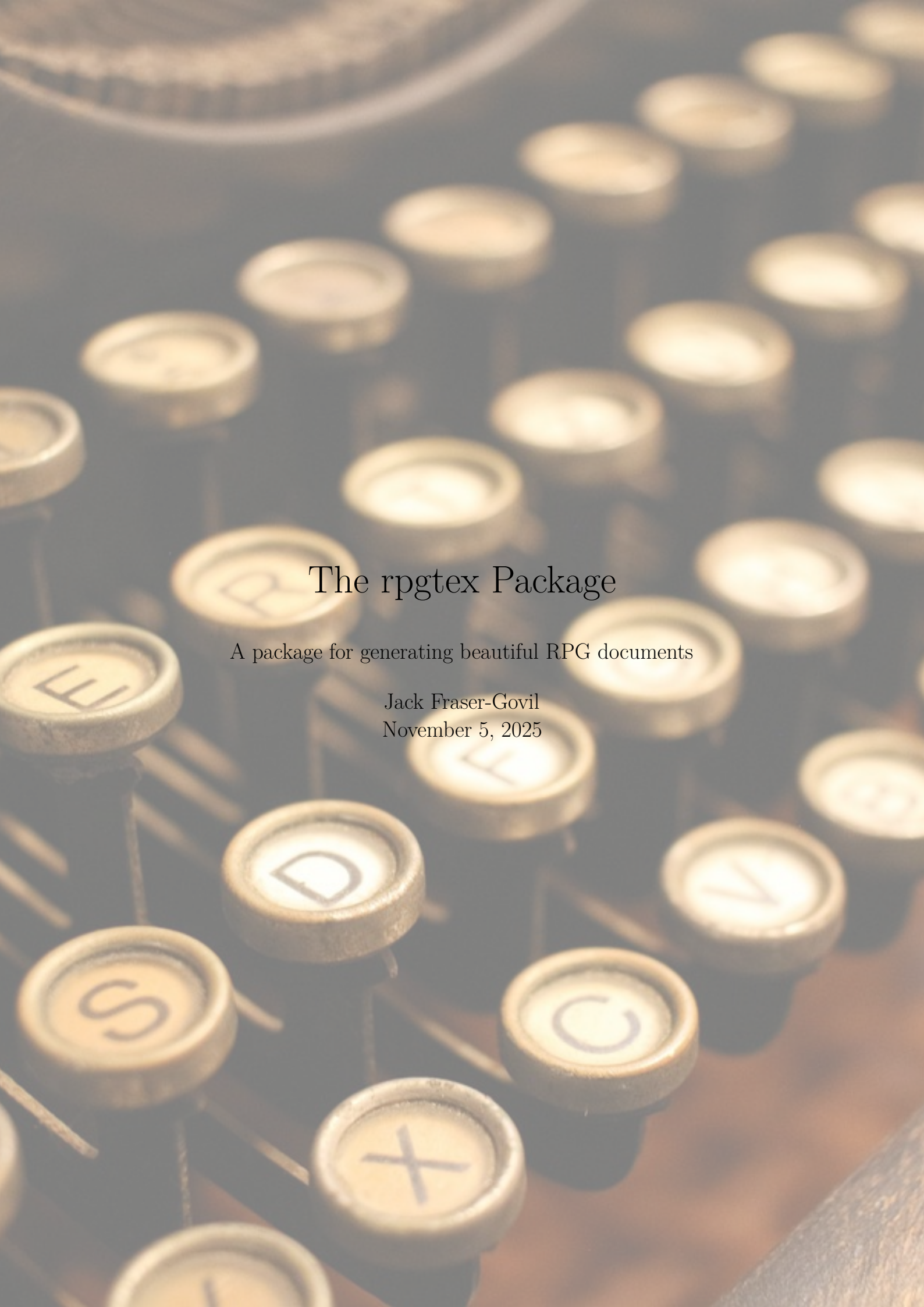

The rpgtex Package

A package for generating beautiful RPG documents

Jack Fraser-Govil
November 5, 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 ``''. 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 `\inputcore/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 **??**). Based on the standard book class, this is designed for larger RPG documents.
2. `rpghandout` (page **??**). Based on the article class, this is designed for shorter documents
3. `rpgcard` (page **??**). A small-document class designed for creating modular `handout' cards for items, spells or abilities.

Compiling rpgtex Documents

`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
```

So long as `rpgtex` is on the user's latex path, and the package properly configured (page 2) no further compilation steps are required. However, for ease of use, we provide the `rpglatex` compiler as part of the `rpgtex` distribution.

The `rpglatex` 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` {m} Compiles latex documents using either xelatex or luatex

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

`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 `bg=full` interface (page 6) to inject code into the latex document, setting the `bg=print` mode and suppressing the background output.	`-p, -print`

Chapter 2: Options & Variables

`rpgtex` defines many dozens to hundreds of variables, most with the (expl3) syntax `\__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.

Package Options

Whether the package is invoked directly, or through a class users have the option to pass options to it which change the behavior of the resulting document:

```
\documentclass[<options>]rpgbook/rpghandout
\usepackage[<options>]rpgtex
```

The options are either in the form of key-value pairs which set internal values, or flags which activate behavior when present. The available options are:

`bg` Controls the presence of the `background paper' and footer decorations.

```
bg = <full / print / none>
```

The value passed must be one of the three options (else an error is thrown). The most obvious effect of these three options is to change whether the `background paper' set by `\RpgSetPaper` (page 13) appears in the document, or the footer decorations set by `\RpgSetFooterDecoration` (page 13) .

Command	Paper	Footer Decorations
`full`	✓	✓
`print`	✗	✓
`none`	✗	✗

This flag also changes the behavior of the rpgcard class (page **??**), and other environments may similarly change their colours or layouts in response to the values passed to this command. Internally these commands are responding specifically to the presence of the paper or the footer flags.

`justified` A flag which, if present, activates justified-text mode. Otherwise, defaults to left-aligned, `ragged right'.

`nomultitoc` A flag which, if present, disabled the multi-column table of contents option

`size` Sets the font size equal to the value, if allowed by the parent class.

```
size = <font-size>pt
```

The allowed values depend on the class being used: the book class (on which rpgbook is based) for example, only accepts values in {10pt,11pt,12pt}. Other classes may accept different values.

`theme` The initial value passed to `\RpgSetTheme` (page 13) when package initialization is finished. The default value is `default`, activating the Default Theme (page **??**).

`themepath` Calls `\RpgSetThemePath` (page 13) , a directory holding multiple theme files used for auto-theme searches if a direct path not given Default value is `\RpgPackagePath/themes`, with the assumption that the theme `name' is stored in `themes/name/name.cfg`.

`columns` Sets the number of columns the document is has

```
columns = <1 / 2>
```

Internally this calls either onecolumn or twocolumn. More advanced column-settings would require the user manually using `multicols`.

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 13) updates the value of `themecolor`, as well as the three `co-varying' colors. Other colors are modified simply using the xcolors interface:

```
\colorlet{narrationcolor}{html}{FFFFF}
```

Command Line Interface

By default, LaTeX does 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.

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

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 4) performs this insertion by default for several predefined variables:

```
rpgtex document.tex -p
```

aliases

```
xelatex "\def\RpgCMDbg=print \input document.tex"
```

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

Fonts

`rpgtex` allows for a high degree of customisation of the fonts and typefaces used for the elements within a document. Fonts can be changed either by the user directly, or (more commonly) by the theme. This is achieved through the `\RpgSetFont` (page 11) command

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

Why didn't my font change?

By default, `\RpgSetFont` doesn't change the actual fonts: it alters internal saved variables which a designer may then assign to a given element. That is, the font `\RpgFontSection` doesn't `hook in' to anything by itself; it only changes the font because most theme documents also call `\titleformat{section}{\RpgFontSection}...., so the assigned value is utilized at the appropriate moment: if you assigned a font to the section, but (for whatever reason) had changed the `titleformat` command, the section font would not update.

If you find that an element doesn't change font after updating the relevant `RpgFont`, check that it is actually being invoked -- and if not, invoke it manually. Once the command is in place, the font will change as expected.

Font Elements

`rpgtex` provides 28 Font Commands by default (themes may provide more). 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 seciton therefore outlines how we have used these elements in the provided themes, though other designers may use them for different purposes.

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=`, rather than setting it within the titlesec specification (`\titleformat \subection\RpgFontSubsection....`).

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.

Font Element	Components	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 9), `\@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 ??) 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 ??).
\RpgFontTipBody	tip-body-family tip-body-style	The font used for the body of an RpgTip environment (page ??).
\RpgFontSidebarTitle	siderbar-title-family siderbar-title-style	The font used for the title of an RpgSidebar environment (page ??).
\RpgFontSidebarBody	sidebar-body-family sidebar-body-style	The font used for the body of an RpgSidebar environment (page ??).
\RpgFontNarration	narration-family narration-style	The font used for all (since they have no title) of an RpgNarration environment (page ??).
\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.

Defining Fonts

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

Font Example

```
\subsection{The Original Font}
Here is some text

\newfontfamily{\myfont}{Arial}
\RpgSetFont{main-body-family=\myfont,
   subsection-style=\slshape\Huge}

\subsection{The New Font}
And after the change is introduced
```

The Original Font

Here is some text

The New Font

And after the change is introduced

Chapter 3: Commands & Macros

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.

```
\title{A title}
\subtitle{The subtitle} %optional
\cover{path/to/cover} %optional
\author{Dr.  W. Riter} %optional

\begin{document}
  \maketitle
  (...)
```

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.

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

If `\RpgUseCoverPage` (page 10) 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{}`), which draws no image.

`\subtitle {m}` `\@subtitle`

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

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

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.

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

This is a Backend Command. When true, \maketitle attempts to use \@cover and then calls \RpgDrawCover (page 10) . If false, it calls \RpgSimpleTitle (page 10) .

\RpgDrawCover {}

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

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

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.
\partpartname	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 10) 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 10) .

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

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.

Fonts & Decorative Text

\RpgContour {{O{}} m}

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

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

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 6).
The contour does not automatically linebreak, but can be controlled manually with a
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 or) will only apply to the first word in the text. Braced commands *can* work, but will cause a compilation error if a
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.

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

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.

RpgDropCap

```
\raggedright \RpgDropCap{T}{he example:
    this text runs over the first line,
    and then revert back to the normal
    font. It almost works! However,
    because it's wrapped in a text box,
    it goes slightly over the edges -
    and would require manual
    calibration.}
```

THE EXAMPLE: THIS TEXT RUNS OVER THE first line, and then revert back to the normal font. It almost works! However, because it's wrapped in a text box, it goes slightly over the edges - and would require manual calibration.

`\RpgSetFont {m}`

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

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

See page 6 for documentation of the available font families.

The values changed by this command are local, and so persist only within a local group.

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.

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

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.

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

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

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 (to avoid `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 `\RpgSetThemednd`)	Output
`\RpgDice { 1d6-2}`	1 (1d6-2)
`\RpgDice {2D6 + 3*2^2}`	19 (2d6+12)
`\RpgDice {1d16}`	8 (1d16)
`\RpgDice {d8-3}`	1 (d8-3)

Theme Commands

`\RpgLayoutOnly {m}` Executes the contents of the command if `layout` mode is active.

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

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}`.

`\RpgSetFooterDecoration {o m}`

Configures an image to be displayed along the bottom of a page as a `footer scroll'.

```
\RpgSetFooterDecortation[<opts>]{path/to/img}
```

When placed within a footer, (i.e. with fancypage), places the image in a node with parameters:
`\node[inner sep=0pt,anchor=south,nearly opaque] at (current page.south)`
`{\includegraphics[width=\paperwidth]{path/to/img}};`
If the package option `bg=none` has been passed, then the image is suppressed.
The following options modify that code as follows:

reverse adds `xscale=-1` to the node arguments, reversing the image (useful for right/left page differences)

tikz-insert=code inserts the code within the tikz environment after the footer scroll. This is not suppressed with `bg=none` and can be used to place chaptermarks / page numbers more precisely than the standard interface allows.

height=<dimexpr> adds `height=dimexpr` to the includegraphics arguments

keepaspectratio adds `keepaspectratio` to the includegraphics arguments

`\RpgSetPaper {}`

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

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

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.

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

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` (page 13) .

`\RpgSetThemeColor {m}`

Sets the `themecolor`, and simultaneously updates the co-varying colors (page 6).

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

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`

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

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.

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

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 {Ot m}`

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

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

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 14
`\RpgPage [p]{example:current page}`	(page 14)
`\RpgPage [b]{example:current page}`	[page 14]
`\RpgPage [c]{example:current page}`	{page 14}
`\RpgPage [(error)]{example:current page}`	?page 14?

`\RpgPlural {o m m}`

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

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

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 4: Environments

RpgCard

The RpgCard environment is designed to allow a writer to create a small, playing-card sized unit which is useful for handing out to players for game elements such as items, spells or abilities.

The backend of the RpgCard is some of the most complex LaTeX code in the package, but this makes it very powerful. We anticipate that users won't access RpgCard directly, but instead a wrapper such as RpgSpell (page **??**) -- however these serve only to format the text within the card environment in a specific way -- the Card itself remains flexible.

The main power of the RpgCard is the ability to automatically *cardbreak*, splitting large internal elements across multiple cards.

The Environment

`RpgCard {O{}}` Splits the contents across a number of playing-card sized units

```
\begin{RpgCard}[<opts>]
      <contents>
\end{RpgCard}
```

Creates a card environment with a height and width defined either by `<opts>`, or the global variables. Text is automatically broken if it exceeds this height, creating multiple cards to hold the text.
The `<opts>` can contain any of the valid inputs to `\RpgSetCard` (see below). Options passed to the environment take precedence to the global variables.

Helper Commands

Examples

Simple RpgCard

```
\begin{RpgCard}
\subsubsection{Text goes here}
               It fits nicely into the
                    card, wrapping over
                    lines\footnote{We can
                    also have a single
                    footnote}.
\end{RpgCard}
```

Text goes here
It fits nicely into the card, wrapping over lines*.

*We can also have a single footnote

Large RpgCard

```
\begin{RpgCard}
{And if we add lots of
    text\footnote{Normally RpgCards
    stack side-by-side before breaking
    over lines -- here they've not got
    enough room, so it's vertically
    stacked.}}

\blindtext
\end{RpgCard}
```

And if we add lots of text*
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.

*Normally RpgCards stack side-by-side before breaking over lines -- here they've not got enough room, so it's vertically stacked.

RpgMap

The RpgMap environment makes it easy to create nested blocks, useful when needing to enumerate the contents of a map. The RpgMap environment uses a stacked counter system and dynamic labelling.

`RpgMap`
`RpgMap*`
`{o}`

Begins a dynamic-stacked environment for generating headed and labelled lists using the `\RpgArea` object to provide entries.

```
\begin{RpgMap}[<opts>]
   <contents>
\end{RpgMap}
```

The starred version of the command is identical in function, but calls `\section*` instead of `\section` (and so on.), suppressing the map elements from the table of contents.
RpgMap uses the counter `RpgAreaDepth` to track how many Maps have been nested. A higher value of this counter results in `smaller' headings being used, beginning with `\section` and progressing to `\subparagraph`.

The permitted options are:

Option	Effects
`header-offset`	An offset added to RpgAreaDepth when determing the heading size to be used (an offset of 0 uses `\section` for the top level map entires, an offset of 1 uses `\subsection`, and so on).
`title`	If non-empty, places the contents in a section one size larger than `RpgAreaDepth+header-offset` (using `\chapter` for the largest possible size). The title is only rendered at the top-level of the map (if RpgAreaDepth=1), otherwise it is ignored. Default value is blank.
`prefix`	A string which is automatically prefixed to the 'number string' of named `\RpgArea` entries in the map. Default is blank.
`blank-prefix`	A string which is automatically prefixed to the `number string' of unnamed (blank) `\RpgArea` entries in the map. Default is ``Area ''
`ref-prefix`	A string prefixed to all labels created by `\RpgArea`, allowing disambiguation of references. Default is ``Map:''

The variables set by options are persistent throughout the nesting - setting `ref-prefix` in one map will mean the same value persists in all encapsulated maps unless manually overridden. Changes do *not* persist once the nesting is finished.

`\RpgArea {o m}`

Creates a formatted entry within the RpgMap environment. The appearance of the title depends on the map-depth and the current header-offset.

```
\RpgArea[<manual-label>]{<area-name>}
```

Creates a `header' depending on the value of `RpgCounter + headeroffset`:

1. section
2. subsection
3. subsubsection
4. paragraph

If outside these values, uses `subparagraph`. The name of the section is preceeded by the `map counter', which is equal to the index within the current map, appended to the `map counter' of any parent maps. The formatting function `RpgMapLevelName` enables hierarchical labelling, such that the third area inside the second map of the first map would be given the counter value `1b-iii'.
If the manual label is set, this is used as the label for this area; otherwise the automatic labelling is used (see below).

`RpgNestedArea o o m`

A wrapper environment for calling `\RpgArea` and then immediately `\begin{RpgMap}`, creating nested map areas in a single call.

```
\begin{RpgNestedArea}[manual-label][<nested-opts>]{<area name>}
   ...
\end{RpgNestedArea}
```

Note that since the RpgMap environment is invoked after the RpgArea is created, the options passed to `<nested-opts>` do not apply to the parent Area, which uses the Map options of its own parent.

Map Labelling & Referencing

Each RpgArea with a non-empty name automatically labels itself using the syntax `\label<ref-prefix><area-name>`. If a manual label was passed to the RpgArea, this is used instead (without the prefix), even if the RpgArea was not named. This is designed to provide disambiguation, as no automatic checks are performed for name collisions.

It is then possible to call `\ref` on this label[1] and `\pageref` or `\RpgPage` (page 14). However, we provide a more powerful referencing interface:

`\RpgMapRef`
`\RpgMapRef* {m}`

Returns the full name of the referenced area, including the map counter. The starred version returns only the map counter.

[1] Though it won't give you anything interesting -- the returned value will be the cumulative number of RpgAreas in the document at that point

Example RpgMap

```
\begin{RpgMap*}[
 header-offset=1,%start at subsection
 title={Example Map}]

 \RpgArea{The Spooky Mansion}
   There are things here

 \begin{RpgMap}[
   title={ignored-as-nested}
   ]
   \RpgArea{Entrance Hall}
       Where you go to enter
   \RpgArea{Kitchen}
       Yum, food
 \end{RpgMap}

 \begin{RpgNestedArea}[][header-offset=2]
   {The Creepy Grounds}
   We set a header-offset for this
        nested block, so....
   \RpgArea{The gardens}
       Note that this is a paragraph,
           not a subsection
   \begin{RpgNestedArea}{The stables}
       Horses live(d) here.

       \RpgArea{Shire horse}
           A very big boi, which still
               keeps the parent's
               header-offset

       \RpgArea{Shetland pony}
           And a tiny one too!
   \end{RpgNestedArea}
 \end{RpgNestedArea}

 \RpgArea{The Graveyard}
   The additional offset-didn't persist;
        we're back to subsections again,
        and the next element will be a
        subsubsection.

   \begin{RpgMap}[
       blank-prefix={Tomb~}
       ]
       \RpgArea{}
           This is an unnamed area - it
               gets given a slightly
               different name
       \RpgArea[tomb-ref]{}
           I might want to refer to this
               later, even though it is
               unnamed.
   \end{RpgMap}
\end{RpgMap*}
```

Example Map

1: The Spooky Mansion

There are things here

1a: Entrance Hall

Where you go to enter

1b: Kitchen

Yum, food

2: The Creepy Grounds

We set a header-offset for this nested block, so....

2a: The gardens. Note that this is a paragraph, not a subsection

2b: The stables. Horses live(d) here.

2b-i: Shire horse. A very big boi, which still keeps the parent's header-offset

2b-ii: Shetland pony. And a tiny one too!

3: The Graveyard

The additional offset-didn't persist; we're back to subsections again, and the next element will be a subsubsection.

Tomb 3a

This is an unnamed area - it gets given a slightly different name

Tomb 3b

I might want to refer to this later, even though it is unnamed.

```
\RpgMapRef{<label-name>}
```

The provided text is fully integrated with `hyerref`, and so they enable click-jumping to the referenced map area.
Using the example map provided above:

Example	Output
`\RpgMapRef {Map:Shire horse}`	2b-i (Shire horse)
`\RpgMapRef *{Map:Shire horse}`	2b-i
`\RpgMapRef *{tomb-ref}`	2b

`\RpgMapRefPage`
`\RpgMapRefPage*`
`{m}`

Appends the page number of the referenced map area to a `\RpgMapRef(*)` command

```
\RpgMapRefPage{<label-name>}
```

Using the example map provided above:

Example	Output
`\RpgMapRefPage {Map:Shire horse}`	2b-i (Shire horse, page 18)
`\RpgMapRefPage *{Map:Shire horse}`	2b-i (page 18)
`\RpgMapRefPage *{tomb-ref}`	2b (page 19)

`\RpgShowMapRefs`

If called, all subsequent `\RpgArea`s will print out a sub-heading listing their macro name. It is not unheard of for a writer to lose track of the labelling name conventions - especially those which are autogenerated. This provides a useful debugging tool for those who don't want to go digging into the aux files.

Example RpgMap with labelling.

```
\RpgShowMapRefs{}
\begin{RpgMap*}[
   header-offset=1,
   title={Example Map}
]

\RpgArea{The Spooky Mansion}
   (\dots)%(skip for example!)
   \RpgArea{The Graveyard}
      (\dots)%(skip for example!)
      \begin{RpgMap}[
            blank-prefix={Tomb~}
         ]
         \RpgArea{}
         Unlabelled area.
         \RpgArea[tomb-ref]{}
            (\dots)%(skip for
                example!)
      \end{RpgMap}
\end{RpgMap*}
```

Example Map

1: The Spooky Mansion

(labelled as `Map:The Spooky Mansion')
(...)

2: The Graveyard

(labelled as `Map:The Graveyard')
(...)

Tomb 2a

Unlabelled area.

Tomb 2b

(labelled as `tomb-ref')
(...)

RpgTable

`RpgTable`
`{o m}`

Begins an environment for creating visually appealing and consistent tables.

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

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 font `\RpgFontTableTitle`.
2. **Auto-headings.** The first row of the tabular environment is automatically rendered in the font `\RpgFontTableHeader`, allowing for trivial header labels.
3. **Font Integration.** The main body of the table is rendered in `\RpgFontTableBody`.
4. **Auto-colouring.** The rows alternate between being transparent and being set to the `tablecolor` variable (page 6). 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:

Standard RpgTable

```
\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:

Header-Suppressed RpgTable

```
\begin{RpgTable}[title={Test
    Table},noheader]{XlX}
Plain & Header & Text
\\
Now there's no & difference & between the
    header and the main
\\
body
\end{RpgTable}
```

Test Table

Plain	Header	Text
Now there's no	difference	between the header and the main
body		

Text Boxes

`rpgtex` defines three `colorbox' environments, which inherit from tcolorbox . These provide a consistent way for a writer to highlight and differentiate blocks of text on the page.

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.

All of the boxes inherit the standard `tcb' style interface, and so tcolorbox options may be passed by the user to control their appearance.

`RpgNarration {o}`

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

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

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.

RpgNarration

```
\begin{RpgNarration}[color=blue!30!white]
 This is text that you would read out
     loud to players, describing a
     scene. It will always be blue,
     even if the theme says otherwise
     -- because the optional argument
     takes priority.
\end{RpgNarration}
```

This is text that you would read out loud to players, describing a scene. It will always be blue, even if the theme says otherwise -- because the optional argument takes priority.

`RpgSidebar {o m}`

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

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

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.

RpgSidebar

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

A Sidebar

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

RpgTip {o m}

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

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

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.

RpgTip

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

A Tip

This is some helpful - but not vital - text.

Index