The rpgtex Package

A package for generating beautiful RPG documents

Jack Fraser-Govil
November 16, 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

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:

bash
```
mkdir -p "$(kpsewhich -var-value TEXMFHOME)/tex/latex/rpgtex" %make sure the directory
    exists
git clone https://github.com/DrFraserGovil/rpgtex.git "$(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:

bash
```
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:

bash
```
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:

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

Because `rpgtex` comes packaged with some resources (namely some fonts) for which TEX does not support relative-pathfinding, it is necessary to provide `rpgtex` with a value for the `\RpgPackagePath`.

`\RpgPackagePath` A variable holding the absolute path to the root directory of the rpgtex install location. We provide a number of different ways to set this variable:

1. The provided `rpglatex` compiler automatically inserts a definition based on a `kpsewhich`, using the Command Line Interface (page 4).
2. If the variable does not exist, the `rpgtex` engine itself tries the following
   (a) Searches for the file `core/rpg-config.cfg`, which (hopefully) contains a definition[a], and `\inserts` it.
   (b) If no file is found, `rpgtex` used the `\write18` utility to dispatch its own calls to `\kpsewhich` and build a config file in the expected location. This requires the compiler be run with `-shell-escape` active
   (c) If this doesn't work, the compiler exits, and suggests that the user manually creates the config file:

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

The extra configuration steps need only be run once -- once the config file exists, the package will be able to configure itself.

[a] In the form of `\edef\RpgPackagePath`{path/to/package}

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.

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}
```

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 25). Based on the standard book class, this is designed for larger RPG documents.
2. `rpghandout` (page 26). Based on the article class, this is designed for shorter documents
3. `rpgcard` (page 27). 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 v) 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 4) to inject code into the latex document, setting the `bg=print` mode and suppressing the background output.	`-p, -print`

PART I
rpgtex For Users

Chapter 1: 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 33) appears in the document, or the footer decorations set by `\RpgSetFooterDecoration` (page 33) .

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

This flag also changes the behavior of the rpgcard class (page 27), 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.

`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`.

`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

`oneside` Forces the document into oneside mode. Disables alternating footers but allows consistent asymmetric margins (such as in this document, where the left margin is larger to accommodate macro names)

`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 (see the class section, beginning on page 25).

`theme` The initial value passed to `\RpgSetTheme` (page 9) when package initialization is finished. The default value is `default`, activating the Default Theme (page 28).

themepath Calls \RpgSetThemePath (page 9) , 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.

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 34) 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}
```

Section Names

Since many themes customise their appearance beyond the simple use of marks (such as chaptermark), we have provided direct access to the current name of the chapter, section, subsection and subsection:

Layout Only

These commands rely on redefining common commands, and therefore are only loaded when in layout mode (i.e. by rpgbook).

\RpgGetName{m} Returns the name given to the last section matching the input command

```
\RpgGetNamechapter/section/subsection/subsubsection
```

If the input value is the name of a vaild section, returns the previous name given to that kind of section. These names can also be `spoofed' by manually calling a \mark (such as \chaptermark etc.), and we also provide the \RpgFakeChapter (page 10) command.

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

Section Names

```
We are in Chapter ‘\RpgGetName{chapter}’,
    section ‘\RpgGetName{section}’.

\subsubsection{An example subsubsection}

Now I am in ‘\RpgGetName{subsubsection}’

\chaptermark{Spoofed Chapter}

I believe I am in a \RpgGetName{chapter},
    though no chapter was actually
    called.

\chaptermark{Options \& Variables} %%put
    it back to the real value!
```

We are in Chapter `Options & Variables', section `Section Names'.

An example subsubsection

Now I am in `An example subsubsection'
I believe I am in a Spoofed Chapter, though no chapter was actually called.

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 vi) 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 34) command

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 7), `\@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 ??).
\RpgFontStatBlockTitle	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.
\RpgFontStatBlockSection	stat-block-section-family stat-block-section-style	The font used for sections within a `statblock' environment (should one be defined).
\RpgFontStatBlockBody	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 2: Commands & Macros

Title & Part Pages

`\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 **??**) 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.

`\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
  (...)
```

`\maketitle` has been completely defined, so commands such as `titlepage` won't work as expected. Instead, the appearance of the title page (or title header) are set by the theme, or user calls to either `\RpgSetCover` (page **??**) or `\RpgSetSimpleTitle` (page **??**) . Which of the two title `modes' is active is controlled by `\RpgSetCoverMode` (page **??**) .
If `cover-mode' has been set, then the image stored in `\@cover` (if there is one) is automatically used as a full-page background image. This is independent of the drawing commands, and occurs before that function is called -- all subsequent drawing occurs over the top of the cover image.

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

The `drawing command' is a control sequence set by `\RpgSetPartPage` (page 33) , which defines a series of tikz functions to place the part title according to the theme specifications.

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

Saves a string to the variable `\@subtitle`. Themes may use this when defining their `\RpgSetCover` 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{}`).

Fonts & Decorative Text

`\emph {m}` `\key {m}`

Uses the `RpgFontEmphasis` font to emphasise text.

```
\emph{text}
\key{text}
```

The `\emph` command is usually a `context aware' emphasis command: equal to `\textit` normally, `\textbf` when the surrounding text is italics etc.
However, for RPGs, it is convenient to be able to identify ***keywords*** in a consistent fashion. The `\emph` command has therefore been redefined to use the `RpgFontEmphasis` font which can be configured to give a desired `keyword formatting'.
The command `\key` has also been provided, as a direct alias for `\emph`.

`\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 3).
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.}
```

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.

Theme Commands

`\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 9) .

`\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

`\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 the control sequence set by `\RpgDiceFormat` (page 34) to perform the output formatting (after performing the regular expression parsing), allowing designers to customise their dice formatting.

`\RpgFakeChapter {m}`

Mimics creating a new chapter with `\chapter` (including adding in to the table of contents) without inserting a `chapter heading'

```
\RpgFakeChapter{fake-name}
```

The value of `fake-name` is passed to the table of contents as a `true' chapter, and an update to `\chaptermark` updates the Section Names (page 3), and thus the footer appearance.

`\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 {O{t} 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 10
`\RpgPage [p]{example:current page}`	(page 10)
`\RpgPage [b]{example:current page}`	[page 10]
`\RpgPage [c]{example:current page}`	{page 10}
`\RpgPage [(error)]{example:current page}`	?page 10?

`\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 3: Environments

RpgCards

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.

We anticipate that users won't access RpgCard directly, but will instead access it through wrappers which utilise the RpgSwitchEnv protocol (page 40) to allow items to be typeset `normally', and in card-mode.

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

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

```
\begin{RpgCard}[<key-value-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 available options are:

Key	Values	Effect
`width`	dimexpr	The outer width of the RPG card
`hmargin`	dimexpr	The horizontal margin between the card boundary and the inner text
`height`	dimexpr	The outer height of the RPG card
`vmargin`	dimexpr	The vertical margin between the card boundary and the inner text
`cardsep`	dimexpr	The horizontal space inserted after a card is finished
`color`	color specifier	The background color of the image. Replaces *tcb/colback*
`opacity`	[0-1]	The opacity of the background of the image. Replaces *tcb/opacityback*
`under-img`	img/path	An image to be used as the background of the card. Clipped to the card background so no overspill occurs.
`underlay`	img/path	An alias for under-img, which replaces *tcb/underlay* to prevent premature tikz expansion.

The formatting of RpgCard otherwise follows that of a tcolorbox , and all other tcb options can be passed in as normal. However, since the title-frame causes issues with the height calculations, RpgCards cannot use the tcb title interface: any attempt to set a title will fail. We provide two aliases, `color` and `opacity`, to make setting the usual values, `colback` and `opacityback`, less verbose and obtuse.
Options passed to the environment take precedence to the global variables.
Note that the RpgCard also temporarily redefines the `\footnote` command (see `\footnote` below).

`\cardbreak {}` Analogous to `\pagebreak`, forces a card to break at the specified location.

```
<before-text>
\cardbreak
<after-text>
```

The command inserts an infinite, but hidden, vertical space penalty, causing the card to break at its location, as if it had been `filled up'.
`\cardbreak` is any empty function outside of the `RpgCard` environment, so it will have no effect on RpgCardSwitch environments.

`\footnote {m}` A redefinition of the standard footnote to account for the quirks of an RpgCard

```
<before-text>\footnote{foottext}<after-text>
```

Due to the multiple processing passes required to ensure that the RpgCard contents fit inside the relevant space, and since footnotes must fit within that space, we had a great deal of difficulty getting footnotes to work as

Simple RpgCard

```
\begin{tikzpicture}%wrap in tikz so we can
    draw dimensions

  \node[anchor=south west,inner
      sep=0pt,outer sep=0pt] at (0,0)
  {
  \begin{RpgCard}[hmargin=0.5cm,vmargin=0.5cm]

    \subsubsection{Text goes here}
      It fits nicely into the card,
          wrapping over lines\footnote{We
          can also have a single footnote}.
  \end{RpgCard}\ignorespaces
  };
  %% Draw the dimenions on top
  %height and vmargin
  \draw[red,<->] (0.7,0)-- node
      [fill=white, right, rotate=90,
      anchor=north]{height}++(0,8.8cm);
  \draw[red,<->] (1.4,0)-- node
      [right]{vmargin}++(0,0.5cm);
  \draw[red,<->] (1.4,8.3)-- node
      [right]{vmargin}++(0,0.5cm);
  %width and hmargin
  \draw[ForestGreen,<->] (0.5,2)--
      node[below]{width}++(4.35,0);
  \draw[ForestGreen,<->] (0,2.4)--
      node[below, rotate=90,
      anchor=east]{hmargin}++(0.5,0);
  \draw[ForestGreen,<->] (4.85,2.4)--
      node[below, rotate=90,
      anchor=east]{hmargin}++(0.5,0);
\end{tikzpicture}
```

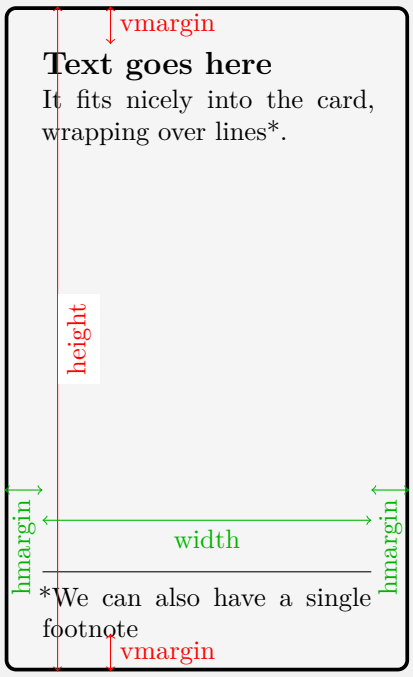

Figure 3.1: An example of a simple RpgCard with the dimensions overlayed

expected (despite them nominally functioning in a tcolorbox).
The compromise is that **only a single footnote can appear in a card**, and the standard `\footnote` command is redefined for the duration of the environment. If multiple footnotes are present, only the final one (in order of execution) will appear. The footnote appears at the bottom of the final card, separated from the main text by a `\hrule`.
An RpgCard-footnote does not increment the global footnote counter.

`\RpgCardSet {m}` Sets the card parameters as global values for all subsequent cards

```
\RpgCardSet{<key/values>}
```

The `<opts>` can contain any of the valid inputs to the `RpgCard` environment: this acts to set them as global default values, but any locally set values will override them.

`\RpgCard!Reset {}` Resets changes made to card parameters back to the defalt value.

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}
```

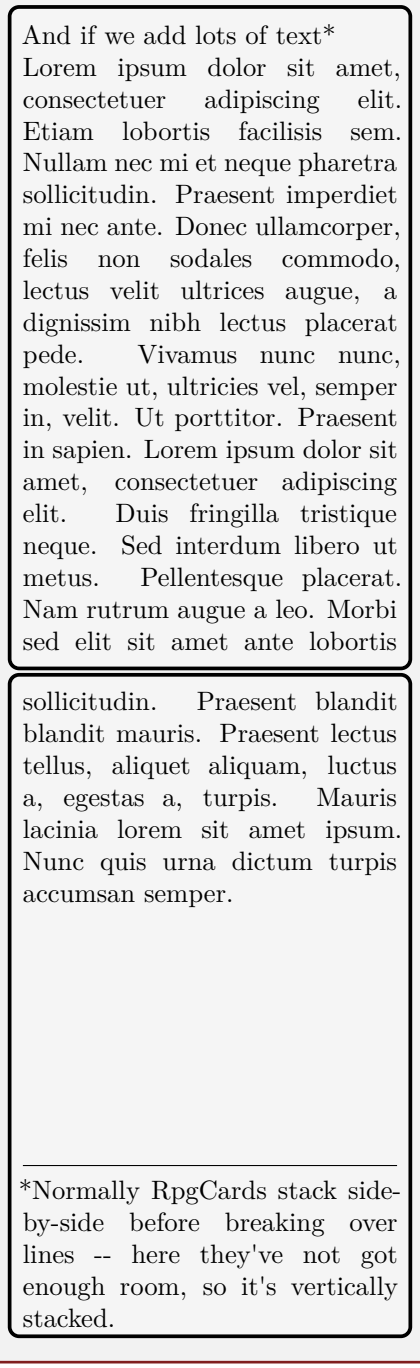

Figure 3.2: An example of an RpgCard with the automatic cardbreaking.

RpgMap

The RpgMap environment is similar to the standard itemize/ description/ enumerate environments, providing a structured way to list elements in an arbitrarily nested fashion. The main difference from the standard list environments are that `RpgMap` (and `\area`, the analogue to `\item`) provide a more elaborate labelling system, useful when needing to enumerate the contents of a map.

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

Begins a nestable map-list environment and enables the `\area` command for adding entries to the map.

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

The starred version of the command is identical in function, with the exception that `\area` 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.

`\area` Adds an area entry into the current RpgMap. The appearance of the entry depends on the current map depth.

`area`

`{O{}O{}}`

```
\area[area-name][<manual-label>]

Or

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

Can be called either as a command (`\area`), or as an environment (`\begin{area}`), with slightly differing behaviours.

As a Command

Design Intent

When called as a command, `\area` is designed to act similarly to an `\item` call inside a `description` environment, providing a familiar interface.

Inserts a `header block' with the following properties:

1. Block is wrapped in a `\chapter`,`\section`, `\subsection` (etc.) depending on the number of nested `RpgMap` environments and the `header-offset`.
2. The header text is one of the following:
   (a) `<prefix> <counter>. <area-name>` if the first optional argument is a non-empty string.
   (b) `<blank-prefix> <counter>` if an `area-name` was not provided.

   `prefix` and `blank-prefix` are arguments passed to the parent RpgMap.
3. The heading is automatically labelled using the following syntax:
   (a) If a `manual-label` argument was provided, this is used as the label; `\label{manual-label}`.

RpgMap with manual nesting

```
\begin{RpgMap}
    \area[Test Zone]
    A description
    \begin{RpgMap}
        \area[Sub Zone]
        A sub-description
        \begin{RpgMap}
            \area[Arbitrary nested zone]
            (we can keep nesting)
        \end{RpgMap}
    \end{RpgMap}
    \area A nameless zone
\end{RpgMap}
```

1: Test Zone

A description

1a: Sub Zone

A sub-description

1a-i: Arbitrary nested zone. (we can keep nesting)

Area 2

A nameless zone

Figure 3.3: An example of RpgMap with area-nesting

(b) If none was provided, and `area-name` is non-empty, then the label is set as `\label{<ref-prefix><area-name>}`. `ref-prefix` is a property of the parent RpgMap, with the default value being `Map:`.

4. Text following the `\area` is rendered as normal text with no modifications.

As an Environment

Fig. 3.3 shows that is possible to nest maps within an `\area` by first adding an area, and then creating a map. We also provide an interface to do this automatically, if `area` is called as an environment.
The syntax and arguments remain identical to the command version, but the body of an `area` environment acts as if it was wrapped in a nested `RpgMap` call.
The following blocks of code are therefore identical in functionality:

```
\begin{area}[name][label]
 <map-contents>
\end{area}
```

```
\area[name][label]
\begin{RpgMap}
 <map-contents>
\end{RpgMap}
```

This is demonstrated in Fig. 3.4.
It is not possible to pass additional options to the internal `RpgMap` call when using the area environment; if the user wishes to vary the commands at different depths, they must use an explicit `RpgMap` environment.

RpgMap with manual nesting

```
\begin{RpgMap}
    \begin{area}[Test Zone]
    A description
        \begin{area}[Sub Zone]
            A sub-description
            \area[Arbitrary nested zone]
                (we can keep nesting)
        \end{area}
    \end{area}
    \area[][nameless-ref] A nameless zone
\end{RpgMap}
```

1: Test Zone

A description

1a: Sub Zone

A sub-description

1a-i: Arbitrary nested zone. (we can keep nesting)

Area 2

A nameless zone

Figure 3.4: An example of RpgMap with area-nesting

Map Labelling & Referencing

Each `area` with a non-empty name automatically labels itself using the syntax `\label{<ref-prefix><area-name>}`. If a manual label was passed to the area, this is used instead (without the prefix), even if the area 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 10). However, we provide a more powerful referencing interface:

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

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

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

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

Example	Output
`\RpgMapRef {Map:Test Zone}`	1 (Test Zone)
`\RpgMapRef *{Map:Sub Zone}`	1a
`\RpgMapRef {nameless-ref}`	2 (Area 2)

`\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
`\RpgMapRef {Map:Test Zone}`	1 (Test Zone)
`\RpgMapRef *{Map:Sub Zone}`	1a
`\RpgMapRef {nameless-ref}`	2 (Area 2)

`\RpgMapShowRefs`

If called, all subsequent `\area`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.

```
\RpgMapShowRefs{}
 \begin{RpgMap}
  \begin{area}[Test Zone]
  A description
      \begin{area}[Sub Zone]
          A sub-description
          \area[Arbitrary nested zone]
              (we can keep nesting)
      \end{area}
  \end{area}
  \area[][nameless-ref] A nameless zone
\end{RpgMap}
```

1: Test Zone

(labelled as `Map:Test Zone')
A description

1a: Sub Zone

(labelled as `Map:Sub Zone')
A sub-description

1a-i: Arbitrary nested zone. *(labelled as `Map:Arbitrary nested zone')*
(we can keep nesting)

Area 2

(labelled as `nameless-ref')
A nameless zone

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

RpgSecret

The RpgSecret environment uses the RpgSwitchEnv system (page 36) to conditionally hide information; this allows a Game Master[2] to use the same text for their own notes as they would for players, but wall off some parts of it as 'not for their eyes'.

`RpgSecret {O{}m}`

A switchable environment (page 36) associated with the switch `ShowSecrets`. When this key is true, the body of the environment is rendered; otherwise it is hidden.

```
\begin{RpgSecret}[<opts>]{<player-text>}
   <GM-text>
\endRpgSecret
```

If the `ShowSecrets` switch has been set to false, then the <player-text> is inserted as plain text with nothing to suggest there's additional text being withheld.
If the switch is true, then the GM text is rendered in a highlighted box, informing the reader that this information is considered secret. If player text was present, this is also shown, and highlighted as the 'publicly available' knowledge.
The optional arguments take the form of key value pairs:

Key	Values	Effect
`title-secret`	string	Sets the name above the `secret' text (if shown). Default value `Secret Text'
`title-public`	string	Sets the name above the `player' text (if secret text is shown, and player text present). Default value is `Normal Text'
`color`	color specifier	Sets the outline color of the `secret box' rendered when the key is set to true.

`\RpgSecretVisible {m}`

An alias for `\RpgSwitch{ShowSecrets}{input}`

RpgSecrets (in hide-mode)

```
%%hide the juicy gossip
\RpgSwitch{ShowSecrets}{false}

\lorumIpsumOne%Snippets of standard lorum
    ipsum
\begin{RpgSecret}{}
  The players don't know it yet, but all
      this pseudo-latin text is a hint
      that devils are involved!
\end{RpgSecret}
 \lorumIpsumTwo{}
 \begin{RpgSecret}{
   Logos kykloforia, thelxis kai ethos,
       tyrannis kai epilysis. Aei thera
       plaka, kyrios meion egestas,
       anagkas proin eira kai thera.
   }

   This bit of text is noteworthy - it's
       been added in after the fact
\end{RpgSecret}
 \lorumIpsumThree
```

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Logos kykloforia, thelxis kai ethos, tyrannis kai epilysis. Aei thera plaka, kyrios meion egestas, anagkas proin eira kai thera. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc.

[2] Or whatever your system calls this role!

RpgSecrets (in show-mode)

```
%%exactly the same as above -- but reveal
    the secrets!
\RpgSecretVisible{true}
\lorumIpsumOne

\begin{RpgSecret}{}
   The players don't know it yet, but all
       this pseudo-latin text is a hint
       that devils are involved!
\end{RpgSecret}

 \lorumIpsumTwo{}

 \begin{RpgSecret}{Logos kykloforia,
     thelxis kai ethos, tyrannis kai
     epilysis. Aei thera plaka, kyrios
     meion egestas, anagkas proin eira
     kai thera.}

   This bit of text is noteworthy - it's
       been added in after the fact
\end{RpgSecret}

 \lorumIpsumThree{}
```

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris.

Secret text
The players don't know it yet, but all this pseudo-latin text is a hint that devils are involved!

Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

Normal text	Secret text
Logos kykloforia, thelxis kai ethos, tyrannis kai epilysis. Aei thera plaka, kyrios meion egestas, anagkas proin eira kai thera.	This bit of text is noteworthy - it's been added in after the fact

Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc.

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 3). 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).**

simple If flag is present, renders using a simple `tabular` (ignoring breakable flag). Useful if not using X-columns, and not wanting to specify a length of the table.

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.

RpgItem, RpgSpell, RpgFeat (etc.)

The prior documented environments have almost entirely been *rules agnostic*. That is, they have primarily been typesetting aids which could equally be used for notes on a whimsical, rules-lite game about squirrels in a forest, or a crunchy, table-heavy futuristic scifi game -- all the theme did was change the appearance of the individual elements.

Some elements, however, are strongly tied not only to an aesthetic theme, but also to the rules of an individual system: a statblock for a Pathfinder monster not only looks wholly different to a hostile Traveller alien, but has different components and arguments.

To this end, we have provided a powerful and flexible interface to allow a designer to create and customise environments. This is the subject of Chapter 11, and makes it somewhat difficult to robustly document the features - as they change behaviour depending on the theme that is being used! When using these functions, always consult the theme-specific documentation.

Advanced Usage

The functions listed below are the `basic interface' for the FeatureForge system; designed for those who are happy to use the format that a theme designer has set up for them.
Those wishing to have a more fine-grained control over the appearance of the objects should consult the full documentation on page (page 36).

By default, all of these environments have the same behaviour (except where noted otherwise)

Feature Forge Example

```
%% Test object does not exist before this
    moment.
%%See FeatureForge for documentation
\RpgMakeFeature{TestObject}{TestCard}{test}
\TestObjectAddProperty{test-key}
    {\testKeyValue} {default}


%card mode defaults to false, so get plain
    text output:
\begin{TestObject}{An Example}
   This is the body text, I can see that my
       key is ‘\getkey{test-key}’
\end{TestObject}

%%Then activate the card mode (and change
    the test-key value)
\TestObjectShowCard{true}
\begin{TestObject} {An Example}
    [test-key=alacrity]
   This is the body text, I can see that my
       key is ‘\getkey{test-key}’
\end{TestObject}
```

An Example

This is the body text, I can see that my key is `default'

An Example

This is the body text, I can see that my key is `alacrity'

Rpg[X]
RpgItem
RpgSpell
RpgFeat
{O{} m O{}}

A card-switching environment which renders the text either in normal text, or as an `RpgCard` (page 11)

```
\begin{Rpg[X]}[<card-opts>]{<name>}[<cmd-opts>] % replace [X] with ‘Item/Feat/Spell’
...
```

All environments following this pattern are based off the Switched-Environment system (page 36), allowing them to render the same text either in normal (`text') mode, or in card-mode, depending on the value set by the user. The optional `card-opts` parameter is passed as the optional parameter to the RpgCard environment, allowing the user to customise the card's appearance.
The `name` parameter defines the object and will (usually) appear in a title/subtitle/subsubtitle.
The main flexibility of the system is delivered through the `<cmd-opts>` argument, which accepts key/value pairs. The keys are determined by the theme (there are none by default), and define rules-based interactions (a D&D theme might add an `armour-class' key, whilst a Forged in the Dark theme might add `tier'). Consult the theme-documentation for the present definition of the keys.

\Rpg[X]ShowCard
{m}

Sets the card mode for all Rpg[X] environments.

```
\Rpg[X]ShowCard{true/false}
```

If set to true, the contents of the specified environment will be rendered in card-mode, or text-mode if false.

\getkey
{m}

Inserts the value passed to `key` in the `cmd-opts` args of the current [X]-environment

```
\begin{Rpg[X]}{Test Object}[key1=value,key2=value]
{
  The value of key1 is \getkey{key1}
}
```

Recovers the value passed to the key (after any post-processing performed by the environment). This allows the user to access the key-values, as well as allowing them to be inserted into the expected formatting, without having to duplicate the information.

See page 39 for more details.

PART II
Classes & Themes

Chapter 4: rpgbook Class

The rpgbook class is designed for writing long form documents such as rulebooks and sourcebooks for RPGs - cases where you need to be able to organise things into parts and chapters!

Features

Inherited Class

The rpgbook class inherits from the extbook class . This is an extension to the basic `book` class to allow more fontsizes to be accepted. Otherwise it behaves near-identically to the standard book class.

The full list of sizes which extbook can accept (and thus allowed inputs for the `size` option (page 2)) is ``eight, nine, ten, eleven, twelve, fourteen, seventeen and twenty points''.

Special Commands

rpgbook inherits the following notable commands from the book class, which are not available in other classes:

`\frontmatter {}` Activates `preliminary formatting' for the introductory sections The initial formatting mimics formatting found in forewords and other miscellaneous text before the `main body' begins:

1. Chapters are un-numbered (as if called with `\chapter*`), despite being entered into the table of contents.
2. Page numbers are changed to lowercase roman (i, ii, etc.)

`\mainmatter {}` Disables the special formatting.The `main matter' is the bulk of the text, and the expected formatting the user requests.
When mainmatter is called, the page number is reset back to 1 -- this may cause the PDF page counter to differ from those which appear in the footer. The values reported by `\pageref` and `\RpgPage` refer to the `footer page numbers', not the PDF page numebrs.

`\backmatter {}` Activates `appendix formatting' Appendix formatting does not change the page numbering, but disables the chapter numbering as in the `frontmatter`

Options

The rpgbook interacts with all of the options detailed on page 2. Note that there is no `forwarding' to the underlying class and that there is a slightly different syntax for, i.e., setting the global font size.

Geometry

The default geometry for an rpgbook is:

Element	Size
Left and right margin	0.65in
Top margin	0.4in
Bottom margin (from main text to page bottom)	0.75in
Bottom margin (from main text to top of footer area)	0.3in
Gap between columns in twocolumn mode	0.25in

Interactions

- RPG books set `\RpgUseCoverPage` (page ??) to true
- The extbook provides the `part` and `chapter`
- The `layout` mode is activated
  1. Unless print mode is active, the page background will use the image set by `\RpgSetPaper` (page 33)
  2. Calling `\RpgSetTheme` clears the page (so that the old theme may complete)
- A table of contents is available and formatted using the ToC-fonts

Chapter 5: rpghandout Class

The rpghandout class is designed for smaller documents where the full structure of a book is unnecessary, such as printouts for players, or individual adventure modules.

Features

Inherited Class

The rpghandout class inherits from the extarticle class . This is an extension to the basic `article` class to allow more fontsizes to be accepted. Otherwise it behaves near-identically to the standard article class.

The full list of sizes which extarticle can accept (and thus allowed inputs for the `size` option (page 2)) is ``eight, nine, ten, eleven, twelve, fourteen, seventeen and twenty points''.

Special Commands

`abstract`
`{O{Abstract}}`
`summary`

Defines a block of text which (if in twocolumn mode) spans both columns, serving as a summary of the document.

```
\begin{abstract}[<abstract-name>]    % (or \begin{summary})
  <abstract-contents>
\end{abstract}
```

We have redefined the abstract environment slightly so that it renders almost identically in both twocolumn and onecolumn mode:
The optional argument determines the `header' text which is printed above the abstract text. This is centered and uses the `\RpgFontAbstractTitle` font, whilst the body text uses `\RpgFontAbstractBody`. The text is placed into a parbox which is 70% the line width.
The `summary` variant is identical, but uses the default header value of `Summary', which might be more familiar to non-technical writers.

Options

The rpghandout interacts with all of the options detailed on page 2. Note that there is no `forwarding' to the underlying class and that there is a slightly different syntax for, i.e., setting the global font size.

Geometry

The default geometry for an rpghandout is:

Element	Size
Left and right margin	0.65in
Top margin	0.4in
Bottom margin (from main text to page bottom)	0.75in
Bottom margin (from main text to top of footer area)	0.3in
Gap between columns in twocolumn mode	0.25in

Interactions

- rpghandout set `\RpgUseCoverPage` (page ??) to false
- The `layout` mode is activated
  1. Unless print mode is active, the page background will use the image set by `\RpgSetPaper` (page 33)
  2. Calling `\RpgSetTheme` clears the page (so that the old theme may complete)
- A table of contents is available and formatted using the ToC-fonts

Chapter 6: rpgcard Class

Chapter 7: `default` Theme

RpgItem: default

The default value of the `RpgItem` (page **??**)

RpgFeat: default

RpgSpell: default

Chapter 8: dnd Theme

RpgItem: dnd Version

The dnd theme defines an override for the generic `RpgItem` (page **??**)

RpgSpell: dnd Version

RpgStatblock: dnd Version

The dnd theme defines a special command which mimics the appearance of a monster statblock - particularly those in the more modern D&D 2024 iteration.

A Monster

Medium aberration, lawful evil

AC 9 (12 with **mage armor**)
HP 16 (3d8+3)
Speed 30 ft., fly 30 ft.
Initiative -1

	STR (12)	DEX (8)	CON (13)	INT (10)	WIS (14)	CHA (15)
mod	+1	-1	+1	+0	+2	+2
save	+3					+4

Senses darkvision 60 ft., passive Perception 12
Languages LaTeX
Challenge 1

A Monster

Medium aberration, lawful evil

AC 9 (12 with **mage armor**)
HP 16 (3d8+3)
Speed 30 ft., fly 30 ft.
Initiative -1

	STR (12)	DEX (8)	CON (13)	INT (10)	WIS (14)	CHA (15)
mod	+1	-1	+1	+0	+2	+2
save	+3					+4

Senses darkvision 60 ft., passive Perception 12
Languages LaTeX
Challenge 1

Chapter 9: scifi Theme

scifi RpgItem

PART 3
rpgtex For Designers

Chapter 10: Designer Commands

The following 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.

The `average user' may safely ignore this section.

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

CoSS Functions

A quirk of some of some of Theme Commands is that they require accessing arguments which were not passed to them. These are ***Control Sequence Setters (CoSS)***: functions which do not execute commands, but instead save them to be executed later.

For example:

```
\RpgSetControlSequence{
This is a (#1)-argument command, but I am using (#2), and even (#3) arguments!
}
```

These CoSS functions do not execute the control sequence, but save it to an intermediary value. The backend of the package looks something like:

```
\NewDocumentCommand{\RpgSetControlSequence}{+m}{
\cs_set:Nn \__rpg_control_sequence:nnn{#1}
}
```

That is, the contents of the CoSS are saved into an expl3 control sequence -- and in this case, one with three arguments (`nnn`). When another internal function comes to execute `__rpg_control_sequence:nnn`, the text will render as:

```
\__rpg_control_sequence:nnn{3}{2}{1}
This is a (3)-argument command, but I am using (2), and even (1) arguments!
```

When working with CoSS functions it is vital to check the documentation to see which arguments are available, as it may not be obvious from the setter's syntax.

Title & Part Pages

`\RpgSetTitleCover {+m}`

Assign the Tikz code for drawing a custom cover page over the top of the `\@cover`-image.

```
\RpgSetTitleCover
{
<custom-tikz-code>
}
```

This is a CoSS Function, with the resulting control sequences being used when `\maketitle` is called (in `cover` mode), allowing the designer to determine where to place the text on the page, and what embellishments accompany it. The stored sequence 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 custom tikz code does not permit any arguments, but the contents of `\@title`, `\@subtitle`, `\@author` and `\@date` are accessible.
If a `\@cover` has been defined, this command is executed after the image is placed, drawing on top of it.

`\RpgSetTitleHeader {+m}`

Assigns the code for typesetting a `header' title - a simple text-only title at the top of the page.

```
\RpgSetTitleHeader
{
<custom-code>
}
```

This is a CoSS Function, with the resulting control sequences being used when `\maketitle` is called in `header` mode, allowing the designer to determine how to structure the text which makes up the `simple' title.
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.

`\RpgSetTitleMode {m}`

If true, `\maketitle` creates a title page to populate, else the title is rendered as an article-like heading.

```
\RpgSetTitleMode{cover/header}
```

When `cover` is set, `\maketitle` attempts to use `\@cover` and then calls the `\RpgSetTitleCover` (page 32) -set tikz code. If false, it uses the value sent to `\RpgSetTitleHeader` (page 32) to render a simple text heading.

`\RpgSetPartPage {+m}`

Assign the Tikz code for drawing a custom part page when activated by `\part` (page 7) .

```
\RpgSetPartPage
{
% #1 = part name
<custom-tikz-code>
}
```

This is a CoSS Function, with the resulting control sequences being used when `\part` is called, allowing the designed to determine where to place the part name on the page, and what embellishments accompany it. The stored sequency 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 command can take one argument (#1), which is equal to the name of the part. The current part counter can be accessed via `\thepart`.
The `\part` command draws a background image (if on is provided), with the contents of this command rendered on top.

Page Appearance

`\RpgSetFooterDecoration {o m}`

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

```
\fancyfoot[LE] % the footer for left-even pages
{
  \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.

`\RpgSetThemeColor{m}`

Sets the `themecolor`, and simultaneously updates the co-varying colors (page 3).

```
\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.

Other

`\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 4 for documentation of the available font families. The values changed by this command are local, and so persist only within a local group.

`\RpgDiceFormat{+m}`

Sets the typesetting of the RpgDice command

```
\RpgDiceFormat
{
% #1 = dice count   #2 = dice size    #3 = added bonus
<custom-code>
}
```

This is a CoSS Function, with the resulting control sequences allowing theme designers to determine how `\RpgDice` is rendered. The default option is: `\RpgDiceFormat{#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.

RpgDice Formatting: The D&D Format

```
Before: \begin{itemize}
 \item \RpgDice{2d8 + 3}
 \item \RpgDice{d8}
\end{itemize}
\ExplSyntaxOn %%Activate expl3 programming
\RpgDiceFormat{ % #1: Dice Number, #2:
    Dice Sides, #3: Modifier
 %%Set dX -> 1dX
 \tl_set:Nn \l__temp_dice{\tl_if_blank:VTF
     {#1}{1}{#1}}
 %%Compute the average result
 \tl_set:Nn \l_tmp_mean_tl { \fp_eval:n {
  floor ( \tl_use:N{\l__temp_dice} * ( #2 +
      1 ) / 2 ) + (#3)
 }}
 %typeset the result
 \l_tmp_mean_tl{}~(\l__temp_dice d#2
 \fp_compare:nNnTF { #3 } { = } { 0
     }{}{#3})
}
\ExplSyntaxOff
After \begin{itemize}
 \item \RpgDice{2d8 + 3}
 \item \RpgDice{d8}
\end{itemize}
```

Before:

- 2d8+3
- d8

After

- 12 (2d8+3)
- 4 (1d8)

`\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}`.

Chapter 11: Feature Forge

In initial drafts of this package, we found ourselves duplicating large swathes of code to produce customizable environments such as RpgItem, RpgSpell and RpgFeature -- these are environments that behave similarly enough to be almost identical in construction, but different enough to warrant their own unique interface.

Rather than forcing designers to write hundreds of lines of code to produce (for example) a slight variant on the RpgSpell environment, we have used the power of expl3 metaprogramming to automatically generate environments as well as configuration hooks and macros with a single line of code.

RpgMakeFeature

`\RpgMakeFeature {m O{} m m}`

Creates a new environment with the specified name. The environment is switchable between card and text mode, and has a number of hooks to customise its appearance and input key values.

```
\RpgMakeFeature{EnvName}[environment-signature]{switch-name}{key-library-name}
```

`\RpgMakeFeature` is a high level meta-function which defines several new functions and environments which persist in the current group (which is most often the global one).

Quick Overview

The following objects are created in the existing group when RpgMakeFeature is called:

`EnvName` The main environment - for example, `RpgSpell`, `RpgItem` etc.

`\[EnvName]ShowCard` A function which toggles the environment between `card-mode` (using `RpgCard` (page 11)) and `text-mode` (the default).

`\[EnvName]CardFormat` A CoSS Function to configure the appearance of the environment whilst in card-mode

`\[EnvName]TextFormat` A CoSS Function configure the appearance of the environment while in text-mode

`\[EnvName]AddProperty` Used to add a key into the key-value pair argument accepted by the main `EnvName` environment.

Creating New Features

This interface makes defining a new feature quick and easy:

1. Call the RpgMakeFeature to initialise all of the functions

   ```
   \RpgMakeFeature{RpgFoo}{FooCards}{foo}
   ```

2. Call `\[EnvName]AddProperty` to create new key-value properties associated with the feature.

   For RpgItem, this might be `weight` and `value`; for RpgSpell it might be `mana-cost` or `range`, and so on.

   ```
   \RpgFooAddProperty{description}{\descriptString}{A basic foo}
   ```

3. Write the formatting functions to determine where the title, the main body text, and the values of the keys are placed, and any formatting that is applied to them.

```
\RpgFooTextFormat{
  \subsection{#1} %header
  \textit{\descriptString}\par %insert the key-defined value here
  #2 % then the body }

\RpgFooCardFormat{
  \begin{center}
     \color{red}
     {\footnotesize \scshape \descriptString}

    {\large\bfseries#1}
  \end{center}
  #2
}
```

Feature Forge Example

```
%card mode defaults to false, so get plain
    text output:
\begin{RpgFoo}{An Example}
This is the body text.

I can sneakily access the key value:
    `\getkey{description}'
\end{RpgFoo}

%%Then activate the card mode (and change
    the test-key value)
\RpgFooShowCard{true}
\begin{RpgFoo}{An Example}[description=An
    advanced foo]
This is the body text.

I can sneakily access the key value:
    `\getkey{description}'
\end{RpgFoo}
```

An Example

A basic foo
This is the body text.
I can sneakily access the key value: `A basic foo'

An advanced foo
An Example

This is the body text.
I can sneakily access the key value: `An advanced foo'

Full Documentation

A more robust explanation of what happens when `\RpgMakeFeature` is called goes as follows:

Step 1: Create Main Environment

`EnvName (varies)`

The main environment created by the FeatureForge system, through a call to `\RpgMakeFeature`. The name of the argument (`EnvName') is the first argument to the constructor function, and serves as the `root' of the names of most of the co-created functions and environments.

```
\begin{EnvName}[card-opts]{<name>}[key-opts]
  (...)
```

The signature (the number and type of arguments) of the environment is set by the `environment-signature` optional argument to RpgMakeFeature, with the default value being `{O{} m O{}}`. There should always be

three arguments, which always have the same meanings, but the specification might force some arguments to be mandatory (i.e. `{O{}m m}`) instead of optional.
The `card-opts` argument is passed to the `RpgCard` (page 11) environment when `card-mode' is active, the `name` argument is (typically) the header or title of the environment, whilst the `key-opts` are passed into the Key Registry system.

Step 2: Text Mode Environment

Defines the text-mode environment and hooks

1. Creates a text-mode environment with the name `__[EnvName]Text`, and a macro which is executed inside that environment, `\__[EnvName]_text_format:nn`.
2. Defines a CoSS Function to set the value of `\__[EnvName]_text_format:nn`

`\[EnvName]TextFormat {+m}` Defines the body of the `EnvName` environment when in text-mode.

```
\[EnvName]TextFormat{
% #1 = name, #2 = body
<custom-code>
}
```

The value of #1 is the name passed as the first mandatory argument of the parent Environment, whilst #2 is the body text of the environment. The designer is able to format these text elements, and also add elements defined by the `key-opts`.

Step 3: Card Mode Environment

Defines the card-mode environment and hooks.

1. Creates a card-mode environment with the name `__[EnvName]Card`, and a macro which is wrapped inside a call to `RpgCard` (page 11) , `\__[EnvName]_card_format:nn`.
2. Defines a CoSS Function to set the value of `\__[EnvName]_card_format:nn`

`\[EnvName]CardFormat {+m}` Defines the body of the `EnvName` environment when in text-mode.

```
\[EnvName]CardFormat{
% #1 = name, #2 = body text
<custom-code>
}
```

This code is excuted from within an `RpgCard` (page 11) environment (which has parameters set by the call to `card-opts`); there is no need for the designer to invoke it themselves.

Step 4: Create Text/Card Switch & Wrapper Function

Defines a new Switch (see `\RpgSwitch` (page 40)), with a name equal to `switch-name`, the second mandatory argument passed to the environment. This does not check if the switch already exists, so it is possible to have multiple environments triggering from the same switch.

This can be used with the default switch interface, but we also provide a wrapper which performs this for the user:

`\[EnvName]ShowCard {m}` Sets the switch associated with the card equal to the value provided.

```
\[EnvName]ShowCard{true/false}    equivalent to    \RpgSwitch{<switch-name>}{true/false}
```

If true, the internal `S:SwitchEnv` (page 36) sets the environment into `card mode' (and hence uses the format set by `\[EnvName]CardFormat`). If false, it enters `text mode' (using the `\[EnvName]TextFormat`-set formatter).
See `\RpgSwitch` (page 40) for more information.

Step 5: Create Parameter Registry

1. Creates an expl3 key list with location {rpg/forge/<key-library-name>}, the final argument passed to the \RpgMakeFeature when the feature is created.

   This key list is used to parse the values of <key-opts>, the final argument to the EnvName argument each time that it is used.

2. Creates a property list , with name [EnvName]_property_list.

3. Defines a wrapper function for adding simple items into the registry

\[EnvName]AddProperty {m m +m}

Adds the specified key in to the key-list. When the key is passed to EnvName, it is saved as a token list .

```
\[EnvName]AddProperty{key-name}{\macro-name}{default-value}
```

The AddProperty command is a wrapper around some basic tl-parsing, calling, adding the following code into the key list at {rpg/forge/<key-library-name>}

```
key-name .tl_set:n = \macro-name,
key-name .initial:n = default-value,
key-name .value_required:n = true,
```

The code also adds the macro into the property list, indexed by the key:

```
\prop_put:cnn{[EnvName]_property_list}{key-name}{\macro-name}
```

Accessing the values directly by macro name is more performant, and therefore the recommended way that designers insert the saved values. However, the property-list allows for a user to access the keys in a more natural fashion, through the \getkey interface.

4. Locally redefines \getkey (page 22):

\getkey {m}

Returns the value passed to the current environment with identifier key-name

```
\getkey{key-name}
```

Inserts the value associated with key-name from the property-list associated with the environment. Although we recommend that designers use the macros that they associated with their keys, it is convenient to allow users to only have to recall one set of associations: the key values they used to define those values in the first place -- at the cost of a minor performance impact.

Advanced Registry Usage

The backend of the Parameter Registry comprises of two expl objects: a key-parser which saves input tokens to provided macros, and a property list which allows you to access those macros using the key-name, after the parsing has completed[a]

Although we have streamlined this interface - making adding a new property into a single-line call rather than the multi-line affair it can often be, this necessarily means losing some of the power that the expl3 interface provides.

For instance, expl3 keys can be set to use *choices*; restricting the input value to one of a predefined set, and some keys may act as flags and not have values associated altogether.

These functions can still be used by a designer by manually adding elements into the underlying objects:

```
\RpgMakeFeature{RpgFoo}{SwitchName} {foo}
                                    directory
\RpgFooAddProperty(...)  %define the normal properties
\ExplSyntaxOn%ensure expl3 mode is active

\keys_define:nn{rpg/forge/foo}
{
 code-property .code =<arbitrary-code>,
 multiple-choice .choice:,
 multiple-choice / a .code:n {code-if-a},
 multiple-choice / b .code:n {code-if-b},
}
```

This would enable the values code-property and multiple-choice to be passed as keys to the RpgFoo environment (and any other environment which was set to use `rpg/forge/foo' as its directory). It would not,

however, automatically add the values into the property list where it can be accessed by `\getkey`.

Sometimes this is not a problem - if the parameter is a flag which changes colors, then there's no reason to expect the user to access this. Otherwise, manually add the desired value into the property list:

```
\keys_define:nn{rpg/forge/foo}
{
  code-property .code =
    {<arbitrary-code>
     \prop_put{RpgFoo_property_list}{code-property} {<value/macro-to-insert>}
  }
(...)
}
```

[a] There's probably a good reason why you can't do this directly from the keys, but for now you need to use this dual-approach!

RpgSwitchEnv

Often it is convenient to be able to toggle between two different environments depending on an external flag. In the context of an RPG this might be for a number of reasons: having a player version and a GM version, or having a screen-readable version versus a printable one.

The overall goal of the RpgSwitchEnv is to reduce the amount of duplication that an author has to do to get the same text in multiple different forms. The forms the basis of the `RpgFeatureForge` system, however it is also used elsewhere; for example the `RpgSecret` (page 17) environment.

Why switch environments?

Whilst it is obviously possible to build an environment which performs the switching for you, we provide a generic interface for switching between *similar environments*.

'Similar' Environments

It is important to note that this system only works for switching between environments which are `similar', insofar as they permit the same number and order of arguments, and interpret their contents similarly.

An itemize and an enumerate are `similar': an itemize and a figure are not.

`RpgSwitchEnv {m o m o m}`

Acts as one of two similar environments based on the value of an input key.

```
\begin{RpgSwitchEnv}{<key>}[opt-1]{env-1}[opt-2]{env-2}
  <contents>
\end{RpgSwitchEnv}
```

The `key` is an input token (a string) which should be in the global *switch-registry* (see below). The value associated with that key determines the behaviour of the Switch-Env:

Value is true The environment acts as `env-1` (with optional arguments `[opt-1]`)

Value is false The environment acts as `env-2[opt-2]`.

Due to the way that token expansion works, it is possible to pass *additional* arguments to this environment:

```
\begin{RpgSwitchEnv}{<key>}[opt-1]{env-1}[opt-2]{env-2}{arg1}{arg2}
```

Formally speaking, `arg1` and `arg2` are a part of the body of the environment; however if both env-1 and env-2 are expecting two arguments, then the token expansion captures them. It is also possible to use a shared optional argument, instead of the unique arguments:

```
\begin{RpgSwitchEnv}{<key>}{env-1}{env-2}[shared-opt]
```

If however, the environments are not similar, and take different numbers of arguments then any excess arguments are inserted into the body of the environment, which can cause unexpected behaviour.
An error is thrown if `key` does not exist in the global registry.

\RpgSwitch {m m}

Change the value of a *switch*, and therefore the behaviour of the associated RpgSwitchEnv

```
\RpgSwitch{<key>}{<value>}
```

Sets the value of the `key` in the *switch-registry* to `value`, which must be a 'bool-ish' text string[a]. If the entry does not exist in the registry, it is created.
After the key is set, all subsequent `RpgSwitchEnv` calls which use that key will have their behaviour altered to match the new key.

[a] That is, either {true,True,1} or {false,False,0}

Example Switching Environment

```
\def\exampleSwitch{
    \begin{RpgSwitchEnv}{test}{enumerate}
        [leftmargin=1cm]{itemize}
      \item item 1
      \item item 2
      \item orangutans
    \end{RpgSwitchEnv}
}
Changing the switch makes the same
    contents appear differently:

\RpgSwitch{test}{true}
\exampleSwitch{}

\RpgSwitch{test}{false}
\exampleSwitch{}

%%Now repeat, but move the optional arg
    to the end as a 'hanging argument'

\def\exampleSwitch{
    \begin{RpgSwitchEnv}{test}{enumerate}
        {itemize}[leftmargin=1cm]
      \item item 1
      \item item 2
      \item orangutans
    \end{RpgSwitchEnv}
}
 Both environments should now be indented:
\RpgSwitch{test}{true}
\exampleSwitch{}

\RpgSwitch{test}{false}
\exampleSwitch{}
```

Changing the switch makes the same contents appear differently:

1. item 1
2. item 2
3. orangutans

- item 1
- item 2
- orangutans

Both environments should now be indented:

1. item 1
2. item 2
3. orangutans

- item 1
- item 2
- orangutans

Index