



The rpgtex Package

A package for generating beautiful RPG documents

Jack Fraser-Govil
December 23, 2025

WELCOME TO THE **RPGTEX** PACKAGE. This **L^AT_EX** package is designed to allow users to flexibly typeset documents associated with Role Playing Games such as *Dungeons & Dragons* -- and many more besides. This package 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 <code>rpgtex</code>	iv	
Configuring <code>rpgtex</code>	v	
Package & Class Usage	v	
Compiling <code>rpgtex</code> Documents	vi	
Part I: <code>rpgtex</code> For Users		
1: Options & Variables		
Package Options	2	
Colo(u)rs	3	
Section Names	3	
Command Line Interface	4	
Fonts	4	
2: Commands & Macros		
Title & Part Pages	7	
Fonts & Decorative Text	8	
Theme Commands	9	
Utility Commands	9	
3: Environments		
RpgCards	11	
RpgMap	13	
RpgSecret	15	
RpgTable	17	
RpgNarration, RpgSidebar and RpgTip	18	
Filigree	20	
RpgItem, RpgSpell, RpgFeat: The FeatureForge	20	
Part II: Classes & Themes		
4: Classes		
rpgbook	23	
rpghandout	24	
rpgcard	25	
rpgdeck	26	
5: <code>default</code> Theme		
Appearance	27	
RpgItem	29	
RpgFeat	30	
RpgSpell	31	
RpgStat	31	
6: <code>dnd</code> Theme		
Appearance	32	
RpgItem	35	
RpgFeat	36	
RpgSpell	37	
RpgStat: D&D Monsters	39	
7: <code>scifi</code> Theme		
scifi RpgItem	46	
Part 3: <code>rpgtex</code> For Designers		
8: Designer Commands		
Title & Part Pages	48	
Page Appearance	49	
Other	50	
9: Feature Forge		
RpgMakeFeature	52	
Full Documentation	53	
RpgSwitchEnv	56	

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 your `LATEX` 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.

^aIn 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 23). Based on the standard book class, this is designed for larger RPG documents.
2. `rpghandout` (page 24). Based on the article class, this is designed for shorter documents
3. `rpgcard` (page 25). 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 <code>xelatex</code> compiler is selected by default, but the <code>-l</code> , <code>-luatex</code> flags set it to use <code>luatex</code> instead.	<code>-l</code> , <code>-luatex</code>
Build Directory	Compilation files (<code>.aux</code> , <code>.log</code> etc.) are stored in a build directory. The default is <code>.build</code> in the calling location, but can be changed with the <code>-b</code> flag	<code>-b <build dir></code>
Output Naming	The name of the output file can be changed from the default (equal to the input tex name)	<code>-o <output name></code>
Multi-pass Compiling	By default, the compiler runs twice in a row to enable references and <code>tikz[remember]</code> commands to function. A full three-compilation suite (necessary for very complex or reference-heavy documents) can be activated with the <code>-f</code> , <code>-full</code> flag	<code>-f</code> , <code>-full</code>
Volume Control	latex is notoriously noisy, producing copious output. By default, this is suppressed and only a summary is printed. The summary can be removed (rendering it completely silent) with the <code>-q</code> command, or the original output recovered in verbose mode; <code>-v</code> . These outputs are always overridden if a compilation error occurs, in which case the full trace is output to the console.	<code>-q</code> , <code>-v</code>
Auto-bibtex	If the <code>-r</code> or <code>-ref</code> flag is set, <code>bibtex</code> is automatically called in between the multi-compilation steps	<code>-r</code> , <code>-ref</code>
Auto-visualisation	If the <code>-show 1</code> option is set (which it is by default), the compiler will call <code>xdg-open <output-file></code> upon completion of the compilation; automatically opening or context-switching to the document. This can be turned off by calling <code>-show 0</code>	<code>-show</code>
Print Mode	A special interface for <code>rpgtex</code> , this uses the <code>bg=full</code> interface (page 4) to inject code into the latex document, setting the <code>bg=print</code> mode and suppressing the background output.	<code>-p</code> , <code>-print</code>

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 49) appears in the document, or the footer decorations set by `\RpgSetFooterDecoration` (page 49) .

Command	Paper	Footer Decorations
<code>full</code>	✓	✓
<code>print</code>	✗	✓
<code>none</code>	✗	✗

This flag also changes the behavior of the `rpgcard` class (page 25), 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 23).

theme The initial value passed to `\RpgSetTheme` (page 9) when package initialization is finished. The default value is `default`, activating the Default Theme (page 27).

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 colors¹ 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` (page 19) environment
2. **tablecolor** The background color of every other row in an `RpgTable` (page 17)
3. **tipcolor** The background color of the `RpgTip` (page 19) environment

narrationcolor The background color of the `RpgTip` (page 19) environment

contourinnercolor The default color of the inner text within a `\RpgContour` (page 8) command

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

fligreecolor The default color of the filigree used in a `RpgFiligree` (page 20) environment.

Calling `\RpgSetThemeColor` (page 50) 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}{FFFFFF}
```

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.

Section Names	
<pre>We are in Chapter ‘\RpgGetName{chapter}’, section ‘\RpgGetName{section}’. \RpgGetName{An example subsection} Now I am in ‘\RpgGetName{subsection}’ \chaptermark{Spoofed Chapter} I believe I am in a \RpgGetName{chapter}, though no chapter was actually called. \chaptermark{Options \& Variables} <i>%%put it back to the real value!</i></pre>	<pre>We are in Chapter ‘Options & Variables’, section ‘Section Names’. An example subsection Now I am in ‘An example subsection’ I believe I am in a Spoofed Chapter, though no chapter was actually called.</pre>

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

Command Line Interface

By default, L^AT_EX 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 T_EX 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 50) 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 section 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 \subsection\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
<code>\RpgFontBody</code>	main-body-family main-body-style	The main body text of the document, which <code>RpgSetFont</code> sets equal to <code>\normalfont</code> . Updating the fontsize here (i.e. using <code>\large</code>) can cause some counterintuitive results since it will <i>only</i> 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.
<code>\RpgFontTitle</code>	title-family title-style	The font used for <code>\@title</code> when <code>\maketitle</code> is called.
<code>\RpgFontSubtitle</code>	subtitle-family subtitle-style	The font used for the value of <code>\@subtitle</code> (page 7), <code>\@author</code> and <code>\@date</code> when <code>\maketitle</code> is called.
<code>\RpgFontPart</code>	part-family part-style	The font used when <code>\part</code> is called.
<code>\RpgFontTocPart</code>	toc-part-family toc-part-style	The font used for a part in the table of contents
<code>\RpgFontChapter</code>	chapter-family chapter-style	The font used when <code>\chapter</code> is called.
<code>\RpgFontTocChapter</code>	toc-chapter-family toc-chapter-style	The font used for a chapter in the table of contents
<code>\RpgFontSection</code>	section-family section-style	The font used when <code>\section</code> is called.
<code>\RpgFontTocSection</code>	toc-section-family toc-section-style	The font used for a section in the table of contents
<code>\RpgFontSubsection</code>	subsection-family subsection-style	The font used when <code>\subsection</code> is called.
<code>\RpgFontSubsubsection</code>	subsubsection-family subsubsection-style	The font used when <code>\subsubsection</code> is called.
<code>\RpgFontParagraph</code>	paragraph-family paragraph-style	The font used when <code>\paragraph</code> is called.
<code>\RpgFontSubparagraph</code>	subparagraph-family subparagraph-style	The font used when <code>\subparagraph</code> is called.
<code>\RpgFontTableTitle</code>	table-title-family table-title-style	The font used for <code><text></code> if <code>\RpgTable</code> (page ??) is called with the <code>title=<text></code> option.
<code>\RpgFontTableHeader</code>	table-header-family table-header-style	The font used for the first row of a <code>\RpgTable</code> .
<code>\RpgFontTableBody</code>	table-body-family table-body-style	The font used for the text within an <code>\RpgTable</code> after the first row.
<code>\RpgFontTipTitle</code>	tip-title-family tip-title-style	The font used for the title of an <code>RpgTip</code> environment (page ??).

<code>\RpgFontTipBody</code>	<code>tip-body-family</code> <code>tip-body-style</code>	The font used for the body of an <code>RpgTip</code> environment (page ??).
<code>\RpgFontSidebarTitle</code>	<code>sidebar-title-family</code> <code>sidebar-title-style</code>	The font used for the title of an <code>RpgSidebar</code> environment (page ??).
<code>\RpgFontSidebarBody</code>	<code>sidebar-body-family</code> <code>sidebar-body-style</code>	The font used for the body of an <code>RpgSidebar</code> environment (page ??).
<code>\RpgFontNarration</code>	<code>narration-family</code> <code>narration-style</code>	The font used for all (since they have no title) of an <code>RpgNarration</code> environment (page ??).
<code>\RpgFontStatBlockTitle</code>	<code>stat-block-title-family</code> <code>stat-block-title-style</code>	The font used for the title of a <code>`statblock'</code> environment - in the dnd theme this corresponds to the <code>monster</code> environment.
<code>\RpgFontStatBlockSection</code>	<code>stat-block-section-family</code> <code>stat-block-section-style</code>	The font used for sections within a <code>`statblock'</code> environment (should one be defined).
<code>\RpgFontStatBlockBody</code>	<code>stat-block-body-family</code> <code>stat-block-body-style</code>	The font used for text within a <code>`statblock'</code> environment (should one be defined).
<code>\RpgFontFooter</code>	<code>footer-family</code> <code>footer-style</code>	The font used for the footer text
<code>\RpgFontPageNumber</code>	<code>page-number-family</code> <code>page-number-style</code>	The font used for the page number within the footer
<code>\RpgFontDropCap</code>	<code>drop-cap-family</code> <code>drop-cap-style</code>	The font used for the large drop-cap letter created by a <code>RpgDropCap</code> (see below).
<code>\RpgFontDropCapInternal</code>	<code>drop-cap-internal-family</code> <code>drop-cap-internal-style</code>	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{title-name}
\subtitle{subtitle-name} % optional
\cover{path/to/cover} % optional
\author{Dr. W. Riter} % optional

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

`\maketitle` has been completely redefined, 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
<code>\part*{partname}</code> <code>\part*[<any text>]{partname}</code> <code>\part[none]{partname}</code> <code>\partpartname</code>	Uses original <code>\part</code> command defined by underlying class.
<code>\part[none]{partname}</code>	Calls the <code>\RpgSetPartPage</code> control sequence on a blank background.
<code>\part[path/to/image]{partname}</code>	Places the corresponding image as a full-page background, and then calls the <code>\RpgSetPartPage</code> drawing command.

The 'drawing command' is a control sequence set by `\RpgSetPartPage` (page 49) , 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` Uses the `RpgFontEmphasis` font to emphasise text.

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

```
\emph{text}
\key{more-text}
```

text
more-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` Renders text with a **contour effect**.

`{O{ } m}`

```
\RpgContour[key/vals]{<text>}
```

The optional argument accepts the following key/value pairs which alter the appearance of the contouring:

Key	Argument Type	Effect
inner	tikz color	Sets the color of the inner text
outer	tikz color	Sets the color of the outer contour around the outside
style	code	A style command (such as <code>\bfseries</code>) which is applied to the text before contouring begins.

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 `\newline` command (not `\\` or `\par`)

Example

```
\RpgContour [inner=red,outer=black]{example}
```

Output

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` Creates a decorative 'drop cap' letter to begin a new chapter with, and modifies the following text.

`{O{ } m m}`

```
\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{T}`{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.}

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

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

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 50) 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
<code>\RpgOrdinal {1}</code>	1st
<code>\RpgOrdinal {2}</code>	2nd
<code>\RpgOrdinal [\textsuperscript]{13}</code>	13 th
<code>\RpgOrdinal [\textbf]{133}</code>	133rd

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

`\RpgPage {0{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
<code>\RpgPage {example:current page}</code>	page 10
<code>\RpgPage [p]{example:current page}</code>	(page 10)
<code>\RpgPage [b]{example:current page}</code>	[page 10]
<code>\RpgPage [c]{example:current page}</code>	{page 10}
<code>\RpgPage [(error)]{example:current page}</code>	?page 10?

`\RpgPlural {s 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 starred version of the command suppresses the number output; so only the potentially-plural word is printed.

The optional argument `[<custom-plural>]` overrides the default logic, allowing for irregular plurals.

Example	Output
<code>\RpgPlural {1}{hat}</code>	1 hat
<code>\RpgPlural {2}{hat}</code>	2 hats
<code>\RpgPlural [octopodes]{1}{octopus}</code>	1 octopus
<code>\RpgPlural [octopodes]{359}{octopus}</code>	359 octopodes
<code>\RpgPlural *[octopodes]{1}{octopus}</code>	octopus
<code>\RpgPlural *[octopodes]{359}{octopus}</code>	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 56) 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
{0{}} 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 <i>tcb/colback</i>
opacity	[0-1]	The opacity of the background of the image. Replaces <i>tcb/opacityback</i>
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 <i>tcb/underlay</i> 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 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.

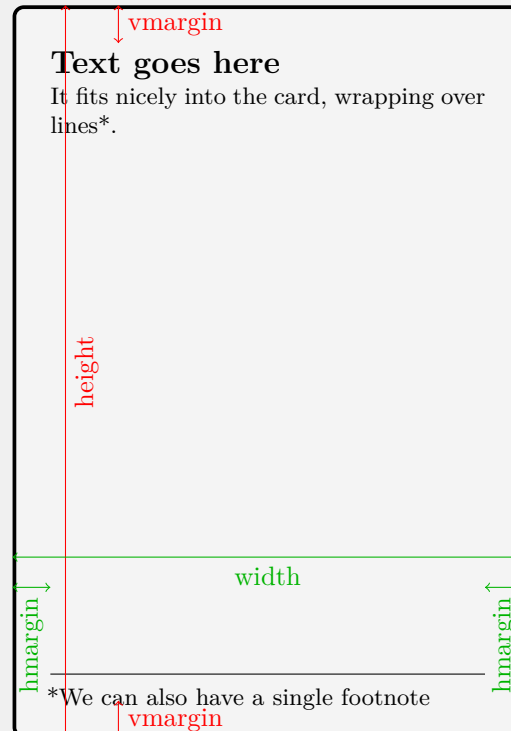
`\RpgCard!Reset`
`{}` Resets changes made to card parameters back to the default value.

```
\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, height=9.7cm,
    width=6.75cm]
    \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; use macros to
  save space!
\heightLines{red}
\widthLines{ForestGreen}

\end{tikzpicture}

\vspace{0.5cm}

%shrink so it fits in display area
\begin{RpgCard}[vmargin=0.1cm,
  hmargin=0.1cm, height=5.2cm, width=3.6cm]
  \raggedright\footnotesize And if we
    add lots of text\footnote{And maybe
      a footnote too?}
  \lipsum[2]
\end{RpgCard}
```



And if we add lots of text*
Nam dui ligula, fringilla
a, euismod sodales,
sollicitudin vel, wisi.
Morbi auctor lorem non
justo. Nam lacus libero,
pretium at, lobortis vitae,
ultrices et, tellus. Donec
aliquet, tortor sed
accumsan bibendum, erat
ligula aliquet magna,
vitae ornare odio metus a
mi. Morbi ac orci et nisl
hendrerit mollis.
Suspendisse ut massa.

Cras nec ante.
Pellentesque a nulla.
Cum sociis natoque
penatibus et magnis dis
parturient montes,
nascetur ridiculus mus.
Aliquam tincidunt urna.
Nulla ullamcorper
vestibulum turpis.
Pellentesque cursus luctus
mauris.

*And maybe a footnote
too?

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

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`
`area`
`{0{}0{}}` Adds an area entry into the current `RpgMap`. The appearance of the entry depends on the current map depth.

```
\area[area-name][manual-label] <body-text>
% or
\begin{area}[area-name][manual-label]
  <body-text>
\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}`.
- (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

It 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}
```

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.

Automated Map 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

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 label¹ 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:

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

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

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

`\RpgMapRefPage*`

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

Using the example map provided above:

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

`\RpgMapShowRefs` If called, all subsequent `\areas` 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.

Visible RpgMap 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

RpgSecret

The `RpgSecret` environment uses the `RpgSwitchEnv` system (page 52) to conditionally hide information; this allows a Game Master² 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` A switchable environment (page 52) associated with the switch `ShowSecrets`. When this key is true, the body of the environment is rendered; otherwise it is hidden.

`{0}{m}`

```

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

```

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:

²Or whatever your system calls this role!

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}`

RpgSecret: Hidden

```
\RpgSwitch{ShowSecrets}{false}
```

```
\lorumIpsumOne%Snippets of standard lorem 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, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur 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.

RpgSecret: Revealed

```
%%exactly the same as above, but reveal
the secrets!
\RpgSwitch{ShowSecrets}{true}
```

```
\lorumIpsumOne%Snippets of standard lorem 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, consectetur 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, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

Normal text

Logos kykloforia, thelxis kai ethos, tyrannis kai epilysis. Aei thera plaka, kyrios meion egestas, anagkas proin eira kai thera.

Secret text

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 is a wrapper for the `tabularx` (or `xltable` or `tabular--` see `breakable` and `simple`) environment, designed to provide attractive environments for the mainstay of RPG books since time immemorial: tables.

```
\begin{RpgTable}[width=0.85\linewidth,
  color=green!30!white]{lX}
  Header 1 & Header 2
  \\
  Text & Text fills up to 85\% of the line
    width then breaks
  \\
  Alternating & This row is transparent
  \\
  Colour & but this one is green
\end{RpgTable}

%suppress header formatting, add title and
  filigree:
\begin{RpgTable}[title={Test
Table},noheader, filigree]{lXlX}
  Plain & Header & Text
  \\
  Now there's no & difference & between the
    header and the main
  \\
  body
\end{RpgTable}
```

Header 1	Header 2
Text	Text fills up to 85% of the line width then breaks
Alternating	This row is transparent
Colour	but this one is green

Test Table

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

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

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

In default mode, RpgTable accepts the standard set of column specifications: {c,l,r,pwidth,...} and the extended tabularx 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 tabular 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 `xltable`, enabling the table to break over pages. **only available in 1-column mode (a fundamental limitation of xltable).**

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.

vskip The vertical spacing above and below the table. Default value is 9pt plus 3pt minus 3pt.

filigree If present, wraps the table in an `RpgFiligree` (page 20) environment, producing ornamental tables

RpgNarration, RpgSidebar and RpgTip

`rpgtex` defines three `colorbox` environments, which inherit from `tcolorbox`: `\RpgSidebar`, `\RpgNarration` and `\RpgTip`. 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 these boxes (`RpgSidebar` and `RpgTip` especially) can be somewhat arbitrary -- any mechanical differentiation between them can be overridden by calls to the `tcbox` backend. 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.

RpgNarration is used for 'read aloud' sections, such as those found at the beginning of adventure modules, or descriptions of environments.

The Three Text Boxes

```
\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}

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

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

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

A Sidebar

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

A Tip

This is some helpful - but not vital - text.

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

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.

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

```
\begin{RpgSidebar}[color=<color>,<tcbbox-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 `<tcbbox-options>` argument can be a list of all the basic `tcolorbox` options (see that documentation). The `color=x` argument is equivalent to calling both `colback=x` and `colbacktitle=x`. Due to the order of processing, if both `color` and `colback` are set, the value of `colback` is used.

Themes may alter the appearance of the sidebar using the `tcb` interface, calling `\tcbset{rpgsidebar/.append style=...}` to overwrite the existing instructions.

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

```
\begin{RpgTip}[color=<color>,<tcbbox-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 `<tcbbox-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.

Filigree

In design, Filigree is a decorative ribbon or other ornament around the outside of an element. We provide both a standard `RpgFiligree` environment and a `tcolorbox` style which implements the filigree.

`rpgfiligree` A `tcolorbox` style command that replaces the existing frame with a pre-defined overlay frame.

```
\begin{any-tcbox-env}[rpgfiligree]
  <body>
\end{any-tcbox-env}
```

This command can be added to the options of any `tcolorbox` environment, and it will add the current filigree as the frame.

`RpgFiligree`
{o} A 'default' `tcolorbox` with the `rpgfiligree` applied to the outside.

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

This environment is equivalent to calling `RpgTip` (page 19) with the `filigree` option enabled. See that documentation for customisation options.

```
%% As environment
\begin{RpgFiligree}
  A basic filigree-wrapped tcbox.
\end{RpgFiligree}

%% As option
\begin{RpgSidebar}[rpgfiligree,
  filigree-size=8pt]{Fancy Sidebar}
  A very fancy sidebar
\end{RpgSidebar}
```

A basic filigree-wrapped tcbox.

Fancy Sidebar

A very fancy sidebar

RpgItem, RpgSpell, RpgFeat: The FeatureForge

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 9, 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 52).

By default, all of these environments have the same behaviour.

Rpg[X]
RpgItem
RpgSpell
RpgFeat
{0} m 0{}

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 52), 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 = value1, key2 = value2]
  The value of key1 is \getkey{key1}
\end{Rpg[X]}
```

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 55 for more details.

```
%% 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}

\vspace{0.5cm}

%%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'`

PART II

Classes & Themes

Chapter 4: Classes

rpgbook

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!

Inherited Class

The rpgbook class inherits from [the extbook class](#) . This is an extension to the basic book class to allow more font sizes 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`

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

Options & Interactions

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.

- RPG books use a cover-title by default (`\RpgSetTitleMode` (page 49) is set to `cover`)
- 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 49)
 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

rpghandout

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.

Inherited Class

The rpghandout class inherits from [the extarticle class](#). This is an extension to the basic `article` class to allow more font sizes 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

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.

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

Options & Interactions

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.

- RPG books use a header-title by default (`\RpgSetTitleMode` (page 49) is set to `header`)
- The `layout` mode is activated
 1. Unless print mode is active, the page background will use the image set by `\RpgSetPaper` (page 49)
 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

rpgcard

The rpgcard class is designed for typesetting individual (or very few) `RpgCard` (page 11) objects, primarily for digital and screen viewing. Printing of RpgCards is better done via a deck (page 26).

Inherited Class

The rpgcard class changes which class it inherits from depending on the *background* option which is passed, which is used as a proxy for 'print mode'.

Normal Mode. By default, the rpgcard class inherits from the `standalone` class. The final output pdf is cropped to the smallest rectangle which contains the output. This makes it ideal for screen-viewing a single card.

Print Mode. If the option `bg=print` or `bg=none` has been passed, the class instead inherits from `article`. This prevents the cropping, and makes it simple to print the card out as a 'normal' size. However, we recommend using the `RpgDeck` class for assembling and printing large numbers of cards.

Special Commands

The rpgcard class does not define any special commands.

Geometry

An rpgcard has 'no geometry' in the normal sense; as a standalone environment it resizes to fit the contents. The document expands horizontally if a `\cardbreak` (page 11) occurs, or if more than one card is added into the document. There is no way to add a linebreak -- this is usually an indication that you want an rpgdeck.

When in print mode, the margins are a uniform 5pt to prevent printing errors.

Options & Interactions

The rpgcard has the following special interactions with the options detailed on page 2:

Option	Normal Behaviour	rpgcard Behaviour
bg	Determines if the 'background paper' and footer decorations are rendered.	Background paper and footers are disabled. Command instead toggles the Inherited class/page geometry of the class (see above).
columns	Sets the number of columns	Ignored

Interactions

- rpgcards cannot set titles or use tables of contents
- rpgcards are generally unsuited for any environment which spans an entire page width, unless wrapped inside another environment. `RpgTable` (page 17) will fail if used 'exposed' in an rpgcard document; but works within an `RpgCard`.
- All switches (`\RpgSwitch` (page 56)) are set to true, setting everything into card-mode.

rpgdeck

The `rpgdeck` class is designed to allow a writer to aggregate a large number of `RpgCard` (page 11) objects into a single document. In particular, it is designed to allow easy consolidation of cards which have been individually typeset using a `rpgcard` (page 25) document.

The `rpgdeck` class is largely identical to the `rpghandout` class, with the following exceptions:

1. The default geometry is a simple 5pt margin on all sides.
2. The ability to activate `twocolumn` mode is deactivated
3. The `clear` pagestyle is set globally; no footer decorations are rendered (only the `\RpgSetPaper` (page 49)).
4. All switches (`\RpgSwitch` (page 56)) are set to true, setting everything into card-mode.

Including rpgcards

The `rpgdeck` activates the `standalone package`, with the `subpreambles=true` option passed.

This allows a writer to `\input` a standalone document, even if it has its own `\documentclass` and `\begin{document}`, which would normally prohibit such an inclusion.

Example Deck-Building

Consider the following `rpgcard` document:

```
% spell1.tex
\documentclass[theme=dnd]{rpgcard}
\def\spellname{<text>}
\begin{document}
  \begin{RpgSpell}{\spellname}
    (...)
  \end{RpgSpell}
\end{document}
```

This generates a single playing-card size description of the spell, and can be compiled on its own and (for example) sent to a player digitally. We can assume that we have multiple such documents, `spell2.tex` and so on.

If we wished to include this card in a larger document:

```
% deck1.tex
\documentclass[theme=dnd]{rpgdeck}
\input{spell1}
\input{spell2}
\input{spell3}
(...)
\end{document}
```

Quirks

In the example above we defined the name of the environment through the macro `\spellname`, which was defined in the preamble. This works fine, because the package invokes `\RequirePackage[subpreambles=true]{standalone}`; the 'subpreambles' are parsed and included, even though they were invoked outside the preamble of the deck-document.

However, the package does this by collecting all the subpreambles first (excluding the `documentclass` declaration), then adding them to the main preamble, and then including the body text. This means that repeated definitions will override each other; if we had used `\spellname` for `spell2` and `spell3`, then *all* of the objects in the final deck would act as if they had `spell3`'s name. This can be avoided by moving the `\def` inside the `\begin{document}`.

Note too that since the theme (`dnd`) of the card was defined in the `documentclass`, this is excluded when it is aggregated; only the 'deck' package options are executed. It is therefore possible to aggregate objects into a deck which uses a different theme than the original items; the items will be rendered using the parent theme. If this is a concern, a manual call to `\RpgSetTheme` (page 9) can be used.

Chapter 5: default Theme

As the name suggests, the `default` theme is the theme which is loaded if no arguments are passed to `theme` when loading the `rpgtex` package. As such, it is largely a re-statement of the default \LaTeX values. This is important, as most other themes begin by first re-inserting the default theme; thereby resetting everything into a 'known state'. This means that they only need to define formatting which differs from the default.

The majority of this document has been typeset in a `rpgbook` class (page 23) using the default theme.

Appearance

Fonts

The default format defines only a single font: the Latin Modern Roman font, the standard \LaTeX font. For consistency with the rest of the library (which uses `fontspec`-defined fonts), we have used an OTF version of LMR which is bundled with the package.

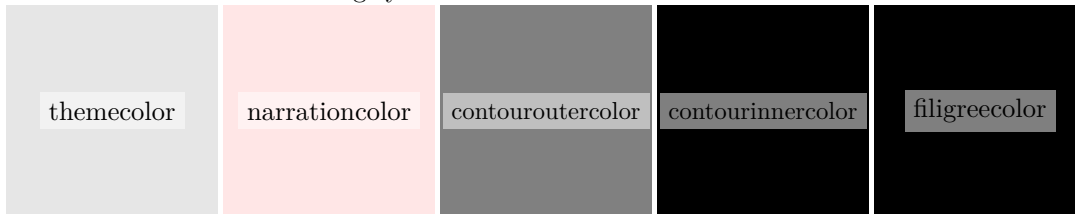
`\lrfont` The name given to the Latin Modern Roman font-family (extended with a small-caps font; Latin Modern Roman Caps) defined by the default theme

The values assigned to the font elements areas follows (note that `\normalfont` is aliased to the value of `RpgFontBody`).

Element	FontFamily	FontStyle
<code>\RpgFontBody</code>	<code>\lrfont</code>	
<code>\RpgFontEmphasis</code>	<code>\normalfont</code>	<code>\bfseries \slshape</code>
<code>\RpgFontTitle</code>	<code>\normalfont</code>	<code>\Huge</code>
<code>\RpgFontSubTitle</code>	<code>\normalfont</code>	<code>\Large</code>
<code>\RpgFontPart</code>	<code>\normalfont</code>	<code>\Huge</code>
<code>\RpgFontChapter</code>	<code>\normalfont</code>	<code>\Huge \bfseries</code>
<code>\RpgFontSection</code>	<code>\normalfont</code>	<code>\huge \bfseries</code>
<code>\RpgFontSubsection</code>	<code>\normalfont</code>	<code>\Large \bfseries</code>
<code>\RpgFontSubsubsection</code>	<code>\normalfont</code>	<code>\large \bfseries</code>
<code>\RpgFontParagraph</code>	<code>\normalfont</code>	<code>\bfseries \slshape</code>
<code>\RpgFontSubparagraph</code>	<code>\normalfont</code>	<code>\slshape</code>
<code>\RpgFontTableTitle</code>	<code>\sffamily</code>	<code>\bfseries \large</code>
<code>\RpgFontTableHeader</code>	<code>\sffamily</code>	<code>\bfseries</code>
<code>\RpgFontTableBody</code>	<code>\normalfont</code>	<code>\small</code>
<code>\RpgFontTipTitle</code>	<code>\sffamily</code>	<code>\bfseries</code>
<code>\RpgFontTipBody</code>	<code>\normalfont</code>	<code>\small</code>
<code>\RpgFontSidebarTitle</code>	<code>\sffamily</code>	<code>\bfseries \normalsize</code>
<code>\RpgFontSidebarBody</code>	<code>\normalfont</code>	<code>\small</code>
<code>\RpgFontNarration</code>	<code>\normalfont</code>	<code>\small</code>
<code>\RpgFontAbstractTitle</code>	<code>\Large</code>	<code>\scshape \bfseries</code>
<code>\RpgFontAbstractBody</code>	<code>\normalfont</code>	<code>\small \slshape</code>
<code>\RpgFontSecretTitle</code>		
<code>\RpgFontCardTitle</code>	<code>\normalfont</code>	<code>\bfseries \Large</code>
<code>\RpgFontCardHeader</code>	<code>\normalfont</code>	<code>\it</code>
<code>\RpgFontCardBody</code>	<code>\normalfont</code>	<code>\small</code>
<code>\RpgFontTocPart</code>	<code>\normalfont</code>	<code>\Large \bfseries</code>
<code>\RpgFontTocChapter</code>	<code>\normalfont</code>	<code>\large</code>
<code>\RpgFontTocSection</code>	<code>\normalfont</code>	<code>\normalsize</code>
<code>\RpgFontStatBlockTitle</code>	<code>\normalfont</code>	<code>\bfseries \LARGE</code>
<code>\RpgFontStatBlockBody</code>	<code>\normalfont</code>	<code>\small</code>
<code>\RpgFontStatBlockSection</code>	<code>\sffamily</code>	<code>\large</code>
<code>\RpgFontFooter</code>	<code>\normalfont</code>	<code>\scriptsize</code>
<code>\RpgFontPageNumber</code>	<code>\normalfont</code>	<code>\scriptsize</code>
<code>\RpgFontDropCap</code>	<code>\normalfont</code>	
<code>\RpgFontDropCapFirstLine</code>		<code>\scshape</code>

Colors

The default theme colors are largely monochromatic:



Backgrounds & Footers

The default theme defines no background or headers (and clears any existing ones). The footer consists of the chapter name and a page number, using the appropriate fonts (`\RpgFontFooter` and `\RpgFontPageNumber`).

Text Boxes

The default text boxes are very simple in design:

The RpgSidebar Has minor decorations at the top and bottom to distinguish it from an RpgTip	The RpgTip Almost entirely undecorated	RpgNarration is also simple; but uses the narration color
---	--	---

Section Headers

Only chapters¹ are numbered; all other section/subsections (etc.) are unnumbered. Otherwise there are no changes made to the titles aside from minor spacing changes, and the assignment of the relevant fonts.

Section Headers	
<pre>\section*{Example} <i>%starred so not added to toc; otherwise the same</i> \subsection{Another Example} \subsubsection{More Examples!}</pre>	Example Another Example More Examples!

RpgDice

The default RpgDice format is $ndx + m$:

Default Dice	
<code>\RpgDice{3d8 -5 +2}</code>	3d8-3

¹If in an rpgbook / other class which support chapters

RpgItem

`\RpgItem` {0}{m 0{}} The default configuration of the `RpgItem` (page 20) environment, used to describe physical objects and equipment.

```
\RpgItemShowCard{true/false} % set the card mode
\begin{RpgItem}[card-opts]{Item Name}[key-values]
  <body text>
\end{RpgItem}
```

As with all FeatureForge environments, has the ability to switch between ``text mode'` and ``card mode'` depending on the value passed to `\RpgItemShowCard` (page 21).

The `default` theme defines two keys for the `RpgItem` object:

Key	Default Value	Effect
<code>description</code>	<code>{}</code>	An italicised byline used to summarise the item
<code>image</code>	<code>{}</code>	If non-empty, define an image path which is used when in card-mode.

Default RpgItem: Text Mode

```
\RpgItemShowCard{false}
\begin{RpgItem}{Joyeuse}[
  description={The Sword Jewellous},
  image={../example/img/joyeuse}
]
  The sword of Charlemagne; this
  jewelled sword gives you +3 to
  heroism checks.
\end{RpgItem}
```

Joyeuse

Requires: The Sword Jewellous

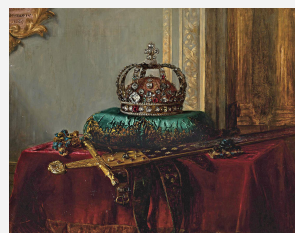
The sword of Charlemagne; this jewelled sword gives you +3 to heroism checks.

Default RpgItem: Card Mode

```
\RpgItemShowCard{true}
\begin{RpgItem}{Joyeuse}[
  description={The Sword Jewellous},
  image={../example/img/joyeuse}
]
  The sword of Charlemagne; this
  jewelled sword gives you +3 to
  heroism checks.
\end{RpgItem}
```

Joyeuse

The Sword Jewellous



The sword of Charlemagne; this jewelled sword gives you +3 to heroism checks.

RpgFeat

`RpgFeat` The default configuration of the `RpgFeat` (page 30) environment, used to describe abilities and character features and choices.

```
\RpgFeatShowCard{true/false} % set the card mode
\begin{RpgFeat}[card-opts]{Feat Name}[key-values]
  <body text>
\end{RpgFeat}
```

As with all FeatureForge environments, has the ability to switch between 'text mode' and 'card mode' depending on the value passed to `\RpgFeatShowCard` (page 21).
The `default` theme defines only a single key for the `RpgFeat` object:

Key	Default Value	Effect
<code>requires</code>	<code>{}</code>	If non-empty, an italicised note is added, indicating the prerequisites for acquiring the ability.

Default RpgFeat: Text Mode

```
\RpgFeatShowCard{false}
\begin{RpgFeat}{Hyperattack}[
  requires={Mega-attack},
]

  You can unleash superhuman speed
  against your enemies. Once per day,
  make \RpgDice{3d10} additional
  attacks.
\end{RpgFeat}
```

Hyperattack

Requires: Mega-attack

You can unleash superhuman speed against your enemies. Once per day, make 3d10 additional attacks.

Default RpgFeat: Card Mode

```
\RpgFeatShowCard{true}
\begin{RpgFeat}{Hyperattack}[
  requires={Mega-attack},
]

  You can unleash superhuman speed
  against your enemies. Once per day,
  make \RpgDice{3d10} additional
  attacks.
\end{RpgFeat}
```

Hyperattack

You can unleash superhuman speed against your enemies. Once per day, make 3d10 additional attacks.

Requires: Mega-attack

RpgSpell

RpgSpell
{0}{m 0{}}

The default configuration of the RpgSpell (page 37) environment, used to describe magical spells or spell-like abilities.

```
\RpgSpellShowCard{true/false} % set the card mode
\begin{RpgSpell}[card-opts]{Spell Name}[key-values]
  <body text>
\end{RpgSpell}
```

As with all FeatureForge environments, has the ability to switch between 'text mode' and 'card mode' depending on the value passed to \RpgSpellShowCard (page 21).

The default RpgSpell is a placeholder; it defines no keys, and defines no special formatting for RpgSpell; on the assumption that 'spells' are so system dependent as to make an attempt to make a default would be pointless.

RpgStat

RpgStat
{0}{m 0{}}

The dnd configuration of the RpgStat (page 39) environment, used to describe monsters and enemies via a *statblock*

```
\RpgStatShowCard{true/false} % set the card mode
\begin{RpgStat}[card-opts]{Creature Name}[key-values]
  <body text>
\end{RpgStat}
```

As with all FeatureForge environments, has the ability to switch between 'text mode' and 'card mode' depending on the value passed to \RpgStatShowCard (page 21).

The default RpgStat is a placeholder; it defines no keys, and defines no special formatting; on the assumption that 'statblocks' are so system dependent as to make an attempt to make a default would be pointless.

CHAPTER 6: DND THEME

The dnd theme is designed to mimic the Appearance of the *Dungeons & Dragons* source books. The specific parameters used to replicate the D&D books are mostly derived from the original DnD-5e-Latex-Template package, with some additional updates in line with the changes made with the 2024 rules update.

APPEARANCE

Fonts

The dnd theme defines a number of font families which are used throughout the theme.

`\bookman` The Bookman Old Style STd typeface, used as the main body text.

`\keplerserif` The KpRoman font, used for **emphasis**

`\kepler` The KpSans font, primarily used in headers and titles

`\gillius` The Gillius ADF No. 2 Font, used as a lighter sans-serif font, often in the body of environments where `\kepler` was used as the title.

`\Royal` **A DECORATIVE FONT**, unsuited for blocks of text. Used for drop-caps

The values assigned to the font elements areas follows (note that `\normalfont` is aliased to the value of `RpgFontBody`).

Element	FontFamily	FontStyle
<code>\RpgFontBody</code>	<code>\bookman</code>	
<code>\RpgFontEmphasis</code>	<code>\keplerserif</code>	<code>\bfseries \scshape \larger [1]</code>
<code>\RpgFontTitle</code>	<code>\normalfont</code>	<code>\bfseries \fontsize {30}{50}\selectfont \color {titlered}</code>
<code>\RpgFontSubTitle</code>	<code>\scshape</code>	<code>\LARGE \color {titlered}</code>
<code>\RpgFontPart</code>	<code>\normalfont</code>	<code>\color {titlered}\fontsize {50}{30}\selectfont</code>
<code>\RpgFontChapter</code>	<code>\normalfont</code>	<code>\titlemode \fontsize {30}{50}\selectfont</code>
<code>\RpgFontSection</code>	<code>\normalfont</code>	<code>\titlemode \Huge</code>
<code>\RpgFontSubsection</code>	<code>\normalfont</code>	<code>\titlemode \huge</code>
<code>\RpgFontSubsubsection</code>	<code>\normalfont</code>	<code>\titlemode \Large</code>
<code>\RpgFontParagraph</code>	<code>\normalfont</code>	<code>\bfseries \slshape</code>
<code>\RpgFontSubparagraph</code>	<code>\normalfont</code>	<code>\bfseries \slshape</code>
<code>\RpgFontTableTitle</code>	<code>\kepler</code>	<code>\bfseries \large \scshape</code>
<code>\RpgFontTableHeader</code>	<code>\kepler</code>	<code>\bfseries</code>
<code>\RpgFontTableBody</code>	<code>\gillius</code>	<code>\small</code>
<code>\RpgFontTipTitle</code>	<code>\kepler</code>	<code>\bfseries \scshape</code>
<code>\RpgFontTipBody</code>	<code>\gillius</code>	<code>\small</code>
<code>\RpgFontSidebarTitle</code>	<code>\kepler</code>	<code>\bfseries \scshape</code>
<code>\RpgFontSidebarBody</code>	<code>\gillius</code>	<code>\small</code>
<code>\RpgFontNarration</code>	<code>\gillius</code>	<code>\small</code>
<code>\RpgFontAbstractTitle</code>	<code>\Large</code>	<code>\scshape \bfseries</code>
<code>\RpgFontAbstractBody</code>	<code>\normalfont</code>	<code>\small \slshape</code>
<code>\RpgFontSecretTitle</code>		
<code>\RpgFontCardTitle</code>	<code>\keplerserif</code>	<code>\color {titlered}\large \bfseries</code>
<code>\RpgFontCardHeader</code>	<code>\normalfont</code>	<code>\normalfont \scshape \footnotesize</code>
<code>\RpgFontCardBody</code>	<code>\normalfont</code>	<code>\small</code>
<code>\RpgFontTocPart</code>	<code>\normalfont</code>	<code>\LARGE \keplerserif \scshape \color {titlered}</code>
<code>\RpgFontTocChapter</code>	<code>\normalfont</code>	<code>\Large \keplerserif \scshape \color {titlered}</code>

\RpgFontTocSection	\normalfont	\normalsize
\RpgFontStatBlockTitle	\scshape	\bfseries \LARGE \color {statblockcolor}
\RpgFontStatBlockBody	\bookman	\small \color {black}
\RpgFontStatBlockSection	\kepler	\color {statblockcolor}\scshape \large
\RpgFontFooter	\normalfont	\color {pagegold}\scriptsize
\RpgFontPageNumber	\normalfont	\color {pagegold}\scriptsize
\RpgFontDropCap	\Royal	
\RpgFontDropCapFirstLine		\scshape

COLORS

The theme provides a large number of colors:

themecolor	narrationcolor	contouroutercolor	contourinnercolor	filigreecolor
bgtan	pagegold	titlered	contourgray	titlegold
default narration	pagenumbers & footer	title font & default contour-inner	default contour-outer	title rules
rulecolor	statblockribbon	statblockbg	BrGreen	PhbLightGreen
statblock triangles			Basic Rules	PHB Part 1
PhbLightCyan	PhbMauve	PhbTan	DmgLavender	DmgCoral
PHB Part 2	PHB Part 3	PHB Appendix	DMG Part 1	DMG Part 2
DmgSlateGray	DmgLilac			
DMG Part 3	DMG Appendix			

The colors from BrGreen through DmgLilac are chosen so that, when set as the themecolor (using \RpgSetThemeColor (page 50)), the appearance of the tables and sidebars is the same as the corresponding part in either the Basic Rules, the Players' Handbook, and the Dungeon Master's Guide.

BACKGROUNDS & FOOTERS

The dnd theme uses a paper-like image as the background, and places a `scroll' on the footer¹. The chapter name is placed in the footer (in pagegold color). The page number is positioned to lie in a divot of the scroll.

¹The scroll alternates direction in twoside documents; this document is written in oneside mode, so there is no alternating.

TEXT BOXES

The dnd theme modifies the text boxes to give them the following appearance:

THE RpgSIDEBAR

Horizontal 'ribbons' and sharp corners differentiate it from RpgTip.

THE RpgTIP

Almost entirely undecorated, with rounded corners.

RpgNarration has 'bars' on the side.

SECTION HEADERS

As in the default mode, only chapters² are numbered; all other section/subsections (etc.) are unnumbered.

Chapters are rendered using a \RpgContour (with the default colors). The subsection environment is embellished with a \hrule (in the titlegold color) which stretches across the page.

```
\section*{Example} %starred so not added to
toc; otherwise the same
\subsection{Another Example}
\subsubsection{More Examples!}
```

EXAMPLE

ANOTHER EXAMPLE

MORE EXAMPLES!

RpgDICE

Following the syntax found in D&D monster statblocks, RpgDice shows the average of the roll, and automatically converts 'dx' into 'ldx':

```
\RpgDice{3d8 -5 +2}
```

10 (3d8-3)

```
\RpgDice{d8 + 0}
```

4 (1d8)

FILIGREE

The dnd implementation of RpgFiligree (page 20) is designed to mimic the decorative elements seen around the class tables in the Player's Handbook:

```
\begin{RpgFiligree}
  Some decorative text
\end{RpgFiligree}

\vspace{1cm}

\begin{RpgTable}[filigree]{1X}
  Header 1 & Header 2
  \\
  Body 1 & Body 2
  \\
  Body 3 & Body 4
\end{RpgTable}
```

Some decorative text

Header 1	Header 2
Body 1	Body 2
Body 3	Body 4

²If in an rpgbook / other class which support chapters

RpgItem

`\RpgItem` The dnd-specialisation of the `RpgItem` (page 20) environment, used to describe physical objects and equipment.

```
\RpgItemShowCard{true/false} % set the card mode

\begin{RpgItem}[card-opts]{Item Name}[key-values]
  <body text>
\end{RpgItem}
```

As with all FeatureForge environments, `RpgItem` has the ability to switch between 'text mode' and 'card mode' depending on the value passed to `\RpgItemShowCard` (page 21). The dnd theme defines the following keys for the `RpgItem` object:

Key	Default Value	Effect
rarity	{}	Common / Uncommon / Rare etc.
type	{}	A descriptor such as 'weapon' or 'wondrous item'
requires-attunement	none	Can either be used as a flag (i.e. a key with no value) in which case the phrase 'Requires attunement' is added. If a value is assigned, it is appended immediately afterwards.
image	{}	If non-empty, define an image path which is used when in card-mode.

The card-variant renders the title in a 'flag ribbon' which expands to fit the title (and shrinks the font size if it would spill over the card boundary)

RpgItem In Action

```
\RpgItemShowCard{false}
\begin{RpgItem}{Joyeuse}{
  rarity=Rare,
  type=weapon (sword),
  requires-attunement, %just a flag, no value
  image={../example/img/joyeuse}
}
  The sword of Charlemagne; this jewelled
  sword gives you +3 to heroism checks.
\end{RpgItem}

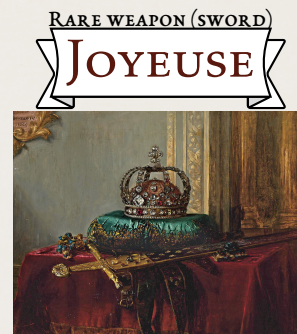
\vspace{0.5cm}\hrule\vspace{0.5cm}

%now the card version
\RpgItemShowCard{true}
\begin{RpgItem}{Joyeuse}{
  rarity=Rare,
  type=weapon (sword),
  requires-attunement={by a Paladin of
    France}, %now has a value...
  image={../example/img/joyeuse}
}
  The sword of Charlemagne; this jewelled
  sword gives you +3 to heroism checks.
\end{RpgItem}
```

JOYEUSE

Rare weapon (sword). Requires attunement

The sword of Charlemagne; this jewelled sword gives you +3 to heroism checks.



Requires attunement by a Paladin of France

The sword of Charlemagne; this jewelled sword gives you +3 to heroism checks.

RPGFEAT

`\RpgFeat` The dnd-specialisation of the `RpgFeat` (page 30) environment, used to describe abilities and character features and choices.

```
\RpgFeatShowCard{true/false} % set the card mode

\begin{RpgFeat}[card-opts]{Feat Name}[key-values]
  <body text>
\end{RpgFeat}
```

As with all FeatureForge environments, has the ability to switch between 'text mode' and 'card mode' depending on the value passed to `\RpgFeatShowCard` (page 21). The dnd `RpgFeat` is largely the same as the default, using the same 'requires' key, and adding an alias:

Key	Default Value	Effect
<code>requires</code>	<code>{}</code>	If non-empty, an italicised note is added, indicating the prerequisites for acquiring the ability.
<code>prerequisite</code>	<code>{}</code>	An alias for 'requires' (since this is what appears on screen).

D&D RpgFeat

```
\RpgFeatShowCard{false}

\begin{RpgFeat}{Grappler}[
  requires={Strength 13 or higher},
]
  \grapplerText %predefined to save
    space. Just plain text!
\end{RpgFeat}

\vspace{0.25cm}\hrule\vspace{0.25cm}

%now the card version
\RpgFeatShowCard{true}

\begin{RpgFeat}[color=DmgLilac]{Grappler}[
  prerequisite={Strength 13 or higher},
]
  \grapplerText
\end{RpgFeat}
```

GRAPPLER

Prerequisite: Strength 13 or higher

You've developed the skills necessary to hold your own in close-quarters grappling. You gain the following benefits:

- You have advantage on attack rolls against a creature you are grappling.
- You can use your action to try to pin a creature grappled by you. To do so, make another grapple check. If you succeed, you and the creature are both restrained until the grapple ends.

GRAPPLER

You've developed the skills necessary to hold your own in close-quarters grappling. You gain the following benefits:

- You have advantage on attack rolls against a creature you are grappling.
- You can use your action to try to pin a creature grappled by you. To do so, make another grapple check. If you succeed, you and the creature are both restrained until the grapple ends.

PREREQUISITE: STRENGTH 13 OR HIGHER

RPGSPELL

`\RpgSpell` The dnd configuration of the `\RpgSpell` (page 37) environment, used to describe magical spells or spell-like abilities.

```
\RpgSpellShowCard{true/false} % set the card mode
\begin{RpgSpell}[card-opts]{Spell Name}[key-values]
  <body text>
\end{RpgSpell}
```

As with all FeatureForge environments, has the ability to switch between 'text mode' and 'card mode' depending on the value passed to `\RpgSpellShowCard` (page 21). The `\RpgSpell` formats the components of a D&D spell by defining the following keys:

Key	Default Value	Effect
school	{}	The spell school (conjuration etc.). Can be empty.
level	0	The level of the spell, expressed as a single integer. 0 is converted into 'Cantrip'.
casting-time	1 action	A plaintext field for the casting time (1 action, 1 reaction etc.)
range	self	The range of the spell.
components	VSM	The Verbal/Somatic/Material components for casting the spell
duration	Instantaneous	The duration of the spell (and its concentration status)
source	{}	An optional field for declaring the source of the spell (i.e. the PHB).

All these fields support a `\footnote`, but note that per the `\RpgCard` documentation (page 11), only a single footnote can be included when in card-mode; successive footnotes override each other. In addition to these keys, we also provide some commands for typesetting some 'boilerplate language' found in D&D spells:

`\RpgCantripScaling` Provides a convenient interface for typesetting the 'level scaling' of cantrips in D&D

```
\RpgCantripScaling[benefit-name]{1st-lvl}{5th-lvl}{11th-level}{17th-level}
```

The command is typeset with some boilerplate text, followed by a `\RpgTable` (page 17), with each argument as an entry used as the corresponding level in the table.

`\RpgSpellUpcast` Provides a convenient interface for typesetting the 'upcasting' (increased benefits when using higher level spell slots) of spells in D&D

```
\RpgSpellUpcast[benefit-of-upcasting]{benefit-per-increase}[slots-per-increase]
```

The default 'benefit' text is *"The damage increases by"*; with the expectation that the mandatory argument is a damage increase (such as 1d6). The slots-per-increase is an integer value (default 1), indicating the number of level gains per increase.

This function can only be called within an `\RpgSpell` environemnt, as it relies on the local value of `\l__rpg_dnd_spell_level_tl`.

```
\RpgCantripScaling[\# of Beams]{1} {3}
{5}{9d10}

\vspace{0.5cm}

%% assume being called within a 3rd level
spell...
\RpgSpellUpcast[The duration increases
by]{1 minute}[2]
```

This cantrip improves with your character level:

Level	1st	5th	11th	17th
# of Beams	1	3	5	9d10

At higher levels. The duration increases by 1 minute for every 2 spell levels above 3rd used to cast this spell.


```

\RpgSpellShowCard{false}
\begin{RpgSpell}{Firebolt}[
  school=Evocation,
  level = 0,
  casting-time=1 action,
  range=120ft,
  components = VS,
  duration=Instantaneous,
  source=Free Basic Rules (2014)
]

```

You hurl a mote of fire at a creature or object within range. Make a ranged spell attack against the target. On a hit, the target takes 1d10 fire damage. A flammable object hit by this spell ignites if it isn't being worn or carried.

```

\RpgCantripScaling{1d10}{2d10}
{3d10}{4d10}
\end{RpgSpell}

```

```

\vspace{1cm}

```

```

\RpgSpellShowCard{true}
\begin{RpgSpell}{Cone of Cold}[
  school=Evocation,
  level = 5,
  casting-time=1 action,
  range=self (60ft cone),
  components = VSM\footnote{A small crystal
    or glass cone},
  duration=Instantaneous,
  source=Free Basic Rules (2014)
]

```

A blast of cold air erupts from your hands. Each creature in a 60-foot cone must make a Constitution saving throw. A creature takes 8d8 cold damage on a failed save, or half as much damage on a successful one. A creature killed by this spell becomes a frozen statue until it thaws.

```

\RpgSpellUpcast{1d8}
\end{RpgSpell}

```

FIREBOLT

Evocation Cantrip

Casting time: 1 action

Range: 120ft

Components: VS

Duration: Instantaneous

Source: *Free Basic Rules (2014)*

You hurl a mote of fire at a creature or object within range. Make a ranged spell attack against the target. On a hit, the target takes 1d10 fire damage. A flammable object hit by this spell ignites if it isn't being worn or carried.

This cantrip improves with your character level:

Level	1st	5th	11th	17th
Damage	1d10	2d10	3d10	4d10

5TH LEVEL EVOCATION Cone of Cold

CASTING TIME 1 ACTION	RANGE SELF (60FT CONE)
COMPONENTS VSM*	DURATION INSTANTANEOUS

A blast of cold air erupts from your hands.

Each creature in a 60-foot cone must make a Constitution saving throw. A creature takes 8d8 cold damage on a failed save, or half as much damage on a successful one. A creature killed by this spell becomes a frozen statue until it thaws.

At higher levels. The damage increases by 1d8 for every spell level above 5th used to cast this spell.

Source: (Free Basic Rules (2014))

*A small crystal or glass cone

RPGSTAT: D&D MONSTERS

Internally, the monster statblock is one of the most complex environments in the `rpgtex` library; the upside being that it makes it quick and easy to generate a monster which follows the standard D&D monster assumptions.

`RpgStat`
`RpgStat*`
`{0}{m 0}{}`

The `dnd` configuration of the `RpgStat` (page 39) environment, used to describe monsters and enemies, using the rules and parameters of spells in D&D 5e. As with all FeatureForge environments, `RpgStat` has the ability to switch between ``text mode'` and ``card mode'` depending on the value passed to `\RpgStatShowCard` (page 21).

```
\RpgStatShowCard{true/false} % set the card mode
\begin{RpgStat}[card-opts]{Creature Name}[key-values]
  <body text>
\end{RpgStat}
```

RPGSTAT VS RpgStat*

`RpgStat*` is identical to the unstarred version, except it automatically activates the `twocolumn` mode, and sets `float-type = figure*` (see below). This makes it suitable for rendering 'boss' (i.e. large) stat blocks whilst writing `twocolumn` documents.

We semantically split the standard `D&D statblock` into two parts:

1. The *statistics* - hit points, armour class, stat modifiers, skill proficiencies and so on. These are highly structured and amount to 'form filling'.

These are defined in the key-value options.

2. The *abilities* - including the traits, actions, bonus actions and so on. These are freeform and largely consist of plain text.

These are defined in the body of the environment.

COMMONER

Medium or Small Humanoid; typically neutral

AC	10	Speed	30ft
HP	4 (1d8)	Initiative	+0

	STR (10)	DEX (10)	CON (10)	INT (10)	WIS (10)	CHA (10)
MOD	+0	+0	+0	+0	+0	+0

Skills

Senses passive Perception 10

Languages Common

Challenge 0 (0xp, proficiency: +2)

TRAITS

Mob Mentality. If a Commoner is subjected to a mind-altering effect (such as the frightened condition, or a charm spell), all Commoners within 10ft are also subjected to this effect.

ACTIONS

Club. *Melee (5ft):* +2 to hit. *On Hit:* 2 (1d4) bludgeoning damage. .

Statistics (key/values)

Abilities (body text)

STATISTICS OPTIONS

There are a large number of options that can be passed to the statistics. Where multiple key names are presented, these are aliased to the same value and behave identically. As with all expl3 keys, repeated values are sequentially overwritten, with only the final value retained.

APPEARANCE

Key	Value	Default Value	Effect
color	tikz-color code	rulered (page 33)	The base color used by the environment. Several colors are automatically derived from this: <ul style="list-style-type: none"> • The 'triangle dividers' use this color • The 'header text' color is 50% darker³ • The border color is 20% lighter (unless otherwise specified) • The themecolor is temporarily set to a value 75% lighter.
outline-color	tikz-color code	--empty--	If non-empty, this color is used for the border of the environment, instead of the one derived from 'color'. No effect when in Card mode.
filigree	(flag)		If this flag is present, replaces the default border with filigree (page 34), using the current filigreecolor. No effect when in Card mode.
filigree-match	(flag)		If this flag is present, replaces the default border with filigree (page 34) and locally changes filigreecolor to be equal to the border color. If both filigree-match and filigree present, filigree-match takes priority. No effect when in card mode.
twocolumn	(flag)		If this flag is present, the environment is rendered in twocolumns, suitable for breaking up statblocks which are full-page width (either when in a onecolumn document, or placed inside a full-width float). Automatically activated by RpgStat*. No effect when in card mode.

POSITIONING

Key	Value	Default Value	Effect
float	!/h/t/b	--empty--	If non-empty, wraps the statblock in a float (of a type determined by float-type), with the float command equal to this value. <div> <p>IMPORTANT!</p> <p>Normally the 'h' option does not work for figure* when in a twocolumn environment. We have implemented a workaround which uses the strip environment to simulate a "immediate full width image". This is very fragile! It is likely to throw compilation errors when on the same page as other floats, or when there is insufficient text before or after the environment begins. Use with extreme caution!</p> </div>
float-type	environment name	figure	No effect when in card mode. The environment which wraps the statblock if a float command is given; setting this to figure* creates a full-width statblock when in a twocolumn document.

³In the sense that it is defined as color!50!black

BASIC DESCRIPTION

Key	Value	Default Value	Effect
nickname	text	<title value>	A shorter or alternative version of the 'main name' which appears at the top of the statblock; this is the value returned by <code>\RpgStatName</code> (page ??) , and used automatically in several places. Useful when you want to give the statblock a grand title, but don't want " <i>Gorgenhar Haluavin, the Devourer of Worlds</i> makes three attacks".
type	text	(empty)	This text is placed in italics underneath the main title. Usually used for size and alignment declarations.
armor-class / ac / AC	text	10	The value given as the armour class of the monster. Can accept strings explaining the armour such as '15 (Natural Armour)' or '10 (13 with <i>mage armor</i>)'
hit-points / hp / HP	text	<code>\RpgDice{1d8}</code>	The value given as the total HP of the monster. The value given is usually as a <code>\RpgDice</code> (page 9) command. This value is expanded after the modifiers are computed, so it is possible to use (i.e.) the constitution modifier as a variable: <code>hp=\RpgDice{8d10 + 8*\stat{con}}</code> will be correctly computed.
speed	text	30ft	Value given as the monster's movement speed. If the substring 'ft' is not found in the input, then it is appended to the end; i.e. <code>speed=30</code> will render as '30ft'.
initiative	text		If this value is non-empty, it is used (without further formatting) as the initiative of the creature. If the value is empty, the computed dexterity modifier is used instead.

ABILITIES & SAVES

Key	Value	Default Value	Effect
proficiency-bonus / proficiency / pb	integer	(empty)	If a value is provided, this is used as the proficiency bonus for all save bonus, skill bonuses and to-hit values. If not provided, a value is computed based on the challenge rating.
str	integer	10	Sets the value of the Strength Score. The associated modifier is automatically computed
dex	integer	10	Sets the value of the Dexterity Score. The associated modifier is automatically computed
con	integer	10	Sets the value of the Constitution Score. The associated modifier is automatically computed
int	integer	10	Sets the value of the Intelligence Score. The associated modifier is automatically computed
wis	integer	10	Sets the value of the Wisdom Score. The associated modifier is automatically computed
cha	integer	10	Sets the value of the Charisma Score. The associated modifier is automatically computed
str-save	(flag)		If present, this flag indicates the monster is proficient in Strength saving throws.
dex-save	(flag)		If present, this flag indicates the monster is proficient in Dexterity saving throws.
con-save	(flag)		If present, this flag indicates the monster is proficient in Constitution saving throws.
int-save	(flag)		If present, this flag indicates the monster is proficient in Intelligence saving throws.
wis-save	(flag)		If present, this flag indicates the monster is proficient in Wisdom saving throws.
cha-save	(flag)		If present, this flag indicates the monster is proficient in Charisma saving throws.

SKILLS & DETAILS

Key	Value	Default Value	Effect
skills	comma-separated list	(empty)	Each skill in this list is iterated over - if the value is one of the 18 D&D skills, it is matched with its usual ability modifier, and a skill bonus is computed in the normal fashion. This bonus is appended to the skill name in the list when displayed. <div> MANUAL SKILLS The automated computing assumes that skills will always be paired with their normal attribute. If a monster uses a variant pairing (i.e. Strength (Intimidation)) then this will need to be manually declared. The automation only occurs if the string exactly matches^a the names of the skills; so athletics, Acrobatics (+25) will cause the 'athletics' to be computed automatically, but the Acrobatics will not count as a match, so will keep its manual bonus declaration. <hr/> ^aWhitespace and variant capitalisation count as 'exact matches' </div>
skills-expertise	comma-separated list	(empty)	Exactly as with the skills key, except the automated bonus adds twice the proficiency bonus. Expertise-skills are listed before other skills in the rendered environment.
languages	text	(empty)	Any languages spoken or understood by the creature. This is used as a simple string (no manipulation)
senses	text	(empty)	Any special senses possessed by the creature. Passive perception is handled separately.
passive-perception	integer	(empty)	If empty, the passive perception is equal to the Wisdom (Perception) bonus (computed from the Wisdom modifier, and if the perception skill was declared as proficient or expertise). If this value is non-empty, override that value with the given one.
challenge / cr / CR	number	(special)	If a value is provided, this sets the display value of the CR. If no value was provided to the CR, one is computed from the proficiency bonus using the usual formula. If no CR was provided and no PB, then the CR defaults to 0. If both a CR and a proficiency bonus are manually provided, then no checks are performed to ensure they 'make sense'. It is possible to have a CR0 creature with a +90 proficiency bonus, if the user manually sets those values. Non-integer values are accepted (i.e. 1/4).
xp	integer	(empty)	If this value is non-empty, it is used as the XP gain defeat. If left empty, the XP is manually computed from an approximate formula which is accurate to $\pm 100xp$ up to CR16: $XP = \begin{cases} 100 + 50 \left(CR^2 + CR + 2CR \lfloor \frac{CR}{7} \rfloor \right) - 7 \lfloor \frac{CR}{7} \rfloor^2 - 5 \lfloor \frac{CR}{7} \rfloor & CR \geq 1 \\ 100 \times CR & \text{else} \end{cases}$ (This is a modified and analytically simplified version of the formula found on a helpful discussion online.)
condition-immunities	text	(empty)	Any condition immunities possessed by the creature.
damage-immunities	text	(empty)	Any damage immunities possessed by the creature.
damage-resistances	text	(empty)	Any damage resistances possessed by the creature.
damage-vulnerabilities	text	(empty)	Any damage vulnerabilities possessed by the creature.

ABILITIES, ATTACKS & SPELLCASTING

The body of the environment is where the user places the traits, abilities and attacks that the creature can use. In order to format these, we provide the following commands.

ABILITY INTERFACE

GLOBAL & LOCAL NAMES

All of the commands we will introduce below follow the established syntax for the `rpgtex` library: `\RpgStat[Command]`. These are placed in the global namespace, and are accessible anywhere - a user may invoke `RpgStat` elements outside of the `RpgStat` environment.

This naming convention - though consistent and avoiding 'common name collisions' - can become cumbersome when used in a dense environment like the `RpgStat`. We therefore provide several *local aliases*. These are macros which can only be used inside the `RpgStat` environment.

With only one exception^a, these aliases are created when the environment begins, using the `\ProvideDocumentCommand` interface -- this means that the alias will only be created if a macro with that name *doesn't exist*.

The local alias is given in **red text**.

^a`\section`, because that will obviously already be defined!

`\RpgStatSection` Creates a section header

`\section`
{m}

`\RpgStatSection`{title-text} body-text

TITLE-TEXT

body-text

Although called a 'section', the visual is most similar to the `\subsection`, though with the text rendered in the `\RpgFontStatBlockSection` font.

`\RpgStatAction` Creates an action 'paragraph' for declaring traits and actions

`\action`
`\trait`
{0{ } m}

`\RpgStatAction`{Trait} body-text

Trait. body-text

`\RpgStatAction`[<cost>]{Action} body-text

Action (<cost> actions). body-text

If a non-empty optional argument is passed, it inserts "[X] actions" into the action name. This is useful for Legendary Actions which might have a variable cost.

`\RpgStatReaction` Creates an action 'paragraph' with an additional 'trigger', indicating when the reaction may be used.

`\reaction`
{m m}

`\RpgStatReaction`{Reaction}{<trigger>}
body-text

Reaction. <trigger>: body-text

`\RpgStatReaction`{Reaction}{} body-text

Reaction. body-text

%empty trigger text does not render the colon

ATTACKS

A thing!

test

```
\begin{RpgStat}{Bom Tombadil}[
  %basics
  type={Medium Celestial, Chaotic Good},
  armor-class = {14},
  hp = {\RpgDice{12d12 + 40}},
  speed = {35 ft.},
  %stats
  str=10, dex= 13, con = 15, int=9, wis
    = 24, cha = 22,
  cha-save, wis-save,
  %skills etc.
  proficiency=8,
  skills={nature (+8)},
  skills-expertise={Performance},
  damage-immunities={all damage
    inflicted by creatures with the
    \textit{Beast} type.},
]
\section{Traits}
\trait{Pacifist} \RpgStatName{}
  cannot deal lethal damage. A
  creature reduced to OHP by
  \RpgStatName{} does not die, but
  instead falls \textit{unconscious},
  even if they are immune to this
  condition.
\section{Actions}
\save{Lullaby}[
  modifier=wis,
  dc=22,
  target=sphere,
  size=30ft,
  failure= {Fall
    \textit{unconscious} for
    \RpgDice{1d4} hours, until they
    take damage, or a creature uses
    an action to shake them awake.},
] Creatures with the
  \textit{deafened} condition or
  \textit{charm} immunity are
  unaffected.
\melee{Begone!}[
  type=spell, modifier=cha,
  dmg=\RpgDice{3d4+\stat{cha}},
  dmg-type=force,
  extra={A creature not native to
    the current plane must make a DC
    22 Charisma saving throw or be
    banished to their native plane}
]
\RpgStatLegendarySection{}
\RpgStatLegendaryDefiance{}
\action{Rebuke the Violent} A
  creature which took hostile action
  in the preceeding turn takes
  \RpgDice{2d10} psychic damage.
\end{RpgStat}
```

BOM TOMBADIL

Medium Celestial, Chaotic Good

AC	14	Speed	35 ft.
HP	118	Initiative	+1
	(12d12+40)		

	STR (10)	DEX (13)	CON (15)	INT (9)	WIS (24)	CHA (22)
MOD	+0	+1	+2	-1	+7	+6
SAVE					+15	+14

Skills Performance (+22), Nature (+8)

Damage Immunities all damage inflicted by creatures with the *Beast* type.

Senses passive Perception 17

Challenge 25 (36000xp, proficiency: +8)

TRAITS

Pacifist. Bom Tombadil cannot deal lethal damage. A creature reduced to OHP by Bom Tombadil does not die, but instead falls *unconscious*, even if they are immune to this condition.

ACTIONS

Lullaby. All enemies within a 30ft sphere must make a DC 22 Wisdom Save.

On Failure: Fall *unconscious* for 2 (1d4) hours, until they take damage, or a creature uses an action to shake them awake.

Creatures with the *deafened* condition or *charm* immunity are unaffected.

Begone! *Melee Spell (5ft):* +14 to hit. *On Hit:* 13 (3d4+6) force damage. A creature not native to the current plane must make a DC 22 Charisma saving throw or be banished to their native plane.

LEGENDARY ACTIONS (3 USES)

Immediately after another creature's turn, Bom Tombadil can expend a Legendary Action to take one of the following actions. Expended actions are regained at the start of each of its turns.

Legendary Defiance. Bom Tombadil repeats a saving throw that they failed in the past hour (using their original modifier). On a success the effect is ended.

Rebuke the Violent. A creature which took hostile action in the preceeding turn takes 11 (2d10) psychic damage.

LEGACY APPEARANCE

The appearance and structure of the RpgStat block is based on the improvements made by the 2024 rules release (with a few modifications of our own). A version that closely aligns with the original 2014 version of the rules can be activated:

`\RpgStatLegacyMode` Overrides the appearance of the RpgStat environment with 2014-style ones.

```
% call outside the environment
\RpgStatLegacyMode{
% subsequent stats use the old visual style
\begin{RpgStat} ...
```

The overrides are purely stylistic; the underlying computation remains the same. The changes persist within a local group.

```
%new style active before these braces
{
  \RpgStatLegacyMode{
  \begin{RpgStat}{Commoner}[
    type={Medium or Small Humanoid;
      typically neutral},
    languages=Common
  ]
  \section{Traits}
    \action{Mob Mentality} If a
      Commoner is subjected to a
      mind-altering effect (such as
      the frightened condition, or a
      charm spell), all Commoners
      within 10ft are also subjected
      to this effect.
  \section{Actions}
    \melee{Club}
  \end{RpgStat}

  % legacy style persists within these
  braces
}

%outside the changes revert to the new style
```

COMMONER

Medium or Small Humanoid; typically neutral

AC	10
HP	4 (1d8)
Speed	30ft

STR	DEX	CON	INT	WIS	CHA
10 (+0)	10 (+0)	10 (+0)	10 (+0)	10 (+0)	10 (+0)

Skills

Senses passive Perception 10

Languages Common

Challenge 0 (0xp)

TRAITS

Mob Mentality. If a Commoner is subjected to a mind-altering effect (such as the frightened condition, or a charm spell), all Commoners within 10ft are also subjected to this effect.

ACTIONS

Club. *Melee (5ft):* +2 to hit. *On Hit:* 2 (1d4) bludgeoning damage. .

Chapter 7: `scifi` Theme

`scifi RpgItem`

PART 3

rpgtex For Designers

Chapter 8: 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 48) -set tikz code. If false, it uses the value sent to `\RpgSetTitleHeader` (page 48) 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 `fancy`), 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 9: 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 `{0{}} m 0{}}`. There should always be

three arguments, which always have the same meanings, but the specification might force some arguments to be mandatory (i.e. `{0{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 56)), 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 52) 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 56) 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
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
{m m +m}`

```
\[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 21):
Returns the value passed to the current environment with identifier `key-name`

`\getkey
{m}`

```
\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>}
    }
  (...)
}
```

^aThere'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 15) 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.

^aThat 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

@subtitle, 7

abstract, 24
action, 43
area, *see* RpgMap, area

backmatter, 23
bg, 2
bookman, 32

cardbreak, 11
columns, 2
cover, 7

emph, 8

FeatureForge
 AddProperty, 55
 CardFormat, 54
 RpgMakeFeature, 52
 ShowCard, 54
 TextFormat, 54

Fonts, *see* RpgFont[X]

footnote, 11
frontmatter, 23

getkey, 21, 55
gillius, 32

justified, 2

kepler, 32
keplerserif, 32
key, 8

lrfont, 27

mainmatter, 23
maketitle, 7

nomultitoc, 2

oneside, 2

part, 7

reaction, 43
Royal, 32

RpgCantripScaling, 37

RpgCard, 11
 Reset, 12
 Set, 12

RpgCMD, 4
RpgContour, 8
RpgDice, 9, 28, 34
 Format, 50

RpgDropCap, 8

RpgFakeChapter, 10

RpgFeat, 20
 AddProperty, 55
 CardFormat, 54
 ShowCard, 21, 54
 TextFormat, 54
 Theme
 default, 30, 36
 dnd, 36

RpgFiligree, 20

rpgfiligree, 20

RpgFont[X], 5

RpgGetName, 3

RpgItem, 20
 AddProperty, 55
 CardFormat, 54
 ShowCard, 21, 54
 TextFormat, 54
 Theme
 default, 29
 dnd, 35
 scifi, 46

rpglatex, vi

RpgLayoutOnly, 50

RpgMakeFeature, *see* FeatureForge

RpgMap, 13
 area, 13
 Ref, 14
 RefPage, 15
 ShowRefs, 15

RpgNarration, 18

RpgOrdinal, 10

RpgPackagePath, v

RpgPage, 10

RpgPlural, 10

RpgSecret, 15
 Visible, 16

RpgSet
 Font, 50

Paper, 49

PartPage, 49

Theme, 9

 Path, 9

ThemeColor, 50

Title
 Cover, 48
 Header, 48
 Mode, 49

RpgSetFooterDecoration, 49

RpgSidebar, 19

RpgSpell, 20
 AddProperty, 55
 CardFormat, 54
 ShowCard, 21, 54
 TextFormat, 54
 Theme
 default, 31
 dnd, 37

RpgSpellUpcast, 37

RpgStat
 Action, 43
 Reaction, 43
 Section, 43
 Theme
 default, 31
 dnd, 39

RpgStatblock
 AddProperty, 55
 CardFormat, 54
 ShowCard, 54
 TextFormat, 54

RpgStatLegacyMode, 45

RpgSwitch
 (command), 56
 Env, 56

RpgTable, 17

RpgTip, 19

section, 43

size, 2

subtitle, 7

summary, 24

theme, 2

themepath, 2

trait, 43