cardgame — Typeset custom game cards*

Adrian Rettich†

Released 2026-04-19

Abstract

The cardgame package allows you to define 'cards', which have various attributes like image, text, and a bunch of layout features. You can then include those cards as images or, more importantly, generate a document containing all of the cards you defined laid out optimally for printing and cutting. You can optionally create card lists for your game's manual, and you can create multiple copies of cards and automatically print the correct number of back sides.

Contents

*This file describes version v1.0, last revised 2026-04-19.

†E-mail: latex@homeworld.space

1 Introduction

The main use of this package is as follows: You have created a board or card game. You want to play it. You do not want to use handwritten cards anymore. Using the cardgame package, you can input your cards in a reasonably intuitive way and let LaTeX handle the layout for you. The package produces a PDF file that contains all of your cards, ready to be cut out. The size is such that the cards fit into standard gaming sleeves; the easiest way to use them without printing on expensive card stock is to put cheap Magic: the Gathering cards or any other pre-existing cards into sleeves, then slide your own cards on top of them. You may use sleeves with transparent back in order to have your own card back design; in this case the generated PDF file contains the correct amount of card backs for you to cut out.

2 Installation

The package is supplied in `dtx` format and as a pre-extracted zip file, `cardgame.zip`. The latter is most convenient for most users: simply unzip this in your local texmf directory and run `texhash` to update the database of file locations. If you want to unpack the `dtx` yourself, running `tex cardgame.dtx` will extract the package whereas `latex cardgame.dtx` or `pdflatex cardgame.dtx` will extract it and also typeset the documentation.

The package requires LaTeX3 support as provided in the l3kernel and l3packages bundles. Both of these are available on CTAN as ready-to-install zip files. Suitable versions are available in MiKTeX 2.9 and TeX Live 2011 (updating the relevant packages online may be necessary). LaTeX3, and so cardgame, requires the e-TeX extensions: these are available on all modern TeX systems.

Typesetting the documentation requires a number of packages in addition to those needed to use the package. To compile the documentation without error, you will need the following packages:

- listings
- float

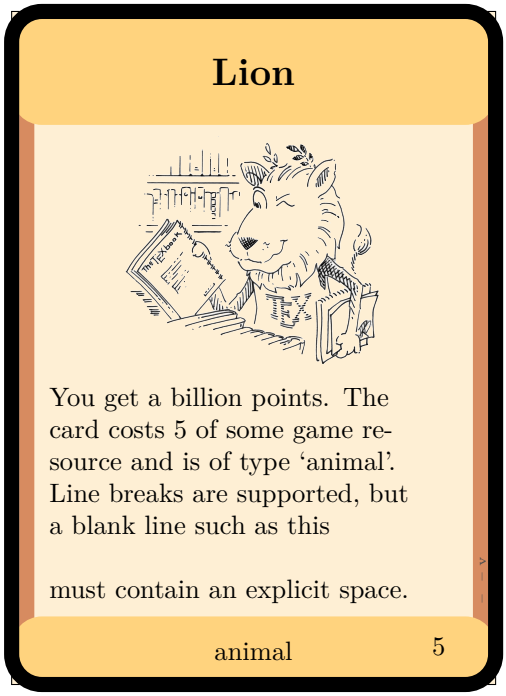
3 Quick Start

If you just want to get to making basic cards and do not need any advanced features, this section provides basic code you can copy and paste into your own document.

```
\usepackage[<options>]{cardgame}
```

You can generate a simple playing card as follows. For the artwork, a PNG with transparency is recommended, but you can of course use any image file. Our examples use the excellent CTAN lion drawn by Duane Bibby. A card has two mandatory arguments: a name and a text. All other options are specified as key-value pairs in the optional argument.

```
  \Card[
  art=lion.png,
  south east content=5,
  tags=animal
  ] {Lion} {
  You get a billion points. The card costs 5 of some game resource
  and is of type ‘animal’.\\
  Line breaks are supported, but a blank line such as this\\\ \\
  must contain an explicit space.
}
  \begin{figure}[H]
    \ShowCard
  \end{figure}
```



Line breaks are supported via explicit backslashes. Do not attempt to have a blank line inside your text argument; it will break because everything on a card is happening inside a TikZ picture. Always use the backslashes.

You can customise many of the features of a card; they are documented in the later sections of this manual. Here is another card showcasing some of these features. Note how you can pass name and text as options if you leave the mandatory arguments empty.

```
\Card[
name=Landscape Lion,
art=lion.png,
text={
This card is in landscape mode.\\
Note how the tags are sorted automatically to have
the same order on all of your cards.},
orientation=landscape,
north east content=5,
north east pre=cost:,
tags={predator,animal},
background colour=Lavender!20,
title colour=BlueViolet!50,
title gradient=BrickRed,
tags colour=BlueViolet!50,
tags gradient=DarkOrchid!50,
frame colour=Blue,
border colour=darkgray
]{}{}
\begin{figure}[H]
  \rotatebox{270}{
  \ShowCard
}
\end{figure}
```

The keys `background color` and so forth are provided as an alternative to the default spelling.

Note how both cards are *defined* before they are ever put inside a figure; the command `\ShowCard` is a convenience function that immediately typesets the most recently defined card. Mostly, if you are designing a game, you just want to print all the cards on as few sheets of cardstock as possible. To this end, you do not use the `\ShowCard` macro. Rather, first define all of your cards. If you want to have more than one copy of a card, simply include `copies=N` for the desired number `N` in the `\Card` call.

Once you have defined all of your cards, call `\PrintCards` to generate a list of *all* cards you have created. Note that you must compile your document twice for this to

work.

You can easily keep your cards sorted in different files by simply having files filled with `\Card` commands and using `\input` to pull them into one PDF file.

By default, cards are sorted by type, subtype, and then title, and one back side is created for each card. You can suppress sorting by passing `sort=false` to `\Cardlist` as an optional argument, and you can suppress creation of back sides by passing `backs=false`.

You can avoid copying options that should be the same for each card by using the `\DeclareCardType` macro.

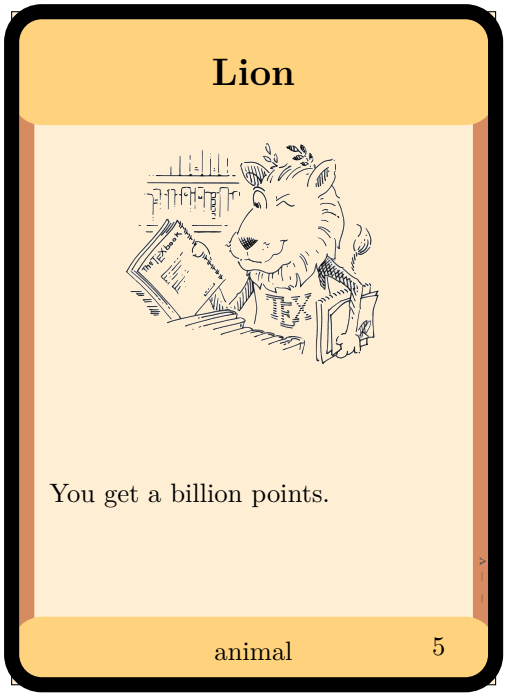
```
  \DeclareCardType{GreenCard}{
  title colour=green
}
```

This provides you with a new command `\GreenCard`, which acts exactly like `\Card` except that it has a green title box by default. If `\DeclareCardType` receives the name of an existing card type as an optional argument, it inherits all options from that card type. Each option can still be overwritten in individual calls.

If you want to include some of your cards in your game manual, give them a `uid`.

```
  \Card[
  art=lion.png,
  south east content=5,
  tags=animal,
  uid=truelion
  ]{Lion}{
  You get a billion points.
}
  \Card[
  art=lion.png,
  ]{Tiger}{
  You lose all your points.
}
```

Now, even though the last card we defined is “tiger”, we can get our lion back by passing its `uid` to the `\ShowCard` command.




```
\begin{figure}[H]
  \ShowCard[truelion]
\end{figure}
```

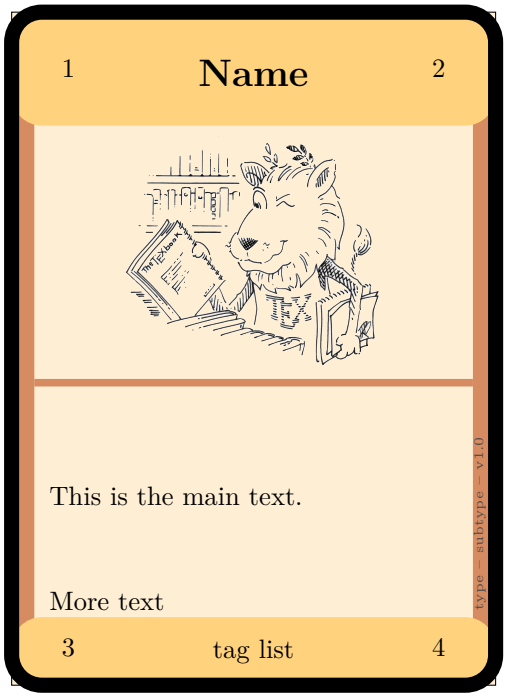
`uid` stands for “unique identifier”, and as such, the package will throw an error if you try to create a second card with the same `uid`, whereas multiple cards with the same `name` are allowed.

This concludes the introductory part of this documentation. Read on if you want to know about the full set of features.

4 The Parts of a Card

In order to communicate information about typesetting of game cards, it is helpful to establish some shared terminology.

All cards generated by cardgame have the following layout.



The parts of this card are as follows.

Border The black outline of the card, showing you where to cut. It should probably not carry information.

Title Box The coloured box at the top, containing the numbers 1 and 2 and the word "name".

Tag Box The bottom box, containing the numbers 3 and 4 and the words "tag list".

Frame The vertical orange bars on the left and right. Do not confuse the frame with the border; the frame's colour usually changes with the card colour, while the border is always black.

Main Area The central area containing the artwork and main text. In this card, it is split in half by a horizontal divider, called the card's separator. In landscape mode, the split is vertical instead.

Artwork The area above the divider line, usually occupied by a picture.

Text The area below the divider line.

Name The card's name is displayed in the centre of the title box.

Tags A card can have any number of "tags" passed as a comma-separated list. How you use them is up to you; they could for example be subtypes. The list is sorted alphabetically and by default is displayed in the tag box, though you can change this.

Corners The card has four corners. Each of the corners can display content. They are called north west (1), north east (2), south west (3), and south east (4).

Playtesting Information If you look carefully, you can see the string "type – subtype – v1.0" displayed in the eastern frame. This optional information tells you the type (and subtype) of card and its newest revision, useful when you are playtesting a game and updating cards as you go. The version information lets you check quickly whether the printed version is currently up to date. This text does not appear by default; set the `playtest` package option to enable it.

Apart from those visible elements, each card has additional attributes that may or may not be visible. They are explained in detail later, but the most important are the following ones.

Type Each card has a "type". This could affect which of various piles in a game the card is a part of. It is not usually displayed on the card itself since it is easier to distinguish cards by colours or symbols. When generating a list of cards, they are sorted by type first.

Subtype Cards can also have a subtype. If you want cards with multiple subtypes, then use the tag list instead. An option lets you control whether to display the subtype in the tag box.

Cost Cards can carry costs. These can, but need not, be numbers, and they can be displayed in one or more corners of the card.

Level Cards can have a level. This is a non-negative integer and can be displayed in one or more corners.

Mark This can contain any text you like. It can be displayed in one or more corners.

Secondary Mark Same principle as mark.

Everything we just talked about makes up the *front side* or *face* of the card. A card also has a *back*, which has the same parts, except that there is no main text and the artwork is centred on the card.

5 Defining Cards

5.1 Options

All card-making macros accept all options listed in this section. Options are given as a comma-separated list of key-value pairs in the form `<key>=<value>` or `<key>={<value>}`. Enclosing the value in curly braces is necessary if it contains a comma or an equals sign.

For options that define boolean values (those that take either `true` or `false` as their value), the equals sign and value can be omitted. Passing `<someboolean>` is the same as passing `<someboolean>=true`.

5.1.1 Options Defining Card Text

`name` The title of the card. If this is too long, it may be hard to fit into the title box. See `modify name` and `title box content` for when that happens. The name can also be passed as the first mandatory argument of the `\Card` macro.

`type` The "type" of card. In reality, this can be any text. It is the primary key for sorting cards. You might want to display this on the back of the card.

`subtype` The secondary key when sorting. Again, this can be any text. If your game is very

complex, you might still want this on the back of the card in order to sort your various draw piles.

`text` The main text of the card. Can also be passed as the second mandatory argument of the `\Card` macro. Text is wrapped automatically. If you want to force a line break, use `\\`. Blank lines in the input are not allowed. If you want to force a blank line in your text, either use the `more text` option below or put a single explicit space on the blank line with a construction like `\\\ \\`.

`more text` Text passed to this option is typeset in the main text area, but at the bottom. The effect is as if you had a `\vfill` between `text` and `more text`. Of course, it is your responsibility to ensure that both texts fit on the card; if they are too long, cardgame will print them overlapping.

`art` This option takes the name of a file that LaTeX can import as a graphic. A PNG file with transparency works well if you want to still see the background colour of the card. Otherwise, your image should bring its own frame *or* you will have to make sure that it has exactly the right dimensions in order to not look strange.

In portrait format, images are scaled (keeping the aspect ratio) to a maximum size of 56mm times 31mm. In landscape format, the maximum size is 40mm times 38mm.

`back art` This option takes the name of a file that LaTeX can import as a graphic. It is set centred on the back of the card. It is scaled just like the face art.

`level` This option takes a non-negative integer, which can be displayed in various places. If it exists, then it becomes the secondary key in sorting cards and the `subtype` is demoted to tertiary key.

`cost` This option is meant to define what game resources a card costs. It can be an integer, but it can also be arbitrary text. It can be displayed in various places.

`mark` This option takes arbitrary text and sets the `mark` attribute of the card. It has no inherent meaning, but it can be displayed in various places.

`secondary mark` This option takes arbitrary text and sets the `secondary mark` attribute of the card. It has no inherent meaning, but it can be displayed in various places.

`tags` This option takes a comma-separated list of tags, all of which are added to the card. By default, tags are sorted alphabetically and displayed in the tag box.

When specifying more than one tag, remember to enclose your list in curly braces. If you really want to, you can specify tags that include commas by nesting braces; for example, `tags={c,{b,a}}` would be sorted as `b,a,c` because `b,a` is a single tag.

5.1.2 Options Modifying the Display of Text

`modify name` Content of this field is executed right before the name of the card is typeset on the card. This allows you to, for example, change the font size for very long names without modifying the name itself, so it is still read correctly by macros like `\ListCards`. By default, the card name uses `\Large\bf`. Use `modify name=` (an empty value) to produce a non-bold normalsize card title.

Nota bene, because this argument is passed to `TiKZ`, you must use the deprecated series of commands `\bf`, `\it`, and so forth to change the font face. The modern macros `\textbf`, `\textit`, and so forth will lead to errors.[1]

`modify text` The same as `modify name`, except it modifies the main text of the card and is empty by default.

`modify more text` The same as `modify text`, but for the `more text` field at the bottom of the text

[1] If someone knows how to do this better, feel free to submit a pull request or just send your solution to `latex@homeworld.space`. Maybe using `TikZ`'s `font` feature is not the correct approach in the first place?

box.

`modify <corner>` `modify <corner> pre` `modify <corner> post`

In each of these options, `<corner>` is a corner specification as described in section 5.1.3. The options modify the main text respectively the `pre` or `post` text of that corner just as `modify text` modifies the main text. The default value of `modify <corner>` is empty, while the other two use `\tiny` by default.

5.1.3 Options Controlling the Placement of Text

The options in this section might be difficult to understand without pictures, so there is an example using most of them at the end of the section.

Apart from the main parts of the card (name, text, and more text) there are six places available for displaying information. These are the centre of the title box, the centre of the tag box, and the four corners. Corners are specified by compass directions: the four corners are `north west`, `north east`, `south west`, and `south east`. Every occurrence of the string `<corner>` in this section can be replaced by any of those four specifications. In landscape format, the cardinal directions are oriented such that the card name is north, not such that the top of the page is north.

In each of these places, you can of course put some text manually. However, you also have the option to link certain information about the card to them, as follows.

`title box` `tag box` `<corner>`

Set these options to link card attributes to the specified location. Possible values are `name`, `type`, `subtype`, `tags`, `level`, `cost`, `mark`, and `secondmark`. For example, the option `north west=mark` means that the value of the `mark` option is set in the upper right corner. This is useful when you define your own card types (see section 5.3). You can set any of these options to be empty in order to not display anything in that position.

The default value of `title box` is `name`. The default value of `tag box` is `tags`. The corners are empty by default.

Even if a link is specified, you can override it on a case by case basis by setting the appropriate `content` option, described next.

`back title box` `back tag box` `back <corner>`

These options work just like those just described, but place text on the back of the card. By default, the `back title box` contains the `type` of the card, while the others are empty.

`title box content` `tag box content` `<corner> content`

These options override the automatic content of the specified location. If you want to put something in the corner of just one card, it is easier to specify this.

You can also use this to shorten the display name of a card while retaining its full name for card lists.

`<corner> pre`

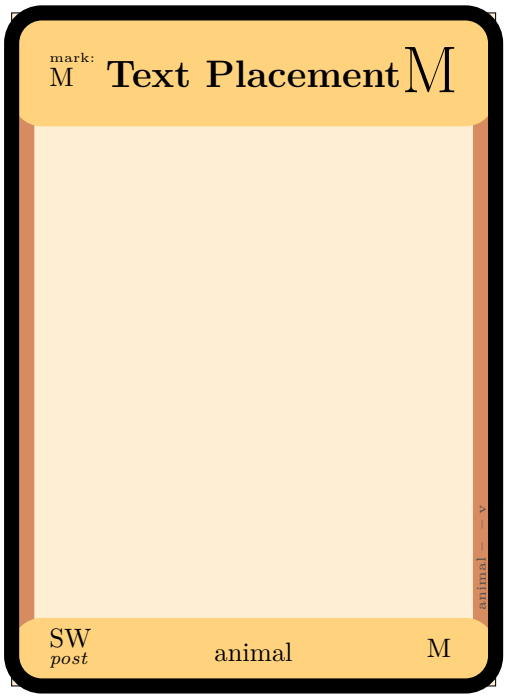
The value of this option is set before the content of the corner, on a separate line. It has a smaller font size by default, but this can be overridden via the `modify <corner> pre` option.

`<corner> post`

Same as the previous option, but set after the corner content.

Here is an example using the options just described. Note how the `mark` is used to place the same content in multiple places without duplicating it in the code.

```
\Card[
type=animal,
mark=M,
tag box=type,
north west=mark,
north east=mark,
modify north east=\Huge,
north west pre=mark:,
south west content=SW,
south west post=post,
south east=mark,
modify south west post=\it\scriptsize,
] {Text Placement} {}
\begin{figure}[H]
  \ShowCard
\end{figure}
```



5.1.4 Colours

cardgame depends on the xcolor package and automatically loads it with the `dvipsnames` option.

For every option containing the word `colour`, a US English equivalent (`color`) is available.

`link colours` Set this Boolean option to `true` in order to link the colours on the back of the card to the colours on the face. By default, they are controlled separately. If this is `true`, then setting any of the subsequent options from this section for the face of the card automatically sets the same option for the back. However, each option can still be overridden by explicitly setting it for the back side.

All other options described in this section take as their value a colour name. This name must be fit to go into a `\colorlet` macro (see the xcolor documentation for details). As such, `black`, `ForestGreen`, and `Dandelion!50` are valid, but if you want to define your own colour by RGB values, you must use xcolor's `\definecolor` first.

Some parts of the card allow gradients. If you specify both a colour and a gradient for a part of the card, then that part will be coloured with a gradient such that the `colour` option is further inside and the `gradient` option is closer to the border of the card. There is an example card at the end of this section.

`background colour`
`back background colour`
This is the background of the main area of the card. Choose a light colour to ensure your text is legible. The default for this is `Dandelion!20`.

`title colour`
`title gradient`
`back title colour`
`back title gradient`
Colour for the title box. The default for both of these is `Dandelion!60`.

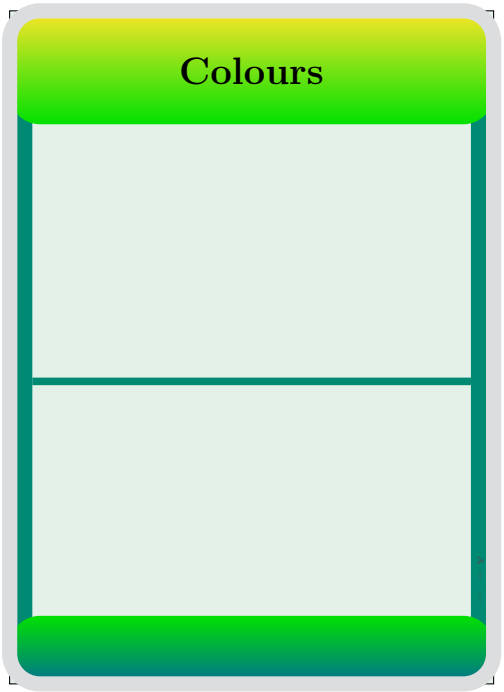
`tags colour`
`tags gradient`
`back tags colour`
`back tags gradient`
Colour for the tag box. If empty, these default to the corresponding `title` value.

`frame colour`
`back frame colour`
The frame's colour. The default value for this is `Bittersweet!60`.

`separator colour`
`back separator colour`
If you set the `separator` option, this controls the colour of the separator bar. If empty, it defaults to the value of `frame colour`.

`border colour`
`back border colour`
The colour for the outer border of the card. The default is `black`.

Here is a card using some of the options described above.




```
\Card[
background colour=ForestGreen!10,
frame colour=PineGreen,
title colour=ForestGreen,
title gradient=Goldenrod,
tags colour=ForestGreen,
tags gradient=MidnightBlue,
border colour=Gray!30,
separator
] {Colours} {}
\begin{figure}[H]
  \ShowCard
\end{figure}
```

5.1.5 Options Controlling the Overall Layout

`orientation` This must be either `portrait` or `landscape`. The default is `portrait`.

`separator` `back separator` These are boolean options. They make the separator in the middle of the main area appear on the face respectively the back of the card.

`rounded corners` This is a boolean option that is `true` by default. Setting `rounded corners=false` disables the rounded corners on the outside of the card border. It does *not* affect any of the corners of boxes on the card, only the part where you cut it. This can prevent ugly white corners around the printed cards if you want to cut your cards with a machine but do not have a corner rounder available.

`back` This is a boolean option and is `false` by default. If set, *only* the back of the card is generated. It is unlikely that you will ever want to set this option; the `\PrintCards` macro can automatically generate the back sides of all cards.

5.1.6 Invisible Attributes

The options described in this section set various attributes of the card which do not usually appear on the card itself, but can be used for control flow, debugging, and by the `\ListCards` macro.

`uid` Set this option to a unique string to later be able to fetch this specific card from your card list. For example, you may have a document containing dozens or hundreds of cards, but in the manual for your game you want to show just one of them as an example. Instead of recreating the card for the manual, give it a uid and `\input` your card list, then typeset only that card by passing its uid to the `\ShowCard` macro.

IDs must be unique. If you specify the same `uid` for more than one card, compilation will abort and tell you to fix it.

Do not use integers as your uids, because cardgame uses them internally. It might work, but it might also not.

`version` You can assign a version number to each card (or, of course, use a card type definition to set all your cards to the same version) in order to track changes while playtesting. If the `playtest` package option is set, then each card's version number is displayed in its frame.

`rulings` Set this option to describe any additional rules, specific interactions with other cards, or FAQs that are too long to appear on the card itself. You can use the `\ListCards` macro to display these rulings in your game's manual.

`copies` Set this to a positive integer to have the `\PrintCards` macro generate that many copies of the card (and that many copies of the back as well).

5.2 Card-Defining Macros

`\Card` The main command to make a card is the `\Card` macro.

`\Card [`⟨*options*⟩`] {`⟨*name*⟩`} {`⟨*text*⟩`}`

A card only has two mandatory parts: a name (usually displayed in the title box) and a text. However, either or both of those mandatory arguments can be empty. If any of them are empty, then the card instead uses its `name` respectively `text` options, if given. If those are empty as well, then the card is generated without a name respectively text, which is perfectly legal.

`options` is a comma-separated list of key-value pairs, as described in the previous sections.

5.3 Defining New Card Types

In order to avoid duplicating information in your document, you can define new "card types" that share any number of default options. This has nothing to do with the `type` attribute of a card, though it would be natural for a card type to have a shared `type` as well.

`\DeclareCardType`

```
\DeclareCardType [⟨type⟩] {⟨name⟩} {⟨defaults⟩}
```

This command defines a new macro `\<name>` (or raises an error if that command already exists). The new macro behaves exactly like the `\Card` macro, except that any key-value pairs you pass to the final argument of `\DeclareCardType` become the default values for cards created by the new macro. You can still overwrite them on a case by case basis.

Card types can inherit from each other: if you pass the name of an existing card type (created by an earlier `\DeclareCardType`) as the optional argument to `\DeclareCardType`, then all the options of that existing type are imported into the new type. You can override any number of them by passing new values in the `<defaults>` argument.

For both the `<type>` and the `<name>` option, do not include the backslash. That is, use `\DeclareCardType[type]{name}`, not `\DeclareCardType[\type]{\name}`.

6 Outputting Things

The commands in this section are the only ones that actually typeset something. In order to output a card, you must first have defined it as described in the previous section.

`\ShowCard`

```
\ShowCard [⟨uid⟩]
```

This macro typesets a single card. It is equivalent to a call to `\tikz` and as such you probably want to wrap it in a `figure` environment.

If no argument is given, then the most recently defined card is shown. Otherwise, the argument should be the `uid` of an existing card; that card is shown.

`\PrintCards`

```
\PrintCards [⟨options⟩]
```

This macro typesets *all* the cards you have defined in this document.

Cards are printed flush with each other, nine to a page (the maximum possible for a home printer).

If you call this in a document that also contains other things, you probably want to call `\newpage` first.

`<options>` is a list of key-value pairs. The following options are recognised.

`sort` This is a boolean. The default is `true` and sorts the cards by type, then level, then subtype, then name. All faces are printed first, then all back sides (if enabled). If `false`, then cards appear in the order in which they are defined in the document, though back sides still appear last.

`backs` This is a boolean. The default is `true`. If `false`, only faces are generated, no back sides.

`\PrintCards` requires two compilations to work properly.

`\ListCards`

`\ListCards[`⟨*options*⟩`]`

This macro generates a table with information about *all* cards you have defined. You can include it in your game manual as a reference. It takes one optional key-value argument, `sort`, which has the same effect as for the `\PrintCards` command.

```
\ListCards[sort=false]
```

Name	Text	Rulings
Lion	You get a billion points. The card costs 5 of some game resource and is of type 'animal'. Line breaks are supported, but a blank line such as this must contain an explicit space.	
Landscape Lion	This card is in landscape mode. Note how the tags are sorted automatically to have the same order on all of your cards.	
Lion	You get a billion points.	
Tiger	You lose all your points.	
Name	This is the main text.	
Text Placement		
Colours		

7 Package Options

The cardgame package accepts the following options.

`sort` This sets the default value of the `sort` option for the `\PrintCards` and `\ListCards` commands.

`backs` This sets the default value of the `backs` option for the `\PrintCards` command.

`playtest` A boolean that defaults to `false`. If `true`, include versioning information in the frame of the card. This documentation has been compiled with `playtest=true`.

8 Issues

If you find a bug or have a request, please contact me at `latex@homeworld.space` or open an issue at https://codeberg.org/BettaGeorge/cardgame.

The implementation of `\ListCards` is rudimentary and to be improved in future versions of this package.

9 Implementation

```
1 ⟨*package⟩
2 ⟨@@=cardgame⟩
3 \ProvidesExplPackage {cardgame} {2026-04-19} {1.0}
4 {Typesetting custom game cards}
5 \RequirePackage{l3keys2e}
6 \RequirePackage[dvipsnames]{xcolor}
7 \RequirePackage{tabularray}
```

TikZ libraries cannot be loaded while explicit syntax is on, even though they can be *used* without problem.

```
8 \ExplSyntaxOff
9 \usepackage{tikz}
10 \usetikzlibrary{calc}
11 \ExplSyntaxOn
```

9.1 Variables and constants

`\c__cardgame_file_message_tl`

A short piece of text that is added to the top of an auto-generated file. Setting this up as a constant means that it can be changed (for example for translation) if necessary.

```
12 \tl_const:Nn \c__cardgame_file_message_tl
13 {
14   \iow_char:N \% ~
15   This~is~an~auxiliary~file~used~by~the~'cardgame'~package.
16   \iow_newline:
17   \iow_char:N \% ~
18   This~file~may~safely~be~deleted.
19   \iow_newline:
20   \iow_char:N \% ~
21   It~will~be~recreated~as~required.
22   \iow_newline:
23 }
```

(*End of definition for* `\c__cardgame_file_message_tl`.)

`defaultbgcolour` `defaulttitlecolour` `defaultframecolour` `defaultbordercolour`

The following are the default colours for things, just because we need to do *something* if the user sets no options. I like yellow.

```
24 \colorlet{defaultbgcolour}{Dandelion!20}
25 \colorlet{defaulttitlecolour}{Dandelion!60}
26 \colorlet{defaultframecolour}{Bittersweet!60}
27 \colorlet{defaultbordercolour}{black}
```

(*End of definition for* `defaultbgcolour` *and others. These functions are documented on page* ??.)

These token lists contain what certain things should be called.

```
\tl_const:Nn \const_portrait_tl {portrait}
\tl_const:Nn \const_cost_tl {cost}
\tl_const:Nn \const_mark_tl {mark}
\tl_const:Nn \const_secondarymark_tl {secondary~mark}
\tl_const:Nn \const_level_tl {level}
\tl_const:Nn \const_type_tl {type}
\tl_const:Nn \const_subtype_tl {subtype}
\tl_const:Nn \const_tags_tl {tags}
\tl_const:Nn \const_name_tl {name}
```

These two comma-separated lists save all existing cards. We are going to write precompiled key lists into them, then later call `\tl_use` on each element to recover the values.

```
\clist_new:N \g_cards_clist
\clist_new:N \g_cardbacks_clist
```

This list saves which uids are currently being used, so we can check whether a newly requested uid is in fact unique.

```
\clist_new:N \g_uid_clist
```

The following increments by one each time a card is created and assigns it a unique identifier, if the user has not provided one.

```
\int_new:N\g_uid_int
```

This list tracks which tags *exist* at all. This is not currently useful, but we might want to generate lists of tags or something like that later.

```
\clist_new:N \g_existingtags_clist
```

debugging messages

```
\msg_new:nnn { cardgame } { tagcount } {
  #1:~#2
}
\msg_new:nnn { cardgame } { cornercontent }{
  Unknown~identifier~'#1';~
  typesetting~nothing~and~continuing.
}
\msg_new:nnn { cardgame } { uid }{
  UID~'#1'~is~already~taken;~assigning~
  a~random~one.
}
```

9.2 Key-Value Options

We now set up the various key-value options. All of these are documented in the user part of this manual, so we just list them here without further comment.

First, the options that can be passed to the package itself.

```
\keys_define:nn { cardgame }
{
  sort .bool_set:N = \l_sort_bool,
  sort .initial:n = { true },
  sort .default:n = { true },

```

```
  backs .bool_set:N = \l_generatecardbacks_bool,
  backs .initial:n = { true },
  backs .default:n = { true },

  playtest .bool_set:N = \l_playtestinfo_bool,
  playtest .initial:n = { false },
  playtest .default:n = { true },
}
\ProcessKeysOptions{cardgame}
```

The options that can be passed to list-producing macros (`\PrintCards` and `\ListCards`) overlap with the package options, but we may want to add more in the future.

```
\keys_define:nn { cardlist }
{
  sort .bool_set:N = \l_sort_bool,
  sort .initial:n = { true },
  sort .default:n = { true },

  backs .bool_set:N = \l_generatecardbacks_bool,
  backs .initial:n = { true },
  backs .default:n = { true },
}
```

By far the most expansive list is the one for the `\Card` command. The meta keys are just a trick to make `color` set `colour`.

```
\keys_define:nn { card }
{
  % OPTIONS DEFINING CARD TEXT
  name .tl_set:N = \l_name_tl,
  name .initial:n = {},
  name .default:n = {},

  type .tl_set:N = \l_type_tl,
  type .initial:n = {},
  type .default:n = {},

  subtype .tl_set:N = \l_subtype_tl,
  subtype .initial:n = {},
  subtype .default:n = {},

  text .tl_set:N = \l_text_tl,
  text .initial:n = {},
  % text .default:n = {},

  more~text .tl_set:N = \l_moretext_tl,
  more~text .initial:n = {},
  % more~text .default:n = {},

  art .tl_set:N = \l_art_tl,
  art .initial:n = {},
  art .default:n = {},

  back~art .tl_set:N = \l_backart_tl,
  back~art .initial:n = {},
```

```
107  back~art .default:n = {},
108
109  cost .tl_set:N = \l_cost_tl,
110  cost .initial:n = { 0 },
111  cost .default:n = 0,
112
113  mark .tl_set:N = \l_mark_tl,
114  mark .initial:n = {},
115  mark .default:n = {},
116
117  secondary~mark .tl_set:N = \l_mark_tl,
118  secondary~mark .initial:n = {},
119  secondary~mark .default:n = {},
120
121  level .int_set:N = \l_level_int,
122  level .initial:n = { 0 },
123  level .default:n = 0,
124
125  tags .clist_set:N = \l_tags_clist,
126  tags .initial:n = {},
127  tags .default:n = {},
128
129  % OPTIONS MODIFYING THE DISPLAY OF TEXT
130
131  modify~name .tl_set:N = \l_modifyname_tl,
132  modify~name .initial:n = { \Large\bf },
133  modify~name .default:n = {},
134
135  modify~text .tl_set:N = \l_textsize_tl,
136  modify~text .initial:n = {\normalsize},
137  modify~text .default:n = {\normalsize},
138
139  modify~more~text .tl_set:N = \l_moretextsize_tl,
140  modify~more~text .initial:n = {\normalsize},
141  modify~more~text .default:n = {\normalsize},
142
143  modify~north~west .tl_set:N = \l_nwtextsize_tl,
144  modify~north~west .initial:n = {\normalsize},
145  modify~north~west .default:n = {\normalsize},
146  modify~north~west~pre .tl_set:N = \l_nwpretextsize_tl,
147  modify~north~west~pre .initial:n = {\tiny},
148  modify~north~west~pre .default:n = {\tiny},
149  modify~north~west~post .tl_set:N = \l_nwposttextsize_tl,
150  modify~north~west~post .initial:n = {\tiny},
151  modify~north~west~post .default:n = {\tiny},
152  modify~north~east .tl_set:N = \l_netextsize_tl,
153  modify~north~east .initial:n = {\normalsize},
154  modify~north~east .default:n = {\normalsize},
155  modify~north~east~pre .tl_set:N = \l_nepretextsize_tl,
156  modify~north~east~pre .initial:n = {\tiny},
157  modify~north~east~pre .default:n = {\tiny},
158  modify~north~east~post .tl_set:N = \l_neposttextsize_tl,
159  modify~north~east~post .initial:n = {\tiny},
160  modify~north~east~post .default:n = {\tiny},
```

```
161   modify~south~east .tl_set:N = \l_setextsize_tl,
162   modify~south~east .initial:n = {\normalsize},
163   modify~south~east .default:n = {\normalsize},
164   modify~south~east~pre .tl_set:N = \l_sepretextsize_tl,
165   modify~south~east~pre .initial:n = {\tiny},
166   modify~south~east~pre .default:n = {\tiny},
167   modify~south~east~post .tl_set:N = \l_seposttextsize_tl,
168   modify~south~east~post .initial:n = {\tiny},
169   modify~south~east~post .default:n = {\tiny},
170   modify~south~west .tl_set:N = \l_swtextsize_tl,
171   modify~south~west .initial:n = {\normalsize},
172   modify~south~west .default:n = {\normalsize},
173   modify~south~west~pre .tl_set:N = \l_swpretextsize_tl,
174   modify~south~west~pre .initial:n = {\tiny},
175   modify~south~west~pre .default:n = {\tiny},
176   modify~south~west~post .tl_set:N = \l_swposttextsize_tl,
177   modify~south~west~post .initial:n = {\tiny},
178   modify~south~west~post .default:n = {\tiny},
179
180   % OPTIONS CONTROLLING THE PLACEMENT OF TEXT
181
182   tag~box .tl_set:N = \l_tagbox_tl,
183   tag~box .initial:n = { tags },
184   tag~box .default:n = {},
185   tag~box~content .tl_set:N = \l_tagboxcontent_tl,
186   tag~box~content .initial:n = {},
187   tag~box~content .default:n = {},
188   back~tag~box .tl_set:N = \l_btagbox_tl,
189   back~tag~box .initial:n = {},
190   back~tag~box .default:n = {},
191
192   title~box .tl_set:N = \l_titlebox_tl,
193   title~box .initial:n = { name },
194   title~box .default:n = {},
195   title~box~content .tl_set:N = \l_titleboxcontent_tl,
196   title~box~content .initial:n = {},
197   title~box~content .default:n = {},
198   back~title~box .tl_set:N = \l_btitlebox_tl,
199   back~title~box .initial:n = { type },
200   back~title~box .default:n = {},
201
202   south~west~content .tl_set:N = \l_swcontent_tl,
203   south~west~content .initial:n = {},
204   south~west~content .default:n = {},
205   south~west~pre .tl_set:N = \l_swpre_tl,
206   south~west~pre .initial:n = {},
207   south~west~pre .default:n = {},
208   south~west~post .tl_set:N = \l_swpost_tl,
209   south~west~post .initial:n = {},
210   south~west~post .default:n = {},
211   south~west .tl_set:N = \l_sw_tl,
212   south~west .initial:n = {},
213   south~west .default:n = {},
214   back~south~west .tl_set:N = \l_bsw_tl,
```

```
  back~south~west .initial:n = {},
  back~south~west .default:n = {},

  south~east~content .tl_set:N = \l_secontent_tl,
  south~east~content .initial:n = {},
  south~east~content .default:n = {},
  south~east~pre .tl_set:N = \l_sepre_tl,
  south~east~pre .initial:n = {},
  south~east~pre .default:n = {},
  south~east~post .tl_set:N = \l_sepost_tl,
  south~east~post .initial:n = {},
  south~east~post .default:n = {},
  south~east .tl_set:N = \l_se_tl,
  south~east .initial:n = {},
  south~east .default:n = {},
  back~south~east .tl_set:N = \l_bse_tl,
  back~south~east .initial:n = {},
  back~south~east .default:n = {},

  north~west~content .tl_set:N = \l_nwcontent_tl,
  north~west~content .initial:n = {},
  north~west~content .default:n = {},
  north~west~pre .tl_set:N = \l_nwpre_tl,
  north~west~pre .initial:n = {},
  north~west~pre .default:n = {},
  north~west~post .tl_set:N = \l_nwpost_tl,
  north~west~post .initial:n = {},
  north~west~post .default:n = {},
  north~west .tl_set:N = \l_nw_tl,
  north~west .initial:n = {},
  north~west .default:n = {},
  back~north~west .tl_set:N = \l_bnw_tl,
  back~north~west .initial:n = {},
  back~north~west .default:n = {},

  north~east~content .tl_set:N = \l_necontent_tl,
  north~east~content .initial:n = {},
  north~east~content .default:n = {},
  north~east~pre .tl_set:N = \l_nepre_tl,
  north~east~pre .initial:n = {},
  north~east~pre .default:n = {},
  north~east~post .tl_set:N = \l_nepost_tl,
  north~east~post .initial:n = {},
  north~east~post .default:n = {},
  north~east .tl_set:N = \l_ne_tl,
  north~east .initial:n = {},
  north~east .default:n = {},
  back~north~east .tl_set:N = \l_bne_tl,
  back~north~east .initial:n = {},
  back~north~east .default:n = {},

  % COLOUR OPTIONS

  link~colours .bool_set:N = \l_linkcolours_bool,
```

```
  link~colours .initial:n = { false },
  link~colours .default:n = { true },
  link~colors .meta:n = { link~colours={#1} },

  background~colour .tl_set:N = \l_col_bg_tl,
  background~colour .initial:n = {},
  background~colour .default:n = {},
  background~color .meta:n = { background~colour={#1} },

  title~colour .tl_set:N = \l_col_title_tl,
  title~colour .initial:n = {},
  title~colour .default:n = {},
  title~color .meta:n = { title~colour={#1} },
  title~gradient .tl_set:N = \l_grad_title_tl,
  title~gradient .initial:n = {},
  title~gradient .default:n = {},

  tags~colour .tl_set:N = \l_col_tags_tl,
  tags~colour .initial:n = {},
  tags~colour .default:n = {},
  tags~color .meta:n = { tags~colour={#1} },
  tags~gradient .tl_set:N = \l_grad_tags_tl,
  tags~gradient .initial:n = {},
  tags~gradient .default:n = {},

  frame~colour .tl_set:N = \l_col_frame_tl,
  frame~colour .initial:n = {},
  frame~colour .default:n = {},
  frame~color .meta:n = { frame~colour={#1} },

  separator~colour .tl_set:N = \l_col_separator_tl,
  separator~colour .initial:n = {},
  separator~colour .default:n = {},
  separator~color .meta:n = { frame~colour={#1} },

  border~colour .tl_set:N = \l_col_border_tl,
  border~colour .initial:n = {},
  border~colour .default:n = {},
  border~color .meta:n = { border~colour={#1} },

  back~background~colour .tl_set:N = \l_bcol_bg_tl,
  back~background~colour .initial:n = {},
  back~background~colour .default:n = {},
  back~background~color .meta:n = { back~background~colour={#1} },

  back~title~colour .tl_set:N = \l_bcol_title_tl,
  back~title~colour .initial:n = {},
  back~title~colour .default:n = {},
  back~title~color .meta:n = { back~title~colour={#1} },
  back~title~gradient .tl_set:N = \l_bgrad_title_tl,
  back~title~gradient .initial:n = {},
  back~title~gradient .default:n = {},

  back~tags~colour .tl_set:N = \l_bcol_tags_tl,
```

```
back~tags~colour .initial:n = {},
back~tags~colour .default:n = {},
back~tags~color .meta:n = { back~tags~colour={#1} },
back~tags~gradient .tl_set:N = \l_bgrad_tags_tl,
back~tags~gradient .initial:n = {},
back~tags~gradient .default:n = {},

back~frame~colour .tl_set:N = \l_bcol_frame_tl,
back~frame~colour .initial:n = {},
back~frame~colour .default:n = {},
back~frame~color .meta:n = { back~frame~colour={#1} },

back~separator~colour .tl_set:N = \l_bcol_separator_tl,
back~separator~colour .initial:n = {},
back~separator~colour .default:n = {},
back~separator~color .meta:n = { frame~colour={#1} },

back~border~colour .tl_set:N = \l_bcol_border_tl,
back~border~colour .initial:n = {},
back~border~colour .default:n = {},
back~border~color .meta:n = { back~border~colour={#1} },

% OPTIONS CONTROLLING THE OVERALL LAYOUT

orientation .tl_set:N = \l_orientation_tl,
orientation .initial:n = {portrait},
orientation .default:n = {portrait},

separator .bool_set:N = \l_separator_bool,
separator .initial:n = { false },
separator .default:n = { true },

back~separator .bool_set:N = \l_bseparator_bool,
back~separator .initial:n = { false },
back~separator .default:n = { true },

rounded~corners .bool_set:N = \l_rounded_corners_bool,
rounded~corners .initial:n = { true },
rounded~corners .default:n = { true },

back .bool_set:N = \l_back_bool,
back .initial:n = { false },
back .default:n = { true },

% OPTIONS FOR INVISIBLE ATTRIBUTES

version .tl_set:N = \l_version_tl,
version .initial:n = {},
version .default:n = {},

rulings .tl_set:N = \l_rulings_tl,
rulings .initial:n = {},
rulings .default:n = {},
```

```
  uid .tl_set:N = \l_uid_tl,
  uid .initial:n = {},
  uid .default:n = {},

  copies .int_set:N = \l_copies_int,
  copies .initial:n = { 1 },
  copies .default:n = 1,
}
```

9.3 Helper Functions

`\insert_attribute:n` Use this in places where we want to typeset one of the various attributes of a card based on the user's preference. The sole argument is the string the user passed. For example, if `south west=mark` is set by the user, then while typesetting the south west corner, we call `\insert_attribute:n{mark}`.

```
\cs_new:Nn \insert_attribute:n {
  \tl_case:NnF{#1}{
    \const_tags_tl {
      \clist_use:Nn \l_tags_clist {,~}
    }
    \const_cost_tl {
      \tl_use:N\l_cost_tl
    }
    \const_level_tl {
      \int_use:N\l_level_int
    }
    \const_mark_tl {
      \tl_use:N\l_mark_tl
    }
    \const_secondarymark_tl {
      \tl_use:N\l_secondarymark_tl
    }
    \const_type_tl {
      \tl_use:N\l_type_tl
    }
    \const_subtype_tl {
      \tl_use:N\l_subtype_tl
    }
    \const_name_tl {
      \tl_use:N\l_name_tl
    }
  }{
    \msg_error:nnV { cardgame }{ cornercontent }{ #1 }
  }
}
```

(*End of definition for* `\insert_attribute:n`*. This function is documented on page* ??.)

9.4 Defining Cards

`\card:n` The main command to make a card. It takes all of its arguments via the key-value interface. It stores the card indexed by its type, subtype, and name, such that we can later sort them alphabetically. It also stores a copy of the card indexed by its uid, because

I could not figure out how to use references. We do not typeset the cards yet; this is handled by separate commands. If automatic card back generation is on, the command calls itself once to generate the back side.

```
\cs_generate_variant:Nn \keys_precompile:nnN { nVN }
\cs_new:Nn \card:n {
  \group_begin:
  \keys_set:nn { card } { #1 }
  % first, if a uid was provided, check whether it is valid.
  % if not, assign one.
  % Do not do any of this for backsides though.
  \bool_if:NF \l_back_bool {
    \tl_if_empty:NTF\l_uid_tl {
      \tl_set:NV\l_uid_tl\g_uid_int
      \int_gincr:N\g_uid_int
    }{
      \clist_if_in:NVT\g_uid_clist\l_uid_tl {
        \msg_error:nnV {cardgame} {uid} {\l_uid_tl}
        \tl_set:Ne\l_uid_tl{\int_use:N\g_uid_int}
        \int_incr:N\g_uid_int
      }
    }
    \clist_gput_right:NV \g_uid_clist \l_uid_tl
  }

  % we simply set a new token list to contain all the keys of the card.
  % since we want to process back sides separately, they go in a special
  % back side list.
  \bool_if:NTF \l_back_bool {
    \tl_new:c {
      cardback @
      \l_type_tl @
      \int_use:N\l_level_int @
      \l_subtype_tl @
      \l_name_tl @
      \l_uid_tl
    }
    \keys_precompile:nnN { card } { #1 } \l_tmpa_tl
    % precompile is a local assignment, and there is no global variant.
    \tl_gset_eq:cN {
      cardback @
      \l_type_tl @
      \int_use:N\l_level_int @
      \l_subtype_tl @
      \l_name_tl @
      \l_uid_tl
    } \l_tmpa_tl

    % also, we somehow need to remember which cards exist.
    % Since it is hard to expand multiple copies during the typesetting loop later,
    % we simply copy the card now.
    \int_set:Nn \l_tmpa_int { 0 }
    \int_while_do:nNnn { \l_tmpa_int } < { \l_copies_int } {
      \clist_gput_right:Nx \g_cardbacks_clist {
        cardback @
```

```
        \l_type_tl @
        \int_use:N\l_level_int @
        \l_subtype_tl @
        \l_name_tl @
        \l_uid_tl
      }
      \int_incr:N \l_tmpa_int
    }
  }{
    \tl_new:c {
      \l_type_tl @
      \int_use:N\l_level_int @
      \l_subtype_tl @
      \l_name_tl @
      \l_uid_tl
    }
    \tl_set:Nn\l_tmpb_tl { #1 }
    \tl_put_right:Nx\l_tmpb_tl {, uid=\tl_use:N\l_uid_tl }
    \keys_precompile:nVN { card } \l_tmpb_tl \l_tmpa_tl
    % precompile is a local assignment, and there is no global variant.
    \tl_gset_eq:cN {
      \l_type_tl @
      \int_use:N\l_level_int @
      \l_subtype_tl @
      \l_name_tl @
      \l_uid_tl
    } \l_tmpa_tl
    % We are currently doing a front side, so we also save the card in the uid list.
    \tl_gset_eq:cN { card_by_uid @ \l_uid_tl } \l_tmpa_tl
    % also, we somehow need to remember which cards exist.
    % Since it is hard to expand multiple copies during the typesetting loop later,
    % we simply copy the card now.
    % Yes, this is unnecessary memory overhead, but when will you
    % ever have so many cards that it matters in practice?
    \int_set:Nn \l_tmpa_int { 0 }
    \int_while_do:nNnn { \l_tmpa_int } < { \l_copies_int } {
      \clist_gput_right:Nx \g_cards_clist {
        \l_type_tl @
        \int_use:N\l_level_int @
        \l_subtype_tl @
        \l_name_tl @
        \l_uid_tl
      }
      \int_incr:N \l_tmpa_int
    }
    \bool_if:NT \l_generatecardbacks_bool {
      \card:n { #1, back }
    }
  }
  \group_end:
}
```

(*End of definition for* `\card:n`. *This function is documented on page* **??**.)

9.5 Generating Output

\typeset_card:nnn

`\typeset_card` is what actually TikZes cards. You should not have to call it directly, but in case you do, here is what it does. The first argument is either empty or a card UID as generated by `\card`, that is, the string `card_by_uid@<uid>`. The second argument is a list of key-value pairs. The third argument is either empty or an integer between 0 and 8. The card to be typeset gets all the characteristics of the card specified in the first argument. Then all the keys specified in the second argument override those. If the first argument is empty, then naturally you must specify all characteristics as key-value pairs in the second argument. If the third argument is empty, a TikZ picture is then produced with no special placement. Wrap it in a figure environment or whatever else you want to do with the picture. If the third argument is an integer, the card is placed at that index plus one on the current page using the overlay mechanism. This is used by the `\print_cards` macro.

An M:tG card is 63x88 mm big. We use 64x89 mm to leave some room for error while cutting while still fitting inside a standard sleeve. An A4 page is 210x297 mm. In particular, a 3x3 portrait layout is optimal, since a landscape layout only admits 2x4 cards.

```
\cs_new:Nn \typeset_card:nnn {

  \tl_if_blank:nF {#3} {
    % We calculate the offset of the lower left corner of the card
    % (henceforth referred to as the "base").
    \int_zero_new:N \l_xoffset_int
    \int_zero_new:N \l_yoffset_int
    \int_set:Nn \l_xoffset_int { % x offset in mm
      (
      \int_mod:nn { #3 - 1 } { 3 }
      ) * 66
    }
    \int_set:Nn \l_yoffset_int { % y offset in mm
      (
      2 -
      \int_div_truncate:nn { #3 - 1 } { 3 }
      ) * 91
    }
  }

  \group_begin:
  \tl_if_blank:nF{#1}{
    \tl_use:c { #1 }
  }
  \tl_if_blank:nF{#2}{
    \keys_set:nx { card } { #2 }
  }

  % The tags should always be sorted the same.
  \clist_sort:Nn \l_tags_clist {
    \str_compare:nNnTF { ##1 } < { ##2 } { \sort_return_same: } { \sort_return_swapped: }
  }
  % And let's keep a handy counter of the tags used.
  \bool_if:NF \l_back_bool {
    \clist_map_inline:Nn \l_tags_clist {
```

```
      \int_if_exist:cF { tagcounter @ ##1 }{
        \int_new:c { tagcounter @ ##1 }
        \clist_gput_right:Nn \g_existingtags_clist { ##1 }
      }
      \int_gincr:c { tagcounter @ ##1 }
    }
  }

  % First, we figure out all the colours to use on the face of the card.
  \tl_if_empty:NTF\l_col_bg_tl{
    \colorlet{bg}{defaultbgcolour}
  }{
    \colorlet{bg}{\tl_use:N\l_col_bg_tl}
  }
  \tl_if_empty:NTF\l_col_frame_tl{
    \colorlet{marg}{defaultframecolour}
  }{
    \colorlet{marg}{\tl_use:N\l_col_frame_tl}
  }
  \tl_if_empty:NTF\l_col_separator_tl{
    \colorlet{separator}{marg}
  }{
    \colorlet{separator}{\tl_use:N\l_col_separator_tl}
  }
  \tl_if_empty:NTF\l_col_title_tl{
    \colorlet{box}{defaulttitlecolour}
  }{
    \colorlet{box}{\tl_use:N\l_col_title_tl}
  }
  \tl_if_empty:NTF\l_grad_title_tl{
    \colorlet{boxgradient}{box}
  }{
    \colorlet{boxgradient}{\tl_use:N\l_grad_title_tl}
  }
  \tl_if_empty:NTF\l_col_tags_tl{
    \colorlet{tagscolour}{box}
  }{
    \colorlet{tagscolour}{\tl_use:N\l_col_tags_tl}
  }
  \tl_if_empty:NTF\l_grad_tags_tl{
    \colorlet{tagsgradient}{boxgradient}
  }{
    \colorlet{tagsgradient}{\tl_use:N\l_grad_tags_tl}
  }
  \tl_if_empty:NTF\l_col_border_tl{
    \colorlet{border}{defaultbordercolour}
  }{
    \colorlet{border}{\tl_use:N\l_col_border_tl}
  }

  % Now, if the back colours are linked, copy everything.
  % If not, initialise them to the default colours. They might get overridden
  % in the next step, stay tuned.
  \bool_if:NTF\l_linkcolours_bool {
```

```
    \colorlet{bbg}{bg}
    \colorlet{bmarg}{marg}
    \colorlet{bseparator}{separator}
    \colorlet{bbox}{box}
    \colorlet{bboxgradient}{boxgradient}
    \colorlet{btagscolour}{tagscolour}
    \colorlet{btagsgradient}{tagsgradient}
    \colorlet{bborder}{border}
  }{
    \colorlet{bbg}{defaultbgcolour}
    \colorlet{bmarg}{defaultframecolour}
    \colorlet{bseparator}{bmarg}
    \colorlet{bbox}{defaulttitlecolour}
    \colorlet{bboxgradient}{bbox}
    \colorlet{btagscolour}{bbox}
    \colorlet{btagsgradient}{bboxgradient}
    \colorlet{bborder}{defaultbordercolour}
  }

  % All back colours that were explicitly specified override the above.
  \tl_if_empty:NF\l_bcol_bg_tl{
    \colorlet{bbg}{\tl_use:N\l_bcol_bg_tl}
  }
  \tl_if_empty:NF\l_bcol_frame_tl{
    \colorlet{bmarg}{\tl_use:N\l_bcol_frame_tl}
  }
  \tl_if_empty:NF\l_bcol_separator_tl{
    \colorlet{bseparator}{\tl_use:N\l_bcol_separator_tl}
  }
  \tl_if_empty:NF\l_bcol_title_tl{
    \colorlet{bbox}{\tl_use:N\l_bcol_title_tl}
  }
  \tl_if_empty:NF\l_bgrad_title_tl{
    \colorlet{bboxgradient}{\tl_use:N\l_bgrad_title_tl}
  }
  \tl_if_empty:NF\l_bcol_tags_tl{
    \colorlet{btagscolour}{\tl_use:N\l_bcol_tags_tl}
  }
  \tl_if_empty:NF\l_bgrad_tags_tl{
    \colorlet{btagsgradient}{\tl_use:N\l_bgrad_tags_tl}
  }
  \tl_if_empty:NF\l_bcol_border_tl{
    \colorlet{bborder}{\tl_use:N\l_bcol_border_tl}
  }

  % Let the drawing begin.
  \tikz[
  remember~picture=\tl_if_blank:nTF{#3}{false}{true},
  overlay=\tl_if_blank:nTF{#3}{false}{true},
  minimum~size=0pt,
  outer~sep=0pt,
  inner~sep=0pt,
  ]{
    % BASE NODE
```

```
660     \tl_if_blank:nTF{#3}{
661       \node[] (base) at (0,0) {};
662     }{
663       \node[] (base) at (
664       $ (current~page.south~west)
665       + (0.8cm,1.5cm)
666       + (\int_use:N \l_xoffset_int mm,\int_use:N \l_yoffset_int mm) $
667       ) {};
668     }
669
670     % card background
671     \draw[
672     fill=\bool_if:NTF\l_back_bool{bbg}{bg}
673     ] (base) rectangle ($(base) + (64mm,89mm)$);
674
675     % colour frame
676     \draw[
677     line~width=2mm,
678     color=\bool_if:NTF\l_back_bool{bmarg}{marg},
679     ] \tl_if_eq:NNTF \l_orientation_tl \const_portrait_tl {
680       ($(base) + (2mm,89mm) $) -- ($(base) + (2mm,0mm) $) ;
681     }
682     {
683       ($(base) + (0mm,2mm) $) -- ($(base) + (64mm,2mm) $) ;
684     }
685
686     \draw[
687     line~width=2mm,
688     color=\bool_if:NTF\l_back_bool{bmarg}{marg},
689     ] \tl_if_eq:NNTF \l_orientation_tl \const_portrait_tl
690     {
691       ($(base) + (62mm,89mm) $) -- ($(base) + (62mm,0mm) $) ;
692     }
693     {
694       ($(base) + (0mm,87mm) $) -- ($(base) + (64mm,87mm) $) ;
695     }
696
697     % SEPARATOR
698     \bool_if:NTF \l_back_bool {
699     \bool_if:NT \l_bseparator_bool {
700       \path[
701       draw=bseparator,
702       line~width=1mm,
703       ]
704       \tl_if_eq:NNTF \l_orientation_tl \const_portrait_tl {
705         ($(base) + (3mm,40mm)$) -- +(58mm,0mm);
706       }{
707         ($(base) + (3mm,44.5mm)$) -- +(58mm,0mm);
708       }
709     }
710       }{
711     \bool_if:NT \l_separator_bool {
712       \path[
713       draw=separator,
```

```
    line~width=1mm,
    ]
    \tl_if_eq:NNTF \l_orientation_tl \const_portrait_tl {
      ($(base) + (3mm,40mm)$) -- +(58mm,0mm);
    }{
      ($(base) + (3mm,44.5mm)$) -- +(58mm,0mm);
    }
  }
  }

  % TITLE BOX
  \tl_if_eq:NNTF \l_orientation_tl \const_portrait_tl {
    \shade[
    bottom~color=\bool_if:NTF\l_back_bool{bbox}{box},
    top~color=\bool_if:NTF\l_back_bool{bboxgradient}{boxgradient},
    draw=none,
    rounded~corners=4mm,
    ]
    ($(base) + (0mm,89mm) $) -- ++(0mm,-15mm) --++(64mm,0mm) -- ++(0mm,15mm) --cycle;
  }{
    \shade[
    right~color=\bool_if:NTF\l_back_bool{bbox}{box},
    left~color=\bool_if:NTF\l_back_bool{bboxgradient}{boxgradient},
    draw=none,
    rounded~corners=4mm,
    ]
    ($(base) + (0mm,0mm) $) -- ++(0mm,89mm) --++(15mm,0mm) -- ++(0mm,-89mm) --cycle;
  }

  % TAG BOX
  \tl_if_eq:NNTF \l_orientation_tl \const_portrait_tl {
    \shade[
    top~color=\bool_if:NTF\l_back_bool{btagscolour}{tagscolour},
    bottom~color=\bool_if:NTF\l_back_bool{btagsgradient}{tagsgradient},
    draw=none,
    rounded~corners=4mm,
    ]
    (base) -- ++(0mm,9mm) -- ++(64mm,0mm) -- ++(0mm,-9mm) --cycle;
  }{
    \shade[
    left~color=\bool_if:NTF\l_back_bool{btagscolour}{tagscolour},
    right~color=\bool_if:NTF\l_back_bool{btagsgradient}{tagsgradient},
    draw=none,
    rounded~corners=4mm,
    ]
    ($(base) + (64mm,0mm) $) -- ++(-9mm,0mm) -- ++(0mm,89mm) -- ++(9mm,0mm) --cycle;
  }

  % Tags
  \node[
  anchor=center,
  align=center,
  font=\normalsize,
  text~width=50mm,
```

```
rotate= \tl_if_eq:NNTF \l_orientation_tl \const_portrait_tl {0} {90} ,
] (tags) at
\tl_if_eq:NNTF \l_orientation_tl \const_portrait_tl
{
  ($ (base) + (32mm,6mm) + (0mm,-1.5mm) $)
}
{
  ($ (base) + (58mm,44.5mm) + (1.5mm,0mm) $)
}
{
  \bool_if:NTF \l_back_bool{
      \tl_if_empty:NF \l_btagbox_tl {
        \insert_attribute:n{\l_btagbox_tl}
      }
  } {
    \tl_if_empty:NTF \l_tagboxcontent_tl {
      \tl_if_empty:NF \l_tagbox_tl {
        \insert_attribute:n{\l_tagbox_tl}
      }
    }{
      \tl_use:N\l_tagboxcontent_tl
    }
  }
};

% CARD BORDER
\draw[
color=\bool_if:NTF\l_back_bool{bborder}{border},
line~width=2mm,
rounded~corners=4mm,
fill=none,
] (base) -- ++(0mm,89mm) -- ++(64mm,0mm) -- ++(0mm,-89mm) --cycle;
% I could not find a way to make a TikZ path have rounded corners
% only on one side, so instead we simply draw over it if the outer
% corners are desired to be unrounded.
\bool_if:NF\l_rounded_corners_bool{
  \draw[
  color=\bool_if:NTF\l_back_bool{bborder}{border},
  line~width=2mm,
  fill=none,
  ] (base) -- ++(0mm,89mm) -- ++(64mm,0mm) -- ++(0mm,-89mm) --cycle;
}

% CARD NAME
\node[
anchor=center,
align=center,
font=\tl_use:N\l_modifyname_tl,
text~width=\tl_if_eq:NNTF \l_orientation_tl \const_portrait_tl {54mm} {70mm} ,
rotate=\tl_if_eq:NNTF \l_orientation_tl \const_portrait_tl {0} {90} ,
] (title) at
\tl_if_eq:NNTF \l_orientation_tl \const_portrait_tl
{
  ($ (base) + (32mm,81mm) $)
```

```
    } {
      ($ (base) + (8mm,44.5mm) $)
    }
    {
      \bool_if:NTF \l_back_bool
      {
        \tl_if_empty:NF \l_btitlebox_tl {
          \insert_attribute:n \l_btitlebox_tl
        }
      }
      {
        \tl_if_empty:NTF \l_titleboxcontent_tl
        {
          \tl_if_empty:NF \l_titlebox_tl {
            \insert_attribute:n \l_titlebox_tl
            }
        }
        {
          \l_titleboxcontent_tl
        }
      }
    };

    % CORNER CONTENT
    % NORTH WEST
      \node[
      anchor=west ,
      align=left,
      font=\tl_use:N\l_nwtextsize_tl,
      rotate= \tl_if_eq:NNTF \l_orientation_tl \const_portrait_tl {0} {90} ,
      minimum~size=5mm,
      ] (nwcontent) at
      \tl_if_eq:NNTF \l_orientation_tl \const_portrait_tl {
        ($(base)+(5mm,81.5mm)$)
      }{
        ($(base)+(7.5mm,5mm)$)
      }
      {
        \bool_if:NTF\l_back_bool {
        \tl_if_empty:NF\l_bnw_tl{
          \tl_use:N\l_bnw_tl
        }
        }{
        \tl_if_empty:NF\l_nwpre_tl{
          \tl_use:N\l_nwpretextsize_tl
          \tl_use:N\l_nwpre_tl\\[-1mm]
        }
        \tl_if_empty:NTF\l_nw_tl{
          \tl_use:N\l_nwcontent_tl
        }{
          \insert_attribute:n \l_nw_tl
        }
        \tl_if_empty:NF\l_nwpost_tl{
          \\[-2mm]
```

```
        \tl_use:N\l_nwposttextsize_tl
        \tl_use:N\l_nwpost_tl
      }
      }
    };
  % SOUTH WEST
    \node[
    anchor=west ,
    align=left,
    font=\tl_use:N\l_swtextsize_tl,
    rotate= \tl_if_eq:NNTF \l_orientation_tl \const_portrait_tl {0} {90} ,
    minimum~size=5mm,
    ] (swcontent) at
    \tl_if_eq:NNTF \l_orientation_tl \const_portrait_tl {
      ($(base)+(5mm,5mm)$)
    }{
      ($(base)+(59mm,5mm)$)
    }
    {
      \bool_if:NTF\l_back_bool {
      \tl_if_empty:NF\l_bsw_tl{
        \tl_use:N\l_bsw_tl
      }
      }{
      \tl_if_empty:NF\l_swpre_tl{
        \tl_use:N\l_swpretextsize_tl
        \tl_use:N\l_swpre_tl\\[-1mm]
      }
      \tl_if_empty:NTF\l_sw_tl{
        \tl_use:N\l_swcontent_tl
      }{
        \insert_attribute:n \l_sw_tl
      }
      \tl_if_empty:NF\l_swpost_tl{
        \\[-2mm]
        \tl_use:N\l_swposttextsize_tl
        \tl_use:N\l_swpost_tl
      }
  }
    };
  % NORTH EAST
    \node[
    anchor=east,
    align=right,
    font=\tl_use:N\l_netextsize_tl,
    rotate= \tl_if_eq:NNTF \l_orientation_tl \const_portrait_tl {0} {90} ,
    minimum~size=5mm,
    ] (necontent) at
    \tl_if_eq:NNTF \l_orientation_tl \const_portrait_tl {
      ($(base)+(59mm,81.5mm)$)
    }{
      ($(base)+(7.5mm,84mm)$)
    }
    {
```

```
      \bool_if:NTF\l_back_bool {
\tl_if_empty:NF\l_bne_tl{
        \tl_use:N\l_bne_tl
      }
      } {
      \tl_if_empty:NF\l_nepre_tl{
        \tl_use:N\l_nepretextsize_tl
        \tl_use:N\l_nepre_tl\\[-1mm]
      }
      \tl_if_empty:NTF\l_ne_tl{
        \tl_use:N\l_necontent_tl
      }{
        \insert_attribute:n \l_ne_tl
      }
      \tl_if_empty:NF\l_nepost_tl{
        \\[-2mm]
        \tl_use:N\l_neposttextsize_tl
        \tl_use:N\l_nepost_tl
      }
 }
   };
 % SOUTH EAST
   \node[
   anchor=east,
   align=right,
   font=\tl_use:N\l_setextsize_tl,
   rotate= \tl_if_eq:NNTF \l_orientation_tl \const_portrait_tl {0} {90} ,
   minimum~size=5mm,
   ] (secontent) at
   \tl_if_eq:NNTF \l_orientation_tl \const_portrait_tl {
     ($(base)+(59mm,5mm)$)
   }{
     ($(base)+(59mm,84mm)$)
   }
   {
     \bool_if:NTF\l_back_bool {
     \tl_if_empty:NF\l_bse_tl{
       \tl_use:N\l_bse_tl
     }
     } {
     \tl_if_empty:NF\l_sepre_tl{
       \tl_use:N\l_sepretextsize_tl
       \tl_use:N\l_sepre_tl\\[-1mm]
     }
     \tl_if_empty:NTF\l_se_tl{
       \tl_use:N\l_secontent_tl
     }{
       \insert_attribute:n \l_se_tl
     }
     \tl_if_empty:NF\l_sepost_tl{
       \\[-2mm]
       \tl_use:N\l_seposttextsize_tl
       \tl_use:N\l_sepost_tl
     }
```

```
}
  };

% PLAYTESTING INFORMATION
\bool_if:NF \l_back_bool {
  \bool_if:NT \l_playtestinfo_bool {
    \node[
    anchor=south~west,
    rotate=\tl_if_eq:NNTF \l_orientation_tl \const_portrait_tl {90}{180},
    ] (type) at
    \tl_if_eq:NNTF \l_orientation_tl \const_portrait_tl {
      ($(base)+(62.7mm,10mm)$)
    }{
      ($(base)+(54mm,87.7mm)$)
    }{
      \tiny
      \textcolor{black!70}{
        \l_type_tl{} ~ -- ~ \l_subtype_tl{} ~ -- ~ v\l_version_tl{}
      }
    };
  }
}

% CARD TEXT
\bool_if:NF \l_back_bool {
  \tl_if_eq:NNF \l_type_tl \const_monsterstring_tl {
    \node[
    anchor=north~west,
    font=\tl_use:N\l_textsize_tl,
    text~width=\tl_if_eq:NNTF \l_orientation_tl \const_portrait_tl {50mm}{40mm},
    rotate= \tl_if_eq:NNTF \l_orientation_tl \const_portrait_tl {0}{90},
    minimum~height=30mm,
    ] (text) at
    \tl_if_eq:NNTF \l_orientation_tl \const_portrait_tl {
      ($ (base) + (5mm,40mm) $)
    }{
      ($ (base) + (20mm,5mm) $)
    }
    {
      \tl_use:N \l_text_tl
    };

    % MORE TEXT
    \tl_if_blank:VF \l_moretext_tl {
      \tl_trim_spaces:N \l_moretext_tl
      \node[
      rotate= \tl_if_eq:NNTF \l_orientation_tl \const_portrait_tl {0} {90} ,
      anchor=south~west,
      align=left,
      font=\l_moretextsize_tl,
      text~width=50mm,
      ] (goalline) at
      \tl_if_eq:NNTF \l_orientation_tl \const_portrait_tl {
        ($ (base) + (5mm,10mm) $)
```

```
        }{
          ($ (base) + (54mm,5mm) $)
        } {
          \l_moretext_tl
        };
      }
    }
  }

  % ART
  \bool_if:NF \l_back_bool {
    \tl_if_empty:NF\l_art_tl{
      \node[
      rotate= \tl_if_eq:NNTF \l_orientation_tl \const_portrait_tl {0} {90} ,
      anchor=center,
      ] (backgraphics) at
      \tl_if_eq:NNTF \l_orientation_tl \const_portrait_tl {
        ($ (base) + (32mm,57mm) $)
      }{
        ($ (base) + (35mm,65.5mm) $)
      }{
        \tl_if_eq:NNTF \l_orientation_tl \const_portrait_tl {
          \includegraphics[
          height=31mm,
          width=56mm,
          keepaspectratio
          ]{\tl_use:N\l_art_tl}
        }{
          \includegraphics[
          width=40mm,
          height=38mm,
          keepaspectratio
          ]{\tl_use:N\l_art_tl}
        }
      };
    }
  }

  % BACK ART
  \bool_if:NT \l_back_bool {
    \tl_if_empty:NF\l_backart_tl{
      \node[
      rotate= \tl_if_eq:NNTF \l_orientation_tl \const_portrait_tl {0} {90} ,
      anchor=center,
      ] (backgraphics) at
      \tl_if_eq:NNTF \l_orientation_tl \const_portrait_tl {
        ($ (base) + (30mm,42.5mm) $)
      }{
        ($ (base) + (35mm,42.5mm) $)
      }{
        \tl_if_eq:NNTF \l_orientation_tl \const_portrait_tl {
          \includegraphics[
          width=56mm,
          height=31mm,
```

```
            keepaspectratio
            ]{\tl_use:N\l_backart_tl}
          }{
            \includegraphics[
            height=38mm,
            width=40mm,
            keepaspectratio
            ]{\tl_use:N\l_backart_tl}
          }
        };
      }
    }
  }

  \group_end:
}
```

(*End of definition for* `\typeset_card:nnn`. *This function is documented on page* **??**.)

`\print_cards:n` This command takes a list of key-value pairs as specificed in the `cardlist` definition. It iterates over all existing cards and calls `\typeset_card:nnn` repeatedly, inserting page breaks along the way.

```
\cs_new:Nn \print_cards:n {
  \keys_set:nn { cardlist } { #1 }

  % If desired, sort all cards alphabetically by type, subtype, and card name.
  \bool_if:NT \l_sort_bool {
    \clist_sort:Nn \g_cards_clist {
      \str_compare:nNnTF { ##1 } < { ##2 }
      { \sort_return_same: }
      { \sort_return_swapped: }
    }
    \clist_sort:Nn \g_cardbacks_clist {
      \str_compare:nNnTF { ##1 } < { ##2 }
      { \sort_return_same: }
      { \sort_return_swapped: }
    }
  }

  % Loop over all cards, figuring out where to place them.
  \int_zero_new:N \l_i_int
  \int_set:Nn \l_i_int { 9 } % this is just a better zero
  \clist_map_inline:Nn \g_cards_clist {
    \int_incr:N \l_i_int
    \int_compare:nNnT { \l_i_int } > { 9 } {
      \newpage
      \pagenumbering{gobble}
      \thispagestyle{empty}
      \int_set:Nn \l_i_int { 1 }
    }
    \typeset_card:nnn { ##1 } {} { \int_use:N \l_i_int }
  }

  % All faces are generated. If back sides are desired, we do those now.
```

```
  \bool_if:NT\l_generatecardbacks_bool {
    \clist_map_inline:Nn \g_cardbacks_clist {
      \int_incr:N \l_i_int
      \int_compare:nNnT { \l_i_int } > { 9 } {
        \newpage
        \pagenumbering{gobble}
        \thispagestyle{empty}
        \int_set:Nn \l_i_int { 1 }
      }
      \typeset_card:nnn { ##1 } {} { \int_use:N \l_i_int }
    }
  }

  % Debugging message: show the tags.
  \clist_sort:Nn \g_existingtags_clist {
    \str_compare:nNnTF { ##1 } < { ##2 } { \sort_return_same: } { \sort_return_swapped: }
  }
  \clist_map_inline:Nn \g_existingtags_clist {
    \msg_term:nnxx { cardgame } { tagcount } { ##1 } { \int_use:c { tagcounter @ ##1 } }
  }
}
```

(*End of definition for* `\print_cards:n`. *This function is documented on page* **??**.)

`\list_card:n` This command produces a single row suitable for including in a tabularray. The only way to do this seems to be to build a token list for the entire table body and then expand it all at once. See: https://tex.stackexchange.com/questions/736567/making-forloop-works-with-tabularray-it-works-with-tabularx

The macro takes as its sole argument an entry from `\g_cards_clist`, which is of course a token list of precompiled key-value assignments.

We use a global token list variable which is cleared with `\tl_gclear` inside `\list_cards` (the function after this one). The reason for this is that we have to wrap the code of `\list_card` inside a group to prevent card attributes from spilling.

```
\tl_new:N\g_cardtblr_tl
\cs_new:Nn \list_card:n {

  \group_begin:
  \tl_use:c { #1 }

  % The tags should always be sorted the same.
  \clist_sort:Nn \l_tags_clist {
    \str_compare:nNnTF { ##1 } < { ##2 } { \sort_return_same: } { \sort_return_swapped: }
  }

  % note: because texts can contain newlines, the x expansion
  % is necessary here because we need to surround the cell content
  % with additional braces.
  \tl_gput_right:Nx\g_cardtblr_tl{{\tl_use:N\l_name_tl}}
  \tl_gput_right:Nn\g_cardtblr_tl{&}
  \tl_gput_right:Nx\g_cardtblr_tl{{\tl_use:N\l_text_tl}}
  \tl_gput_right:Nn\g_cardtblr_tl{&}
  \tl_gput_right:Nx\g_cardtblr_tl{{\tl_use:N\l_rulings_tl}}
  \tl_gput_right:Nn\g_cardtblr_tl{\\}

```

```
  \group_end:
}
```

(*End of definition for* `\list_card:n`. *This function is documented on page* **??**.)

`\list_cards:n` The internal command to make the table of cards. It takes as its argument key-value pairs as defined by `cardlist`.

```
\cs_new:Nn \list_cards:n {
  \keys_set:nn { cardlist } { #1 }
  % sort all cards alphabetically by type, subtype, and card name.
  \bool_if:NT \l_sort_bool {
    \clist_sort:Nn \g_cards_clist {
      \str_compare:nNnTF { ##1 } < { ##2 }
      { \sort_return_same: }
      { \sort_return_swapped: }
    }
    \clist_sort:Nn \g_cardbacks_clist {
      \str_compare:nNnTF { ##1 } < { ##2 }
      { \sort_return_same: }
      { \sort_return_swapped: }
    }
  }

  \tl_gclear:N\g_cardtblr_tl
  \clist_map_inline:Nn \g_cards_clist {
    \list_card:n { ##1 }
  }

  \begin{tblr}[expand=\g_cardtblr_tl]{
    colspec = {XXX}, colsep = 8mm, hlines = {2pt, white},
    row{odd} = {azure8}, row{even} = {gray8},
    row{1} = {2em,azure2,fg=white,font=\bfseries\sffamily},
    row{2-Z} = {1em}
    }
    Name & Text & Rulings\\
    \g_cardtblr_tl
  \end{tblr}
}
```

(*End of definition for* `\list_cards:n`. *This function is documented on page* **??**.)

9.6 User-Exposed Functions

For usage notes, see the user manual.

`\Card` There is probably a better way to do this than these nested conditionals, but my knowledge of how LaTeX3 handles arguments and key-value pairs is not deep enough.

```
\NewDocumentCommand\Card{ O{} m m }{
  \IfBlankTF{#2}{
    \IfBlankTF{#3}{
      \card:n{
        #1,
      }
    }{
```

```
      \card:n{
        #1,
        text={#3},
      }
    }
  }{
    \IfBlankTF{#3}{
      \card:n{
        #1,
        name={#2}
      }
    }{
      \card:n{
        #1,
        text={#3},
        name={#2}
      }
    }
  }
}
```

(*End of definition for* `\Card`*. This function is documented on page* **??**.)

`\DeclareCardType` If no parent type is given, this just creates a wrapper for `\Card`. To inherit from a parent, we just wrap said parent.

```
\NewDocumentCommand\DeclareCardType{O{} m m}{
  \IfBlankTF{#1}{
    \expandafter\NewDocumentCommand\csname#2\endcsname{O{} m m}{
      \Card[#3,##1]{##2}{##3}
    }
  }{
    \expandafter\NewDocumentCommand\csname#2\endcsname{O{} m m}{
      \csname#1\endcsname[#3,##1]{##2}{##3}
    }
  }
}
```

(*End of definition for* `\DeclareCardType`*. This function is documented on page* **??**.)

`\ShowCard` If no argument is given, we access the last element of the global card list. Otherwise, we reconstruct the name under which the card was saved by uid.

```
\cs_generate_variant:Nn\typeset_card:nnn{onn}
\NewDocumentCommand\ShowCard{ o }{
  \IfNoValueTF{#1}{
    \typeset_card:nnn{\clist_item:Nn\g_cards_clist{-1}}{}{}
  }{
    \typeset_card:onn{card_by_uid @ #1}{}{}
  }
}
```

(*End of definition for* `\ShowCard`*. This function is documented on page* **??**.)

`\PrintCards`

```
\NewDocumentCommand\PrintCards{ O{} }{
  \print_cards:n{#1}
}
```

(*End of definition for* `\PrintCards`*. This function is documented on page* **??**.)

`\ListCards`

```
\NewDocumentCommand\ListCards{ O{} }{
  \list_cards:n{#1}
}
```

(*End of definition for* `\ListCards`*. This function is documented on page* **??**.)

```
⟨/package⟩
```