The recipecard class*

Ben Reish
ben.reish@alumni.oc.edu

October 23, 2006

Abstract

The point of this package is to typeset recipes. I tried cooking.sty and did not like the results so I am making my own. `Recipecard` is an alternate method for typesetting recipes. Created by Ben Reish ©2005

Contents

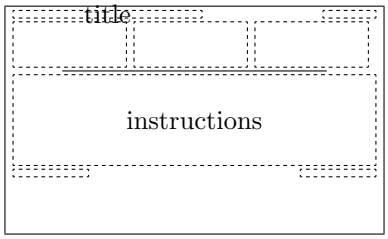
1 Introduction

For those who yearn to have typed note cards for their recipes, recipecard has been created. The user can print out his or her recipes in this class and then cut the pages along the box outlines for pieces that will fit on a notecard. Hopefully, multiple dimensions of note cards will be accommodated. This class is based on the article class. This class is issued under the LaTeX 2_ε Project Public License.

The `recipecard` class was created to look like a recipe card. The card begins with a title and across from the title is the number of servings the recipe makes. Then comes three columns of ingredients if there are enough ingredients and depending on the number of ingredients per column, which can be changed

*This document corresponds to recipecard.dtx v2.0, dated 2006/10/03.

from recipe to recipe. After the ingredients comes the instructions on making the recipe. Finally, if the user has used the `\cookingtime` and `\cooktemp` commands, those two are below the instructions.



2 Usage

2.1 Class Options

The `recipecard` class has three Class Options which can be selected at the `\DocumentClass` statement. Each option tells the class what size the card output will be. The options are: `fivebysevn` (5 inches by 7 inches), `fourbysix` (4 inches by 6 inches), and `threebyfive` (3 inches by 5 inches). The `fivebysevn` option is intended for use with 5 inch by 7 inch note cards and places two note cards per page. The `fourbysix` option is intended for use with 4 inch by 6 inch note cards and places two note cards per page. The `threebyfive` option is intended for use with 3 inch by 5 inch note cards and places two note cards per page. Originally, the code should have been able to place four 3 inch by 5 inch note cards per page, but that just did not work. One other thing that did not work is the ability to automatically break long recipes onto separate note cards. That is something that the user will have to take care of when the recipe prints.

The default option is the `fivebyseven` option. A sample `\Documentclass` statement might look like: `\documentclass[fourbysix]{recipecard}`.

2.2 User Commands

`\ingredient` The `\ingredient {`⟨*ingredient*⟩`}` command adds an ingredient to the next recipe. It is used before the `recipe` environment because the list it is creating is printed at the beginning of the next instance of the `recipe` environment.

`\changeingrdlistnum` The `\changeingrdlistnum {`⟨*num*⟩`}` command will change the number of ingredients per column to ⟨*num*⟩. The user can create his or her own command to shorten up the command name. It should be used before the instance of the `recipe` environment that it is supposed to change. The user can make use of this command as often as he or she likes. The command will change all the following instances of the `recipe` environment as well. For a more uniform look to the printed card, the user should divide the number of ingredients per recipe by 3 and round up.

`recipe` The `recipe` environment is the staging area for each recipe. Use it to group all the information about each recipe into one area. Its call out looks like:

`\begin{recipe}` {⟨*title*⟩} {⟨*servings*⟩} ... `\end{recipe}`. The instructions for making the recipe go in between the `\begin` and `\end` commands. The `\cooktemp` and `\cookingtime` commands go in with the instructions.

`\cooktemp` The `\cooktemp` {⟨*temp*⟩} {⟨*deg*⟩} macro adds a cooking temperature to the card. The ⟨*temp*⟩ is the numerical temperature. The ⟨*deg*⟩ is the units. Put 'C' in for ⟨*deg*⟩ for Celsius or 'F' for Fahrenheit. For example, if the user wanted 350° Fahrenheit for the cooking temperature, the command would look like: `\cooktemp{350}{F}`

`\cookingtime` The `\cookingtime` {⟨*time*⟩} command adds a bold face "Cook Time: ⟨*time*⟩" to the bottom of the card.

3 Implementation

3.1 Class Definition

```
1 \RequirePackage{calc,ifthen,boxedminipage}
```

The user will need the `calc`, `ifthen`, `geometry`, and `boxedminipage` packages for the `recipecard` class to operate correctly. These should be available with the user's distribution of LaTeX 2ε or from *CTAN* at WWW.CTAN.ORG.

The `geometry` package is used because it seems easier than attempting to use the old style layout commands.

After requiring the `geometry` package, the code creates new boolean variables and token registers for use later in the class.

```
2 \RequirePackage[letterpaper,noheadfoot]{geometry} %showframe
3 \newboolean{fiveseven} \newboolean{foursix} \newboolean{threefive}
4 \newtoks{\@ta} \newtoks{\@tb} \newtoks{\@listone} \newtoks\@listtwo
5 \newtoks\@listthree \def\@nil{}
6
7 \renewcommand{\normalsize}{\fontsize{10pt}{12pt}\usefont{T1}{ptm}{m}{n}%
8     \selectfont}
```

These commands are general specifications needed by any class. First, select the default font. Then set up the page layout in general. After that, create a couple of length dimensions so that each size of notecard can have its own height and width of the ingredient columns.

```
9 \setlength{\textwidth}{7in} \setlength{\textheight}{10.25in}
10 \setlength{\paperwidth}{8.5in} \setlength{\paperheight}{11in}
11 \newlength{\@ingredientlist} \newlength{\@cardheight}
12 \newcommand{\textdegree}{\textsuperscript{$\circ$}}
```

The `\textdegree` command is needed for the `\cooktemp` command which will be described later. It is made available to the user at any point in the class by defining a command for it.

3.2 Options

Next, the class-specific options are defined.

```
13 \DeclareOption{fivebyseven}{\geometry{%
14     body={7in,10.25in},left=.75in}% centering,right=.75in
```

```
    \setlength{\@ingredientlist}{2in} \setlength{\@cardheight}{5in}%
    \setboolean{fiveseven}{true}}
```

The `fivebyseven` option resets the page layout so that two 5 inch by 7 inch notecards can be placed on the front of one page. There is a little extra space so that the user can cut the pieces out without harming either notecard. This option sets the ⟨*cardheight*⟩ to 5 inches and the ingredient list column width, ⟨*ingredientlist*⟩, to 2 inches. Finally, it sets the `fiveseven` boolean to true so that any other things the class might need on a size specific basis can be set inside an `if` statement later as necessary.

```
\DeclareOption{fourbysix}{\geometry{%
    body={6in,8.15in},left=.75in}%right=1.75in
    \setlength\@ingredientlist{1.75in}%
    \setlength\@cardheight{4in} \setboolean{foursix}{true}}
```

The `fourbysix` option resets the page layout so that two 4 inch by 6 inch notecards can be placed on the front of one page. There is a little extra space so that the user can cut the pieces out without harming either notecard. This option sets the ⟨*cardheight*⟩ to 4 inches and the ingredient list column width, ⟨*ingredientlist*⟩, to 1.75 inches. Finally, it sets the `foursix` boolean to true so that any other things the class might need on a size specific basis can be set inside an `if` statement later as necessary.

```
\DeclareOption{threebyfive}{\geometry{%
    landscape,body={10.25in,6.25in},left=.375in}%,right=.375in
    \setlength\@ingredientlist{1.5in}%
    \setlength\@cardheight{3in} \setboolean{threefive}{true}}
```

The `threebyfive` option resets the page layout so that four 3 inch by 5 inch notecards can be placed on the front of one page. There is a little extra space so that the user can cut the pieces out without harming either notecard. This option sets the ⟨*cardheight*⟩ to 3 inches and the ingredient list column width, ⟨*ingredientlist*⟩, to 1.5 inches. Finally, it sets the `threefive` boolean to true so that any other things the class might need on a size specific basis can be set inside an `if` statement later as necessary. Note: The class does not print four 3" by 5" notecards to the page. It only gets two to the page.

```
\DeclareOption{nothing}{\relax}

\DeclareOption*{\typeout{What's \CurrentOption?}}

\ExecuteOptions{fivebyseven,nothing}

\ProcessOptions\relax
```

Once the options are defined, the class must be given a default option to use if none is specified by the user. In this case, the `fivebyseven` option is chosen.

```

```

3.3 User Commands

The following commands are used to fill in the recipe cards' information.

`\ingredient` The `\ingredient` command is used to define the list of ingredients in the recipe. It is the user-friendly way to add an ingredient. It uses a counter, ⟨*ingred@cnt*⟩, to keep track of how many ingredients have been added. The counter is used later. Then the command calls `\@ddtoNgrList` and passes its argument off to that command. The argument of this command can be any length, but remember that it will be wrapped to the next line if it is too long because of the width of the ingredient columns.

```
\newcounter{ingred@cnt}\setcounter{ingred@cnt}{0}
\newcommand{\ingredient}[1]{%
    \stepcounter{ingred@cnt}
    \@ddtoNgrdList{#1}
    %\typeout{\string\ingredient{#1}}
}
```

`\changeingrdlistnum` The `\changeingrdlistnum` macro allows the user to modify the number of ingredients listed vertically before the class switches to the next column of ingredients. The default value is 7. The number of rows in the ingredient list columns is stored in the ⟨*ingred@list*⟩ counter.

```
\newcounter{ingred@list} \setcounter{ingred@list}{7}
\newcommand{\changeingrdlistnum}[1]{%
    \setcounter{ingred@list}{#1}
}
```

`recipe` The `recipe` environment has several things going on. First, the class creates the card width dimension, ⟨*@cardwidth*⟩. Depending on the size of cards the user wants at the `\documentclass` instance, the length of ⟨*@cardwidth*⟩ is different. Any other global settings that change due to the size of the card are to be implemented here. Then the class creates several lengths and a save box called `\@reccardbox` to be used with the `\begin{lrbox}` command. The boolean variables set during the options section of the class come into play here.

```
\newdimen{\@cardwidth}
\ifthenelse{\boolean{fiveseven}}{\setlength\@cardwidth{7in}}{}
\ifthenelse{\boolean{foursix}}{\setlength\@cardwidth{6in}}{}
\ifthenelse{\boolean{threefive}}{\setlength\@cardwidth{5in}%
    \changeingrdlistnum{4}
    \renewcommand{\normalsize}{\fontsize{8pt}{10pt}%
    \usefont{T1}{ptm}{m}{n}\selectfont}}{}
```

For the 3 inch by 5 inch card, the font needs to be smaller to help fit the same amount of information on the card. The `\normalsize` command is renewed to allow the smaller font.

```
\newsavebox{\@reccardbox} \newdimen{\@reccardh} \newdimen{\@rectemp}
\newdimen{\@hruleoffset} \newdimen{\@rectempa}
\newdimen{\@rectempb} \newdimen{\@rectempc} \newdimen{\BR@recd}
\setlength\@rectemp{\@cardheight-2\fboxsep-2\fboxrule-17pt}
\setlength{\@hruleoffset}{(\@cardwidth-2\fboxsep-2\fboxrule-.714\@cardwidth)/2}
```

The code sets the length of ⟨*@rectemp*⟩ to the height of the card minus two times the separation distance for a framed box minus two times the thickness of the

framed box line minus seventeen points for the title font. The ⟨*@hruleoffset*⟩ length is the distance needed to center the horizontal rule that goes below the ingredients and above the instructions.

```
\newenvironment{recipe}[2]{%
    \ifthenelse{\equal{#2}{\@empty}}{\def\@recserv{}}%
        {\def\@recserv{Serves: #2}}
    \def\@rectitle{#1 \raggedright}
```

The author assumes that one would enter a title for each recipe so the most likely unused argument would be the second one which is the number of serving the recipe makes. The code checks to see if the second argument is empty. If it is the code defines the internal command, `\@recserv` to be empty. If the it is not empty, the code defines `\@recserv` as "Serves: ⟨*servings*⟩." Then the code defines the internal command `\@rectitle` as the first argument of `recipe` environment, ⟨*title*⟩ and applies the `\raggedright` command so that LaTeX does not try to stretch the title all the way across the card.

Then the code checks the second and third ingredient lists, `\@listtwo` and `\@listthree`, for emptiness. The code assumes that there will be at least one or two ingredients per recipe and therefore does not check the first list. If either is empty, the code enters a blank list `\item` command into the empty list to avoid an error when LaTeX processes an empty ingredient list token in a list environment.

```
    \ifthenelse{\equal{\the\@listtwo}{\@empty}}{\@listtwo={\item {}}}{}
    \ifthenelse{\equal{\the\@listthree}{\@empty}}{\@listthree={\item {}}}{}
```

Next, the code begins an `lrbox` environment which is like a `\savebox`, but as an environment, it can have more than just static text put in it. This is used so that the whole card in its entirety (title, ingredients, instructions, cooking times and temperatures) can be placed in a `\boxedminipage` command in the ending of the `recipe` environment. The `\boxedminipage` command creates the outline for the card for cutting purposes.

Inside the `lrbox`, the code starts a `minipage` environment which will contain all the text on the card. Its width is the card width minus twice the thickness of the line that surrounds the `boxedminipage` minus twice the separation distance between the line and the inside text.

Then the code changes the font for the title and servings; then places the title with a horizontal fill white space between it and the number of servings; and then switches back to normal font.

```
    \begin{lrbox}{\@reccardbox}
    \begin{minipage}[t]{\@cardwidth-2\fboxsep-2\fboxrule}
    \noindent\fontsize{14.4}{17} \usefont{T1}{pzc}{mb}{it}%
    \@rectitle\hspace{\fill}\@recserv\normalsize\normalfont\par
```

Here begins three separate `minipage` instances; one for each list of ingredients. The first and second `minipage` instances are separated with a 3 point space, as are the second and third `minipage` instances. Each `minipage` is ⟨*@ingredientlist*⟩ wide which is set depending upon the size of card the user is wanting.

Inside each of the three `minipage` instances is a `\begin{list}` command. This command defines a `list` environment that better meets the needs of the

`recipecard` class. The ⟨*leftmargin*⟩ distance indents the text by $\frac{1}{4}$ inch which makes the text that is word wrapped to the next line indent a noticeable amount. The ⟨*itemindent*⟩ pulls the first line of each item back out to the edge of the line instead of being indented in a $\frac{1}{4}$ inch. The code also adds a `\raggedright` command to the end of the `list` definition which causes LaTeX to not try to stretch the text of the item across the whole line. After the `list` is defined, the contents of `\@listone` are placed in the `list` environment and the environment is closed.

```
65      \begin{minipage}[t]{\@ingredientlist}
66          \begin{list}{}{\setlength\leftmargin{.25in}%
67              \setlength\itemindent{-.25in}\raggedright}\the\@listone%
68              \end{list}\end{minipage}
69          \typeout{first box}
70      \hspace{3pt plus 0pt minus 6pt}
71      \begin{minipage}[t]{\@ingredientlist}
72          \begin{list}{}{\setlength\leftmargin{.25in}%
73              \setlength\itemindent{-.25in}\raggedright}\the\@listtwo%
74              \end{list}\end{minipage}
75          \typeout{second box}
76      \hspace{3pt plus 0pt minus 6pt}
77      \begin{minipage}[t]{\@ingredientlist}
78          \begin{list}{}{\setlength\leftmargin{.25in}%
79              \setlength\itemindent{-.25in}\raggedright}\the\@listthree%
80              \end{list}\end{minipage}
81          \typeout{third box}
82      \hspace{\fill}
83      \par
```

Following the three lists of ingredients, the code skips 3 points vertically and then skips the value of ⟨*@hruleoffset*⟩ horizontally. Then the code places a line down to separate the ingredients from the instructions. Next, the code skips 3 points vertically before starting the instructions.

The `\everypar` token is expanded at the beginning of each new paragraph. This code places a 1em space (␣) indent at the start each paragraph. Because of this, to obtain a fully left aligned sentence or statement, the user will need to use a command like: `\hspace{-1em}`.

```
84      \vspace{3pt} \hspace{\@hruleoffset}%
85      \rule{.714\@cardwidth}{0.7pt}%
86      \par\vspace{3pt} \hspace{1em}%
87      \everypar={\hspace{1em}}
88      }{%
```

Here begins the commands that are executed at the end of the user-entered text. The code places `\@cooktime` and `\@cooktemp` at the bottom left and right, respectively, of the instructions. Then the code ends the overall `minipage` and `lrbox`.

```
89      \par \noindent \@cooktime \hspace{\fill} \@cooktemp%\par
90      \end{minipage}\end{lrbox}
91      \vspace{-.25in} \hspace{-21pt}
```

Here the code inserts the contents of the `lrbox`, `\@reccardbox`, into a boxed minipage. Then the clean up code starts. Zero the counter, empty the list tokens,

and empty the cooking commands. The cooking commands use a global definition because they are placed inside the `recipe` environment, which causes the changes made inside the environment to be reset when the environment ends. The global definition steps out of the environment and makes the changes so that they do not reset when the environment closes.

```
    \begin{boxedminipage}[t]{\@cardwidth}%
    \rule[-\@rectemp]{0pt}{\@rectemp} \hspace{-4pt}
    \usebox{\@reccardbox}%
    \end{boxedminipage}
    \ifthenelse{\boolean{threefive}}{\hspace{.5ex}}{\par\vspace{.35in}}
    \setcounter{ingred@cnt}{0}%
    \@listone={}
    \@listtwo={}
    \@listthree={}
    \gdef\@cooktime{} \gdef\@cooktemp{}
    \everypar={}
    %\typeout{\string\pagetotal\space\the\pagetotal}
    %\typeout{\string\@listone\space`\the\@listone'}
    }
```

`\cookingtime` The `\cookingtime {`⟨*time*⟩`}` macro adds a cook time to the card. This command uses an internal command, `\@cooktime`, to place text on the recipe card. The `recipe` environment has the internal command called out right after the instructions which causes `\@cooktime` to be expanded whether or not the user has used `\cookingtime`. If the user has not used `\cookingtime`, then the `\@cooktime` command expands to an empty space. If he or she has used it, then `\@cooktime` is defined as "Cook Time: ⟨*time*⟩" which is expanded when the `recipe` environment calls the `\@cooktime` command. The `\hspace{-1em}` is because this command is always at the left of the new paragraph, which because of the `\everypar` command in the `recipe` environment, has a 1em indent before it. The negative one here moves the text back up to the edge. The author wanted the cook time to stand out so the bold face font was used.

```
\def\@cooktime{}
\newcommand{\cookingtime}[1]{%
    \def\@cooktime{\hbox{\hspace{-1em}\bfseries Cook Time: #1}}
}
```

`\cooktemp` The `\cooktemp` command uses an internal command, `\@cooktemp` to insert the cooking temperature into the recipe. As with `\@cooktime`, `\@cooktemp` is called by the `recipe` environment regardless of it being empty or not. Again, the author wanted the cooking temperature to stand out so the code uses the bold font.

```
\def\@cooktemp{}
\newcommand{\cooktemp}[2]{%
    \def\@cooktemp{\hbox{\bfseries %
        Temperature: #1\textdegree\hspace{-1.5pt}#2}}
}
```

3.4 Non-User Commands

Here are the commands that only the class may call. These commands help to hide and safeguard the inner workings of the class.

`\@ddtoNgrdList` The `\@ddtoNgrdList {`⟨*ingredient*⟩`}` adds the latest ingredient to one of the token lists for the printout of the recipe card. This is an internal function and is not user friendly. It calculates which column to put the current ingredient into using the number of ingredients listed so far and the maximum number allowed in each column.

```
\newcounter{@tempa}\newcounter{@tempb} \newcounter{@tempc}
\newcommand{\@ddtoNgrdList}[1]{%
    \setcounter{@tempa}{\theingred@list+1}%
    \setcounter{@tempb}{2*\theingred@list+1}%
    \setcounter{@tempc}{3*\theingred@list+1}%
```

After initializing the limit numbers for the columns of ingredients, decide what column (1, 2, or 3) to put the latest ingredient into. If this is the first time through the sequence, clear the lists.

```
    \ifthenelse{\value{ingred@cnt}=1}{%
    \@ta={} \@listone={} \@listtwo={} \@listthree={}%
    }{}
    \ifthenelse{\value{ingred@cnt}<\value{@tempa}}{%
        %\typeout{\string\@listone :\space\the\@listone}
        \expandafter\@ta\expandafter=%
            \expandafter{\the\@listone \item #1}
        \@listone=\@ta}{%
        \ifthenelse{\(\value{ingred@cnt}>\value{@tempa}%
            \or \value{ingred@cnt}=\value{@tempa}%
            \)\and\value{ingred@cnt}<\value{@tempb}}{%
```

This if statement states, "if the value of ⟨*ingred@cnt*⟩ is greater than the value of ⟨*@tempa*⟩ (which is the number of ingredients per column plus one) or equal to the value of ⟨*@tempa*⟩" and "the value of ⟨*ingred@cnt*⟩ is less than the value of ⟨*@tempb*⟩ (which is the number of ingredients per column times two plus one)." If the previous is true the code then evaluates the following statement.

```
            \expandafter\@ta\expandafter=%
                \expandafter{\the\@listtwo \item #1}
```

The `\expandafter`'s skip the following token, so the previous statement, on the first time through reads "`\the\@listtwo`" which expands to the contents of `\@listtwo` and then returns to the beginning of the statement. Then the code reads back through the statement. It has used the `\expandafter`'s so they are not there for the second reading of the statement. The second time through, the statement reads, "`\@ta={\item` ⟨*ingredient1*⟩ `...` `\item` (now it inputs the contents of the first argument of the command)." Now `\@ta` is the whole of `\@listtwo` plus the statement, "`\item` ⟨*ingredient5*⟩" (for example). Lastly, the code sets `\@listtwo` to the value of `\@ta`.

```
            \@listtwo=\@ta}{%
            \ifthenelse{\(\value{ingred@cnt}>\value{@tempb}%
```

```
                \or \value{ingred@cnt}=\value{@tempb}\)\and%
                \value{ingred@cnt}<\value{@tempc}}{%
                \expandafter\@ta\expandafter=%
                    \expandafter{\the\@listthree \item #1}
                \@listthree=\@ta}{%
```

The first two columns are pretty self-explanatory. If the ingredient counter (⟨*ingred@cnt*⟩) is greater than the number of items per column (⟨*ingred@list*⟩) times 3, then an error is necessary. To continue processing, the class adds the offending item to the last column any way but puts out an error. As noted in the error message, the way around the error is to use `\changeingrdlistnum` command to change the number of ingredients allowed per column.

```
                \ifthenelse{\(\value{ingred@cnt}>\value{@tempc}%
                    \or\value{ingred@cnt}=\value{@tempc}\)}{%
                    \setcounter{ingred@cnt}{2}
                    \expandafter\@ta\expandafter=%
                        \expandafter{\the\@listthree \item #1}
                    \@listthree=\@ta}{%
                    \ClassError{recipecard}{More than \the@tempc\space ingredients for
                    one recipe card}{Unfortunately, the card
                    design only allows for three columns of a total of
                    \the@tempc\space ingredients. Hint: change the value of
                    \string\changeingrdlistnum.
                    Congratulations! You have used more ingredients
                    than this Class was designed for.}
                }
            }
        }
    }
    %\typeout{ingred@cnt\space\theingred@cnt}
}
```

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

Change History

v1.0

General: Initial version 1

v1.2

\@ddtoNgrdList: Implemented the ability to change how many rows were in each column of the ingredients list. 9

\changeingrdlistnum: Added the ability to manipulate how many rows are in each column of the ingredients list. 5

v1.3

\cookingtime: Added the cook time to the card. 8

\cooktemp: Added the cooking temperature to the card. 8

recipe: Added support for 5x7 notecards. 5

v1.5

recipe: Added support for 4x6 and 3x5 notecards. 5

v1.52

recipe: Fixed a bug in the 3x5 notecard display that was dropping the third list of ingredients to the bottom of the first list.

Also added an indent to paragraphs in the instructions section of the recipe environment. 5

v1.53

General: Added figure to help describe the layout of the recipe card. 2

v1.6

General: Added more verbage to the explanations of how the code operates. 1

v1.7

General: Added sections and subsections to class documentation. 1

v2

General: Added table of contents to first page. 1