The `MakeCookbook` Bundle
Make a Cookbook Using LaTeX

Terrence P. Murphy Rosalie A. D'Amico

December 2, 2018

Contents

Part I
Introduction

1 About

The `makecookbook` bundle contains the files needed to create a nice quality family cookbook in a form ready to submit to most print-on-demand companies. Modifiable choices have been made regarding standard book features such as trim size, margins, headers/footers, chapter heading formatting, front matter (copyright page, table of contents, etc.) and back matter (index). Commands and environments have been created to format the food stories and recipes. The user will need to: (1) supply their own food stories and recipes(!), and (2) select (install if necessary) the needed fonts.

The design, layout and typography for cookbooks varies substantially, so we necessarily take a "point of view" on the desired look of the cookbook. However, even if your goal is a significantly different layout, you may find this work helpful in thinking through and implementing your design.

Please note that no new document class or package is included here. Rather, we provide a modifiable preamble and a small number of other files that, together, fully support creation of all of the *internal* pages of a cookbook (i.e., everything except the cover art). We may refer to the `makecookbook` "package" in this documentation – by that we mean package in a broader sense and do *not* mean an actual .sty style package.

2 Preliminaries

2.1 Requirements

2.1.1 Compiler

The `makecookbook` bundle uses the `fontspec` package. That means it must be compiled with either LuaTeX or XeTeX. We have only tested with LuaTeX. However, we have not used Lua code, so we expect you should be successful (after a possible tweak or two) with an XeTeX compile.

2.1.2 Fonts

The `makecookbook` bundle assumes you have installed the three fonts listed below. (All are licensed under the SIL Open Font License, Version 1.1). To have a successful compile "out of the box", these fonts must be installed on your system:

Serif	*EB Garamond*	TrueType	Version from Google Fonts
Sans Serif	*Lato*	TrueType	Version from `www.latofonts.com`
Script	*Italianno*	OpenType	Version from Google Fonts

Please note: we assume that the *EB Garamond* and *Italianno* fonts were obtained from Google Fonts, and that the *Lato* font was obtained from `http://www.latofonts.com`. These fonts *do not* required installation of any font-related packgages other than `fontspec`.

Beginning on page 16, we discuss how you can easily substitute your own favorite OpenType (including TrueType flavored) fonts, subject only to certain requirements regarding feature set.

2.2 License

The `makecookbook` bundle includes an example cookbook with seven recipes. Those recipes are courtesy of Rosalie D'Amico[1]. You are, of course, welcome to try them! They are included in the bundle to provide

[1] D'Amico, Rosalie A. *Mama, How Do You Make....* (self-published, 2018).

real-world examples of using LaTeX code to enter recipes. We only ask that you consider those recipes as for you personal use and not (without attribution) for further food-related publication (further publication OK in a LaTeX context).

2.3 Contact Information / Feedback

No doubt, this documents contains typos, poorly explained (or unexplained) items, and other errors. It is equally likely the `makecookbook` package includes coding errors. We value your feedback. Please report problems or make other suggestions to Terry Murphy: `latex@rd-tpm.com`.

2.4 Version Information / Change History

Version 0.85 dated December 2, 2018. This is the initial version of the `makecookbook` bundle.

2.5 The LaTeX Community / StackExchange

In this `makecookbook` package, much of the code (except for the mistakes!) is not original. When we ran into difficulties, we usually turned to `tex.stackexchange.com` for help. We often found solutions in previous questions and answers, but sometimes had to ask our own questions. The LaTeX community is amazingly generous with their time.

You may see in the text or in the comments to the code something like "See Q 59619". That is a reference to a specific question and answer on `tex.stackexchange.com`. You can often track it down by a Google search of "latex 59619"; if that doesn't work, try "tex.stackexchange.com 59619".

3 Installing `makecookbook`

Go to the CTAN homepage of the `makecookbook` package: `https://ctan.org/pkg/makecookbook`. On that page you can download the complete zip file using the download button next to "*Download the contents of this package in one zip archive*". After unzipping, you will have the following files and directories:

```
makecookbook/
|   README
|   makecookbook-doc.tex
|   makecookbook-doc.pdf
|
|--- mycookbook/
     |  makecookbook.tex
     |  makecookbook.pdf
     |  cb-preamble.tex
     |  cb-lettrine.cfl
     |  cb-idxstyle.ist
     |
     |--- tex/
     |       cb-frontmatter.tex
     |       cb-chapterA.tex
     |       cb-chapterB.tex
     |
     |--- img/
             cb-imageA.jpg
             cb-imageB.jpg
```

Copy the `mycookbook` directory (including all its subdirectories and files) to you LaTeX project area. Choose a location so that the `mycookbook` directory is the root directory of your cookbook project.

Next, download and install[2] the three required fonts. Get *EB Garamond* and *Italianno* from Google

[2]We are not experts on font installation. Because we load fonts by filename, our understanding is that you just need to follow the normal instructions of your operating system for installing a system font. That's all we did for our Windows 10 system. If you run into problems, first verify that the installed font is available for other non-LaTeX programs. Beyond that, check the LuaTeX documentation, check the `fontspec` documentation, or ask a question at `tex.stackexchage.com`.

Fonts. Get *Lato* from `http://www.latofonts.com/`. We highly recommend that you download the three fonts from those sources to ensure you have the same versions as we have used, with the same filenames.

Once the fonts are installed, you will be able to build the initial version of your cookbook by doing a LuaTEX compile (or two) of `makecookbook.tex`.

If you decide to substitute you own fonts for any of the three fonts, you cannot compile until you modify `cb-preamble.tex` to associate your selected fonts with your cookbook project. Instructions on how to make the required modifications are given, beginning on page 16.

4 How This Document is Organized

Before we discuss the details of the `makecookbook` files and our LATEX code, we first (in Parts II and III) look at the cookbook-writing project at a higher level:

Part II - Managing/Organizing Your Cookbook Files. Keeping a book size project well organized is critical. We start by describing our approach to organizing the cookbook files.

Part III - Elements of a Book/Cookbook. Next, we provide an *overview* of the key elements that make up a cookbook. This include elements common to most any book (title page, table of contents, chapters, index, etc.), as well as elements particular to a cookbook (recipe names, ingredients, steps, etc.). All of the elements listed here will be discussed in detail further below.

Then, we dig deeper into the elements that make a book/cookbook:

Part IV - Elements of a Book - the Details. We list and discuss the key elements that are common to most books. We describe the choices we made regarding each element and, where appropriate, we describe how you can make a different choice for that element.

Part V - Elements of a Book - the Details (Fonts). Due to the length of this discussion, and due to the importance of font selection and usage, we break this topic out separately.

Part VI - Elements of a Cookbook - the Details. We list and discuss the elements that are particular to cookbooks. For each element, we describe the LATEX commands and environments you will use to include that element in your cookbook, including any options that have been programmed into those commands and environments. Using the files provided and the information discussed here, you should have the tools to make a complete cookbook.

Part VII - The PDF File. We discuss the sometimes conflicting requirements for building a PDF file for a digital cookbook (with bookmarks and links) versus building a PDF file for submittal to a print-on-demand company.

Part VIII - Examples. We present and discuss several recipe examples. This allows you to see real world examples of the LATEX commands and environments needed to make your cookbook.

Part II
Managing/Organizing Your Cookbook Files

Writing a cookbook is a large project. Early on, you should think through how you will manage your cookbook files. We describe below our fairly standard approach to organizing a book-size document in LaTeX:[3]

- Create a directory devoted exclusively to the cookbook. We will call that directory the *root directory* of the cookbook (in the files distributed with the `makecookbook` package, we call that directory `mycookbook`). All other directories used in the cookbook project will be referenced relative to the root directory. The root directory holds: (1) a one-page `makecookbook.tex` file that includes the instructions needed to pull all of the cookbook files together, (2) the `cb-preamble.tex` file that includes the LaTeX packages and programming code used by the cookbook, and (3) any special files needed by `cb-preamble.tex` (in our case, the `cb-lettrine.cfl` and `cb-idxstyle.ist` files).
- Create two subdirectories under the root directory, one called `tex` and one called `img`. The `tex` directory holds a `.tex` file for each chapter of the cookbook (plus `cb-frontmatter.tex`). The `img` directory holds any image files used in the cookbook.

It is helpful to see the full contents of the `makecookbook.tex` file. This file brings in the preamble, the front matter and the cookbook chapters and handles a few other "housekeeping" items:

```
\documentclass[11pt]{book}
\input{cb-preamble}

\begin{document}
  \frontmatter
  \include{./tex/cb-frontmatter}

  \mainmatter
  \include{./tex/cb-chapterA}
      .
      . (you include here all of the chapters of your cookbook)
      .
  \include{./tex/cb-chapterB}

  \backmatter
  \CookbookIndex{}
\end{document}
```

[3] See, e.g.: https://en.wikibooks.org/wiki/LaTeX/Modular_Documents

Part III
Elements of a Book/Cookbook

Here we present an *overview* of the elements that make up a book (in general), plus the special additional elements that make up a cookbook.

An excellent source for the elements of a book is *A Few Notes on Book Design* by Peter Wilson[4]. Although the entire article is well worth reading, see in particular *Chapter Two - The Parts of a Book*.

1 Elements of a Book

We list here some of the typical elements of a book. The first five are very basic elements:

Trim Size The physical size (height x width) of the paper used to print the book.

Margins In its simplest form, this is the top, bottom, inner and outer margins of the text area. Allowances must also be made for the areas where any header or footer is printed.

Headers and Footers This is where (above or below the text area) things like page number and current chapter name/number are printed.

Chapter Title Formatting The chapter name/number as printed on the first page of each new chapter.

Fonts Selection of fonts and choice of point size both play a fundamental role in all books.

Odds and Ends We will briefly consider a few other elements, such as color, drop caps and images.

Next we list the elements associated with the three traditional areas of a book: the front matter, the main matter and the back matter:

Front Matter This is the first part of the book, and includes an assortment of preliminary information. Typically, in this order, there is a title page, a copyright page, a dedication page, and a table of contents, all of which may be followed by one or more short chapters such as a preface or acknowledgments.

Main Matter This is the heart of the book. We follow the typical case where the main matter consists only of the chapters of the book.

Back Matter We follow the typical case where the back matter contains the book index. This is where you might include other ancillary information such as a bibliography, an appendix, notes, etc.

2 Elements of a Cookbook

Chapter Intro In a cookbook, chapters tend to be organized into logical units such as cookies, desserts, pasta, appetizers, etc. Following the chapter name, there will usually be some introductory text for that chapter. We call that the *chapter intro*.

Recipe Name and Yield A new recipe is introduced with a recipe name and, often, with information regarding the "yield" of the recipe (e.g., "makes 36 cookies" or "serves 4 to 6", etc.).

Recipe Intro/Story Often there is a story or other information specific to a recipe.

Ingredients and Steps Following the recipe name/yield and (possibly) a recipe story, there is a list of the ingredients, plus the steps required to make the recipe.

Attribution If the source of the recipe is known, it is proper to acknowledge that source.

Additional Comments/Advice There may be some recipe side notes and/or advice that is not properly included within the recipe, but is noted afterwards.

[4] https://ctan.org/pkg/memdesign

Part IV
Elements of a Book – the Details

1 Terminology

We define here some of the terms used below. Our book is two-sided, so visualize an open book with both the left and right page visible.

The terms ***recto*** and ***verso*** (from the Latin) refer to the text written or printed on the right (front) side and on the left (reverse or back) side of a leaf of paper. By book publishing convention, the first page of a book, and the start of each chapter of a book, is on a recto page. That means all recto pages will have odd page numbers and all verso pages will have even page numbers.

Still visualizing our open book, the ***outer margin*** is the right margin on a right/recto page and the left margin on a left/verso page – the margin *away from* the book binding. Similarly, the ***inner margin*** is the left margin on a recto page and the right margin on a verso page – the margin *closest to* the book binding.

2 Trim Size and Margins

In deciding on the trim size of the cookbook, we started with two requirements: (1) for the main text, we wanted an easily readable font size of at least 11 points, and (2) with very few exceptions, we wanted all recipes to fit on one page. With trial and error on margins and possible use of multiple columns, plus research on best practices in typography, and testing of various fonts, we ended up with the trim size and margins described below. Then we used the `geometry` package to set those parameters.

We chose an industry standard trim size of 8 x 10 inches. That trim size allows for printing by most print-on-demand printing companies.

It is tempting to set small margins to allow for more text on the page. Typographers have given much thought to these matters and it just isn't that simple. A useful discussion is found in Chapter 2 of the KOMA-script documentation[5]. As a result, we increased our margins well beyond our initial instincts. Even so, there is still not as much margin space as some experts might like.

We set the outer margins to 1.0 inch. When the book is open, you see both sides of the inner margin together; it is therefore recommended that you set the inner margins to one-half the outer margin (here, 0.5 inch). Visually, it is as if you had three margins: a 1-inch verso outer margin, a 1-inch combined recto/verso inner margin, and a 1-inch recto outer margin. You should also add a ***binding offset*** to compensate for the part of the inner margin that disappears into the binding – we have set the binding offset (for *each* side of the inner margin) to 0.375 inches.

Because we are using footers but not headers, we have set the top margin to 0.75 inches and the bottom margin to 1.0 inch. That leaves the text area at 6.125 x 8.25 inches, about 63% of the full page size.

We have set the distance between the bottom of the text area and the baseline of the footer to 40 points, which seems to give a nice separation between the text area and the footer.

The above is obtained with the following `geometry` package settings:

```
\usepackage{geometry}

\ifCookbookDraft
\geometry{paper=letterpaper,     % the physical paper size during draft mode
  layoutsize={8in,10in},         % always use intended final paper size for layout
  layouthoffset=0.25in,          % center the "layout" horizontally
  layoutvoffset=0.5in,           % center the "layout" vertically
  %showframe,                    % use when needed
  showcrop}
\else
\geometry{papersize={8in,10in}}  % the physical paper size in final production mode
\fi
```

[5] https://ctan.org/pkg/koma-script

```
\geometry{nomarginpar,          % do not reserve space for margin notes
  bindingoffset=0.375in,
  inner=0.5in,
  outer=1in,
  top=0.75in,
  bottom=1in,
  footskip=40pt}                % default seems to be 27pt
```

When you submit your cookbook to a print-on-demand printer, the PDF file's metadata must show the paper size equal to the intended trim size. In the `geometry` package, this is done by making sure that `papersize` equals `layoutsize`. However, during the time you are working on your cookbook (in draft mode), you want to set the `geometry` package `papersize` equal to the physical size of the paper coming out of your local printer. In our case, we are printing draft pages in `letterpaper` size (8.5 x 11 inches). You manage this draft vs. final difference, by setting the `\newif` value of `\ifCookbookDraft` to `\CookbookDrafttrue` (draft) or `\CookbookDraftfalse` (final). You will find these settings near the very top of `cb-preamble.tex`.

Just modify the above `geometry` settings for your needed trim size, draft paper size, margins, etc.

3 Headers and Footers

LaTeX sets headers and footers with the `pagestyle` command. Several pagestyles are predefined by LaTeX, but you are also allowed to define your own. We are using the `book` document class, which by default uses: (1) the predefined `empty` pagestyle (no header or footer on the page), (2) the predefined `plain` pagestyle (no header, the footer contains only a centered page number), and (3) a modified version of the predefined `headings` pagestyle.

In the cookbook, we use the `empty` pagestyle, a modified version of the `plain` pagestyle, plus a `main` pagestyle that we define. We use the `fancyhdr` package to redefine the `plain` pagestyle and to define our own `main` pagestyle, as follows:

```
\usepackage{emptypage, fancyhdr}        % for emptypage, see Q360739
  % NOTE: RO = right/odd; LE = left/even; CE = center/even; CO = center/odd
\fancypagestyle{plain}{%
  \fancyhf{}                            % clear the header and footer
  \renewcommand{\headrulewidth}{0pt}    % use 0 to disable header ruler line
  \renewcommand{\footrulewidth}{0.2pt}
  \fancyfoot[RO, LE] {Page \thepage}}

\makeatletter             % \makeatletter must be OUTSIDE the command - see Q 444532
\fancypagestyle{main}{    % identical to plain, except in mainmatter, where
                          % it includes \leftmark in the center of the footer
  \fancyhf{}
  \renewcommand{\headrulewidth}{0pt}
  \renewcommand{\footrulewidth}{0.2pt}
  \fancyfoot[RO, LE] {Page \thepage}
  \fancyfoot[CE,CO]{\if@mainmatter \leftmark\fi}} %  See Q340125
\makeatother
```

The above code does as follows:

- By loading the `emptypage` package, we force all completely empty pages to use the `empty` pagestyle.
- The redefined `plain` pagestyle: (1) has no header and no header-area rule line, (2) has a footer-area rule line that is 0.2 points thick, and (3) prints the page number at the outer margin – the right side of the footer on *recto* (odd numbered) pages and the left side of the footer on *verso* (even numbered) pages.
- The `main` pagestyle is identical to the redefined `plain` pagestyle, except, *in the mainmatter area only*, it prints the `\leftmark` in the center of the footer on both even and odd numbered pages. In the `book` class, the `\leftmark` is in all-caps and looks like "CHAPTER 2. COOKIES".

With our pagestyles defined, we now describe how they are used. We follow a very typical approach to headers and footers in printed books:

- Empty Page: if a page is otherwise completely empty, we use the `empty` pagestyle. This rule applies through the frontmatter, mainmatter and backmatter, *and supersedes all other rules*. An empty page happens, for example, when a chapter ends on a recto page. Since all new chapters start on a recto page, the intervening verso page is completely empty.
- Frontmatter: We employ the `empty` pagestyle from the beginning (title page) up to and including the page just before the `\tableofcontents` page. Then, from that page through the end of the frontmatter, we employ the (redefined) `plain` pagestyle. We follow the default rule for the `book` class, where frontmatter page numbers are indicated by small roman numerals. Note that there is "pagination" (page counting) from the first page, but only printing of the page number after switching to the `plain` pagestyle.
- Mainmatter: We employ the `main` pagestyle, except on the first page of each `\chapter`, which switches to the redefined `plain` pagestyle (this switch is the default book class `\chapter` behavior, so requires no coding by us). Page numbers are arabic, with page 1 being the first page of the first mainmatter chapter.
- Backmatter: We employ the `plain` pagestyle. Page numbers are arabic, and continue with the pagination from the mainmatter.

Our code only needs to issue two `\pagestyle` commands: (1) we include `\pagestyle{empty}` as the first line of `cb-frontmatter.tex`, and (2) we include `\pagestyle{main}` just after the `\tableofcontents` command in `cb-frontmatter.tex`. (Recall that the `main` pagestyle is identical to the `plain` pagestyle in the frontmatter and backmatter). To implement our rule for empty pages, we load the `emptypage` package, which automatically applies the `empty` pagestyle to all empty pages.

The `fancyhdr` documentation is a good source of additional information on these matters.

4 Chapter Title Formatting

We use the `titlesec` package for the formatting of chapter titles. Following are some key concepts of the `titlesec` package:

- The ***label*** means the basic chapter information, such as "Chapter 6".
- To obtain the current ***label***, you use the `\chaptertitlename` command (to obtain "Chapter") and the `\thechapter` command (to obtain the current chapter number). Importantly, per the `book` class default, both of those commands are blank in the frontmatter and backmatter, so we only obtain a non-empty ***label*** in the mainmatter.
- The ***title body*** means the text identifying the current chapter, such as "Sauces and Chutneys".
- the ***title*** means the entire chapter title (*label* plus *title body*).
- Two commands are provided to change the ***title*** format. The `\titleformat` command is used for the "internal" format (i.e., shape, font, label, etc.) and the `\titlespacing` command defines the "external" format (i.e., spacing before and after, etc.).

Following is our code:

```
\usepackage{titlesec}

\titleformat{\chapter}[display]    % [display] puts the label in a separate paragraph
  {\filleft\FontChapterLabel}      % The format for the whole title (label and title body text)
  {\chaptertitlename\ \thechapter} % This defines the text for the label
  {1pt}                            % The horizontal separation between label and title body:
           % Next is optional code preceding the title body. We change
           % the title body font from the initial setting above.
           % we include \raggedleft because text may exceed one line.
  {\titlerule\vspace{1ex}\raggedleft\FontChapterTitle}

\titlespacing*{\chapter}           % The starred version kills the indentation of the
                                   %     paragraph following the title.
  {0pt}                            % amount to increase left margin
  {20pt}                           % vertical space before title
  {20pt}                           % verticle space between title and text
```

The `titlesec` documentation is a good source of additional information on these matters.

5 Odds and Ends

5.1 Color

An important decision in a cookbook is whether to print in black and white or color. The printing costs can be substantially different. One option is to first print a small number of books in black and white, live with the book for a while, edit as necessary, and then move to color printing when you are fully satisfied with the final product.

Our approach is to define all colors used in the cookbook in one place in the preamble. We use the `xcolor` package and the `\definecolor` command to create our defined names for all of our colors. That allows us (in one place) to change those color definitions to accommodate either a black and white or color cookbook.

You should check with your print on demand printer to determine whether they prefer (or require) either the CMYK or RGB color model. In our code, below, we provide examples of both color models:

```
\usepackage{xcolor}
\definecolor{clrWhite}{cmyk}{0.00, 0.00, 0.00, 0.00}    % true white
\definecolor{clrBackTip}{rgb}{1.0, 0.95, 0.95}          % red!5!white
\definecolor{clrFrameTip}{rgb}{0.75, 0.0, 0.0}          % red!75!black
\definecolor{clrBackCheffy}{rgb}{1.0, 1.0, 1.0}         % white
\definecolor{clrFrameCheffy}{rgb}{0.0, 0.0, 0.75}       % blue!75!black
\definecolor{clrBackNotes}{rgb}{1.0, 1.0, 1.0}          % white
\definecolor{clrFrameNotes}{rgb}{0.0, 0.75, 0.0}        % green!75!black
\definecolor{clrLettrineBig}{gray}{0.5}
\definecolor{clrLettrineSmall}{gray}{0.5}
\definecolor{clrIngTitle}{cmyk}{0.00, 1.00, 1.00, 0.00} % true red
\definecolor{clrEditNote}{cmyk}{0.00, 1.00, 1.00, 0.00} % true red
\definecolor{clrHyperRef}{cmyk}{0.00, 1.00, 1.00, 0.00} % true red
```

See the `xcolor` package for helpful information on the CMYK and RGB color models.

5.2 Drop Cap

A *drop cap letter* is a single letter (usually at the beginning of a chapter or important paragraph) that is larger than the following text. The practice began more than 2,000 years ago. Originally, the drop cap was very ornate and several lines high. As typesetting took hold in the mid 15th century, the typesetter would leave the necessary blank space to allow for a hand-drawn drop cap. The drop cap served two purposes: (1) it was a decorative element and (2) it assisted the reader by dividing the text into different parts. By the latter part of the 19th century, drop caps had mostly lost their ornamental flourish and usually consisted only of a larger letter (maybe sized to two or three lines) signifying the start of a chapter or section.

There was a substantial reduction in the use of drop caps at the beginning of the 20th century. More recently, there has been a bit of a revival. Our review of modern cookbooks found that it now quite common, but hardly universal. We have elected to use drop caps in the chapter intro and in the recipe story, but our code does not require it. We use the `lettrine` package to implement our drop caps ("lettrine" is the French word for drop caps). As is customary, the `lettrine` package (optionally) uses a different font style for the first several characters following the drop cap (by default they are set to small caps). We call these special characters the *drop caps text*. Our code:

```
\usepackage{lettrine}
\renewcommand{\LettrineFontHook}{\MyScriptFont\color{clrLettrineBig}}
\renewcommand{\LettrineTextFont}{\color{clrLettrineSmall}\FontLettrineText}
\renewcommand{\DefaultOptionsFile}{cb-lettrine.cfl}
```

We use the `\LettrineFontHook` command to set the font (and color) of the *drop cap letter* to our script font. We use the `\LettrineTextFont` command to set the font (and color) of our *drop caps text* to a small-caps version of the current roman font. Finally, we use the `\DefaultOptionsFile` command to point to a `lettrine` package configuration file (here `cb-lettrine.cfl`) where we fine-tuned the height (in rows) and width of each alphabetic letter that may be used as a *drop cap letter*. See the `lettrine` package documentation for more details.

5.3 Images/Photos

There is very little material here on using images in your cookbook. We use the well-documented `graphicx` package and its`\includegraphics` command as the starting point for displaying photos. Our advice is to use non-LaTeX programs to put your images in final form (any rotation or cropping needed, as well as any needed adjustments to size or dpi).

By way of example, we include the code for our `\SideBySide` command, used for displaying two images side-by-side. Some comments on the setup:

- Recall that all of our `*.tex` files (and therefore our "current directory") will either be in: (1) the root directory of the cookbook project, or (2) the `tex` subdirectory under that root directory. Also, all of our image files will be in the `img` subdirectory under that root directory. We use the `\graphicspath` command to tell LaTeX to look for the image files either: (1) in the `img` subdirectory under the current directory, or (2) in the `img` directory that is a sibling of the current directory.
- We use the `caption` package to manage any caption we put under an image. With `skip=2pt`, the vertical distance between the image and caption is set to 2 points. With `labelformat=empty`, we have an empty caption label. With `font={rm,it}`, the caption font is `\rmfamily` and italic. The `caption` package options can be selected in the `\usepackage` command, or separately in the `\captionsetup` command. To demonstrate the latter, we select the `justification=centering` option in `\captionsetup`.
- The `\SideBySide` command has two mandatory arguments. For the two side-by-side images, **Arg #2** is the left image and **Arg #3** is the right image.
- **Arg #1** is an optional key-value argument with three possible entries. `VertAlign=` defaults to `c` (vertically align images at their centers) and can also be set to `t` (align at their tops) or `b` (align at their bottoms). `LeftCaption=` and `RightCaption=` default to `\empty` and can be use to include a caption under one or both of the images.

```
\graphicspath{{img/}{../img/}} % look in img directory (subdir of book root or sibling of tex)
\usepackage[skip=2pt, labelformat=empty, font={rm,it}]{caption} %
\captionsetup{justification=centering} % this is needed to have multi-line captions centered

\pgfkeys{
  /SideBySide/.is family, /SideBySide,
  default/.style = {VertAlign = c, LeftCaption = \empty, RightCaption = \empty},
  VertAlign/.estore in    = \VerticalAlign,
  LeftCaption/.estore in  = \LeftText,
  RightCaption/.estore in = \RightText,
}

\NewDocumentCommand \SideBySide{O{\empty} m m} % Q 5769
{
\pgfkeys{/SideBySide, default, #1}%
%
\begin{figure}[htb]
  \centering
  \begin{minipage}[\VerticalAlign]{0.49\textwidth} % align at t= top, c = center, b = bottom
    \centering
    \includegraphics[width=0.97\textwidth]{#2} %
    \ifx\LeftText\empty\relax\else\caption{\LeftText}\fi
  \end{minipage}\hfill%
  \begin{minipage}[\VerticalAlign]{0.49\textwidth}
    \centering
    \includegraphics[width=0.97\textwidth]{#3} %
    \ifx\RightText\empty\relax\else\caption{\RightText}\fi
  \end{minipage}
  \end{figure}
}
```

6 Front Matter, Main Matter and Back Matter

6.1 Introduction

Almost all books have a structure consisting of three sections: the front matter, the main matter and the back matter. We describe here the organization and contents of those three sections. Although there are many options regarding the organization and contents of those three sections, we limit our discussion to the actual (and very typical) structure we have selected.

6.2 Front Matter

The **front matter** (sometimes called "preliminaries") is the first section of a book. Because the front matter is normally the last section of a book to be completed, small roman numerals are traditionally used for the front matter page numbers. That way, last-minute changes will not require renumbering the main text.

For our cookbook, the front matter is contained in the file `cb-frontmatter.tex`. We now describe, in order of appearance, the parts that make up the front matter.

6.2.1 Title Page

It is actually typical for old-line publishers to start with a half-title page (just the book name and no other information) and then the full title page (including author's name and possibly other information). We include only a full title page.

6.2.2 Copyright Page

Here we provide the copyright information. As is common, we also includes information on the publisher, the book edition and edition history, the ISBN number (if any), and the Library of Congress number (if a USA book). We also include here information about the production, design and fonts (the "colophon"). *You will need to make several edits to this page – see the "front matter helper commands" section of the preamble.*

6.2.3 Dedication

The dedication, if any, follows the copyright page and is the only element on that page.

6.2.4 Table of Contents

We provide the page number for each chapter in the book, and for the index. If your book has an appendix (or similar), you should also include that in the table of contents.

With the book class, no package is required to include chapters, sections and subsections in the table of contents. Because our cookbook has no sections or subsections, we list only chapters (plus the index) in our table of contents. We insert the table of contents at the desired location in the front matter simply by issuing the `\tableofcontents` command. Section 6.4 (page 14) shows how we add the index to our table of contents.

6.2.5 Other

Following the table of contents, there may be some or all of the following (each as a separate chapter and each included in the table of contents): foreword, preface, acknowledgments, introduction, or similar.

6.3 Main Matter

The **main matter** (sometimes called the "body matter") follows the front matter. This is the actual text of your book. We follow the typical case where the main matter consists only of the chapters of the book. The first page of the main matter is page 1 of the book (Arabic numbering).

6.4 Back Matter

The **back matter** (sometimes called the "end matter") follows the main matter. Ours is a typical case, where the back matter contains only the book index. The back matter is where you might include other ancillary information such as an appendix, notes, a glossary, a bibliography, etc. Our index consists only of the recipes in the cookbook. See page 22 and the `\RecipeNameAndYield` command for how we put our recipes in the index.

Following is the code to include the index in the back matter. First, from `cb-preamble.tex` we have:

```
\usepackage{imakeidx}   % supports creation of an index (here, a recipe index)
\makeindex[intoc]       % make the *.idx file; intoc = include this Index in TOC (Q 59619)

\NewDocumentCommand \CookbookIndex{}{
  \cleardoublepage % flush all material and clear until you start new odd numbered (recto) page
  \phantomsection\addcontentsline{toc}{chapter}{\indexname} % see also Q 59619
  \printindex
}
```

And from `makecookbook.tex` we have:

```
\backmatter
\CookbookIndex{}
```

With the above code, you get the standard index format for the `book` document class (plus inclusion in the table of contents). We have also added code that modifies the format of the index in three ways: (1) it slightly changes the hanging indent of long index items, (2) it right justifies the page number associated with the index item, and (3) it "dot fills" between the index item and the page number. Our code:

```
\makeatletter
\def\@idxitem{\par\hangindent 10pt} % not needed unless you want to fine tune hanging indent
\newcommand{\betterdotfill}         % see Q 396898
  {\leavevmode \nobreak\cleaders \hb@xt@ .44em{\hss .\hss }\hskip .5em plus 1fill \kern \z@}
\makeatother
\makeindex[options=-s cb-idxstyle]  % use cb-idxstyle.ist for style; Q 132465 & Q 396898
```

You might want to temporarily comment out this additional code, just to compare the results to the standard `book` class index format.

The above code: (1) slightly changes the hanging indent for index items that are longer than a single line (this code is definitely *not* necessary), (2) defines the `\betterdotfill` command (more on this below), and (3) uses the `\makeindex` command to load an "index style file" named `cb-idxstyle.ist` to modify the format of the index[6]. The `cb-idxstyle.ist` file is a text file with the following contents:

```
delim_0 "\\betterdotfill "
delim_1 "\\betterdotfill "
delim_2 "\\betterdotfill "
```

What is with that strange `\betterdotfill` command (courtesy of Enrico Gregorio in Q 396898)? If you still have the default setup of the `makecookbook` package (including fonts, margins, etc.), we can demonstrate why it is needed. In the above `cb-idxstyle.ist` file, change the three `betterdotfill` entries to standard `dotfill` entries:

```
delim_0 "\\dotfill "
delim_1 "\\dotfill "
delim_2 "\\dotfill "
```

Now compile and look at the index entry for the recipe named "G – Potato Salad with Sherry Shallot Vinaigrette". That recipe is *exactly* the wrong size, causing `\dotfill` to fail. The page number is not right justified and there is no dot fill. Enrico's magic code[7] solves the problem by forcing the last part of "wrong-sized" recipe names to spill over to the next line.

[6] See `https://ctan.org/tex-archive/indexing/makeindex/paper/ind.pdf` for documentation on the index style file.

[7] Don't ask us to fully explain Enrico's code, although there are strong hints in *The TeXbook* by Donald E. Knuth.

Part V
Elements of a Book – the Details (Fonts)

1 Introduction

The METAFONT font selection scheme in original TEX was developed beginning in the late 1970's. Although advanced for its time, it is quite limiting by today's standards. In LATEX, a 'New Font Selection Scheme' (NFSS) was released in 1989 and then updated in 1993. While certainly a great improvement, allowing package writers to make many new fonts available, it is still a cumbersome and limiting system.

Backward compatibility has been a limiting factor in LATEX development, due to: (1) the expectation that older documents will still compile and produce identical output and (2) the large ecosystem of third-party packages. Until recently, one of the most important advances was the `pdfTeX` compiler, which produced PDF output directly from a TEX or LATEX compile. However, `pdfTeX` is still an 8-bit system with the same font limitations. While there are packages and methods for incorporating Unicode in a `pdfTeX` compile, it is both kludgy and incomplete.

Recently, the XeTeX and LuaTeX compilers have been released. They are Unicode-based 32-bit systems. Both can load any OpenType (including TrueType) font installed on your computer – all you need is the `fontspec` package. That allows easy use of the advanced typographic features of OpenType. It must be noted that these two new compilers have some subtle incompatibilities with TEX and LATEX . However, those incompatibilities are minimal and will impact a small group of uses – primarily those with old legacy code. Of course, `pdfTeX` is still available for any such legacy code.

To allow for more modern font handling, we therefore decided to use the LuaTeX compiler. As between LuaTeX and XeTeX, we decided that LuaTeX is a better choice due to its more complete support of the `microtype` package and for the option (which we have not used) of Lua scripting.

1.1 Fonts

We discussed on page 4 the three fonts you must install if you want a successful compile "out of the box". Here we discuss the attributes required of substitute fonts. We begin by again listing the three font, along with the weights and shapes required of substitute fonts:

Serif *EB Garamond* (version from Google Fonts). Any substitute font should have regular and bold weights, both to include the italics shape. Also, the regular weight should include the small caps shape.

Sans Serif *Lato* (version from `www.latofonts.com`). Any substitute font should have regular, semi-bold and bold weights, all to include the italics shape.

Script *Italianno* (version from Google Fonts). We only need the regular font weight and shape.

In addition to the above, the substitute fonts must be OpenType (including TrueType) and should support the `Ligatures=TeX` font feature described in the `fontspec` package manual. Finally, the *serif* and *sans serif* fonts should support the following (there are workarounds if they do not):

- These glyphs: *degree* (char 176), *copyright* (char 169), *center dot* (char 183), and *bullet* (char 8226).
- The `Fractions=On` OpenType font feature, as described in the `fontspec` package manual.

On page 18 and following, below, we discuss the simple changes to `cb-preamble.tex` you must make to substitute your own favorite fonts.

2 Line Length and Font Size

Part of the font selection process is choosing a proper point size for the intended use. And these decisions directly impact other issues, such as the choice of trim size and margins. We consider here the important relationship between point size and line length.

Lines of text can be less comfortable to read if they are either too long or too short. Following are a couple of (sometimes contradictory) rules of thumb for the body text of a printed book[8]:

- Line length should be between 45 and 75 characters per line. For printed works with multiple columns, 40 to 50 characters per line is preferred.
- Line length should be about 30 times the point size of the font, with an acceptable range between 20 and 40 times. We call that number the *point multiple*, and calculate it by dividing the length (in points) of the line by the point size of the font. We assume that 1 inch equals 72.27 points (often rounded to 72 points in desktop publishing applications). Our 6.125-inch text margin is therefore about 443 points wide.

Now let's relate the above rules of thumb to our four different layouts of body text.

- The introductory text at the beginning of each chapter has a font size of 14 points. Dividing 443 by 14, we have a nearly ideal *point multiple* of about 31.6. We roughly calculate the characters per line at between 78 and 80 – a little high, but we find the text easily readable and note the *point multiple* is right where it needs to be.
- The introductory text at the beginning of each recipe has a font size of 13 points. Also, the text is indented on each side by 20 points, giving a text width of 403 points. Dividing 403 by 13, we have a nearly ideal *point multiple* of about 31. We roughly calculate the characters per line at about 78 – again, a little high, but we find the text easily readable and note the *point multiple* is right where it needs to be.
- The recipe ingredients and steps are presented in a two-column format, with a font size of 10.95 points. Listing of ingredients should not be considered "body text", so we focus on the recipe steps. With a 10 point separation between the two columns, each column is about 217 points wide. Dividing 217 by 10.95, we have a *point multiple* of just about 20 – at the low end, as might be expected for multi-column text. We roughly calculate the characters per line at 48, right where it should be for multi-column text.
- The copyright page uses a font size of 9 points. This is a "fine print" page and should not be considered as body text. It is of no concern that the characters per line and *point multiple* are not within prescribed parameters.

3 Our Selected Fonts

Below we provide additional detail on the fonts and font sizes we selected.

3.1 Serif Font

For the `makecookbook` package, we selected the *EB Garamond* font. Garamond (produced in many versions) is an old-style serif typeface, named for sixteenth-century Parisian engraver Claude Garamond. It and related typefaces are very popular for printing body text in books. It is a classic design that does not shout "look at me" but just unobtrusively makes for very easy to read text.

EB Garamond is the primary font used in the cookbook. It is used everywhere in the cookbook that is not specifically mentioned below for the other fonts. Some of the key places where this font is used:

- When listing the steps to make a recipe. This is in a two-column environment, using a 10.95 point font.
- At the beginning of recipes, when telling a story about the recipe or describing other special information regarding the recipe. This is in a one-column environment, indented 20 points on both the left and right sides. We use a 13 point font.
- At the beginning of each chapter, when discussing the recipes in that chapter. This is in a one-column full-width environment, using a 14 point font.
- On the copyright page, where it is typical to have a smaller font. We use a 9 point font.
- In the chapter title, this font is used as the chapter ***label*** (e.g., "Chapter 7"), using a 14.4 point font.

[8] See, e.g.: `https://en.wikipedia.org/wiki/Line_length` and the references cited there.

3.2 Sans Serif Font

For the `makecookbook` package, we selected the *Lato* font. *Lato* is described as "clean and modern" as well as "transparent" (unobtrusive), meeting our goal for easy to read but low-key body text fonts. This is the only other font used for body text. It is used in two places:

- When listing recipe ingredients. This is in a two-column environment, using a 10.95 point font. We use the semi-bold face to provide a better visual contrast between recipe ingredients and recipe steps.
- When a recipe has multiple sections of ingredients/steps, we use this font for the title of a recipe section. It is still 10.95 points, but in italics with a bold face.

3.3 Script

For the `makecookbook` package, we selected the *Italianno* font. We believe it provides an interesting contrast to our two body text fonts, while still remaining readable. It is used in three places:

- In the chapter title, it is used for the name of the chapter. We use a 40 point font size.
- In recipes, it is used for the recipe name. We use a 24 point font size.
- As the large *drop cap letter*. (Font size determined by the `\lettrine` package).

Side Note

We have a confession to make. For our cookbook, we do not use the *EB Garamond* or *Lato* font. (We do use *Italianno*). In their place, we use the *Adobe Garamond Pro* and *Adobe Myriad Pro* commercial fonts. Although both implementation of Garamond are excellent, for our main font we wanted a well-tested and reliable font (*EB Garamond* is still in development). For our secondary font, we just very much like the look and readability of the Myriad typeface. Of course, we could not use a commercial font in this package.

4 Code to Implement our Font Usage

4.1 Load Our Fonts

The first step is to load our three fonts using the `fontspec` package:

- Our *serif* font is loaded using the `\setmainfont` command. After that, our code refers to this family as the `\rmfamily`.
- Our *sans serif* font is loaded using the `\setsansfont` command. After that, our code refers to this family as the `\sffamily`.
- Our *script* font is loaded using the `\newfontfamily` command. The `\newfontfamily` command allows us to name the family (similar to `\rmfamily`). We name our script font `\MyScriptFont`.

All three font loading commands include a mandatory argument to identify the font, plus various optional key-value augments. The `fontspec` package provides several options for identifying the font. We use their "by file name" option. The easiest way to understand this is to look at the six font files that make up *EB Garamond's* regular, regular italic, bold, bold italic, semi-bold and semi-bold italic font faces:

- `ebgaramond-regular.ttf`
- `ebgaramond-italic.ttf`
- `ebgaramond-bold.ttf`
- `ebgaramond-bolditalic.ttf`
- `ebgaramond-semibold.ttf`
- `ebgaramond-semibolditalic.ttf`

The leading part of the font name (`ebgaramond`) is entered as the mandatory argument. The `.ttf` file extension is entered in the optional key-value argument: `Extension=.ttf`. The six font faces are identified by their filename, with a `*` representing the leading part of their filename. The complete code to load the fonts (see the `fontspec` package for more details):

```
\usepackage{fontspec}
\usepackage{microtype}

\setmainfont{ebgaramond}[
  Extension=.ttf,
  UprightFont=*-regular,
  ItalicFont=*-italic,
  BoldFont=*-bold,
  BoldItalicFont=*-bolditalic,
  FontFace={sb}{n}{*-semibold},
  FontFace={sb}{it}{*-semibolditalic},
  Ligatures=TeX,
  Numbers=Lining]

\setsansfont{lato}[
  Extension=.ttf,
  UprightFont=*-regular,
  ItalicFont=*-italic,
  BoldFont=*-bold,
  BoldItalicFont=*-bolditalic,
  FontFace={sb}{n}{*-semibold},
  FontFace={sb}{it}{*-semibolditalic},
  FontFace={k}{n}{*-black},
  FontFace={k}{it}{*-blackitalic},
  Ligatures=TeX,
  Numbers=Lining]

\newfontfamily\MyScriptFont{Italianno}[
  Extension=.otf,
  UprightFont=*-Regular-OTF,
  Ligatures=TeX]
```

As mentioned above, in our cookbook, we use the *Adobe Garamond Pro* and *Adobe Myriad Pro* fonts. To replace *EB Garamond* and *Lato* with those fonts, it is as simple as changing the `\setmainfont` and `\setsansfont` commands with (you can do the same to substitute your favorite fonts):

```
\setmainfont{agaramondpro}[
  Extension=.otf,
  UprightFont=*-regular,
  ItalicFont=*-italic,
  BoldFont=*-bold,
  BoldItalicFont=*-bolditalic,
  FontFace={sb}{n}{*-semibold},
  FontFace={sb}{it}{*-semibolditalic},
  Ligatures=TeX,
  Numbers=Lining]

\setsansfont{myriadpro}[
  Extension=.otf,
  UprightFont=*-regular,
  ItalicFont=*-it,
  BoldFont=*-bold,
  BoldItalicFont=*-boldit,
  FontFace={sb}{n}{*-semibold},
  FontFace={sb}{it}{*-semiboldit},
  FontFace={k}{n}{*-black},
```

```
FontFace={k}{it}{*-blackit},
Ligatures=TeX,
Numbers=Lining]
```

4.2 Handle Special Font Faces

We handle the two special font faces, semi-bold and black, giving them commands that are similar to the built-in commands used, for example, by bold (i.e., similar to the `\textbf` and `\bfseries` comands):

```
\NewDocumentCommand \sbseries {}{\fontseries{sb}\selectfont}
\DeclareTextFontCommand{\textsb}{\sbseries}
\NewDocumentCommand \kseries {}{\fontseries{k}\selectfont}
\DeclareTextFontCommand{\textk}{\kseries}
```

4.3 Commands to Select Our Fonts

Next, we provide the commands needed to select the fonts. This gives us a level of indirection between the actual font used and the more abstract name we use for that font. In referencing/selecting a font, we will only use the below command names:

```
\NewDocumentCommand \FontSteps          {}{\rmfamily\mdseries}
\NewDocumentCommand \FontStepsDefault   {}{\rmfamily\mdseries}
\NewDocumentCommand \FontIngredients    {}{\sffamily\sbseries}
\NewDocumentCommand \FontIngDefault     {}{\sffamily\sbseries}
\NewDocumentCommand \FontIngTitle       {}{\sffamily\bfseries\itshape}
\NewDocumentCommand \FontChapterIntro   {}{\rmfamily\fontsize{14}{16.8}\selectfont}
\NewDocumentCommand \FontRecipeStory    {}{\rmfamily\fontsize{13}{15.6}\selectfont}
\NewDocumentCommand \FontCopyrightPage  {}{\rmfamily\fontsize{9}{11}\selectfont}
\NewDocumentCommand \FontChapterLabel   {}{\rmfamily\fontsize{14.4}{18}\selectfont}
\NewDocumentCommand \FontChapterTitle   {}{\MyScriptFont\fontsize{40}{48}\selectfont}
\NewDocumentCommand \FontRecipeName     {}{\MyScriptFont\fontsize{24}{29}\selectfont}
\NewDocumentCommand \FontLettrineText   {}{\rmfamily\scshape\sbseries}
\NewDocumentCommand \FontTitleColorBox  {}{\rmfamily\Large\bfseries}
\NewDocumentCommand \FontTitlepageTitle {}{\MyScriptFont\fontsize{40}{48}\selectfont}
\NewDocumentCommand \FontTitlepageAuthor{}{\rmfamily\sbseries\scshape\fontsize{14.4}{18}\selectfont}
```

4.4 Special Handling of Fractions

For a cookbook, it is important to have a consistent way to display "nice" (and readable) fractions. For cookbooks, fractions are almost always of the "split level" type. Based on a suggestion in Q416164, we began by using the `xfrac` package and the `\sfrac` command, as follows:

```
\usepackage{xfrac}
\def\fr#1/#2 {\sfrac{#1}{#2} }
\def\frx#1/#2 {\sfrac{#1}{#2}}
```

The first `\def` allows you to type `\fr`1/2 to get nicely formatted ½ followed by a space. The second `\def` allows you to type `\frx`1/2 to get nicely formatted ½ where the space following your entry is "gobbled" by the `\def` because there is a space between `\def` and {. This second form is needed in the less common case when you want a parenthesis, comma, period or other character to immediately follow the fraction (i.e., no space between). Thus, to have a period immediately follow a nice ½ fraction, you would enter `\frx`1/2 . (Note the space between the 2 and the period).

Later we noted the discussion in Q234857 and the possibility of using the OpenType `Fractions=On` option. Our serif and sans serif fonts include full support for `Fractions=On`. So we replaced the `\sfrac` command with the `\addfontfeatures` command (below). Although `\sfrac` does a good job of producing "nice" fractions, the fractions look even nicer when you use an OpenType font that fully supports `Fractions=On`. The new `\def` still uses the same "space trick" as the old `\def`. NOTE: some OpenType fonts "support" `Fractions=On` only partially – if there is an internal "pre-made" fraction they will use that, but things don't otherwise look so nice (or consistent). Check you font.

Our final code (you can revert to `\sfrac` if you desired font does not support this feature):

```
\def\fr#1/#2 {{\addfontfeatures{Fractions=On}#1/#2} }
\def\frx#1/#2 {{\addfontfeatures{Fractions=On}#1/#2}}
```

4.5 Commands for Certain Glyphs

The degree, copyright, bullet and center (or mid) dot glyphs all have standard unicode character codes, and all are available in our serif and sans serif fonts. We therefore provide our own commands for these glyphs, rather than using the associated LaTeX commands. Our goal is to avoid the possible loading by LaTeX of other unnecessary fonts to create those glyphs. A font uses only for a few glyphs can cause some confusion, and even failed print jobs, at some of the print on demand companies.

Similarly, to avoid use of the math command `$\cdot$` (and possible loading of a math font), we "rolled our own" using our `\CtrDot` glyph and appropriate kerning to build the `\CtrDots` (plural) command.

If you use a font that does not provide these glyphs, use the LaTeX macros `\textdegree` or `\copyright` or `\textbullet` or (math) `$\cdot$`.

We also include here the code to use two "glyph-like" images: `\ChefHat` and `\Oven`. We made those two images using the `tikz` package and then saved them each in a TeX box. They are used in selected places, in the same fashion as normal font glyphs.

```
\NewDocumentCommand \TextDegree {}{{\char176}}   % or ^^^^00B0
\NewDocumentCommand \Copyright  {}{{\char169}}   % or ^^^^00A9
\NewDocumentCommand \TextBullet {}{{\char8226}}  % or ^^^^2022
\NewDocumentCommand \CtrDot     {}{{\char183}}   % or ^^^^00B7
\NewDocumentCommand \CtrDots    {}{{\CtrDot\kern 0.2em\CtrDot\kern 0.2em\CtrDot\kern 0.2em}}

\newsavebox{\HatBox}
\AtBeginDocument{\savebox{\HatBox}[\hatwidth]{\MakeChefHat}}%
\NewDocumentCommand \ChefHat    {}{\usebox{\HatBox}}%
\NewDocumentCommand \ChefNote   {}{{\raisebox{.4ex}{\ChefHat}}}

\newsavebox{\OvenBox}
\AtBeginDocument{\savebox{\OvenBox}[\ovenwidth]{\MakeOven}}%
\NewDocumentCommand \Oven       {}{\usebox{\OvenBox}}%
```

4.6 Point Size of Default Roman and Sans Fonts

The `IngredientsAndSteps` environment and the `\RecipeStory` and `\ChapterIntro` commands (discussed further below) need to know certain default font parameters (in points) for the fonts they use. We calculated those values beforehand and stored the hard-coded values:

```
\def\StdIFontSize{10.95}
\def\StdSFontSize{10.95}
\def\StdIBaseline{13.6}
\def\StdSBaseline{13.6}
\def\RStoryFontBaseline{15.6pt}
\def\CIntroFontBaseline{16.8pt}
```

Part VI
Elements of a Cookbook – the Details

1 Chapter Introduction

In a cookbook, chapters tend to be organized into logical units such as cookies, pasta, appetizers, etc. Following the chapter name, there will usually be some introductory text for that chapter. We call that the *chapter intro*.

Use the `\ChapterIntro` command to enter your *chapter intro* text. The command sets the default (roman serif) font to a larger size (14 points). A simplified version of the `\ChapterIntro` command follows. The actual code (page 31) includes an optional argument that you will rarely use:

```
\NewDocumentCommand \ChapterIntro {+m}{\FontChapterIntro{#1\par}} % the \par needed by \lettrine
```

As noted, the `\ChapterIntro` command sets a default font face and size. However, you are free to use most any of the LaTeX text markup commands. You might make some text **bold** or *italic*, or use an `enumerate` environment (e.g., to list you favorite cookie recipes). In our book, we have elected to use the `\lettrine` command to create a "drop cap" effect – see the separate discussion of this on page 12.

2 Recipe Name and Yield

A new recipe is introduced with a recipe name and, often, with information regarding the "yield" of the recipe (e.g., "makes 36 cookies" or "serves 4 to 6", etc.). To enter this information, we use the `\RecipeNameAndYield` command. It uses a key/value interface of the form:

```
\RecipeNameAndYield{⟨key=value, ...⟩}
```

The key/value options are as follows:

- [**Name=**] Required. This is the recipe name. it is the only required key/value option.
- [**Yield=**] Optional. You can optionally provide the "yield" of the recipe.
- [**NoIdxName=**] Optional. As a general rule, the recipe name (as entered in the **Name=** option) is included in the Index to the cookbook (in the back matter). However, if you set this key to the number '1', the recipe name will not be placed in the Index. The purpose of this option is to allow you to make enties in the `\index` that are somewhat different than the actual recipe name. For example, perhaps you want to slightly shorten the recipe name for the Index.
- [**IndexA=**] Optional. This is where you would enter alternate text for this recipe's index.
- [**IndexB=**] Optional. Use this if you want another recipe index entry.
- [**IndexC=**] Optional. Use this if you want yet another recipe index entry.
- [**XRefLabel=**] Optional. Normally, recipes do not have a cross-reference `\label` created. Use this key/value option if you want to create a cross-reference `\label` for this recipe (the value entered here is the `\label`). That allows you to reference this recipe's page number from another location in the cookbook.

Side Note

This is a good place to remind you of the special handling needed when the `value` element of a key/value entry includes a comma. Suppose, for example, that your recipe is *Pasta with Sausage, Tomatoes and Cream*. If your key/value entry is **Name=Pasta with Sausage, Tomatoes and Cream**, then they key/value parser will think the `Name=` entry ends at the comma. The solution is to put an extra set of curly braces around the value, such as: **Name={Pasta with Sausage, Tomatoes and Cream}**. That tells the parser to treat the internally braced text as one unit.

Following is the code defining the `\RecipeNameAndYield` command:

```
\pgfkeys{
  /RecipeNameAndYield/.is family, /RecipeNameAndYield,
    default/.style = {Name = 0, NoIdxName = 0, XRefLabel = \empty, Yield = \empty,
      IndexA = \empty, IndexB = \empty, IndexC = \empty},
    Name/.estore in      = \RecipeName,
    NoIdxName/.estore in = \NoIndexName,
    IndexA/.estore in    = \IdxA,
    IndexB/.estore in    = \IdxB,
    IndexC/.estore in    = \IdxC,
    XRefLabel/.estore in = \XRefLbl,
    Yield/.estore in     = \RecipeYield,
}

\NewDocumentCommand \RecipeNameAndYield {m}{%
  \pgfkeys{/RecipeNameAndYield, default, #1}%
    % Put the recipe name in the Index, unless the user sets NoIdxName = 1:
  \ifnum\NoIndexName=1 \relax\else\index{\RecipeName}\fi
    % Can add up to three other Index entries:
  \ifx\IdxA\empty\relax\else\index{\IdxA}\fi
  \ifx\IdxB\empty\relax\else\index{\IdxB}\fi
  \ifx\IdxC\empty\relax\else\index{\IdxC}\fi
    % Did the user ask us to set up a label for cross-reference?:
  \ifx\XRefLbl\empty\relax\else\RecipeLabel{\XRefLbl}\fi
    % Add a bookmark (only adds bookmark if hyperref is active):
  \ifnum\NoIndexName=1 \RecipeBookmark{\IdxA}\else\RecipeBookmark{\RecipeName}\fi
    % Now write the recipe name and (possibly) the yield
  \begin{center}%
    {\FontRecipeName{\RecipeName}}\par%
    \ifx\RecipeYield\empty\relax\else {\textit{\RecipeYield}}\par \fi%
  \end{center}%
}
```

3 Recipe Story

Often there is a story or other information specific to a recipe. In our design, that story follows the recipe's `RecipeNameAndYield` and precedes its `IngredientsAndSteps`. We call it the *recipe story*.

For reasons of readability, and for visual interest, we indent both the left and right margin of the *recipe story* by 20 points and use a font size of 13 points. However, like the chapter intro, you are free to use most any of the LaTeX text markup commands[9]. A simplified version of the `\RecipeStory` command follows. The actual code (page 29) includes an optional argument that you will rarely (if ever) use.

```
\def\RecipeStoryIndent{20 pt}
\NewDocumentCommand \RecipeStory {+m}{%
  \FontRecipeStory
  \leftskip=\RecipeStoryIndent \rightskip=\leftskip
  {#1\par}  % the \par needed by \lettrine
}
```

4 Ingredients and Steps

Following the recipe name/yield and (possibly) a recipe story, there is a list of the ingredients, plus the steps required to make the recipe. We have formatted this section into two columns, with the ingredients in a semi-bold face of the sans serif font and the steps in a normal face of the roman (serif) font. Both fonts use the default font size of 10.95 points.

[9] But see Q 66332 and Q 183569. You cannot use a list environment or similar.

The code to accomplish this uses the `IngredientsAndSteps` environment to establish the two-column format, and then the `\ListIngredientsAndSteps` command to list the ingredients and steps. We discuss them below.

4.1 The `IngredientsAndSteps` Environment

A simplified version of the `IngredientsAndSteps` environment follows. The actual code (page 29) is somewhat complicated. It has no mandatory arguments, but has several optional `key=value` style arguments that you will rarely (if ever) use.

When used without optional arguments, the `IngredientsAndSteps` environment is very simple. Its only purpose is to establish the two-column format, using code that is equivalent to the following:

```
\NewDocumentEnvironment{IngredientsAndSteps}{}
  {\begin{multicols}{2}}  % The "before environment" setup
  {\end{multicols}}       % The "after environment" cleanup
```

We simply provide a wrapper around the `multicols` environment where we select the two-column option.

4.2 The `\ListIngredientsAndSteps` Command

We use the `\ListIngredientsAndSteps` command to list the ingredients and provide the steps needed to make the recipe. There is one optional argument and two mandatory arguments, as follows:

- [**Arg 1**] Optional. In some recipes, there may be multiple sections of ingredients and steps. For example, the first section might be the ingredients and steps to make the cake, and the second section the ingredients and steps to make the frosting. In that case, you may want a label associated with each ingredients and steps section. The optional argument is the name of that section, formatted in bold italics and using the `clrIngTitle` color.
- [**Arg 2**] Required. This is the list of ingredients. Each ingredient entry is separated by a blank line (i.e., the paragraph indicator in LaTeX text entry). Each paragraph becomes an ingredient in the list, formatted as `\raggedright` (not right justified), with a separation of 2 points between paragraphs.
- [**Arg 3**] Required. This is the recipe steps. Each step entry is separated by a blank line (again, the paragraph indicator). Each paragraph becomes a recipe step. Unlike the ingredients, the steps are formatted as justified (the document default). We use the `enumerate` list environment to list the steps by number.

Following is the code defining the `\ListIngredientsAndSteps` command:

```
\NewDocumentCommand \IngredientsHeading {O{0} m O{2}}%
  {\begingroup \setlength{\parindent}{0pt} \ifnum #1 > 0 {\vspace{#1 pt}}\fi
  \FontIngTitle\color{clrIngTitle} #2\par\vspace{#3 pt} \endgroup}%

\NewDocumentCommand \ListIngredientsAndSteps {o +m +m}{%
  \IfValueT {#1}{\IngredientsHeading{#1}[3]}%
  {\begingroup\ifx\relax#2\relax\else\FontIngredients{}\IngredientsList{#2}\fi\endgroup}% arg2
  {\begingroup\ifx\relax#3\relax\else\FontSteps{}\RecipeSteps{#3}\fi\endgroup}%         % arg3
}

\NewDocumentCommand \IngredientsList { >{\SplitList{\par}} +m}{%
  \setlength{\parskip}{2pt}\raggedright%
  \ProcessList{#1}{\ProcessIngList}}

\newcommand\ProcessIngList[1]{\hangindent1em #1\par}

\NewDocumentCommand \RecipeSteps { >{\SplitList{\par}} +m}{%
  \begin{enumerate}[itemsep=2pt plus 1 pt minus 1pt, parsep=0pt plus 1pt,
        topsep=4.5pt plus 2.0pt minus 1.0pt, leftmargin=*]
    \ProcessList{#1}{\item}
  \end{enumerate}
}
```

As indicated in the description of the optional argument, please note that one `IngredientsAndSteps` environment may contain more than one `\ListIngredientsAndSteps` command.

5 Attribution

If the source of the recipe is known, it is proper to acknowledge that source. In that case, after the listing of ingredients and steps, we include an attribution. It is formatted as right justified and italics, in the default 11 point roman/serif font. The code:

```
\NewDocumentCommand \Attribution{m}{{\hspace*\fill}{\textit{#1}}}%
```

6 Additional Comments/Advice

There may be some recipe side notes and/or advice that is not properly included within the recipe, but is noted afterwards. We provide three separate environments to display that information – the `Tip`, the `Cheffy`, and the `ChefNotes` environments – all based on the `tcolorbox` package. All three environments take one mandatory arguments, which is the text (and any associated LaTeX markup commands) to be included in the body of the `tcolorbox` environment.

For example, the following code...:

```
\begin{Tip}
  {You can substitute \fr1/2 cup of vegetable or light olive oil for the butter.}
\end{Tip}

\begin{Cheffy}
  {Using some almond flour results in biscotti with a delightful crunchy texture without making
  them  hard. I encourage you try useing 25\% almond flour.}
\end{Cheffy}

\begin{ChefNote}
  {The amount of flour might vary if you use all-purpose flour versus "00" flour. If Tipo "00"
  flour, you might need a little more flour. Use the texture of your dough as a guide.}
\end{ChefNote}
```

...gives you the following results:

Food for Thought

You can substitute ½ cup of vegetable or light olive oil for the butter.

Let's Get Cheffy

Using some almond flour results in biscotti with a delightful crunchy texture without making them hard. I encourage you try using 25% almond flour.

Chefnotes

The amount of flour might vary if you use all-purpose flour versus "00" flour. If Tipo "00" flour, you might need a little more flour. Use the texture of your dough as a guide.

Of course, given the needs of your cookbook, you can easily change the title and color of these three environments. The LaTeX code follows (see page 12 for information on the colors used below):

```
\usepackage{tcolorbox}
\tcbset{fonttitle=\FontTitleColorBox}
```

```
\NewDocumentEnvironment{Tip}{+m}
  {
  \begin{tcolorbox}[colback=clrBackTip,colframe=clrFrameTip, title=Food for Thought]
    {#1}%
  \end{tcolorbox}
  }

\NewDocumentEnvironment{Cheffy}{+m}
  {
  \begin{tcolorbox}[colback=clrBackCheffy,colframe=clrFrameCheffy, title=Let's Get Cheffy]
    {#1}%
  \end{tcolorbox}
  }

\NewDocumentEnvironment{ChefNotes}{+m}
  {
  \begin{tcolorbox}[colback=clrBackNotes,colframe=clrFrameNotes,title=Chefnotes]
    \ChefNote{}{#1}%
  \end{tcolorbox}
  }
```

7 Odds and Ends

This section includes a number of cooking-related commands that were included in our preamble. Some you may find useful; others, not so much. Even if not right for your cookbook, we hope you can use the ideas here to make your own "helper" commands.

7.1 Various Simple but Useful Commands/Defines

We present the code immediately below and selectively discuss that code following:

```
\def\nl{\par}    % see Q 96247 for why we defined \nl for use with pgfkeys

\NewDocumentCommand \PreheatC{m}{Preheat oven to #1\Degrees convection.\thinspace\Oven}%
\NewDocumentCommand \PreheatR{m}{Preheat oven to #1\Degrees regular oven.\thinspace\Oven}%

\NewDocumentCommand \Tbl{o}{\IfNoValueTF{#1}{tablespoon }{tablespoon#1}}%
\NewDocumentCommand \tsp{o}{\IfNoValueTF{#1}{teaspoon }{teaspoon#1}}%
\NewDocumentCommand \Pd{o}{\IfNoValueTF{#1}{pound }{pound#1}}%
\NewDocumentCommand \Ounce{o}{\IfNoValueTF{#1}{ounce }{ounce#1}}%
\NewDocumentCommand \Degrees{o}{\IfNoValueTF{#1}{\TextDegree{} }{\TextDegree{}#1}}%

\NewDocumentCommand \AxB{m m o}{{#1\thinspace{x}\thinspace#2}\IfValueT{#3}{#3}}%
\NewDocumentCommand \AxBxC{m m m o}{%
  {#1\thinspace{x}\thinspace#2\thinspace{x}\thinspace#3}\IfValueT{#4}{#4}}%

\NewDocumentCommand \Inch{m}{{#1-inch}}%
\NewDocumentCommand \EditNote{m}{{{\color{clrEditNote} #1}}}%
\NewDocumentCommand \Quote{m}{{``#1''}}%

\NewDocumentCommand \IngredientsSeparator{}{{\FontStepsDefault \CtrDots\CtrDots}}%
\NewDocumentCommand \SeparateParagraphs{}{{\vskip 5pt}}%
\NewDocumentCommand \Recipe{+m}{{\textit{\textsb{#1}}}}%
```

The definition of `\nl` is needed to get around a problem with the `pgfkeys` package (for handling `key=value` arguments), because `pgfkeys` does not allow use of `\par` in a `value`.

There is a story behind the `\Preheat` commands. Traditionally, cookbooks have assumed the reader has only a regular (not convection) oven. We went back and forth on whether we should stick with tradition.

When we surveyed friends and family most likely to use our cookbook, almost all had convection ovens. If we gave cooking times and temperatures only for regular ovens, those times and temperatures would be wrong for the vast majority of our readers. Our compromise was to include an `\Oven` image (drawn using the `tikz` package) after the `\Preheat` instructions. That oven acts as a reminder to the reader that they may need to adjust time and temperature based on their particular oven type. Of course, even if the reader has the indicated oven type, it also acts as a reminder that each oven is different – they should adjust if their oven normally cooks hotter or cooler. If the `\Oven` image is too quirky for you, just remove it from the `\Preheat` commands.

The five commands (beginning with `\Tbl` and ending with `\Degrees`) first came about due to indecision. When first entering our recipes, we couldn't decide whether "teaspoon" and "tablespoon" should be spelled our or abbreviated. We put off the decision by using these commands, which could be redefined later based on our final decision. We quickly ran into a problem in cases where we wanted no space after the command, such as plurals (needing an 's' after tablespoon), punctuation or closed parenthesis. That lead to the current approach, with an optional argument that is appended without an space; without the optional argument, a space is automatically inserted.

The next two commands (`\AxB` and `\AxBxC`) are used to get the right spacing when you want "5 x 7" or "5 x 7 x 2". By using `\thinspace`, we believe the results are better: e.g.: "5 x 7" and "5 x 7 x 2". We also use the same optional argument technique as with teaspoon.

We use the `\EditNote` command to insert editing notes (in red) in the cookbook, allowing comments and notes at places that need some further work. The `\Quote` command ensures that quote marks around text are the correct shape for beginning and ending quotes.

In some recipes, there is a logical separation of ingredients into two parts. The `\IngredientsSeparator` command provides a visual marker (6 center dots) for that separation.

7.2 \BakeUntil

No doubt, your collection of recipes come from all sorts of sources, many no longer even known to you. There is no standard way to describe the steps to make a recipe, so your recipes likely vary substantially in the ordering and wording used to describe common recipe steps. It is good goal to make your cookbook recipes as consistent as possible in this regard.

One place we constantly ran across this issue is in "Bake Until" instructions. We decided to standardize our instruction by use of a `\BakeUntil` command. This command takes one key/value style argument.

The key/value options are as follows:

- [**Min=**] Required. The minimum baking time.
- [**Max=**] Optional. The maximum baking time.
- [**TPick=**] Optional. If equal to 1, then the "until" is "until toothpick tests clean"
- [**GBrown=**] Optional. If equal to 1, then the "until" is "until golden brown"
- [**Until=**] Optional. Use this if you want any other "until" phrase.

Here are three examples:

```
\BakeUntil[Min=15, Max=18, GBrown=1]
\BakeUntil[Min=15, TPick=1]
\BakeUntil[Min=15, Max=18, Until=until firm to the touch]
```

The results:

Bake about 15-18 minutes, until golden brown
Bake about 15 minutes, until toothpick tests clean
Bake about 15-18 minutes, until firm to the touch

Following is the code defining the `\BakeUntil` command:

```
\pgfkeys{
  /BakeUntil/.is family, /BakeUntil,
  default/.style = {Min = 0, Max = 0, TPick = 0, GBrown = 0, Until = \empty},
    Min/.estore in    = \BkMin,
```

```
    Max/.estore in    = \BkMax,
    TPick/.estore in  = \BkTPick,
    GBrown/.estore in = \BkGBrown,
    Until/.estore in  = \BkUntil,
}

\NewDocumentCommand \BakeUntil{m}{%
  \pgfkeys{/BakeUntil, default, #1}%
  \newcount\MyCount                \MyCount = \BkMax%
  \newcount\MyCase%
  {Bake about \BkMin}%
  \ifnum\MyCount > 0 {-\BkMax}\fi
    { minutes}%
  \ifx\BkUntil\empty  \MyCase = 0 \else \MyCase = 1 \fi%
  \ifnum\BkTPick  = 1 \MyCase = 2 \fi%
  \ifnum\BkGBrown = 1 \MyCase = 3 \fi%
  \ifcase\MyCase%                          '0' case - do nothing
    \or {, \BkUntil}%                      '1' case - use the "Until" text
    \or {, until toothpick tests clean}%  '2' case - TPick
    \or {, until golden brown}%           '3' case - GBrown
  \fi
}
```

7.3 A Few Other Commands

Copyright. We establish the environment for the copyright page. In that environment: (1) the font size is set to 9 points, (2) there is no indentation of the first line of a paragraph, and (3) all of the text is pushed down (as far as possible) to the bottom of the page.

```
\NewDocumentEnvironment{CopyrightPage}{} % See 31186
  {\FontCopyrightPage\setlength{\parindent}{0pt}\par\vspace*{\fill}}
  {\clearpage}
```

Hidden Lines. When listing recipe ingredients and steps, the `multicol` environment matches the lengths of the first and second columns. There are some cases where the results don't look quite right. For example, the ingredients may take up *almost* all of the first column, but not quite. So the listing of steps begins with one (lonely) line in the first column. We use the `\InsertHiddenLines` command to insert an extra (non-visible) ingredient that tricks the `multicol` environment into moving the first line of the recipe steps to the second column.

The `\InsertHiddenLines` command takes one parameter: the number of hidden lines to insert:

```
\newcount\LineCount
\NewDocumentCommand \InsertHiddenLines{m}{%
  \LineCount = #1
    {%
    \newcount\foo \foo=0
    \loop                            % \loop starts the construct ended by \repeat
      \phantom{.}\par \advance \foo by 1
    \ifnum \foo < \LineCount \repeat % \repeat also "serves as" the \fi to the \ifnum
    }%
}
```

8 Handling of Long Recipes

The code for the `\RecipeStory` command and the `IngredientsAndSteps` environment is actually more complicated than was shown above. In both cases, the actual code includes a rarely-used optional parameter. For a longer recipe, these optional parameters allow certain tweaks to the recipe layout that may allow you to keep the recipe name plus the full text of the ingredients and steps on one page.

8.1 The "Real" \RecipeStory Command

In our standard setup, the recipe story comes after the recipe name and before the ingredients and steps. One potential problem is that the length of the recipe story may cause the ingredients and steps to continue past the bottom of the current page. To avoid that problem, we designed the `\RecipeStory` command to include an option of dividing the recipe story into two parts. That allows one part (or none) to print before the ingredients and steps and the remaining part to print after.

The parameters for the actual `\RecipeStory` command:

- [**Arg #1**] Optional. Permitted values: non-negative integers less than or equal to the number of lines in the recipe story. If a value is provided, then the given number of lines (possibly zero) of the recipe story will be printed *before* the ingredients and steps. To print the remaining lines after the ingredients and steps, you issue the `\Finish RecipeStory` command.
- [**Arg #2**] Required. This is the text of the recipe story. As mentioned above, this may also include many of the LaTeX text markup commands. (However, if you use the optional argument, your LaTeX code should not change the height of the baseline from the predefine `\RStoryFontBaseline` value).

The actual code follows:

```
\def\RecipeStoryIndent{20 pt}
\newbox\StoryBox
\newbox\StoryBoxA
\NewDocumentCommand \RecipeStory {o +m}  % arg1 = optional = lines for first part
{                                        % arg2 = story text
\setbox\StoryBox\vbox
  {
  \FontRecipeStory
  \leftskip=\RecipeStoryIndent \rightskip=\leftskip        % Q 66332
  {#2\par}           % the \par needed by \lettrine
  }
  \IfNoValueTF {#1}  % NoValue=TRUE or NoValue=FALSE?
  {\unvbox\StoryBox} % NoValue=TRUE, so flush the full StoryBox
  {                  % handle the NoValue=FALSE case
  \ifnum #1 > 0      % > 0 means we split the story into two part; print the first part here
    {                % = 0 means ALL of story is deferred
    \setbox\StoryBoxA=\vsplit\StoryBox to #1\dimexpr \RStoryFontBaseline %
    \unvbox\StoryBoxA%
    }
  \fi
  }
}

\NewDocumentCommand \FinishRecipeStory{}%
  {\ifvoid\StoryBox \else \smallskip\unvbox\StoryBox\par\medskip\fi} %
```

8.2 The "Real" IngredientsAndSteps Environment

There is one more thing to try if splitting the recipe story, above, does not work to fit the recipe name plus ingredients and steps all on one page. We have coded the `IngredientsAndSteps` environment to provide some additional options.

In it default mode, the `IngredientsAndSteps` environment simply: (1) creates a two-column environment, and (2) sets the `\FontIngredients` and `\FontSteps` font selection commands to use their default font size of 10.95 points.

Using an optional parameter with a key/value interface, the the `IngredientsAndSteps` environment can be used to change (temporarily) the default text width and default font size. By increasing the text width and/or decreasing the font size for the ingredients and steps, it might be possible to fit the recipe on one page. The key/value options are as follows:

- [**AdjIFont=**] Default value = 1. This is a floating point number that indicates the size you want the `\FontIngredients` font, computed as a ratio of the default size. For example, if AdjIFont=0.92, then the

font would be set to 92 percent of its default size. IMPORTANT: This is the "controlling" ratio for all font parameters. If any of the other font parameters are set to 0 or not entered, then that other parameter will have the same value as set here.

- [**AdjIBaseline=**] Default value = 0. This is a floating point number that indicates the size you want the `\FontIngredients` baseline, computed as a ratio of the default baseline size. For example, if AdjIBaseline=0.92, then the ingredient font's baseline would be set to 92 percent of its default size. If 0 or not entered, then AdjIBaseline = AdjIFont.
- [**AdjSFont=**] Default value = 0. This is a floating point number that indicates the size you want the `\FontSteps` font, computed as a ratio of the default size. For example, if AdjIFont=0.92, then the font would be set to 92 percent of its default size. If 0 or not entered, then AdjSFont = AdjIFont.
- [**AdjSBaseline=**] Default value = 0. This is a floating point number that indicates the size you want the `\FontSteps` baseline, computed as a ratio of the default baseline size. For example, if AdjIBaseline=0.92, then the steps font's baseline would be set to 92 percent of its default size. If 0 or not entered, then AdjSBaseline = AdjIFont.
- [**AddWidth=**] Default value = 0. This is amount (taken as points) that you want to increase both the left and right margins. So, if AddWidth=5, then the text width will be increased by 5 points on both the left and right margins.

Following is the code defining the `IngredientsAndSteps` environment:

```
\usepackage{multicol}           % allows multiple column environments
\usepackage[strict]{changepage} % for \adjustwidth
\usepackage{xfp}                % for fpeval floating point macro

\pgfkeys{ /IngredientsAndSteps/.is family, /IngredientsAndSteps,
    default/.style = {AdjIFont = 1, AdjIBaseline = 0,
                      AdjSFont = 0, AdjSBaseline = 0,
                      AddWidth = 0, RaggedCols   = 0},
  AdjIFont/.estore in     = \AdjustIFont,
  AdjIBaseline/.estore in = \AdjustIBaseline,
  AdjSFont/.estore in     = \AdjustSFont,
  AdjSBaseline/.estore in = \AdjustSBaseline,
  AddWidth/.estore in     = \AddPageWidth,
  RaggedCols/.estore in   = \RaggedColumns,
}

\newdimen\IFont  \newdimen\IBase  \newdimen\SFont  \newdimen\SBase  \newdimen\AddWidth

\NewDocumentEnvironment{IngredientsAndSteps}{o}
{   % Do the before-environment setup:
\IfNoValueTF {#1}%
    % Handle the simple case = no optional key/value argument:
  {\AddWidth = 0 pt}
    % Handle the special case = there IS an optional key/value argument:
  {
  \pgfkeys{/IngredientsAndSteps, default, #1}%

  \IFont    = \AdjustIFont pt
  \IBase    = \AdjustIBaseline pt
  \SFont    = \AdjustSFont pt
  \SBase    = \AdjustSBaseline pt
  \AddWidth = \AddPageWidth pt

  \ifdim\IBase = 0 pt \IBase = \IFont \fi
  \ifdim\SFont = 0 pt \SFont = \IFont \fi
  \ifdim\SBase = 0 pt \SBase = \IFont \fi

  \RenewDocumentCommand\FontIngredients{}{\FontIngDefault\fontsize{\fpeval{\StdIFontSize * \IFont}}
    {\fpeval{\StdIBaseline * \IBase}}\selectfont}
```

```
  \RenewDocumentCommand\FontSteps{}{\FontStepsDefault\fontsize{\fpeval{\StdSFontSize * \SFont}}
    {\fpeval{\StdSBaseline * \SBase}}\selectfont}
  }
    % Now, finish setup:
\adjustwidth{-\AddWidth}{-\AddWidth}
\begin{multicols}{2}%
}   % End of the before-environment setup
    % Now do the after-environment cleanup:
{
\end{multicols}
\endadjustwidth
    % We need to restore the fonts to default size IF there was an optional key/value argument:
\IfValueT {#1}
  {
  \RenewDocumentCommand \FontIngredients {}{\FontIngDefault}
  \RenewDocumentCommand \FontSteps       {}{\FontStepsDefault}
  }
} % End of the "after environment" cleanup
```

9 Adding Images in the Chapter Intro

You may wish to strategically place images in the middle of the *chapter intro* text. For example, on the first page of a chapter, you may begin with the *chapter intro* text, then include image(s) at the bottom of that first page, and then complete the *chapter intro* text on the next page.

9.1 The "Real" `\ChapterIntro` Command

The code for the `\ChapterIntro` command is shown below. It allows for splitting the *chapter intro* into two parts and is essentially identical to the code for the `\RecipeStory` command. For the use of the optional first argument, see the `\RecipeStory` command (page 29). To print the second part of the *chapter intro* text, you issue the `\FinishChapterIntro` command.

```
\newbox\IntroBox
\newbox\IntroBoxA
\NewDocumentCommand \ChapterIntro {o +m}  % arg1 = optional = lines for first part
{                                         % arg2 = chapter intro text
\setbox\IntroBox\vbox
  {
  \FontChapterIntro {#2\par}% the \par needed by \lettrine
  }
\IfNoValueTF {#1}    % NoValue=TRUE or NoValue=FALSE?
  {\unvbox\IntroBox} % NoValue=TRUE, so flush the full IntroBox
  {                  % handle the NoValue=FALSE case
  \ifnum #1 > 0      % > 0 means we split the story into two part; print the first part here
    {                % = 0 means ALL of story is deferred
    \setbox\IntroBoxA=\vsplit\IntroBox to #1\dimexpr \CIntroFontBaseline %
    \unvbox\IntroBoxA  %
    }
  \fi
  }
}

\NewDocumentCommand \FinishChapterIntro{}%
  {\ifvoid\IntroBox \else \smallskip\unvbox\IntroBox\par\medskip\fi} %
```

Part VII
The PDF File

We have two separate goals for the PDF file created by LaTeX. We need a PDF file in the proper form to submit to a print-on-demand printer. We also need a PDF file that is a digital version of the cookbook, but also includes bookmarks and links. Unfortunately, the two styles of PDF files are not identical and therefore require different compile-time options.

The best way to show the differences between the digital and print-on-demand compiles is to first describe our setup for the digital cookbook. We will then describe the changes needed to generate the print-on-demand PDF file.

1 The Digital Cookbook

In the digital version of the cookbook, we would like to include bookmarks and other links to content. Specifically, we would like:

- Top-level bookmarks equal to the entries in the Table of Contents.
- Second-level bookmarks for each recipe (associated with its top-level cookbook chapter).
- For the recipe Index, links back to each recipe by clicking on the recipe's indicated page number.
- Within the cookbook main text, where there is a page number cross-reference to another recipe, a link back to that recipe by clicking on the cross-reference page number.

We implement these bookmarks and links by using the `hyperref` and `bookmark` packages. First, we load the packages (and set the color of the links):

```
\usepackage[bookmarks=true,colorlinks=true, allcolors=clrHyperRef]{hyperref}
\usepackage{bookmark} % Q 247158 shows Heiko (the author) loads both hyperref and bookmark
```

To create a bookmark/link for each recipe, we define the `\RecipeBookmark` command, which will be called by the `\RecipeNameAndYield` command. (The [1] in `\pdfbookmark` makes the recipe a second-level bookmark):

```
\newcounter{ctrRecipe} % initially set to zero by \newcounter macro
\NewDocumentCommand \RecipeBookmark{m}{%
  \ifx\%1\empty\relax\else\stepcounter{ctrRecipe}\pdfbookmark[1]{#1}{Recipe\arabic{ctrRecipe}}\fi
}
```

To create a linkable `\label` to a recipe, we define the `\RecipeLabel` command (note: `\phantomsection` is from `hyperref` and marks the linkable location of the recipe). Our `\RecipePageNo` command then uses `\pageref` to display the page number and (via `hyperref`) provide a link to the recipe `\label`:

```
\NewDocumentCommand \RecipeLabel{m}{\phantomsection\label{recipe:#1}}
\NewDocumentCommand \RecipePageNo{m}{\pageref{recipe:#1}}
```

Finally, we define the `\CookbookIndex` command (called from `cb-frontmatter.tex`) to create the recipe Index, make the Index a top-level part of the bookmarks, and include the Index in the Table of Contents:

```
\NewDocumentCommand \CookbookIndex{}
{
\cleardoublepage % flush all material and clear until you start new odd numbered (recto) page
\phantomsection\addcontentsline{toc}{chapter}{\indexname} % see also Q 59619
\printindex
}
```

2 The Print-On-Demand Cookbook

There are generally stricter requirements for PDF files submitted for print-on-demand printing. We discuss the various issues below.

2.1 Embedded Fonts

Your print-on-demand company will require that all of your fonts are embedded in the PDF file. In most cases, it will be sufficient to include an "embedded subset" (i.e., includes only the glyphs that were actually used). However, we note Q 24005, where the questioner stated that the print-on-demand company lulu.com requires full font embedding. If that happens to you, a solution is provided at that question. But, as stated there, many font licenses disallow full font embedding. Check your font license if that situation arises.

We note that our compiled PDF file includes an embedded subset of our fonts without any special or additional steps. You should still double-check your PDF file to make sure your fonts are embedded.

As discussed at Section 4.5 (page 21), to avoid any unnecessary font issues, it is best to avoid loading any font you do not really need. For that reason, we relied on our already loaded fonts (rather than relying on LaTeX macros) for the degree, copyright, bullet and center (or mid) dot glyphs.

2.2 Bookmarks, Annotations, and Comments

Your PDF file should not include any bookmarks, links, annotations or comments. Because our digital cookbook includes bookmarks and links, we will need different compile-time options. We handle that by making a few small changes to the preamble. Near the top of the preamble, we use `\newif` to create a new conditional: `\ifHyperRef`. We set the value to `\HyperReftrue` if we are making the digital cookbook, and to `\HyperReffalse` if we are making the printed cookbook. For example, for the printed cookbook:

```
\newif\ifHyperRef        \HyperReffalse
```

We also modify the first two code snippets shown in the above discussion of the Digital Cookbook. Below, the `hyperref` and `bookmark` packages are only loaded if we have `\HyperReftrue`. If `\HyperReffalse`, we only need to set the `\phantomsection` and `\RecipeBookmark` commands to do nothing:

```
\ifHyperRef
  \usepackage[bookmarks=true,colorlinks=true, allcolors=clrHyperRef]{hyperref}
  \usepackage{bookmark}

  \newcounter{ctrRecipe}
  \NewDocumentCommand \RecipeBookmark{m}{%
   \ifx\%1\empty\relax\else\stepcounter{ctrRecipe}\pdfbookmark[1]{#1}{Recipe\arabic{ctrRecipe}}\fi
}
\else
  \providecommand\phantomsection{}% Q 44088
  \NewDocumentCommand \RecipeBookmark{m}{}%
\fi
```

Side Note

You may run into anomalies in your compile if you switch back and forth between `\HyperReftrue` and `\HyperReffalse`. Because the `*.ind` index file was created with the "other" option, in the first switched compile, the contents of that file may contain unknown commands causing compile errors. Just run the compile again and all should be OK.

2.3 Trim Size, Crop Marks and Other Printer's Marks

Almost all print-on-demand companies require: (1) that the trim size indicated in the PDF file matches the trim size requested for the printed cookbooks, and (2) that there are no crop marks or other printer's marks in the submitted PDF file.

The `\ifCookbookDraft` logic discussed on page 10 handles the trim size, plus the crop marks and other printer's marks made by the `geometry` package. We also use `\ifCookbookDraft` logic to only load the `layout` and `lipsum` packages when we are in draft mode:

```
\ifCookbookDraft
  \usepackage{lipsum, layout}
\fi
```

Note that some print-on-demand printers (e.g., Kindle Direct Publishing) will not accept a PDF file that includes filler from the `lipsum` package (or similar).

2.4 PDF/X

Although very few print-on-demand companies *require* PDF/X compliant documents, it is almost always their preferred format. We have not taken the extra steps needed for full PDF/X compliance. However, you may wish to familiarize yourself with this (very large) topic. A good starting point is the documentation for the `pdfx` package.

If you use `pdfx` or other methods of entering metadata (such as title, author, etc.) into you PDF file, be sure the entered information ***exactly*** matches the information you give to your print-on-demand company in their submittal forms.

2.5 Other Print-on-Demand Issues

Of course, document security of any type must not be used. For the best results, all images should be sized at 100%, flattened to one layer and inserted into your file at a minimum resolution of 300 dots per inch. You should expect that all transparent objects will be flattened. See also Section 5.1 on page 12 for consideration relating to color.

Part VIII
Examples

In this part, we present and discuss some example code (mostly for entering recipes). You will see "in action" the use of various commands and environments, and you will see how we use some of the helper macros. We assume you have downloaded and installed the `makecookbook` package, and that you are ready to compile the sample cookbook that uses `mycookbook` as its root directory.

Recall that the `mycookbook/tex` directory holds two example chapters of recipes. To make those recipes easier to reference here, we name the first recipe **"A – (With Recipe Name Here)"**, the second recipe **"B – (With Recipe Name Here)"**, and so on. Here, we will refer to the included recipes by their first alphabetic letter only (e.g., "see recipe **D**").

1 `\RecipeNameAndYield` Command

1.1 Recipe A

Recipe **A** presents the simplest case of the `\RecipeNameAndYield` command. We enter only the recipe name (no yield, no special Index entries, no cross-reference label). Our actual code:

```
\RecipeNameAndYield{Name=A -- Lemon Roasted Potatoes}
```

1.2 Recipe B

In recipe **B** the `\RecipeNameAndYield` command includes an entry for `Yield=`. We also wanted the Index to hold only the actual recipe name (without the leading **B**), so we included `NoIdxName=1` and `IndexA=Pumpkin Pancakes`.

On second thought, we decided a leading **B** was needed so you could use the index to find the recipe. We therefore added `IndexB=B!Pumpkin Pancakes`, which also allowed us to demonstrate the sub-entry indicator character ![10] and create a sub-entry for this recipe. (Yes, it looks a little funny. In the real world, a sub-entry only looks right if there are multiple sub-entries under the entry).

Check the Index entries for recipe **A** and recipe **B** to see the differences. Please note that you *do not* need to include `NoIdxName=1` when you use `IndexA=` (You can have *both* the original name and up to three alternate names).

```
\RecipeNameAndYield{Name=B -- Pumpkin Pancakes, Yield=Yield: 6 Pancakes, NoIdxName=1,
  IndexA=Pumpkin Pancakes, IndexB=B!Pumpkin Pancakes}
```

1.3 Recipe C

Recipe **C** demonstrates use of `XRefLabel=` to create a cross-reference label for this recipe. (See recipe **D** for the reference back to this recipe). Our actual code:

```
\RecipeNameAndYield{Name=C -- Pesto, XRefLabel=Pesto}%
```

1.4 Recipe D

Recipe **D** demonstrates two things regarding the `\RecipeNameAndYield` command: (1) the use of `\nl` to manage multi-line text for the recipe or yield, and (2) the use of an extra set of curly braces when the key/value text includes commas. Our actual code:

```
\RecipeNameAndYield{Name=D -- Pasta Genovese,
  Yield={(Pasta with Pesto, Potatoes and Green Beans)\nl Makes 4 to 6 servings}}
```

[10] see the `makeindex` package at `ctan.org/pkg/makeindex` for documentation of the `\index` command.

2 \RecipeStory Command

2.1 Recipe A

Recipe **A** presents the typical case, using the `\RecipeStory` command *without* the optional argument. We also demonstrate use of the `\lettrine` command. Our actual code:

```
\RecipeStory{\lettrine{T}{his is a recipe} story. \lipsum[66]}
```

2.2 Recipe E

Recipe **E** presents the special case, using the `\RecipeStory` command *with* the optional argument. Note that our recipe is the same as recipe **A**, except we: (1) increased the length of the recipe story, and (2) included the optional argument to the `\RecipeStory` command ([8]) that prints only the first 8 lines of the recipe story before the recipe, with the balance printed at the location of the `\FinishRecipeStory` command. Our actual code:

```
\RecipeStory[8]{\lettrine{T}{his is a recipe} story. \lipsum[1-3]}
```

3 IngredientsAndSteps Environment

3.1 Recipe A

Recipe **A** presents the typical case, using the `IngredientsAndSteps` environment *without* the optional (key/value style) argument. In that case, they only thing the `IngredientsAndSteps` environment does is set up a two-column environment (using the `multicol` package) for entering the ingredients and steps. Our actual code (with pseudo-code for the `\ListIngredientsAndSteps` command):

```
\begin{IngredientsAndSteps}
  \ListIngredientsAndSteps{...ingredients...}{...steps...}
\end{IngredientsAndSteps}
```

3.2 Recipe F

Recipe **F** presents the special case, using the `IngredientsAndSteps` environment *with* the optional (key/value style) argument. Here, we want to shrink the ingredients and steps fonts, and and their baselines, to 92% of their default size. We also want to increase both the left and right margins by 5 points. Our actual code (with pseudo-code for the `\ListIngredientsAndSteps` command):

```
\begin{IngredientsAndSteps}[AdjIFont=0.92, AddWidth=5]
  \ListIngredientsAndSteps[Tomato Meat Sauce]{...ingredients...}{...steps...}
  \ListIngredientsAndSteps[Béchamel]{...ingredients...}{...steps...}
  \ListIngredientsAndSteps[Assembly]{...ingredients...}{...steps...}
\end{IngredientsAndSteps}
```

Note that these values allow the recipe to fit on one page (try that recipe without any optional key/value entries). You might also try playing with the optional key/value entries by entering your own values. For example, set both fonts and their baselines to different values.

4 \ListIngredientsAndSteps Command

4.1 Recipe A

Recipe **A** presents the typical case, using the `\ListIngredientsAndSteps` command *without* its optional argument. The first required argument is the list of ingredients, and the second is the list of steps. Our actual code:

```
\ListIngredientsAndSteps
  {                                                          % begin ingredients
  2 \Pd[s] baby Dutch gold potatoes, washed and cut in half

  \fr1/2 cup water

  \fr1/4 cup extra-virgin olive oil

  ...and the rest of the ingredients
  }                                                          % end ingredients
  {                                                          % begin steps
  \PreheatC{375} \ChefNote

  In a \Inch{\AxB{9}{13}} baking dish, combine all ingredients except parsley.

  Roast for 30 minutes.

  ...and the rest of the steps
  }                                                          % end steps
```

4.2 Recipe F

Recipe **F** presents the special case, using the `\ListIngredientsAndSteps` command *with* its optional argument. This normally is needed when you have more than one listing of ingredients and steps (e.g., one for making the cake and one for making the frosting). Our actual code (mostly pseudo-code, and for clarity we also show the `IngredientsAndSteps` environment, but not its optional arguments):

```
\begin{IngredientsAndSteps}
  \ListIngredientsAndSteps[Tomato Meat Sauce]
  {
  ...ingredients...
  }
  {
  ...steps...
  }
  \ListIngredientsAndSteps[Béchamel]
  {
  ...ingredients...
  }
  {
  ...steps...
  }
  \ListIngredientsAndSteps[Assembly]
  {
  ...ingredients...
  }
  {
  ...steps...
  }
\end{IngredientsAndSteps}
```

4.3 Recipe I

Recipe **I** presents another special case. You might wish to divide the ingredients into separate groupings, each with a heading, but only have one grouping of recipe steps. We use the `\ListIngredientsAndSteps` command *without* its optional argument. But we use the `\IngredientsHeading` helper command (page 24) to enter those heading within the listing of ingredients. Note the use of optional `Arg #1` (for the second heading) – that is the number of points of vertical skip before the heading. (Not really needed here, but it would be useful if the heading was not at the top of a column).

4.4 Recipes B, C and G

Our `IngredientsAndSteps` environment uses the `multicols` environment (from the `multicol` package). It creates a two-column environment for entering ingredients and steps. One of the features of the `multicols` environment is that it balances the lengths of the first and second column. In some instances, the result may not look right to you. For example, the ingredients may take up *almost* all of the first column, but not quite. To balance the two columns, the `multicols` environment may put one line from the recipe steps in the first column.

To work around the above problem, you can use the `\InsertHiddenLines` command. That command takes one mandatory parameter, which is the number of hidden lines to insert. Now look at the printed output for recipes **B**, **C** and **G**. You can probably see that we inserted, respectively, 1, 2 and 3 hidden lines at the bottom of the first column of those recipes.

Recipe **B** also uses the `\IngredientsSeparator` command. That command provides a visual separation between ingredients. In this case, it separates the dry ingredients from the wet ingredients.

5 `\ChapterIntro` Command

5.1 Chapter One

Chapter One presents the typical case, using the `\ChapterIntro` command *without* the optional argument. We also demonstrate use of the `\lettrine` command. Our actual code:

```
\ChapterIntro{
  \lettrine{T}{his is the} chapter intro. \lipsum[1-3]
}
```

5.2 Chapter Two

Chapter Two presents the special case, using the `\ChapterIntro` command *with* the optional argument. We wanted two side-by-side images at the bottom of the first page of the chapter. After a trial run, we calculated that the first 16 lines of the *chapter into* text would fit (with the images) on page one. We print the balance of the *chapter into* text with the `\FinishChapterIntro` command. Our actual code:

```
\ChapterIntro[16]{\lipsum[1-3]}
\SideBySide[LeftCaption={left caption}, RightCaption={right caption}]{image-a}{image-b}
\FinishChapterIntro{}
```