

# **Encoding Guidelines for the Historic American Cookbook Project**

The following set of guidelines is meant to be used as an aid in training new coders for the Historic American Cookbook Project. Coders currently working on the project will also find it a useful reference material in keeping their coding consistent with what has been done in the project so far. A simpler version of this document can be found at <http://digital.lib.msu.edu/cookbooks/project.cfm?about=encoding>, a page created by Ruth Ann Jones at the beginning of the project to describe the coding process as it was originally conceived. This version of the encoding guidelines includes more detailed, step-by-step instructions for the coding process, as well as a number of updates and changes that have been added to the encoding guidelines from the time the project began through the end of June, 2003. The website may be used as a quick reference, however, and most of the information appearing in tables in this document has been copied verbatim from that source.

## **BEFORE YOU BEGIN**

The following is a list of terms that are probably unfamiliar to you, but which will be used to describe the coding process throughout this packet. It may be useful for you to learn them if you are interested in the theory behind what we are doing. I got most of this information from the glossary in the XMetaL 2.0 User Guide, then added some notes specific to this project.

**DTD** = A set of declarations, written in a formal notation defined in the XML standards, that define the structure of a set of documents. Among other things, a DTD declares all of the element names that can appear in a document, the hierarchy in which they can be arranged, the type of content they can have, and which attributes they can have. The DTD used in the Feeding America project is named `cookbook.dtd` and can be found in `K:/cooking/cookbook.dtd`, and on each coding workstation in `C:/Program Files/SoftQuad/XMetaL 2/Rules/cookbook.dtd`.

**DOCTYPE declaration** = Document type declaration; a declaration at the top of an XML document that specifies which DTD applies to the document, and may contain some extra markup declarations. XMetaL will automatically generate a DOCTYPE declaration when you select a rules file to accompany a new XML document.

**Element** = a structural building block of an XML document. Blocks of text are contained in elements according to their function in the document: for example, headings, lists, paragraphs, and links are all surrounded by specific elements. Essentially, each basic tag in the cookbook tagset (`<p>`, `<emph>`, `<list>`, `<recipe>`, etc.) is an element. You can view a complete list of the elements used in this project by opening the Element List window in XMetaL (choose **Element List** from the **View** menu) while coding a cookbook.

**Empty element** = An element that has only a single tag (which may have attribute values) and cannot have any content. In XML, an empty tag looks like `<TAGNAME/>`. There are

only three empty elements in the cookbook tagset: <pb> (page break), <lb> (line break), and <gap> (used to mark missing text).

**Required element** = An element's sub-element that the DTD has declared must be present in order for the document to be valid. The <cookbook> element, for example, **must** contain <meta>, <front>, and <body>, or the document will not validate.

**Attribute** = A value that is associated with an element but is not part of the **content** of the element. (The content of an element is the text that appears between the opening and closing tags of that element.) Many formatting properties are represented by attributes: for example, text size, italicized text, and text alignment. Most of you should be familiar at least minimally with the **rend=** attribute, which is a required attribute for the <emph> tag in the cookbook tagset (and is optional for most other elements as well). You can view and edit any attribute through the Attribute Inspector in XMetaL.

**Required attribute** = An attribute that the DTD has declared must be present in order for the document to be valid. The <emph> element in the cookbook tagset **must** contain a **rend=** attribute, or the document will not validate. Required attributes for any given element will be displayed in bold in the XMetaL Attribute Inspector.

**Attribute inspector** = An XMetaL window that enables you to view and edit the attributes of any element. To display the Attribute Inspector, press **F6** or choose **Attribute Inspector** from the **View** menu.

## GETTING STARTED

All books that have reached the coding phase in our project should be posted on the local network in K:/cookery/xml in progress/. The files should be in plain text format, and should already have been file compared to eliminate errors. You will use XMetaL to code the books, but opening a plain text file in XMetaL will generate errors, so I recommend opening the file in NoteTab first and then copying and pasting the file into XMetaL.

Here are the steps you should follow when starting a new cookbook:

- 1) **Record on the cookbook progress sheet that you have begun coding the cookbook.** There is a spreadsheet in K:/cookery/spreadsheets/ called cookbookprogresssheet.xls, which Stephanie created to track the progress of all the books in the project. When you begin coding a cookbook, please find your book in the list of cookbooks and add your name in the "Coded By" column, followed by "in progress" (to let Stephanie know that the book is not complete, but still being coded). If you have picked up a coding project where someone else left off, just add your name next to theirs in the "Coded By" box.
- 2) **Open the text file for the book in NoteTab Light.** All text files should be in K:/cookery/xml in progress/. If you have trouble locating the text file for any book, ask Stephanie for assistance.

- 3) **Open the metadata template in NoteTab Light.** The metadata is a section of text that is used to catalog the book online. Because the information entered into the metadata is basically the same for all books, Ruth Ann has created a template for this portion of the coding that can simply be pasted in at the beginning of the text. The template has been saved as a text file and is located here: K:/cookery/xml in progress/metadata.txt.
- 4) **Copy and paste the metadata template into the text file for the book.** Copy the whole of the metadata from the metadata.txt file (you can "select all" by pressing Ctrl+a, and copy with Ctrl+c), then paste the text you have copied into the **very beginning** of the book text file. Make sure you paste it in **before** the first page break—otherwise this will cause problems later!
- 5) **Copy the file which now contains both the metadata and the book.** I would recommend using Ctrl+a again.
- 6) **Open XMetaL.** There should be a shortcut on your desktop; if not, use the Start menu to open it. The file path (through Start) is Programs→SoftQuad Applications→XMetaL 2.0.
- 7) **Create a new file in XMetaL.** Use the "New..." command from the "File" menu at the top of the screen. XMetaL will then prompt you to choose which type of new file you would like to create; choose "Blank XML Document" from the two options listed. (If "Blank XML Document" does not appear on the screen, make sure you are in the "General" tab of the "New" document window.)

XMetaL will then prompt you to "Choose a DTD or Rules File." There should be a file called cookbook.dtd (the file extension may not display on your computer—in that case, you want the file named cookbook, **not** cookbook.rlx) in the window that pops up. Double click on this file.

XMetaL will now display a blank white screen, at the top of which is a line of text which looks something like this:

```
<?xml version="1.0"?>
<!DOCTYPE cookbook SYSTEM "C:\Program Files\SoftQuad\XMetaL
2\Rules\cookbook.dtd">
```

This is the DOCTYPE declaration for the document. Leave this text in the file, but move the cursor past it—you may want to hit enter a couple of times to leave yourself space between it and the cursor. Then

- 8) **Paste the metadata/book file into XMetaL.** There should be three things in the file you have created: the DOCTYPE declaration at the top, the metadata, and the text of the book, *in that order*.
- 9) **Wrap the <cookbook> tag around the entire book.** By entire book I mean everything but the DOCTYPE declaration—so the cookbook tag should wrap around everything from the opening <meta> tag in the metadata to the very end of the book (usually a <pb type="back cover">). The Element List in XMetaL works much like the clipboards in NoteTab Light—if you double click on a tag, the tag will be inserted wherever the cursor appears, or (if text is highlighted) around a chunk of highlighted text. The Element List should appear in the lower right-hand corner of your screen in XMetaL; if it does not appear, click on View, then Element List. It would be a good idea to turn the Attribute

Inspector on as well, which is done in the same manner. I would recommend either typing in the <cookbook> tag yourself (<cookbook> at the beginning, </cookbook> at the end of the file), or pasting in both the opening and closing tags by double-clicking "cookbook" in the Element List and then moving the end tag to the end of the document. Highlighting the entire book from metadata to back cover would simply take too long. (The "select all" command does not work in XMetaL.)

- 10) **Save the file as "[bookcode].xml", "[bookcode]"** meaning the four-letter code assigned to that book. You should save the file in the directory K:/cookery/xml in progress/.

## CODING THE WRAPPER ELEMENT <cookbook>

Before you move on to coding the metadata and the text of the book, you need to define some attributes for the <cookbook> tag. The following two tables provide information about these attributes and how they are used.

ATTRIBUTES ESTABLISHED FOR THE WRAPPER ELEMENT <cookbook>	
type=	Required. This contains general categories which can characterize an entire cookbook, a chapter or section, or (infrequently) individual recipes or formulas. Allowed values are general, charity, famous, frugal, restaurant, invalid, histperiod, and encyclopedia. See Table 3 for definitions.
chefschoo=	Optional, but should be used if a cookbook is identified as type=famous. Fill in the name of the chef or cooking school.
histperiod=	Optional, but should be used if a cookbook is identified as type=histperiod. Fill in the name of the historical period (such as "Temperance movement" as given in the cookbook.
class1=	Required. These are categories for foods and other types of activities described in the cookbooks. Allowed values are fruitvegbeans, meatfishgame, eggscheesedairy, breadsweets, soups, accompaniments, beverages, <i>generalfood</i> , menus, medhealth, household, farmgarden, childrear, etiquette, restaurant, servants, marketing, <i>generalnonfood</i> , <i>foodandnonfood</i> . The values shown in italics will probably be the ones used most often at this level, when you are describing an entire cookbook. However, some books will have focus on certain types of foods and one of the other values will be appropriate. The same values are used for individual recipes. See the Coding Recipes section for definitions.

class2=	Optional. Same allowed values as class1; use this if necessary to represent a secondary focus of a particular cookbook.
region=	Required. If a recipe is identified with a specific place or region in the U.S. <b>or</b> with a particular ethnic group, use this attribute. Allowed values are northeast, south, midwest, west, ethnic, and general. Use the U.S. Census map to decide which region a place is in.
subregion=	Optional but should be used if the region= attribute is used. Fill in the more specific region as identified by the cookbook.
ethnicgroup=	Optional, but should be used if a cookbook or portion of a cookbook is identified as <element region="ethnic">. Fill in the name of the group as identified in the cookbook.
occasion=	Optional. If a cookbook is identified with a special occasion, use this attribute. Allowed values are Thanksgiving, Christmas, wedding, birthday, patriotic, spring, summer, fall, winter, other.
bookID=	Required. The bookID consists of the year of publication followed by the four-letter ID code of the book.

<b>VALUES FOR THE TYPE= ATTRIBUTE OF THE WRAPPER ELEMENT &lt;cookbook&gt;</b>	
"general"	General works that do not fall into one of the special categories listed below.
"famous"	Cookbooks by a famous chef (Julia Child, Fannie Farmer) or produced by a well-known cooking school. The list of famous chefs for this project is: Child, Hale, Randolph, Leslie, Beecher, Harland, Corson, Farmer, Lincoln, Parloa, and Rorer. If you code a book by one of these authors, choose type="famous" and put the chef's and/or school's name in chefschool=
"charity"	Cookbooks produced by church or community groups for fundraising.
"frugal"	Works on cooking economically or using inexpensive ingredients.
"restaurant"	Works featuring large-scale recipes for restaurants.
"invalid"	Works on cooking for invalids or treating various conditions through diet (e.g. diabetes cookbooks).
"histperiod"	Works based on the cooking of a specific historical period. Put the name of the period in the <i>attribute</i> histperiod=. For example, a Civil War cookbook: <cookbook type="histperiod" histperiod="Civil War">

"encyclopedia"	Works organized as a dictionary or encyclopedia: that is, articles arranged alphabetically by topic.
----------------	------------------------------------------------------------------------------------------------------

## THE MAJOR STRUCTURAL ELEMENTS OF A COOKBOOK

A <cookbook> consists of four sections:

<meta>	metadata, expressed as Dublin Core elements: required
<front>	frontmatter: required
<body>	main body of book: required
<back>	backmatter: not required, but normally it will be used at least minimally to hold the <pb> reference for the back cover image

The metadata, as I have already mentioned, is a portion of the file that we use in cataloging our digital texts online. No, you don't have to worry about what Dublin Core elements are (but if you're interested, you can ask Ruth Ann). The only thing you have to do with the metadata is paste in the template and follow Ruth Ann's instructions for filling the content of the elements within <meta>.

The frontmatter consists of any pages that fall before the main body of a book begins; this includes front covers, title pages, copyright statements, tables of contents, introductions, and the like.

The body, or main body of the book, is (I hope) fairly self-explanatory. This is where all the real content of a book lies—recipes, general chapters, etc.

The backmatter is similar to the frontmatter, only it falls *after* the main body of the book. Indexes, appendices, advertisements, and the like commonly appear within the backmatter (though it is not usually as extensive as the frontmatter).

If you ever need clarification about where a book's frontmatter, body, or backmatter begins or ends, please ask Stephanie or Ruth Ann.

## CODING THE METADATA

After you code the cookbook tag, move on to the <meta> template portion of the code. The tags within <meta> each begin with a dc, such as <dcTitle> and <dcContributor>. The data that goes into these fields will be used for cataloging the book, so **accuracy is absolutely essential** when coding this portion of the book. You will be entering most of this information manually, and it won't be

checked for spelling errors after you finish. Typos make our department look sloppy, so please be careful!

That said, coding the metadata is actually quite easy. If you look at the metadata in XMetaL, a large portion of the text in the template will appear in purple. This is because that text appears in **comment tags** (`<!-- -->`). These tags are useful because any text within comment tags will be ignored by XMetaL during rules checking and document validation; its presence will not affect the code at all. It will also be ignored by web browsers. Comment tags, then, are a useful way for a coder to leave him or herself notes and reminders within the text without interfering with the text or coding at all. Ruth Ann has used them in the metadata template to provide instructions for completing each of the elements within `<meta>`. To code the metadata, all you need to do is follow her instructions. After entering the proper information, delete the comment tags (no purple text should remain when you are finished). If you have any questions concerning the metadata, see Ruth Ann.

## STRUCTURAL ELEMENTS OF THE FRONTMATTER AND BACKMATTER

Divide the `<front>` and `<back>` of each book into `<div>` sections based on their content. Each `<div>` must have a type attribute indicated. Allowable values are: advertisement, appendix, backcover, contents, copyrightstmt, dedication, frontcover, glossary, halftitlepage, illustration, introduction, index, preface, titlepage, and other. Be sure to include page breaks within the `<div>` tag where appropriate. Keep in mind, also, that you may have more than one `<div>` on a single page.

This list is meant to be fairly comprehensive, so don't use "other" unless none of the other values fit at all. For example, an editor's note is pretty similar to a preface, so tag it as "type=preface." Use "type=contents" for lists of illustrations, figures, and other special items as well as the usual table of contents. Tables of weights and measures that appear as a reference in the front or back of the book may be coded as "type=appendix," whether or not they are labeled as one. (This does not mean that you should never use "type=other," however. I have used it quite regularly to mark blank pages and inscriptions, for example. Just think of it as a last resort.)

If necessary, a `<div>` can be divided into `<subdiv>` elements. However, it shouldn't be necessary very often. An exception might be a lengthy introduction that is divided into two parts, each with its own heading. Normally, the paragraph tag `<p>` will be all you need within a `<div>`.

## GENERAL FORMATTING ELEMENTS

Within a <div> or <subdiv> in the front or backmatter, and also within a <chapter>, <section>, or <subsection> within the main body of the text (which will be explained later), the following elements may be used to markup the text. Some of these, such as <p>, <pb>, and <emph>, you should already be familiar with. Others, such as <lb>, <ref>, and <table>, you may not recognize. I will provide more detailed instructions on how to code the more complex of these elements (particularly <table> and <list>), but most are fairly self-explanatory.

<p>	Paragraph: to subdivide <recipe> or <formula>, as needed.
<pb>	Page break. Follow the same rules as for other typing projects, i.e. <pb n="pagenumber" id="book079.jpg">
<hd>	Heading. Use for the titles of chapters, sections, etc.
<lb>	Line break. Can be used any time there are special line breaks, as on the title page. Keep in mind that hitting "enter" to break a line in XMetaL will not tell the web browser to display a line break at that point; the <lb> tag must be there for the line to render correctly.
<emph>	Use only for linguistic emphasis (see notes on formatting for explanation).
<alt>	Originally intended to give the 20 <sup>th</sup> century equivalents of archaic terms, the use of the <alt> tag has evolved over the course of the project. It is now used to correct typos that appeared in the original text, and also to insert text into recipes to facilitate easier searching of the collection. A detailed section on how to use the <alt> tag appears later in this document.
<term>	May be used within <p> to indicate a term being discussed in an encyclopedic cookbook. It is preferable, however, to use <term> within <list> to markup glossary entries that do not contain multiple paragraphs.
<definition>	May be used within <p> to mark the first paragraph of an entry in an encyclopedic cookbook. Like <term>, <definition> should be used within <list> when possible.
<ref>	Use to create an internal link within the document via target and id pairs, which will be explained later. The <ref> tag is used most often to indicate a footnote within the text, or to create links to pages from an index or table of contents. More information will be provided about both footnotes and indexes later in this document.
<list>	Use to indicate a list of items—common examples of lists in the cookbooks project would be menus and lists of ingredients.



<table>	Use to indicate a table within the text—i.e., text arranged in multiple rows and columns. Do <i>not</i> use the <table> tag to represent columns that have been created simply to preserve space on a page—e.g. indexes, menu items, or lists of ingredients arranged in multiple columns. These should be coded with the <list> tag.
<illustration>	Use to indicate an illustration within the text. More information on coding illustrations will be provided later in this document.
<gap>	Use to indicate text that is missing or completely illegible. More notes on how to use the <gap> tag appear later in this document.
<unclear>	Use to indicate text that is clearly visible but difficult to accurately transcribe, such as a handwritten inscription. See the section on "WHEN TO USE <gap> AND <unclear>" for further explanation.
<attribution>	Use to indicate a section of text that is attributed to someone other than the editor or author of the book being tagged. The <attribution> tag should go around the attributed party's name, whether it be a person, an organization, or the title of another book. This tag should be used almost exclusively within <recipe> (see Low-Level <recipe> Elements chart in the Coding Recipes section), but may be used elsewhere as well.

## ARRANGING THE TEXT WITH FORMATTING ATTRIBUTES

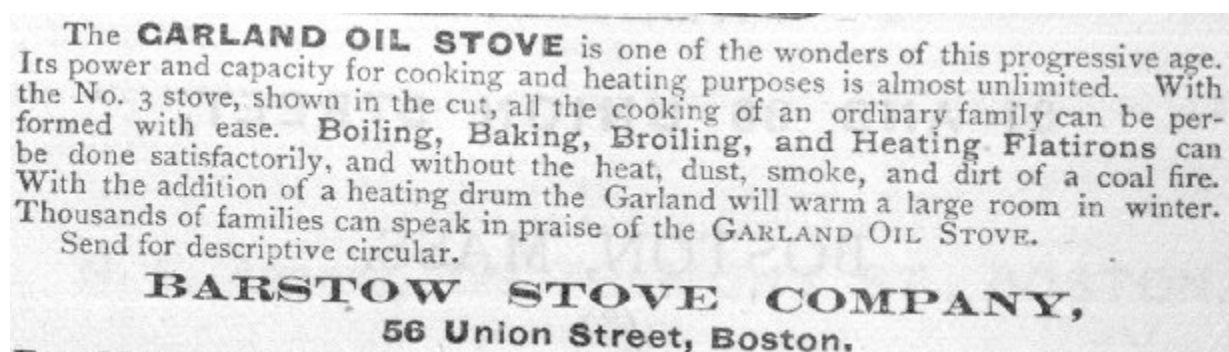
Within most of the cookbooks in our collection, and within the front and backmatter of those books especially, text is arranged in various styles and page alignments to improve the appearance of the book. When coding, we want to preserve the original appearance of the book in our electronic text to the best of our ability. To this end, all of the tags in the cookbook tagset include attributes that allow you to change the way the text appears on the page.

<b>FORMATTING ATTRIBUTES THAT OCCUR IN MOST ELEMENTS IN THE COOKBOOK DTD</b>	
align=	Allowed values are center, right, indent1, and indent2.
rend=	Allowed values are bold, italic, and ornate.
size=	Allowed values are larger and smaller. (This means larger or smaller <i>than the text immediately surrounding</i> the tagged text.)
placement=	Allowed values are heading and inline. Heading means on a line by itself. Inline means not on a line by itself, like the text in a paragraph.

height=	This is an attribute available only in <ref>, used mainly for formatting footnote marks (but it may be used anywhere you find a superscript or subscript within the text). Allowed values are subscript (below the line of text) and superscript (above the line of text).
---------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Although the <emph> tag may be used to add these attributes without the use of any other tag, it is preferred that we streamline the coding by adding them to other elements and deleting the <emph> wherever possible. Thus, formatting is often defined within other tags, such as <p>, <hd>, <doctitle>, etc.

For example, here is a piece of an advertisement from *Mrs. Lincoln's Boston Cook Book*:



The coding for this advertisement looks like this (codes are displayed in bold):

```
<p>The <emph rend="bold">GARLAND OIL STOVE</emph> is one of the wonders of
this progressive age. Its power and capacity for cooking and heating purposes
is almost unlimited. With the No. 3 stove, shown in the cut, all the cooking
of an ordinary family can be performed with ease. <emph rend="bold">Boiling,
Baking, Broiling, and Heating Flatirons</emph> can be done satisfactorily,
and without the heat, dust, smoke, and dirt of a coal fire. With the addition
of a heating drum the Garland will warm a large room in winter. Thousands of
families can speak in praise of the GARLAND OIL STOVE.</p>
```

```
<p>Send for descriptive circular.</p>
```

```
<p align="center" rend="bold">BARSTOW STOVE COMPANY,<lb/>
56 Union Street, Boston.</p>
```

Where all text within a paragraph could be considered bold, I coded the bold attribute within the paragraph tag. (Note that, if you put a formatting attribute inside a larger tag, *all text within that tag* will be formatted according to that attribute.) However, for text within a paragraph that is emphasized but otherwise unremarkable, it is necessary to use the <emph> tag.

Note that "rend=" is a required attribute for the <emph> tag; thus, <emph> cannot be used to code text alignment or size only. Any text surrounded by an <emph> tag must be either bold, italicized, or ornate text.

Although it is good to be as accurate as possible when formatting text, there is no need to get too picky about text size and alignment if the tags just won't accommodate the original page formatting. Particularly in advertisements and on title pages, text sizes and alignments often vary more widely than our code does. Notice that I did not code the BARSTOW STOVE COMPANY text as `size="larger"`; this is because there was a block of text on the same page that was much bigger than that particular line, so I just didn't bother. Ultimately, we only have three different sizes of text; keep this in mind as you are coding, and feel free to ignore spots that are too troublesome. I often brush over the fine details of the page in order to make the coding more efficient, particularly in places where lines with similar formatting can be grouped into paragraphs and broken with `<lb/>`. The rundown:

Text in any tag defaults to left alignment.

<code>align="center"</code>	Centers the text.
<code>align="right"</code>	Moves the text to the right side of the page.
<code>align="indent1"</code>	Indents a line of text one tab space over. (A tab space that is typed in will be ignored by the browser, and the text will appear left-aligned as normal.)
<code>align="indent2"</code>	Indents a line of text two tab spaces over.
<code>rend="bold"</code>	Makes the text bold.
<code>rend="italic"</code>	Makes the text italic.
<code>rend="ornate"</code>	Changes the typeface of the text to an ornate style.
<code>size="larger"</code>	Makes the text larger than the surrounding text.
<code>size="smaller"</code>	Makes the text smaller than the surrounding text.

There is one more formatting attribute that is included in all tags, and that is `"placement="`. The placement attribute can be used to offset headings without using the `<hd>` tag. Thus far in the project, it has been used almost exclusively to offset headings within a `<recipe>`, for reasons that will be discussed in the section on coding recipes. You may use it in other areas of the cookbook as well, but be sure you understand what it does before you try.

The `"placement="` attribute has two possible values:

<code>placement="heading"</code>	Takes an element contained within a larger structural element (usually <code>&lt;p&gt;</code> ) and moves it to a "heading" position—that is, separates it from the other text inside the larger element by putting it on its own line and adding a blank line between it and the rest of the text. May be used with any smaller element within <code>&lt;p&gt;</code> (although using it with <code>&lt;hd&gt;</code> would be redundant).
<code>placement="inline"</code>	Takes a heading ( <code>&lt;hd&gt;</code> ) contained within a paragraph ( <code>&lt;p&gt;</code> ) and restrains it to an "inline" position, so that the text may still be recognized as a heading without being placed on its own line above the rest of the paragraph.

Note that the "placement=" attribute is designed to format elements that are *contained within* a larger structural element—not within elements that *are themselves* larger structural elements. A tag containing the 'placement="heading"' attribute will only be offset if it is surrounded by text to be offset from. If the attribute is used in a place that would be offset normally anyway, it would be redundant and unnecessary. The <p> and <hd> tags are automatically set apart from the surrounding text, so a 'placement="heading"' attribute in either of these tags is unnecessary. The 'placement="inline"' attribute would also never be necessary within <p> (since you cannot have a <p> within a <p>, and remember, this is only for elements contained within a larger structural element), but may be used with a <hd> inside a <p> if the headings in your book are not offset from the surrounding text.

The use of the "placement=" attribute inside recipes and formulas will be discussed in a later section.

## A NOTE ON SPACING

As mentioned in the description of the <lb> tag, simply hitting "enter" to break a line in XMetaL will not create a line break when the document is viewed in a web browser. Likewise, hitting the spacebar more than once will not add extra blank spaces to a line of text; only one blank space will be displayed when the document is viewed online, even if you hit the spacebar ten times in a row. This is because web browsers, when displaying coded documents such as the ones we are creating in this project, ignore all extra *whitespace*—that is, all characters that appear as blanks on a display screen or printer, including the space character, the tab character, and carriage returns (line breaks).

To create a line break in the middle of a line, as already explained, you need to use the <lb> tag; but to create extra spaces (as though you had hit the spacebar more than once) in the middle of a line, you will need to use a special character code. The code for a single blank space is **&nbsp;**—non-breaking space. If you need to make a long line of blank spaces, you can intersperse the character code with regular space characters—the web browser will read one space character between two other characters normally.

Normally, creating a long string of blank spaces is not very important in the cookbook coding—but it can come in handy when you are coding tables of contents or indexes that have a lot of quotation marks you would like to line up with the words above them, for example.

## CODING TITLE PAGES AND OTHER FRONT AND BACKMATTER

Title pages vary widely in terms of formatting and content, but they all contain three basic elements: the title, author, and imprint (publisher's information) of the book. Ruth Ann has created three special tags to code these chunks of text *only on the title page of the book*: <doctitle>, <docauthor>, and <docimprint>. Be careful not to forget these tags; it's easy to overlook them, since you only use them once in any given book. Any other text on a title page should be placed in <p> tags, only adding attributes to accommodate formatting. Illustrations, of course, may be coded as normal.

Indexes, tables of contents, and glossaries should be coded as lists, with target and ID pairs and <term> and <definition> tags where appropriate. I have included more detailed instructions on how to code these in a later section.

Any pages in the front and backmatter that contain nothing but a stamp, bookplate, or other information that has been added by the library may be coded as a blank page. If a stamp or bookplate from the MSU libraries appears on the same page as a chunk of meaningful text, just ignore it. (Delete it if it's been typed.)

Other pages in the front and backmatter are coded almost exclusively in <p> and <hd> tags, with added formatting where appropriate.

## THE BODY: STRUCTURAL ELEMENTS

The <body> structure, like front and backmatter structure, is also limited in the number of text division levels. The body can be divided into chapters, which can be divided into sections, which can be divided into subsections.

It is not necessary to use all of these levels; only use as many as necessary to reflect the actual structure of the book. The DTD allows the <recipe> element to be located immediately within the <body> element or within <chapter>, <section>, or <subsection>. For example, Thomas Bullock's *The Ideal Bartender* has no chapter divisions at all, simply a title page and a series of recipes, so the <recipe> elements would be placed immediately within the <body> element.

This means the following structures are possible, from simplest to most layered:

```
<cookbook>
  <meta></meta>
  <front></front>
  <body>
    <recipe></recipe>
  </body>
</cookbook>
```

```

<cookbook>
  <meta></meta>
  <front></front>
  <body>
    <chapter>
      <recipe></recipe>
    </chapter>
    <chapter>
      <recipe></recipe>
    </chapter>
  </body>
</cookbook>

```

```

<cookbook>
  <meta></meta>
  <front></front>
  <body>
    <chapter>
      <section>
        <recipe></recipe>
      </section>
      <section>
        <recipe></recipe>
      </section>
    </chapter>
  </body>
</cookbook>

```

```

<cookbook>
  <meta></meta>
  <front></front>
  <body>
    <chapter>
      <section>
        <subsection>
          <recipe></recipe>
        </subsection>
        <subsection>
          <recipe></recipe>
        </subsection>
      </section>
    </chapter>
  </body>
</cookbook>

```

## WHEN TO USE THE "CLASS=" ATTRIBUTE FOR <chapter>, <section>, and <subsection>

The "class=" attribute (for possible values, see Coding Recipes section) is optional for the <chapter>, <section>, and <subsection> elements. As of May 2002, "class=" should be used only for the *smallest* of these three elements in use in a particular book or portion of a book whenever possible. Theoretically, this will avoid having the search engine produce duplicate search results. So:

If you have a <chapter> with no smaller divisions, you *must* assign a "class=" value to that chapter. Because the attribute is optional within <chapter>, rules checking will not warn you if you forget, so please be careful. Omitting the "class=" in this case will make the information contained in that chapter more difficult for visitors to our website to find.

If you have a <chapter> that is divided up entirely into <section>s—this means *all* of the text within the chapter is contained in <section> tags, not just occasional portions of it—you *must* assign a "class=" to each of the sections, and you should *not* assign a "class=" to the <chapter>. Once again, be very careful not to omit the "class=" values in the <section> tags. It's up to you to notice if they're missing.

If you have a <chapter> that contains mostly recipes but also the occasional <section>, you should assign a "class=" value to *both* the <chapter> *and* the <section>s contained within that chapter. This is done to ensure that all of the text within the <chapter> has been classified in some way or another, and is also classified in as much detail as possible (i.e. in the smallest portions possible).

The above rules apply to the <section>/<subsection> level of coding as well.

## THE BODY: RECIPES AND FORMULAS

**Within the <chapter> or <section> or <subsection> elements** (which hold the major portions of the text) the majority of the text should be tagged as one of three types:

<recipe>	Directions for making something edible, and intended as a food or beverage. This category does not include medicines taken internally.
<formula>	Directions for making something non-edible (or not intended as a food or beverage), such as laundry starch, fabric dyes, or medicines.

<p>	General commentary that is not part of a recipe or formula, such as advice on how to choose foods in the marketplace, foods that go together well, table manners, etc. Some of the books will also have extensive sections on other domestic matters such as childrearing, care of invalids, advice on household management, etc.
-----	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Some recipes and formulas will contain more than one paragraph, and (as one might expect) these are indicated with <p> tags. However, although many recipes are complete in a single paragraph, these must also have a set of <p> tags immediately within the <recipe> tags.

Although this is somewhat repetitive, it serves two purposes. In order for the style sheet to produce a consistent screen display, the coding also needs to be consistent, whether a recipe has one paragraph or two. This means we must either use the extra <p> tags, or have no <p> tags at all within recipes and use <lb> or some other construction to indicate a second or third paragraph. Since the typists are already inserting <p> tags, the first choice makes more sense.

## CODING RECIPES AND FORMULAS

(Note: I have included a number of sample recipes and their coding at the end of this packet; if you wish, you can refer to them as you read through this section.)

When you first identify a recipe or formula--which can be ANY set of concrete instructions on how to make, preserve, construct, clean, exterminate, etc. anything at all--the first thing you must do is enclose it in <recipe> or <formula> tags. After you have added the tags themselves, you MUST define a "class1=" value for each recipe, and a "class=" value for each formula. Recipes may have a second class ("class2=") as well, but formulas have only one. Class values must be chosen from the following list (which, incidentally, is the same list of options used for the <cookbook>, <chapter>, <section>, and <subsection> tags):

DEFINITIONS OF "CLASS=" VALUES FOR FOOD TOPICS	
"fruitvegbeans"	Preparing and preserving fruits, vegetables, beans, and legumes of all kinds; proper storage conditions; nutritional value of these foods. This includes mushrooms and nuts.
"meatfishgame"	Preparing or preserving beef, lamb, mutton, poultry, seafood, and wild game such as venison, squirrel, buffalo, etc. Include organ meats such as kidney, brains, tripe, etc. Also, storing these foods; nutritional value of these foods.



"eggscheesedairy"	Making cheese or other dairy products (i.e. yogurt) and recipes which have eggs, cheese, or dairy products as the major ingredients (i.e. puddings, creams, custards, quiche). Also, storing these foods; nutritional value of these foods.
"bread sweets"	Breads, cereals, grains, baked goods, and pasta: crackers, muffins, tarts, pies, cakes, pancakes, oatmeal, rice, macaroni, etc. Also, sweets or desserts even if they are not baked, such as fudge, boiled sugar candies, icings for cakes, ice cream, etc. Also, storing these foods; nutritional value of these foods.
"soups"	Soup recipes. This category takes precedence over "fruitvegbeans" and "meatfishgame" -- i.e. asparagus soup goes here, not in "fruitvegbeans"; beef broth goes here, not in "meatfishgame". Also, storing these foods; nutritional value of these foods.
"accompaniments"	This category encompasses foods meant to season or flavor other foods, or which are used in the preparation of other foods, rather than being eaten alone. This includes recipes for sauces, jams and preserves, and condiments such as mustard or pesto, as well as directions for using or preparing herbs or spices. Yeast is also classified under "accompaniments." Also, storing these foods; nutritional value of these foods.
"beverages"	Anything meant to be drunk instead of eaten. Milk or eggnog goes here, not in "eggscheesedairy." Fruit juice goes here, not in "fruitvegbeans." Also, storing these foods; nutritional value of these foods.
"marketing"	Advice on how to choose items at market, or descriptions of articles found at market; for example, an explanation of the different cuts of meat offered for sale in a marketplace, or instructions on how to choose the best ears of corn from market.
"generalfood"	Applies only to <cookbook>, <chapter>, and <section>. This is to be used when two or more categories are covered by the material. Most cookbooks will be class1="generalfood" because they cover all types of food.
"menus"	Restaurant menus, "bills of fare" and other portions of text that list foods that go together well. This can only be used with <cookbook>, <chapter>, or <section>.

"medhealth"	Foods and beverages prepared especially for invalids, infants, or small children, but that are not medicinal and may be eaten even by those in good health.
-------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------

<b>DEFINITIONS OF "CLASS=" VALUES FOR NON-FOOD TOPICS</b>	
"medhealth"	Information about health, nutrition, hygiene, beauty, or care of the sick, infants, and small children. Examples might include: "a tincture for mouth and gums" (i.e. toothpaste), "tonics" or nutritional supplements like cod liver oil, or poultices for dressing wounds.
"household"	Information about household management. Examples of <formula> under this category would include directions for preparing things like laundry starch or fabric dyes.
"farmgarden"	Anything related to the raising of food or livestock. Examples might include advice on caring for an orphaned calf or making a spray to ward off potato beetles.
"childrear"	Advice on raising children.
"etiquette"	Advice on good manners, how to behave in social situations, directions on hosting parties, etc.
"restaurant"	Advice on managing a restaurant or hotel, or (in the case of Tunis Campbell) training employees of a restaurant or hotel.
"servants"	Use this for topics concerning servants in private homes. Hotel employees should be listed under "restaurant" (because that is actually shorthand for "restaurants and hotels".)
"generalnonfood"	Anything that doesn't fit in one of the categories above.

<b>And, a very general category, probably only applicable at the &lt;cookbook&gt; level:</b>	
"foodandnonfood"	For those encyclopedic works that address many sorts of foods and many sorts of noncooking topics such as gardening, nursing the sick, organizing household work, etc.

If in doubt as to which class a recipe, chapter, section, subsection, or cookbook should be placed in, use the following rules as a guideline:

a) Classify according to the nouns in the recipe title, not the adjectives. For example, should "bread pudding" be classed as "breadweets" or "eggscheesedairy"? The noun in this case is "pudding" which is basically the eggs-and-milk part of the recipe. Therefore, classify this in "eggscheesedairy."

b) If that doesn't work, look at the ingredients. Classify according to what seems to be the "biggest" ingredient in the recipe.

We are trying to use the class2= attribute as seldom as possible when coding these books; in most cases, you should be able to narrow down the class value for any particular recipe into a single class. If you feel you must, however, go ahead and assign a class2. Here are some cases where class2= values have been assigned that we would like to keep consistent:

- **Meat pies, or pastries containing meat:** assigned a class1="breadweets" and a class2="meatfishgame". Hopefully this will narrow the search for people looking specifically for meat pies as opposed to fruit pies.
- **Sandwiches:** assigned a class1="breadweets" and a class2= which is dependent on the type of filling used in the sandwich. An egg salad sandwich would have a class2="eggscheesedairy"; a ham sandwich would have a class2="meatfishgame".
- **Medical recipes:** Any recipe that makes a food intended for invalids, the elderly, infants, or small children (but which could be eaten by people in good health) should be classified with a class1= to match the type of food the recipe makes and a class2="medhealth". These will not always be made obvious by the recipe title alone; for example, you may have a recipe for lemonade which contains a side note reading "This is a pleasant and cooling beverage in fevers." If a comment like this appears anywhere in the recipe, add a class2="medhealth".
- Any recipe which is titled "\_\_\_\_\_ and \_\_\_\_\_" (for example, "pork and beans" or "spinach and eggs") should be coded with a class1= to match the first item and a class2= to match the second, assuming both items are equally important to the recipe and are used in approximately equal proportions. "Pork and beans" would have a class1="meatfishgame" and a class2="fruitvegbeans"; "spinach and eggs" would be class1="fruitvegbeans" and class2="eggscheesedairy." Do not bother to assign a second class if the second item is used only as a garnish, however.

In any kind of classification system, there always will be differences of opinion about what category a certain thing should go in; this is normal and to be expected. Try to be consistent, but don't agonize over individual recipe classifications.

Before moving on to further tagging within the recipe, you should also check for other attributes that are unique to the recipes. These include:

<b>ATTRIBUTES WITHIN THE &lt;recipe&gt; TAG</b>	
region=	Use if the recipe comes from a specific region. Acceptable values are "west", "midwest", "south", "northeast", and "ethnic". (Don't use region="ethnic" at the recipe level, however; using the "ethnicgroup=" attribute for an ethnic recipe will be sufficient.)
subregion=	If the "region=" value is not descriptive enough, you may add a subregion (perhaps a state name). You may type whatever is appropriate in this field.
ethnicgroup=	If the recipe is ethnic, enter an ethnicgroup. You may type whatever is appropriate in this field.
occasion=	Use when a recipe is meant to be prepared for a specific occasion or season. Acceptable values are "Thanksgiving", "Christmas", "wedding", "birthday", "patriotic", "spring", "summer", "fall", "winter", "other". A recipe for Election Cake, for example, would go under "patriotic", while a recipe for Lent would be labeled as "other".
alcoholic=	This attribute applies only to beverages. Acceptable values are "yes" or "no", but you should only use this tag to mark alcoholic beverages, and always choose "yes". "no" is only there for the sake of contrast.
id=	Use only if each recipe has its own number that is referred to in the index of the book. You may type whatever you wish in this field as long as each id= value is unique (you should probably use a scheme such as "r001", "r002", etc.), and then use these values to create target and id pairs when coding the index. This will link the numbers in the index to the individual recipes, not just to page numbers.

Some of the cookbooks, such as the *Great Western Cook Book* or *Aunt Babette's Cook Book*, will contain recipes that *all* need to be coded as a certain region, ethnicgroup, etc. In these cases, you may want to leave yourself a note at the end of the file reminding yourself to do a find and replace on the string "<recipe " to paste these attributes in automatically; it will save you some time. Be careful, though, since even certain regional and ethnic cookbooks may contain recipes that are not from the same region or ethnic group as the majority of the cookbook; for example, if you were coding a Northeastern cookbook and came across a recipe that claims to be "a Southern dish", you would code it as <recipe region="south">, NOT <recipe region="northeast">. Also, if the title of a recipe includes an ethnicity--"German Waffles", for example--it is acceptable to include that ethnicity as an attribute (<recipe ethnicgroup="german">). Do NOT use the titles of recipes to assign regions, however--a recipe for "New Jersey Potatoes" would NOT be coded as <recipe region="northeast">.

After adding the appropriate attributes to a recipe tag, you may begin coding the lower-level elements which appear within <recipe> and <formula>. These elements are as follows:

<b>LOW-LEVEL ELEMENTS FOR MARKING UP RECIPES AND FORMULAS</b>	
<purpose>	This is the title of the recipe or the statement of what the directions will produce. In older cookbooks, this is often located in the first sentence: "To make a bread pudding..." In later cookbooks this is usually a heading located before a list of ingredients. When the <purpose> appears as a heading on the page, use <purpose placement="heading">. (Don't "double-tag" it as <purpose><hd>Chocolate Cake</hd></purpose>)
<process>	<i>(ignore)</i> This would be used for verbs: braise, boil, etc. Use this very sparingly, for actions that are uncommon in 20th century cooking. Don't tag words like "stir" or "roast." Do tag words or phrases like "let the batter sweat overnight".
<ingredient>	<i>(perl)</i> This would be used for ingredients in a recipe or the items used to make a formula: things like madder root or walnut hulls would be ingredients for a fabric dye formula.
<implement>	<i>(perl)</i> Objects used to perform some action in a recipe or formula. Ignore common items like spoons, bowls, pots and pans. (This tag may also be used outside of a recipe or formula.)
<measurement>	<i>(ignore)</i> Use this to flag unusual measurement terms such as gill or teacup-full.
<contributor>	Use in church and charity cookbooks when contributors of individual recipes are listed.
<attribution>	Use when a recipe is attributed to someone else besides the editor or author of the book being tagged. <attribution>"This is based on Julia Child's recipe for boiled turnips."</attribution> Use this tag only when a person's actual name is mentioned in the recipe; do not use it to tag phrases such as "A gentleman from Missouri" and the like. You may also use it to tag the names of organizations or titles of books accredited with any given recipe. (This tag may also be used outside of a recipe or formula.)
variation	Use for variations on a recipe. Usually this means an instruction to follow the same cooking directions and set of ingredients but with one or two substitutions.

As you may have noticed, two of the tags (<process> and <measurement>) have an *ignore* note at the beginning of their description. This is because, due to time

constraints, we will not be coding these two features of the cookbooks (although we are making an attempt to keep track of the types of obscure processes and measurements that come up in the books, which will be explained later). Therefore, do not code processes or measurements within the recipes.

Also, two more tags (`<ingredient>` and `<implement>`) have a *perl* note in their descriptions. Because it would be both exhausting and time-consuming to attempt to tag every ingredient and implement in every recipe by hand, we have created two different perl scripts to tag these elements for us. Because the perl scripts are dependent on the major structural tagging already being present in the file before they are run, you will have to code each book in three different stages: once to put in the major structural tags, once to check the ingredient coding within recipes and formulas, and once to check the implement coding. I will describe to the two latter stages of coding in a later section. For now, however, just concentrate on the `<purpose>`, `<contributor>`, `<attribution>`, and `<variation>` tags, which are added in the first stage of coding.

The descriptions above should be sufficient to explain the use of the `<attribution>` and `<contributor>` tags. `<purpose>` and `<variation>`, on the other hand, can cause a certain amount of confusion when you first begin coding. For that reason, I will provide a little more elaboration on those two tags here.

The `<variation>` tag, as stated above, encloses an alteration of a recipe that consists of the same set of instructions, but with one or two substitutions. Sometimes it is difficult to tell what actually constitutes a variation; some are obvious, and some are a little more subtle. You will have to use your own judgment on what you code as a variation and what you don't, but here are a few suggestions on how to decide:

Anything that produces an end product that is different from the originally stated purpose of the recipe is definitely a variation. For example, if you come across a recipe for strawberry pie, and in the final paragraph there is a sentence that reads "For raspberry pie, use raspberries," that final sentence would be coded as a variation. In this type of variation, you may also define a second purpose to reflect the end product of the variation (as opposed to the recipe): in this case, "raspberry pie" would be the purpose of the variation.

Anything that substitutes one or two ingredients for another ingredient in the recipe may be coded as a variation, whether the end product of the recipe changes or not. (If the end product of the recipe does not change, defining a second purpose is not necessary.)

If you come across a statement suggesting an addition or substitution to the recipe, such as "You may add lemon juice if you like" or "The nuts may be omitted," you don't really need to code it as a variation. I generally don't bother tagging such simple alterations as these. However, if you feel the alteration changes the recipe a great deal, you may tag them—it's your call.

As for the purpose tag—as you have already read, it is used to mark the title of the recipe or the statement of what the directions will produce. It is very important not to neglect to put in the purpose tag, for two reasons. One, this tag is not required by the DTD, so if you forget to put it in, rules checking will not catch your error. Two, the <purpose> tag is used in close conjunction with the search engine on our website to help users locate specific recipes. Therefore, it is very important that the tag be both present and enclosing a chunk of text that is descriptive enough to help a user find a recipe through our website.

This second condition can be a problem in some of the older cookbooks and, more commonly, within variations on a recipe. Often, there is no chunk of text that can be tagged as <purpose> that clearly states what the recipe or formula is meant to make (or which process the recipe or formula is meant to teach you) in these instances. This is a problem because of the way the search engine will work at the end of the project.

If you visit the Feeding America homepage (<http://digital.lib.msu.edu/cookbooks/>) and click on the "Search the Collection...Within Text" link from the navigation bar on the left, three different search fields will appear: "Find recipes for," "Find Recipes that use," and "Find instructions for household remedies, etc." These search options are specialized searches that we have designed for this project which make it possible for the user to limit his or her search to only certain parts of the text, not the full text. In coding terms, these searches narrow the text to be searched to *only* the text that appears within specific tags—in the case of "Find recipes for" and "Find instructions for household remedies, etc.", only text within <purpose> tags is searched for the user-defined term; in the case of "Find Recipes that use," only text within <ingredient> tags is searched.

When one thinks about the nature of the site we are creating in this project, the usefulness of this feature is readily apparent. Imagine that someone would like to search our collection of cookbooks for apple pie recipes. If the user were to do a full-text search for "apple pie", he or she would likely come up with a number of recipes that included phrases such as "Prepare the crust just as you would for Apple Pie" and the like, which are not actually recipes for apple pie themselves. A search limited to text within <purpose> tags, however, would be much more useful in this instance. Since <purpose> tags mark the end result of a recipe, only recipes that *produce* apple pie should be selected as results for a search within purpose; recipes that only *mention* apple pie should be excluded. So if users take advantage of the "Find recipes for" search, they should be much more successful at finding the information they are looking for when searching for specific recipes. If this still confuses you, try a few searches in either the "Find recipes for" or "Find instructions for household remedies, etc." fields and see what you come up with. The words displayed in bold are the words tagged as <purpose> within those recipes.

What all this means in practical terms is that, while coding, you have to pay attention to the text you put inside a <purpose> tag. Think of it in terms of the search engine, and what sort of terms a visitor to our website might enter when trying to find that recipe. If the recipe heading that appears in the original text

isn't descriptive enough, for example, or if you're coding a variation that doesn't have its own heading but still requires a purpose separate from the main recipe heading, you may want to use the `<alt>` tag to add the missing information. This use of the `<alt>` tag, then, is purely a means of artificially inserting extra information that will be used by the search engine to locate a recipe. If you put the `<alt>` tag *inside* the `<purpose>` tag, then type extra terms within the `synonym1=` or `synonym2=` attributes within `<alt>`, those coder-defined synonyms will also be searched when a user tries the "Find recipes for" or "Find instructions for..." searches on our website.

It may be a little challenging to determine which phrase you should mark with the `<purpose>` tag within a recipe. Usually the recipe heading will be appropriate; if there is no recipe heading, use your best judgment to mark something within the recipe itself, and use the `<alt>` tag if necessary.

One other side note concerning purposes: when coding the purpose of a *medhealth formula*, Ruth Ann would like us to tag the condition the formula is supposed to treat with `<purpose>` tags whenever possible. Please keep this in mind when coding medhealth formulas. Unfortunately, the medical condition treated by the formula does not always appear in the heading for the formula; often, it will be buried somewhere inside the formula, and the formula will have some other title describing its product, such as "green ointment" or "healing salve." To address this, I have occasionally added an `<alt>` tag within `<purpose>` and listed the medical conditions the formula treats there.

### **A note on formatting `<purpose>` tags**

The `placement="heading"` attribute, mentioned earlier in the formatting section, is used almost exclusively to format the `<purpose>` tag inside a recipe, when the purpose appears as the heading of a recipe. As mentioned in the chart above, Ruth Ann would prefer us to use `placement="heading"` within `<purpose>` to mark the heading of a recipe, not separate `<hd>` tags in addition to the `<purpose>` tag, which would be rather redundant. So if you are coding a book in which the purpose of the recipe is stated in a heading, you will need to do the following:

- 1) Add the `placement="heading"` attribute to the `<purpose>` tag.
- 2) Add other formatting attributes (`align`, `rend`, `size`) to the `<purpose>` tag when appropriate.
- 3) Move the first `<p>` tag in the recipe to a point between the `<recipe>` tag and the `<purpose>` tag.

Notice that the first `<p>` tag in the recipe is moved to include `<purpose>`. This may seem strange, but it is necessary because *all text in a recipe MUST be inside `<p>` tags*. (This is useful to remember when coding lists of ingredients as well—lists can stand on their own in the main body of the text, but must be wrapped in `<p>` tags when they appear inside a recipe.) Also, the way a `placement="heading"` tag works is to take some of the text *within a paragraph* and separate it from the rest of the paragraph with a blank line—thus offsetting the text to make it look like a heading. If the `placement="heading"` tag is used in isolation, then—for example,



say you put it inside a `<p>` tag that enclosed a single line of text—it would serve no purpose whatsoever. To mark headings outside of recipes, please use only the `<hd>` tag; reserve the use of the `placement="heading"` attribute for purposes within recipes.

### The three-stage coding process

I mentioned earlier that, because of our use of the perl script to automatically tag ingredients and implements in the book, the coding is done in three stages. Everything I have covered up until now has been part of the first phase of coding, which involves essentially coding everything except the `<ingredient>` and `<implement>` tags in a book. In the second stage of coding, you will be scanning over ingredient tags that have been automatically pasted in by the ingredients perl script; in the third and final phase, you will be checking the implement tags pasted in by the implements perl script.

Now, as you might expect, there are a **lot** of ingredients in each cookbook. While the perl script does tag most of these accurately, there are a lot of places where it will paste in tags where they are unnecessary, or skip tags where they are needed. To correct these errors, we need to go through the books in a fair amount of detail; each recipe will need to be scanned to remove and add tags as appropriate. This process can get to be quite exhausting—but it does have its advantages.

For one thing, if you made any coding mistakes your first time through the book—say you missed a purpose tag or an attribute in a recipe—you'll have the chance to notice and correct the error when you're going through the book a second time in more detail. As an extension of this idea, the fact that we go through each book in detail twice allows you to be a *little* sloppy in your first run through the recipe coding. I underlined recipe because it is important not to miss details of the coding *outside* the recipes. The perl script is designed to only code ingredients inside of recipes, so when scanning for ingredients, you will only be scanning parts of the book that contain recipes. At any rate—what I'm getting at is that if you want to go a little faster through your first stage of coding and not be too careful about catching every variation in all the recipes, it is acceptable at this point to simply concentrate on classifying the recipes and coding purpose tags where appropriate. If you do this, however, you will have to be careful about catching the variations the second time through.

### WHEN TO USE THE `<alt>` TAG

The `<alt>` tag is used for a variety of purposes in the cookbook coding. One is to correct typographical errors in the cookbooks--this means words that are actually misspelled, not simply spelled in an old form. The word "yelks", for example, I would leave alone; but if I came across the word "ylks" in the same cookbook, I would code it as

```
<alt synonym1="yelks">ylks</alt>
```

to correct the typographical error. (Note that the correct spelling is entered in a synonym1= attribute, which only appears within the <alt> tag.)

To address the issue of archaic spellings, which is what the <alt> tag was originally created for, I have been keeping an excel spreadsheet to record all notable variations in spelling that I come across, located in K:\cooking\spreadsheets\archaic spellings.xls. You are welcome to add to this spreadsheet at any time. Theoretically, we will be creating a perl script at some point to tag these words, but we haven't gotten that far yet.

The other way that I commonly use the <alt> tag is to help define the <purpose> of a recipe. If a recipe or variation does not contain a chunk of text that fully and accurately describes what the recipe or variation is supposed to make or do, an <alt> tag may be used within <purpose> to make the text more searchable. In this case, the <alt> tag would appear immediately within <purpose>, with the synonym1= (and possibly synonym2=) attribute containing the terms or information the coder would like to add. See the section on coding recipes for a more detailed description of how and why we use the <alt> tag in this manner.

## TARGET AND ID PAIRS

Target and ID "pairs" are used to refer a reader from one part of a document to another. They're referred to as pairs because whenever you use `<ref target="example">` in one part of a book, there has to be another portion tagged as `<element id="example">`. When a matching "target=" and "id=" pair exist in a document, the final website will allow you to click on the text surrounded by the `<ref target="(example)">` tag to jump to the place marked with the corresponding "id=(example)" value. In other words, you are creating an internal link within the document whenever you set up a target/id pair.

The "id=" attribute can be used in many different elements in the cookbook DTD. It is required in `<pb>`, so that is the most frequent use. (This is why, when creating indexes and tables of contents, we use the page break id numbers as target values to guide the user to a particular page; by doing so, we can avoid defining new id values purely for the sake of the index.) It can also be used in lists, chapter headings, illustrations, recipes, etc.

I have included some examples of concrete applications of target and id pairs later in this document, under the headings of "CODING FOOTNOTES" and "CODING LISTS, INDEXES, TABLES OF CONTENTS, LISTS OF ILLUSTRATIONS, AND GLOSSARIES OF TERMS." Footnotes and tables of contents/indexes are the most common applications of the target/id pair, so it may help to look over the examples to get a better idea of how these pairs work.

## EXTERNAL REFERENCES

The cookbook collection will be accompanied by several groups of supplementary material: author biographies, essays on individual books and cooking genres, a glossary of cooking terms, and a gallery of museum objects. We will need to create links to this material from within the cookbook texts. The attributes `xref=` and `item=` will be used to do that. These attributes can be used with most elements.

We are hoping to write a perl script to automatically insert `xref=` and `item=` values for us; Dave is currently working on this, so you don't have to worry about coding them for the moment. At the end of the project, though, you will likely be checking over the `xref=` and `item=` coding in the same way that you check over ingredient and implement coding, so please be aware that these attributes exist, and keep in mind what they are supposed to be tagging.

The allowed values for `xref=` are authors, essays, objects, glossary (the same four categories named above).

The value for `item=` is a code referring to the particular author, essay, object, or glossary entry. We'll need to create standardized lists for these.

Example: one of the museum objects being photographed is a Dutch oven. When a recipe mentions using this item, it would be tagged like this:

```
"Let the stew simmer for three hours in a <implement xref="objects"
item="dutchoven">Dutch oven</implement> placed among the coals."
```

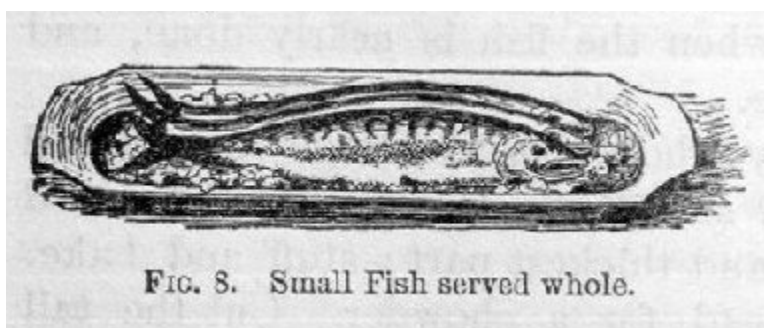
## ILLUSTRATIONS

Follow these guidelines to decide if something *is* an illustration: If you see an abstract design used to fill a little space at the end of a chapter, don't tag it at all. If you see something decorative that *is* an identifiable object (a little row of spoons, perhaps) tag it as an illustration even if it's only there to fill space.

ILLUSTRATION TAGS	
<illustration>	Contains <caption> and <description>. The caption is optional, since there might not be one. The description is required.
<caption>	Wrap this tag around the picture's caption.
<description>	Write a brief description of the picture. See the Sunday school books website ( <a href="http://digital.lib.msu.edu/ssb/">http://digital.lib.msu.edu/ssb/</a> ) for examples of good descriptions.

An additional note about `<description>`: Keep in mind that whatever you type in the description field of the illustration will be displayed online, so be careful to avoid typos, and please try to write in complete sentences or descriptive statements. Don't skip articles like *a*, *an*, or *the*, and use punctuation when necessary. Also, if you're describing a picture that has a caption, try not to repeat the caption in your description; for example, in the illustration shown below, "An illustration of small fish served whole." would not be a good description.

For your reference, here is an example of a captioned illustration and the code I used to transcribe it:



`<illustration>`

`<caption>FIG. 8. Small Fish served whole.</caption>`

`<description>An illustration of three long, skinny fish arranged on a rectangular platter.</description>`

`</illustration>`

(A quick note: it actually is important that the `<caption>` come before `<description>` inside an `<illustration>` tag, so don't confuse the order of these two elements. If you do, the document will not validate.)

If an illustration is embedded within a paragraph and you're not sure where to put it, just move it to a point between paragraphs. Illustrations may occur within paragraphs, however, so if you have an illustration that spans an entire page, you can place the illustration code right where it falls in the normal text.

## EDITORIAL NOTES

The TEI tagset, which we used to code the Sunday school books, has numerous tags for indicating unusual characteristics of a book that cannot be clearly understood through the transcription of the text. For example, there is a tag `<inscription>` for marking up handwritten inscriptions, a tag `<foreign>` for marking

words in a foreign language, and an element `<q>` to indicate a quotation from another source.

The cookbook tagset attempts to reduce the number of tags needed for situations like these by including an element `<ednote>`. When you encounter something that needs a bit of explanation, go ahead and put it in, wrapped in `<ednote>` tags. The XSL stylesheet (used to define how various tags will display in a web browser at the end of the project) will be written to display `<ednote>` material inside square brackets with a heading [Note: ] so it will be clear to the reader that this is an addition to the original text. In general, put the `<ednote>` **after** the place that needs explanation. For example, on the title page of Amelia Simmons:

```
<div type="titlepage">

    <p>John Hammond</p>
    <ednote>Handwritten inscription.</ednote>

    <docTitle>American Cookery...</docTitle>

</div>
```

Editorial notes in the cookbooks project are generally used to mark handwritten inscriptions and text that has been edited or moved due to coding requirements. Most oddities in the text (such as poems, footnotes, and quotes) do not require an editorial note; the user should be able to tell the nature of such text by looking at the transcription or page image.

## WHEN TO USE `<GAP>` AND `<UNCLEAR>`

The `<gap>` tag is used to mark places where pieces of text are completely missing from the cookbook. This could mean missing pages, or just a word that has been obscured by a stain or a printing error. The `<gap>` tag has a **required** attribute, "extent=". In the extent field, you must type in how much of the text is missing. Ruth Ann wants "extent=" values to be standardized so that she can define how they will be displayed in the browser later on in the project. This tag hasn't come up enough for us to make an extensive standardized list yet, but for now, use the following values:

extent="one word"	One word missing
extent="two words"	Two words missing
extent="several words"	More than two words missing
extent="one page"	One page missing
extent="two pages"	Two pages missing
extent="several pages"	More than two pages missing

If you come across something that you feel should have a different "extent=" value, please ask Ruth Ann about it.

Note that the <gap> tag should only be used to mark *missing* text; if you can make out enough of the word to make an educated guess, use the <unclear> tag. This tag is much more straightforward--just type your best guess at the word, and wrap the <unclear> tag around the text you're not sure of. You will probably use this most often in handwritten inscriptions.

## LOOKING UP SPECIAL CHARACTERS

Occasionally you may come across a special character that is not included in the basic list the typists use to input character codes; in these cases, you must look up the XML character code yourself and paste it into the file. Ruth Ann has included links to webpages that provide these codes at <http://www.lib.msu.edu/jonesr/sgml/wkst.htm>; I recommend that you set this as your homepage on your browser if it isn't already, as it also includes links to MAGIC and the public cookbooks website, both of which will be important reference pages while you are coding. If you need help finding the code for any character, see Ruth Ann.

## CODING LISTS, INDEXES, TABLES OF CONTENTS, LISTS OF ILLUSTRATIONS, AND GLOSSARIES OF TERMS

<list> TAGS	
<list>	Use to indicate a list of items. A <list> contains a series of <item>s.
<hd>	Use to mark headings within a list.
<item>	The individual lines or sections of a list. Can contain <term>, <definition>, and <ref>.
<term>	Use only when <list> is being used to encode a glossary-type section.
<definition>	Use only when <list> is being used to encode a glossary-type section.
<ref>	Use for page numbers in a table of contents. See further explanation below.

The basic structure of any list in the cookbooks project is:

```

<list>
  <item></item>
  <item></item>
  <item></item>
</list>

```

where the `<list>` tag goes around the entire list, and `<item>` tags mark each item in the list. Any text in a list may, of course, be formatted and realigned just like any other text in the book. It is also possible to use the `<hd>` tag at any point within `<list>` to code subheadings in a list, although not all lists will have them.

Since indexes, tables of contents, and the like in the cookbooks are coded as lists, they can be very time-consuming. They're long, they require a good deal of reformatting, and they contain references to page numbers, which must be wrapped in `<ref>` tags and assigned a "target=" value to link the user back to that particular page in the text.

When coding an index or table of contents, the first thing I do is fix the formatting. In the case of most indexes, we can't match the formatting exactly with our cookbook code, since there are no allowances for multiple columns on a page. Instead, we put everything into one column. Even after that, some indexes need a little tweaking; most books will have long index entries wrapped onto two or more lines, but I move each single item onto a single line of code. I also try to make all the dots end somewhat evenly.

After I finish the formatting, I go back and add `<item>`, `<ref>`, and `<hd>` tags as appropriate. I find that "cut" and "paste" commands save a lot of time when pasting in these tags, since the tagging on each line will essentially be identical apart from the `<ref target=" ">` value and any odd variations in alignment.

Here is an example of some coding from an index page (tags in bold>):

Maryland Corn Cakes . . . . .	92	Graham Wafers . . . . .	96
Dodgers, or Dabs, or Puffs . . . . .	93	Wafer Biscuit . . . . .	96
Indian Bannock . . . . .	93	Gluten Wafers . . . . .	97
 WAFFLES AND GRIDDLE-CAKES.			
Distinction between Griddle- Cakes, Pancakes, etc. . . . .	97	Corn Meal Griddle-Cakes . . . . .	100
Waffle Iron . . . . .	98	Graham Griddle-Cakes . . . . .	100
Waffles . . . . .	98	Huckleberry Griddle-Cakes . . . . .	100
Lemon Syrup . . . . .	98	Rice or Hominy Griddle-Cakes . . . . .	100
		Bread Griddle-Cakes . . . . .	100

```

<item>Graham Wafers ..... <ref
target="linc116.jpg">96</ref></item>
<item>Wafer Biscuit ..... <ref
target="linc116.jpg">96</ref></item>
<item>Gluten Wafers ..... <ref
target="linc117.jpg">97</ref></item>
<lb/>
<lb/>
<hd align="center">WAFFLES AND GRIDDLE-CAKES.</hd>
<lb/>
<lb/>
<item>Distinction between Griddle-Cakes, Pancakes, etc. .... <ref
target="linc117.jpg">97</ref></item>
<item>Waffle Iron ..... <ref
target="linc118.jpg">98</ref></item>
<item>Waffles ..... <ref
target="linc118.jpg">98</ref></item>
<item>Lemon Syrup ..... <ref
target="linc118.jpg">98</ref></item>

```

In this example, the "WAFFLES AND GRIDDLE-CAKES." line is a subheading in the list, so it is coded as <hd> and center-aligned. <lb> tags were used to add spaces between sections (note that you need two in a row to make one blank line), and "Distinction between Griddle-Cakes, Pancakes, etc." was all condensed into one line of code. (If combining the lines like this leaves you with a line that is just too long to fit with any reasonable number of dots, just put some set number of dots at the end of the line (like three or five) and ignore the fact that that line doesn't line up with the others. To be perfectly honest, the ends of the lines won't line up no matter what we do, unless we code each list like this as a table, which would be too time-consuming.)

Notice that the <ref target=" "> values are the same values that go in the <pb id=" "> field for the page being referenced. The easiest way to find these values is simply to look back at the typing copy. Often, you can calculate the number in your head by figuring out the difference between the image number and the actual page number; for example, in the above book, the image number = the page number + 20. When coding this index, I simply pasted <ref target="linc.jpg">\*\*\*</ref> tags around each page number, then went back and added image numbers to each <ref> tag based on the page number.

If you run across indexes that reference a range of page numbers, such as "113-120" or "12, 13, 14", wrap the <ref> tag around all numbers and set the target as the first page in the sequence. If you have a reference to numbers that are not in sequence, such as "113, 120" or "12, 14", you must tag each number with <ref> tags individually.

I have had one or two books so far in which the index referred the reader back to a certain recipe number instead of a page number; these are the most time consuming, since you must then assign an ID number to each recipe in order to link



everything back to the index. Luckily, in that case, you can follow the numbering scheme that the book has laid out, and you don't need to worry about calculating page IDs.

One other type of list that may appear in the front or backmatter of a book is a glossary of terms; although very easy to code, glossaries do require two special tags that are not used elsewhere: `<term>` and `<definition>`. As you can probably guess, `<term>` goes around the word being defined, and `<definition>` goes around the explanation of that word. Thus, the basic structure of a glossary code would look something like this:

```
<list>
  <item>
    <term></term><definition></definition>
  </item>
  <item>
    <term></term><definition></definition>
  </item>
</list>
```

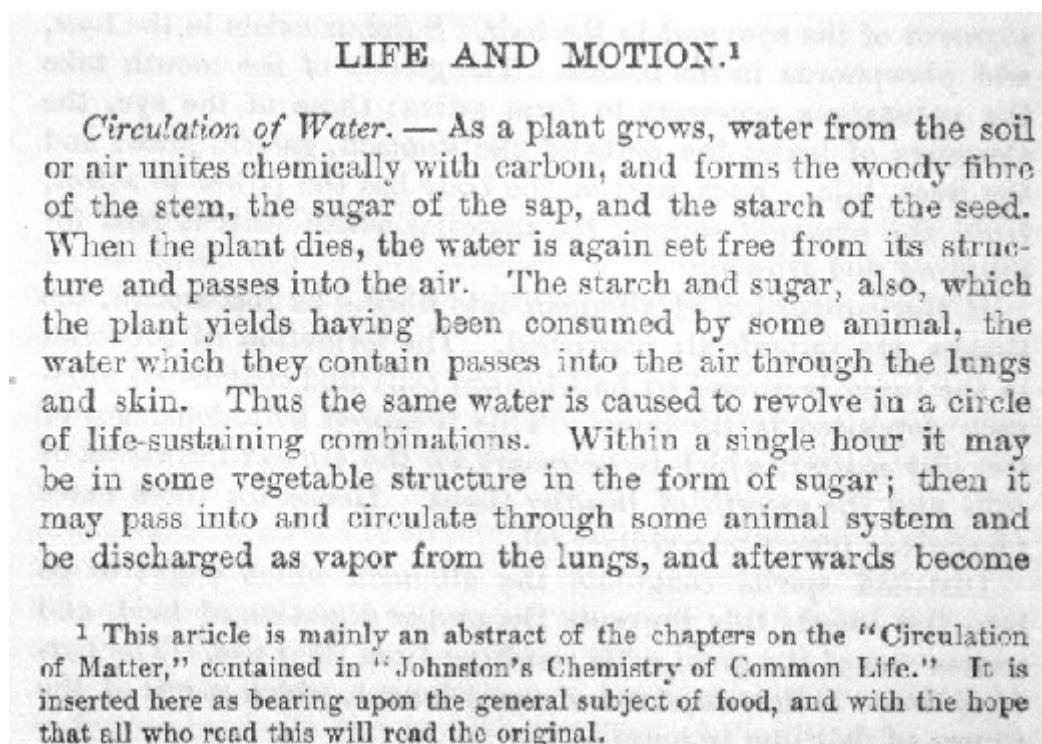
Lists are a fairly common occurrence in the cookbooks even outside of the front and backmatter; it is not uncommon to see ingredients printed in a list form, and I usually code menu sections as lists as well (each menu item gets its own `<item>` tags, and don't worry about formatting—in most cases, it's too strange to be accommodated by our code anyway). In most lists within the body of the text, all you need are the `<list>` and `<item>` tags.

## CODING FOOTNOTES

Footnotes are coded as `size="smaller"` paragraphs in the cookbook project, with a target and ID pair (a link, like those used in indexes and tables of contents) to allow the user to move instantly from the footnote mark to the footnote itself. To code a footnote:

- 1) Wrap a `<ref>` tag around the footnote mark (usually a number or an asterisk).
- 2) Add the `height="superscript"` attribute to the `<ref>` tag if appropriate.
- 3) Establish a `"target="` value for the footnote. I always use the pattern `target="n*"`, where `*` is the number of the footnote, counting from the first footnote at the beginning of the book. Note that, when referencing footnotes, each target/ID pair must be unique; you may use any scheme you like, as long as you don't repeat yourself.
- 4) Wrap a `<p>` tag around the footnote itself, if there isn't one already.
- 5) Add the `size="smaller"` attribute to the `<p>` tag.
- 6) Enter the same value you used for the `"target="` in the `<ref>` tag as the `"id="` in the `<p>` tag.

The fact that we code footnotes as regular paragraphs can complicate things in certain situations. For example, if you come across a page like this:



The footnote on this page appears in the middle of a paragraph. Since you can't have a <p> within a <p>, this poses a coding problem. In order to get around this, move the footnote to the nearest convenient location and add an <ednote> explaining that you have moved it. My code for this section of text looks like this:

```
<hd align="center" size="larger">LIFE AND MOTION.<ref target="n3"
height="superscript">1</ref></hd>
```

```
<ednote>The following note appears on the bottom of page 478 in the
original text.</ednote>
```

```
<p id="n3" size="smaller"><ref height="superscript">1</ref> This
article is mainly an abstract of the chapters on the "Circulation of
Matter," contained in "Johnston's Chemistry of Common Life." It is
inserted here as bearing upon the general subject of food, and with
the hope that all who read this will read the original.</p>
```

```
<p><emph rend="italic">Circulation of Water.</emph>--As a plant grows,
[etc.]
```

In this case, I placed the footnote in front of the regular paragraph and added an <ednote> describing the original position of the footnote. When you code, you may place the footnote anywhere between paragraphs, but please do not place it before the reference mark.

Note that I used the `<ref>` tag to make the footnote mark inside the footnote itself a superscript. This is the only way to code superscripts and subscripts anywhere in the text; the `<ref>` tag is the only one in the tagset that allows the "height=" attribute. It is therefore acceptable to use a `<ref>` tag to code superscripts and subscripts without assigning a "target=" value—without a "target=", you are not creating a link.

Use the above `<ednote>` as a template for what to type when you move a footnote.

## CODING TABLES

<b>&lt;table&gt; TAGS</b>	
<code>&lt;table&gt;</code>	Use to indicate a table within the text—i.e., text arranged in multiple rows and columns. Do <i>not</i> use the <code>&lt;table&gt;</code> tag to represent columns that have been created simply to preserve space on a page, e.g. indexes or lists of ingredients arranged in two columns.
<code>&lt;hd&gt;</code>	May be used to mark headings within a table—not column headings, but headings that appear outside of the row and column structure of a table.
<code>&lt;row&gt;</code>	Use to indicate a row within a table.
<code>&lt;cell&gt;</code>	Use to indicate a cell within a row.

<b>FORMATTING ATTRIBUTES USED ONLY WITHIN &lt;table&gt;</b>	
<code>columns=</code>	Required. Use within the <code>&lt;table&gt;</code> element to indicate the number of columns in a table. (You do not need to indicate the number of rows.)
<code>colspan=</code>	Use within the <code>&lt;cell&gt;</code> element to stretch a cell across two or more columns. The value equals the number of columns you want to stretch the cell across.
<code>rowspan=</code>	Use within the <code>&lt;cell&gt;</code> element to stretch a cell across two or more rows. The value equals the number of rows you want to stretch the cell across.

Tables are a bit more tricky to code than the lists covered in the last section. The tagging itself isn't all that difficult, but deciding where to put which tag can be a little problematic, especially in old books like the ones we're coding. In an ideal world, all tables would be divided evenly into columns and rows, like this:

a	b	c	d
e	f	g	h
i	j	k	l
m	n	o	p

In this case, the coding of the table would be very straightforward. You start with the `<table>` tag, define the "columns=" attribute according to the number of columns you have in the table, and then go row by row, cell by cell, tagging the contents of each cell in order. The basic structure for the above table would look like this:

```
<table columns="4">
```

```
  <row>
    <cell>a</cell>
    <cell>b</cell>
    <cell>c</cell>
    <cell>d</cell>
  </row>
```

```
  <row>
    <cell>e</cell>
    <cell>f</cell>
    <cell>g</cell>
    <cell>h</cell>
  </row>
```

```
  <row>
    <cell>i</cell>
    <cell>j</cell>
    <cell>k</cell>
    <cell>l</cell>
  </row>
```

```
  <row>
    <cell>m</cell>
    <cell>n</cell>
    <cell>o</cell>
    <cell>p</cell>
  </row>
```

```
</table>
```

Notice that, in a regularly formatted table like this one, the number of cells in each row equals the number of columns in the table; if the two numbers don't match, you've probably made a mistake somewhere.

Unfortunately, not all tables will be so nicely laid out. In some instances, a table will be formatted more like this:

a		b	
c	d	e	f
g	h	i	j
	k	l	m

with single cells stretching across two or more columns or rows.

Those of you who have some experience coding tables in HTML should be familiar with the "colspan=" and "rowspan=" attributes. These two attributes, used only inside the <cell> tag of a table, can be used to stretch a single cell across two or more columns, or two or more rows. In order to accommodate table coding situations such as the one above, we have added these attributes to the cookbook tagset as well. The way they work is this:

When you want to stretch a cell across two or more columns, you need to put the "colspan=" attribute inside the code for that cell. In the table above, cells "a" and "b" need a "colspan=2" attribute to stretch them across two columns each. You do *not* need to code the cell twice to match the number of columns in the table; the "colspan=2" attribute essentially makes that one cell equal to two cells in width.

When you want to stretch a cell across two or more rows, you need to put the "rowspan=" attribute inside the code for that cell. In the table above, cell "g" will need a "rowspan=2" attribute to stretch it across the third and fourth rows in the table. Once you have added the "rowspan=" attribute to a cell, you do *not* need to code anything for that cell in the following row—the browser will automatically stretch the cell across the rows, filling the space you will be leaving blank.

It can be a little difficult to visualize how these two attributes work, so for your reference, here's the basic structure for the above table:

```
<table columns="4">
```

```
  <row>
```

```
    <cell colspan="2">a</cell>
```

```
    <cell colspan="2">b</cell>
```

```
  </row>
```

```

<row>
  <cell>c</cell>
  <cell>d</cell>
  <cell>e</cell>
  <cell>f</cell>
</row>

<row>
  <cell rowspan="2">g</cell>
  <cell>h</cell>
  <cell>i</cell>
  <cell>j</cell>
</row>

<row>
  <cell>k</cell>
  <cell>l</cell>
  <cell>m</cell>
</row>

</table>

```

Because visualization of the coding is important when coding tables, I would recommend that, if you come across a table that is not already divided clearly into rows and columns and cells, you draw in the lines yourself, right on the typing copy. This will help you determine where you need to use "rowspan=" and "colspan=" attributes, if you need to use them at all.

In addition to the "rowspan=" and "colspan=" attributes, each cell may also contain the general formatting elements "align=" and "rend=". "align=" in particular is used quite often to center or right-align the text inside a cell. "rend=" may be used if all of the text inside a cell is in italics or bold.

For your reference, here's a simple example of a table and its code:

<i>Baking Bread, Cake, and Puddings.</i>				
Loaf bread	.	.	.	40 to 60 m.
Rolls, biscuit	.	.	.	10 to 20 "
Graham gems	.	.	.	30 "
Gingerbread	.	.	.	20 to 30 "
Sponge cake	.	.	.	45 to 60 "
Plain	"	.	.	30 to 40 "
Fruit	"	.	.	2 to 3 hrs.

```

<table columns="3">
<hd rend="italic" align="center">Baking Bread, Cake, and
Puddings.</hd>
<row>
<cell>Loaf bread . . . . .</cell>
<cell align="right">40 to 60</cell>
<cell align="center">m.</cell>
</row>
<row>
<cell>Rolls, biscuit . . . . .</cell>
<cell align="right">10 to 20</cell>
<cell align="center">"</cell>
</row>
<row>
<cell>Graham gems . . . . .</cell>
<cell align="right">30</cell>
<cell align="center">"</cell>
</row>
<row>
<cell>Gingerbread . . . . .</cell>
<cell align="right">20 to 30</cell>
<cell align="center">"</cell>
</row>
<row>
<cell>Sponge cake . . . . .</cell>
<cell align="right">45 to 60</cell>
<cell align="center">"</cell>
</row>
<row>
<cell>Plain &nbsp; " &nbsp; " . . . .</cell>
<cell align="right">30 to 40</cell>
<cell align="center">"</cell>
</row>
<row>
<cell>Fruit &nbsp; " &nbsp; " . . . .</cell>
<cell align="right">2 to 3</cell>
<cell align="center">hrs.</cell>
</row>
</table>

```

That being an example of the simplest type of table you will come across, here's an example of the most complicated type of table you will see in our collection:

HINDQUARTERS 1, 2, 3, 4, 5	Full Round 1, 2, 3	1—Leg or Shin. 2—Round: divided into Top, Bottom and Leg Bone. 3—Flank: of mixed fat and lean, containing Flank Steak nested in "cod" fat.
	"Full" or "Hip and," Loin 4, 5	4—Short Loin: including Sirloin and, on under-side, part of Filet or Tenderloin. 5—Full Hip: divided into Short Hip (containing the large end of Filet) and Top Sirloin (or "Butt").

Charts like this do occasionally appear in the cookbooks, and the best way we have to render them in text is with the table coding. Unfortunately, it's a little hard to tell where to divide up the columns, rows, and cells in tables like these. This is where I would especially recommend drawing in lines yourself, like this:

HINDQUARTERS 1, 2, 3, 4, 5	Full Round 1, 2, 3	1—Leg or Shin.
		2—Round: divided into Top, Bottom and Leg Bone.
		3—Flank: of mixed fat and lean, containing Flank Steak nested in "cod" fat.
	"Full" or "Hip and," Loin 4, 5	4—Short Loin: including Sirloin and, on under-side, part of Filet or Tenderloin.
		5—Full Hip: divided into Short Hip (containing the large end of Filet) and Top Sirloin (or "Butt").

This should help you code the table accordingly. If you do ever come across a chart which seems too complicated to code, ask Ruth Ann for her opinion; she may allow you to code it as an illustration (but be sure to ask first).

A quick side note: Elongated curly brackets, such as those in the table above, cannot be accurately rendered in our cookbook tagset. It is better to render *something* in their place, however, than to just leave them out. I usually type a single curly bracket in each cell crossed by the elongated bracket when I code, as you may deduce from the table diagram above. You may try other methods if you like—just try to visualize the way the table will look when displayed in a web browser.

Also: to create a blank cell, please insert a non-breaking space character (code &nbsp;) in between the two cell tags.

Tables and lists can be very similar in appearance; if you're ever in doubt as to whether you should code something as a table or as a list, just use your best judgment. If it is crucial that the data line up perfectly, then you should probably code the text as a table; if not, either one would be fine.

Since the typists haven't been given instructions on how to type tables, you may have to clean up the text a bit (or even type the data in yourself) before you begin coding. I recommend arranging the data in a way that would facilitate cutting and pasting, since this speeds up the process a great deal.

If you have a table that is printed in two columns—not a table *with* two columns, I mean a table that has been divided into two columns (like an index) to save printing space on a page—you may either code it exactly as it appears on the page, or move it all into one column. Either way is acceptable, but do not delete any text (including column headings, even if they're redundant) in doing so. If the table would be extremely wide if coded as it appears on the page, I would recommend



moving it into one column to be sure it will fit the page width on our website (which, if you take a look, isn't very wide at all).

## CHECKING FOR CURLY BRACKETS

When you reach the end of your first run through a book, it would be a good idea to search the entire document for curly brackets (`{` or `}`). Since the typists use these to write notes to the coder throughout the book (and these are not a part of the original book), any text inside curly brackets should be deleted before the book is published online. A search for curly brackets at the end of the coding process will make sure you've caught and addressed all of the notes from the typists.

## THE VALIDATION PROCESS

Once you have finished coding the entire book, you will need to check to see if the file validates—that is, you will use XMetaL's automatic rules checking function to make sure the code you have put in conforms to the rules laid out in the cookbook DTD. To begin the validation process, click on the button that looks like a piece of paper with a large blue checkmark superimposed on it, or choose "Validate Document" from the "Tools" menu. XMetaL will bring up a window that lists all of the errors in the document. Double click on the first item in the list; XMetaL will move your cursor to the place in the document where the error occurred. It is then up to you to determine what the source of the problem is. If you look carefully at the description XMetaL has provided in the validation box, it will probably give you a hint at what has gone wrong; for example, if you get an error that says "<tag> is not allowed at this point within its container <tag>," you probably missed an ending tag, or did not nest your elements correctly (a common error of this type would be forgetting to include the <purpose> of a <recipe> in the first <p> tag of the recipe—remember, everything within a recipe *must* be inside <p> tags).

Once you spot and correct the error, close the validation box and then reopen it again before you click on the next error; you will find the contents of the box may change significantly with each error that you correct, so take them one at a time. Fixing one problem may solve a whole series of others.

If you ever have difficulty understanding the source of an error, you may ask Ruth Ann or one of the other coders for help; however, I highly recommend you take the time to solve the errors on your own. Finding your own mistakes is one of the best ways to learn the coding process and come to understand the cookbook tagset. It may be a little confusing, but hang in there!

## CHANGING THE DTD

If, while coding, you come across a place in a book that simply cannot be accurately represented by the codes we have defined and the rules we have set for them, consult with Ruth Ann. It is possible to change the rules to accommodate the contents of the books, but Ruth Ann must be the one to make the final decision as to whether this is necessary. In most cases, you will probably find a way to work around the problem without changing the DTD; but don't be afraid to ask if you feel a change is necessary.

## CODING INGREDIENTS

Once you have completed the first stage of coding on a cookbook and completed the validation process, please tell Elizabeth that you are ready to begin ingredient coding on your book. She will run the ingredients perl script on the book, then leave a copy of the file with marked-up ingredients in K:/cookery/xml in progress/, named [bookcode]2.xml ([bookcode] being the four-letter code for the book). When she tells you the file is ready, you can open the new file in XMetaL and begin scanning for ingredients.

When you open the marked-up file, everything the perl script identified as an ingredient will be marked with `<i>` tags (`<i>`, `</i>`). These are an artificial tag we have created to facilitate scanning for ingredients; if we put the full tag in right away (`<ingredient>`, `</ingredient>`), the tags would take up so much space on the screen that it would be difficult to try to look past them. The only problem with using the shorter `<i>` tags, then, is that the document won't validate while the `<i>` tags are still present. Don't worry about this for the moment—just run a search to replace `<i>` with `<ingredient>` and `</i>` with `</ingredient>` when you finish scanning. After that, you can check for validation a second time.

So when you begin scanning, the first thing you should do is search for the first `<i>` in the document. Theoretically, it should be inside the first `<recipe>` or `<formula>` of the book. We do not want `<ingredient>` tags to appear anywhere except inside of formulas and recipes, so if the first `<i>` tag is stuck in the middle of the frontmatter, you will have to delete it (and all other tags that appear outside of `<recipe>` and `<formula>` tags, if you come across more). If it is inside the first `<recipe>` or `<formula>`, you're good to go.

There are three main types of errors generated by the perl script that you will have to watch out for when ingredient scanning. First, you will have to make sure that every item that is used as an ingredient in any given recipe is coded with `<i>` tags *at least once* (it doesn't matter if it's tagged more than once—it would be a waste of your time to try to remove all the extra tagging). The perl script is fairly comprehensive, but it doesn't catch all the terms that could possibly come up as ingredients in all of the recipes and formulas.

Second, you will need to make sure that nothing that *isn't* an ingredient in any particular recipe has `<i>` tags around it. Due to some quirks in the perl script, certain words which would never be considered ingredients in any of the recipes may come up tagged—such as "`<i>hard</i>`" and "`<i>corner</i>`". You will need to remove these extra tags. Also, only words that are being used as ingredients *in that recipe* should have `<i>` tags around them—take out `<i>` tags that are inserted around words that could be ingredients in other recipes, but are not ingredients in the recipe you are looking at. For a common example, if you see the phrase "`<i>butter</i> the size of an <i>egg</i>" when scanning for ingredients, you should take out the <i> tags around "egg," which is a measurement in this case, not an ingredient. "<i>Cream</i> the <i>butter</i> and <i>sugar</i>" is another common example—in this example, take away the tags around "Cream," which is a process in this context. Also, words inside the <purpose> of a recipe should not be tagged as ingredients of the recipe—the perl script should have skipped all <purpose> tags automatically, but the process isn't perfect, so you may have to remove some <i> tags from <purpose> as well.`

Third, you will need to watch for spots where the perl script neglected to paste in an end tag (`</i>`), which happens quite often around vegetables in particular. All you need to do to fix this error is paste in the end tag where appropriate. Along the same lines, the perl script may occasionally double-tag a word, or insert two beginning or ending tags in a row; in this case, just clear out the extra tags.

There will probably be times when you question whether a certain item should be considered an ingredient in a recipe or not. For example, if you have to boil a vegetable in water but end up draining all the water out before you continue with the recipe, is "water" really an ingredient in the end product of the recipe? If you use salt and ice to freeze ice cream, but neither substance goes in the ice cream mixture itself, are they ingredients? What about garnishes, sauces, or side dishes that are mentioned in a recipe for roast beef?

The answer to all these questions is "when in doubt, tag it." As a reference librarian, Ruth Ann will tell you that having too much information is always better than having not enough—so even if we're not using the classic definition of the word "ingredient" in this project, at least we're being thorough (and hopefully visitors to our website will be able to find the information they're looking for). I have been tagging anything that is used in the preparation and eventual serving of a dish as an ingredient in the recipe for that dish. When you are scanning for ingredients, please do the same.

While you are going through the book to check the ingredient tagging, it is a good idea to watch for any minor mistakes you may have made in the coding up until this point—missed recipe attributes, variations you didn't notice your first time through, and the like. If you notice anything you left out or forgot, add it now. You won't get a chance to look at the book in so much detail again, so if you don't catch a mistake at this point, you probably won't catch it at all.

## THE TERMS SPREADSHEET

One of the features we would like to add to our finished cookbook website will be a glossary of obscure or outdated cooking terms—ingredients, processes, measurements, and the like that would probably be unfamiliar (and maybe even incomprehensible) to a modern reader. Peter Berg, the head of Special Collections, will be in charge of selecting and defining these terms; and at the beginning of the project, he was in charge of scanning the books and looking for them as well. However, since the coders at the DMC end up practically reading the book at the ingredient stage of coding anyway, we offered to find obscure terms for him and keep track of them in a spreadsheet, which he can use as a resource to pull possible glossary entries from.

Each coder, then, has his or her own copy of a terms spreadsheet—named "terms[coder's name].xls," located in K:/cooking/spreadsheets/. Each of these spreadsheets is a copy of a spreadsheet that I started at the beginning of the project; that spreadsheet is in the same folder, titled "cookbookterms.xls." If you don't have your own copy of the terms spreadsheet yet, take the following steps to make one:

- 1) Open the K:/cooking/spreadsheets/ folder, either through Explore or My Computer.
- 2) Copy the Microsoft Excel spreadsheet "cookbookterms."
- 3) Paste the copy into the same folder. The new file will be titled "Copy of cookbookterms."
- 4) Rename the file "terms[Yourname]". The file is now ready for use.

If you take a look at your terms spreadsheet, you will notice that it is divided into five columns, titled "obscure or antiquated terms," "Artifacts," "Processes," "Measurements," and "Delete?" Each of these columns contains a list of terms sorted according to the nature of the term. As you scan for ingredients, you will be adding to the list whenever you come across a term that you don't recognize, or that you feel the average modern reader would not recognize.

The "obscure or antiquated terms" column is for obscure ingredients, as well as any other miscellaneous old words that a modern reader might not recognize ("apothecary," for example). "Artifacts" is for actual tools and kitchen implements, which may be mentioned either inside or outside the recipe portions of the book. "Processes" is for any cooking processes that may be unfamiliar to someone who is not an experienced chef (see the terms already in that column for examples). "Measurements" is for obscure measurement terms which would be either unfamiliar or just plain not useful to a modern reader ("as much as will lie on a sixpence" is one of my favorites). Finally, the "Delete?" column is for varieties of fruits and vegetables (such as "Bartlett Pears" and "Bellflower Apples"), as well as brand names ("Magic Yeast" or "St. Louis Flour"). (The reason the column is titled "Delete?" is because I doubt we will feel the need to define any of the terms in this column at the end of the project—but I thought I'd record them anyway, just in case.)

When you add a term to any of these lists (and you don't have to pay attention to whether the term is already in the list—we'll be filtering out duplicate entries at the end), please type the word, followed by the code of the book you found it in and the page number on which you found it in parentheses (see current entries for examples). We're keeping track of the book and page number so that Peter has a reference to check the context of the word when he is writing definitions. You may add the term in any blank space in the appropriate column—you don't have to place it neatly at the bottom of the current columns, since we'll just be alphabetizing the columns at the end anyway, and that will sort out any blank spaces.

One last note concerning the spreadsheet, and this is **very important**—Whenever you add a term to the "Artifacts" category, please write the word down on a separate piece of paper and give the list to Elizabeth when you complete the ingredient stage of coding. Any new terms in this category will be added to the implements perl script, which will need to be run on your book before you scan for implements (which is the last and easiest stage of the coding, yay!). To ensure that we catch all of the implements in each book, we want to add these terms to the perl script before we run it on your book—so don't forget to make this list!

## IMPLEMENTS CODING

Once you finish scanning a book for ingredients and have re-validated the document, bring the list of new "Artifacts" you have found in your book to Elizabeth and let her know the file is ready for implement coding. Elizabeth will update the perl script to make sure your artifacts are tagged, then run the perl script on the file. She will let you know when the implement coding is ready to be checked. The new file name will be [bookcode]3.xml, and it should be in DmcWork1:/cooking/xml in progress/.

Checking the implements tagging isn't nearly as time-consuming as checking ingredients tagging; there simply aren't as many implements in the cookbooks as there are ingredients, especially since we're only bothering to code implements that might be unfamiliar to a modern reader (we don't have to code words like "bowl," "pan," "spoon," and the like). Therefore, when you check the implement coding, all you have to do is search for the implement tag (shortened to <im>), checking each place the computer has tagged a word, then taking out or leaving in the tags as necessary. Unlike the <ingredient> tag, <implement> can appear anywhere in the book (as long as the document validates)—so you don't need to confine the implement coding to recipes and formulas.

The implements script is fairly accurate because it is not nearly as expansive as the ingredients script, but it will tag certain words which *can* be implements but usually aren't; "press," "slice," "baker," and "waiter" come to mind. For words like these, you may have to read some of the context to determine whether it is actually being used as an implement or not. You will also need to be careful about words that are usually implements, but may be used as measurements as well—such as "tumbler"

or "wine glass." If these are being used as measurements, we don't want them marked with `<implement>` tags.

After you have searched the entire document and looked at every `<im>` tag, run another search and replace the string `<im>` with `<implement>` and `</im>` with `</implement>`. After you check for validation and correct any errors that come up, you're finished!

## FINISHING UP

All you have to do after you finish the implement coding is record on the cookbook progress sheet (K:/cooking/spreadsheets/cookbookprogresssheet.xml) that you have finished coding the book by adding the date after your name; let Elizabeth know that you have finished the coding, and give her your tracking sheet; and file the typing copy **in alphabetical order** in the file cabinet with the other finished books (Stephanie can show you where this is if you don't know). That's it—you're ready to begin another coding project!

## TROUBLESHOOTING

If, at any stage of the coding process, you come across something you cannot correct—for example, a cut off page image, a word in another language (such as Greek or Hebrew) that you cannot transcribe, or information missing from the metadata—it is okay to leave those parts of the book uncoded for the time being, but you should do two things to be sure we don't forget to fix the problem later:

- 1) Leave yourself a note in comment tags (`<!-- -->`) in the xml document to remind yourself where the error has occurred, and what needs to be done to fix it.
- 2) Tell Elizabeth about the error so that she can document it—at the end of the project, you will be finding solutions to all the documented errors that have not yet been fixed in the project. (If the problem is a cut off page image, be sure to tell Stephanie about it also so she can arrange for the image to be rescanned as soon as possible.)

Most questions that you have about general coding should be easy to address if you ask Elizabeth, Ruth Ann, or one of the other coders for advice. However, if there is something that no one knows how to fix immediately, be sure you remember to document the problem—otherwise these errors will slip through the cracks.

## EXAMPLES OF CODED RECIPES

The following pages contain a few examples of recipes that I have pulled from the books I've coded, showing a few of the things that you might run across while coding. These are by no means representative of *everything* you will find in the cookbooks you code; but hopefully they will give you a good overview, as well as some concrete examples of some of the tags that might be a little confusing. The sample code beneath each recipe is marked with structural tags only; ingredients and implements have not been tagged in these recipes yet. Important structural tags are in bold.

*A basic recipe (no heading)*

*Brain Sauce.* — Clean the brains, remove the red membrane, and soak in cold water. Put them into *one pint* of *cold water* with *one tablespoonful* of *lemon juice* and *half a teaspoonful* of *salt*. Boil ten minutes; then plunge into cold water. Make *one pint* of *drawn butter sauce*; flavor with *lemon* and *parsley*; add the brains chopped fine, and when hot serve.

```
<recipe class1="accompaniments"><p><purpose rend="italic">Brain
Sauce.</purpose>--Clean the brains, remove the red membrane, and soak
in cold water. Put them into <emph rend="italic">one pint</emph> of
<emph rend="italic">cold water</emph> with <emph rend="italic">one
tablespoonful</emph> of <emph rend="italic">lemon juice</emph> and
<emph rend="italic">half a teaspoonful</emph> of <emph
rend="italic">salt.</emph> Boil ten minutes; then plunge into cold
water. Make <emph rend="italic">one pint</emph> of <emph
rend="italic">drawn butter sauce;</emph> flavor with <emph
rend="italic">lemon</emph> and <emph rend="italic">parsley;</emph> add
the brains chopped fine, and when hot serve.</p></recipe>
```

*A basic recipe (with a heading)*

### **To Stew Sweet Breads.**

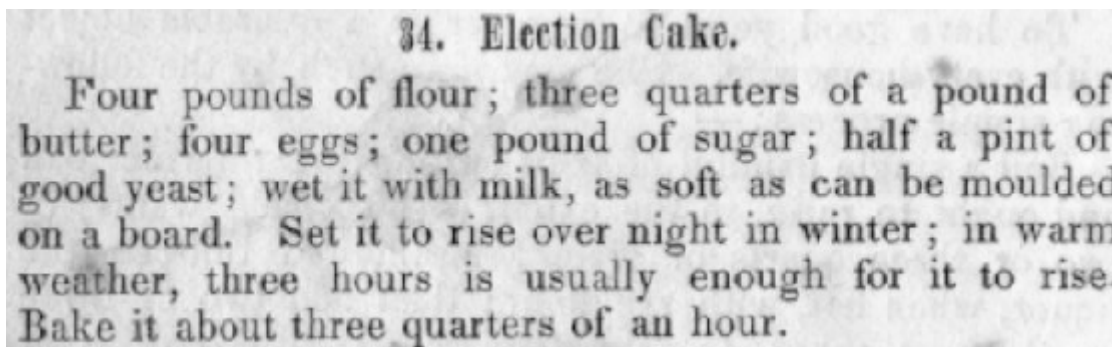
Stew them in a little water, with butter, flour, and a little cream; season with salt, pepper, parsley and thyme.

```
<recipe class1="meatfishgame"><p><purpose rend="bold" align="center"
placement="heading">To Stew Sweet Breads.</purpose>
```

```
Stew them in a little water, with butter, flour, and a little cream;
season with salt, pepper, parsley and thyme.</p></recipe>
```

*\*Notice that the first <p> tag in the recipe above appears between the <recipe> and <purpose> tags—the placement="heading" tag sets the purpose off from the rest of the text in the same <p> tag.*

*A "patriotic" recipe from a regional cookbook with an index that links to recipe numbers, not page numbers*



```
<recipe region="northeast" class1="breadsweets" id="a034"
occasion="patriotic"><p><purpose align="center"
placement="heading">34. Election Cake.</purpose>
```

```
Four pounds of flour; three quarters of a pound of butter; four eggs;
one pound of sugar; half a pint of good yeast; wet it with milk, as
soft as can be moulded on a board. Set it to rise over night in
winter; in warm weather, three hours is usually enough for it to rise.
Bake it about three quarters of an hour.</p></recipe>
```

*A recipe from a regional cookbook with a common type of variation*

#### CORN PUDDING.

Let the corn be very young and tender; scrape from the cob about a quart; put it in a quart of milk, three eggs, a few grains of salt, and a small tea-cupful of sugar; beat it up well, and let it bake slowly for two hours.

Tomatoes are very good, cooked the same way.



```
<recipe region="west" class1="eggscheesedairy"><p><purpose
align="center" placement="heading">CORN PUDDING.</purpose>
```

Let the corn be very young and tender; scrape from the cob about a quart; put it in a quart of milk, three eggs, a few grains of salt, and a small tea-cupful of sugar; beat it up well, and let it bake slowly for two hours.</p>

```
<p><variation><purpose><alt synonym1="tomato
pudding">Tomatoes</alt></purpose> are very good, cooked the same
way.</variation></p></recipe>
```

*A "medhealth" recipe with a <contributor> and <attribution>*

### Invalid's Gingerbread.

[From Mrs. Cornelius' YOUNG HOUSEKEEPERS' FRIEND.]

One pint of molasses, one cupful sugar, one teaspoonful of soda, one teaspoonful of ginger, one of salt, with flour enough to roll out very thin. It is improved by keeping a week or two.

EMMA M. E. SANBORN, M. D.

```
<recipe class1="breadweets" class2="medhealth"><p
align="center"><purpose rend="bold" placement="heading">Invalid's
Gingerbread.</purpose>
```

```
<attribution>[From Mrs. Cornelius' YOUNG HOUSEKEEPERS'
FRIEND.]</attribution></p>
```

```
<p>One pint of molasses, one cupful sugar, one teaspoonful of soda,
one teaspoonful of ginger, one of salt, with flour enough to roll out
very thin. It is improved by keeping a week or two.</p>
```

```
<p align="right"><contributor>EMMA M. E. SANBORN, M.
D.</contributor></p></recipe>
```

*A recipe in an ethnic cookbook with the ingredients arranged in a list*

198

## **FARINA CAKES**

**(Pasticcini di semolino)**

Farina, six and a half ounces.

Sugar, three and a half ounces.

Pine-seeds, two ounces.

Butter, a small piece.

Milk, one quart.

Four eggs.

A pinch of salt.

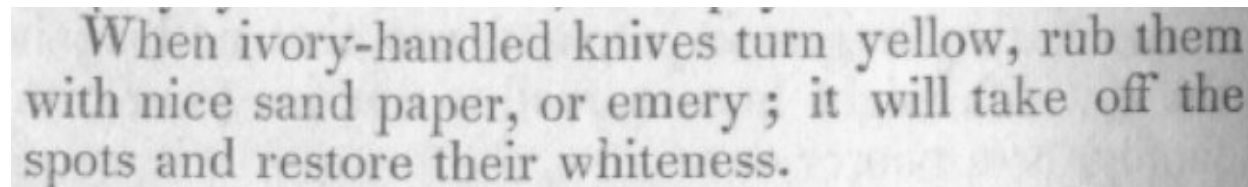
Taste of lemon peel.

Cook the farina in the milk and when it begins to thicken pour the pine-seeds, previously chopped fine and pounded with the sugar, then the butter and the rest, less the eggs which must be put in last when the mixture has completely cooled. Then place the whole well mixed in little molds, greased evenly with butter and sprinkled with bread crumbs ground fine, and bake.

```
<recipe ethnicgroup="italian" class1="breadsweets"
id="r198"><p><purpose align="center" rend="bold"
placement="heading">198<lb/>
FARINA CAKES<lb/>
(Pasticcini di semolino)</purpose>
<list>
<item>Farina, six and a half ounces.</item>
<item>Sugar, three and a half ounces.</item>
<item>Pine-seeds, two ounces.</item>
<item>Butter, a small piece.</item>
<item>Milk, one quart.</item>
<item>Four eggs.</item>
<item>A pinch of salt.</item>
<item>Taste of lemon peel.</item></list></p>
```

**<p>**Cook the farina in the milk and when it begins to thicken pour the pine-seeds, previously chopped fine and pounded with the sugar, then the butter and the rest, less the eggs which must be put in last when the mixture has completely cooled. Then place the whole well mixed in little molds, greased evenly with butter and sprinkled with bread crumbs ground fine, and bake.**</p></recipe>**

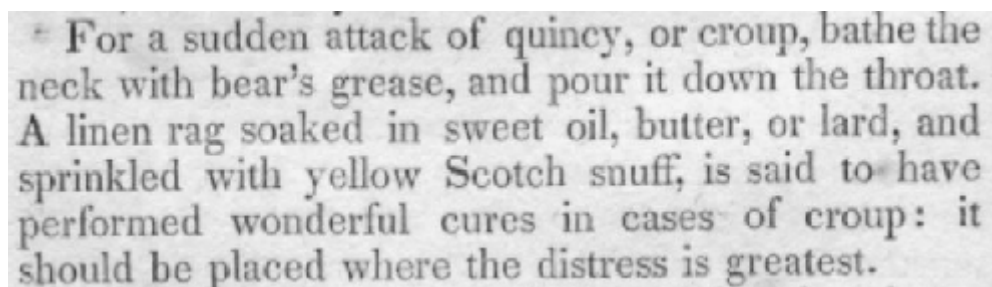
*A "household" formula that is not set off from the surrounding text (watch for these, particularly in chapters on household management)*



When ivory-handled knives turn yellow, rub them with nice sand paper, or emery ; it will take off the spots and restore their whiteness.

**<formula class="household"><p><purpose>**When ivory-handled knives turn yellow,**</purpose>** rub them with nice sand paper, or emery; it will take off the spots and restore their whiteness.**</p></formula>**

*A "medhealth" formula (also not set off) with a variation*



\* For a sudden attack of quincy, or croup, bathe the neck with bear's grease, and pour it down the throat. A linen rag soaked in sweet oil, butter, or lard, and sprinkled with yellow Scotch snuff, is said to have performed wonderful cures in cases of croup: it should be placed where the distress is greatest.

**<formula class="medhealth"><p>**For a sudden attack of **<purpose>**quincy, or croup,**</purpose>** bathe the neck with bear's grease, and pour it down the throat. **<variation>**A linen rag soaked in sweet oil, butter, or lard, and sprinkled with yellow Scotch snuff, is said to have performed wonderful cures in cases of croup: it should be placed where the distress is greatest.**</variation></p></formula>**