openrecipeformat Documentation

Release latest

Jan 04, 2021

CONTENTS

The Open Recipe Format was designed to be as simple as possible for basic applications, while also being flexible enough to handle various complexities that may be encountered in production environments. This guide outlines the basic file format, what needs to be defined at a bare minimum, and how to put it all together.

At its heart, the Open Recipe Format (ORF) uses an established file format called YAML. It's not a difficult format to get used to, but using a syntax checker will save you from a lot of headaches. An excellent online resource can be found at *http://yaml-online-parser.appspot.com/*.

At its most basic level, a recipe consists of a name, a list of ingredients, and a list of steps to put those ingredients together. At this point, the most complex of these is the ingredient list. Each item in the list consists of an amount, a unit (which is often implied, in traditional recipes), and a corresponding ingredient.

Let's put together a very basic recipe for giving an apple to a friend. In a non-computer parsable format, we might say, "Give an apple to a friend." To make this easier for the computer to use, we need to be more specific:

```
recipe_name: Giving an Apple to a Friend
ingredients:
    - apple:
        amounts:
            - amount: 1
              unit: each
steps:
    - step: Give an apple to a friend.
```

This is as basic as a recipe gets. Because we are striving to be specific, there is nothing implied in this recipe; if we were writing it out traditionally, we would say *1 each apple* instead of *1 apple*. This may feel awkward to some, and developers may choose to allow their users to say *1 apple* in their software, but when they save it to an ORF file, it needs to be more specific.

Let's expand this recipe, to include more fruit, and more steps for handling the fruit. This recipe is for a very basic fruit salad, which is unlikely to win any awards, but will still feed your friends.

```
recipe_name: Basic Fruit Salad
yields:
    - servings: 6
ingredients:
    - apple:
        amounts:
            - amount: 1
              unit: each
    - banana:
        amounts:
            - amount: 1
              unit: each
    - orange:
        amounts:
            - amount: 1
              unit: each
    - grapes:
        amounts:
            - amount: 1
              unit: cup
steps:
    - step: Cut the apple into cubes.
    - step: Cut the banana into slices.
    - step: Peel the orange, and divide into segments.
    - step: Combine all ingredients in a bowl.
    - step: Mix to combine.
```

This recipe is fine for 6 people, but what if you need to feed 18 people? An important consideration that most recipe software handles is scaling recipes. However, many commercial environments require multiple scalings to be available in a tidy, concise format. In our case, we would make the following changes to our recipe to handle both at the same time:

```
recipe_name: Basic Fruit Salad
yields:
    - servings: 6
    - servings: 18
ingredients:
    - apple:
        amounts:
            - amount: 1
              unit: each
            - amount: 3
              unit: each
    - banana:
        amounts:
            - amount: 1
              unit: each
            - amount: 3
              unit: each
    - orange:
        amounts:
            - amount: 1
              unit: each
            - amount: 3
              unit: each
    - grapes:
        amounts:
            - amount: 1
              unit: cup
            - amount: 3
              unit: cup
```

A classic use case for this scenario is a bakery, which may need to bake batches of 50, 100, 250, or even more cookies at a time, depending on the occassion. Unfortunately, ingredients are the product of nature, and not all recipes scale as nicely as we would like. It may be that in the case of our bakery, 50 cookies call for 1/2 tablespoon of baking soda, but 250 cookies only calls for 2 tablespoons. It is up to the baker to discover this, but when they do, it will be aggrivating to them to not be able to make note of that in their software.

Speaking of notes, many professional cooks and bakers like to make bench notes when experimenting with recipes. These items of information do not necessarily fit into the classic "steps" format. Fortunately, ORF allows for "notes" to be saved, both alongside specific steps, and outside of the steps. Let's go back to our first recipe, and add some notes.

```
recipe_name: Giving an Apple to a Friend
ingredients:
    - apple:
        amounts:
            - amount: 1
              unit: each
        notes:
            - Use whole apples
            - Pears may be substituted, but produce a different flavor and mouthfeel
steps:
    - step: Give an apple to a friend.
```

(continues on next page)

(continued from previous page)

```
      notes:
        - You can also give an apple to an enemy.
notes:
    - This is a friendly recipe; giving, rather than throwing, is recommended.
```

This recipe contains notes for individual ingredients and steps, as well as for the recipe as a whole. While it is not expected that most cooks, even in the professional kitchen, will take advantage of this, it is fully expected that food scientists will have the opportunity to do so.

Food scientists have a whole new set of information that they need to manage. Precision is key for these individuals, and ORF is designed to handle features that are key to their job success. Consider the following:

```
ingredients:
    - apple:
        usda_num: 09003
        amounts:
            - amount: 1
              unit: each
        processing:
            - whole
            - raw
        substitutions:
            - pears:
                usda_num: 09252
                amounts:
                    - amount: 1
                      unit: each
        notes:
            - Use whole apples
            - Pears may be substituted, but produce a different flavor and mouthfeel
steps:
    - step:
          Gather the apples.
      haccp:
          control_point: The apples must be clean
      notes:
          - Some people like green
          - Some people like red
    - step:
          Hand out the apples.
      haccp:
          critical_control_point: Wash hands with soap and warm water before␣
↪distributing.
```

This set of ingredients and steps includes a scary amount of detail, for the average layperson. However, it paves the way for the industrialization and safety of this recipe, allowing the user to specify keywords related to processing, and HACCP (Hazard Analysis Critical Control Points) instructions to be declared. References to the USDA Standard Reference are also included, allowing the recipe to make use of nutritional data provided by the USDA.

Now that you have been through a brief overview of the format and its capabilities, the other tutorials and references will make much more sense.

CHAPTER

ONE

OPEN RECIPE FORMAT REFERENCE

1.1 oven_fan

Setting to be used with convection oven. Possible values are “Off”, “Low” and “High”. If not specified, it is assumed to be “Off”. If specified, all software should display and print this value. If not specified, it is up to the software whether or not it is displayed and/or printed, but it should be consistent.

1.2 oven_temp

Starting oven temperature, if the oven is used.

1.2.1 amount

The number of degrees.

1.2.2 unit

Should be C or F.

1.3 oven_time

How long the dish should spend in the oven. This is an overall value, which refers to the recipe as a whole. If multiple oven times are used, they should be specified in the recipe.

1.4 ingredients

A list of dicts, defining which food items are to be added to the recipe. These items should be listed in the order in which they are to be used. Bearing this in mind, a particular item may be listed multiple times, if it is to be used multiple times and/or at different quantities in a recipe.

To be clear, it is preferable to list “1 1/2 cups of sugar” and then “1/2 cup of sugar” (as specified below) than to list “2 cups sugar, divided”.

1.4.1 ingredient

A dict of items, describing an ingredient, and how much of that ingredient to use.

1.4.1.1 amounts

A list of dicts which describe the amounts to use. Normally, the list will only contain one dict. In cases where multiple yields need to be stored (i.e. 50 cookies vs 100 cookes vs 250 cookies), each yield will have its own dict in this list, in the same order as the recipe's yield field.

amount

The amount of the `unit` to use.

unit

The unit, relevant to the `amount`.

1.4.1.2 processing

A list of tags which describe the processing of this item. For instance, "whole", "large dice", "minced", "raw", "steamed", etc.

1.4.1.3 notes

Any notes specific to this ingredient.

1.4.1.4 substitutions

This field is a list of ingredients, in exactly the same format as a regular ingredient list item, minus the `substitutions` field. For instance, it must contain `amounts`, and may also contain `processing`, `usda_num`, `notes`, etc.

1.4.1.5 usda_num

This corresponds with the index keys in the USDA Standard Reference. It is generally used for easy lookup of nutritional data. If possible, this should be used, and USDA data, when available, is preferable to any other nutritional data source.

1.5 notes

This is a field that will appear in several locations. The recipe itself may have noted, each ingredient may have notes, and each step may have notes.

1.6 recipe_name

The name of this recipe.

1.7 recipe_uuid

This functions somewhat like a network MAC address. It needs to contain an identifier for the company or the software package, and then a unique identifier for the recipe itself. This is easiest to handle when a recipe is managed either by a website (which likely has its own internal primary keys) or a piece of commercial software that has been registered to a user, using a registration key.

Ideally, a central source should be set up for companies to register their unique company identifier, to avoid UUID collisions.

1.8 source_book

If this recipe was originally pulled from a book, then the book information should go here. Recipe software should make an intelligent effort to include correct information in the correct fields, rather than just dumping everything into a generic notes field.

1.8.1 authors

This is a list. Refers to the author(s) of this recipe. Can be the same as `source_authors`, if appropriate. If there was only one author, then they would be the only item in the list.

1.8.2 title

Title of the book. This is a single value, not a list.

1.8.3 isbn

International Standard Book Number, if available.

1.8.4 notes

Any information about the book that does not fit into another field.

1.8.5 X-<field>

A lot of different information about a book can be stored. Until a field has been officially accepted into the spec, it should start with a capital X, followed by a dash.

1.9 source_authors

Does not refer to the person who entered the recipe; only refers to the original author of the recipe. If this recipe was based on another recipe by another person, then this field should contain the name of the original author.

1.10 source_url

The URL that this recipe was copied from, if applicable. In the case of a recipe-hosting website, this may refer to the official URL at which the recipe is hosted.

1.11 steps

A list, in order, of steps to be performed on the recipe. Each item in the list is a dict, as specified below.

1.11.1 step

The only item in the dict that is absolutely required.

1.11.2 haccp

A dict, which can contain either a `control_point` or a `critical_control_point`. Should not contain both.

1.11.2.1 control_point

Refers to specific HACCP guidelines relevant to this step.

1.11.2.2 critical_control_point

Refers to specific HACCP guidelines relevant to this step, which are critical to the safety outcome of this recipe. For instance, "Cook until the food reaches an internal temperature of 165F."

1.11.3 notes

A list of notes relevant to this step. Often known as "bench notes" to professionals.

1.12 yields

Refers to how much food the recipe makes. This is a list, which will normally contain one dict. In cases where multiple yields need to be stored (i.e. 50 cookies vs 100 cookes vs 250 cookies), each yield will have its own dict in this list.

1.12.1 amount

The amount, relevant to the `unit`.

1.12.2 unit

Generally "servings", but up to the user. Can be "packages", "cups", "glasses", etc.

1.13 X-<field>

A lot of different information about a recipe can be stored. Until a field has been officially accepted into the spec, it should start with a capital X, followed by a dash.

CHAPTER

TWO

OPEN NUTRITION FORMAT REFERENCE

2.1 usda_num

Official record number matching an item in the USDA Standard Reference.

2.2 usda_name

If `usda_num` is specified, this name may or may not match the actual name in the database.

2.3 amount

2.4 unit

2.5 proximates

2.5.1 water

nutrient_no: 255

2.5.2 energy

nutrient_no: 208

2.5.3 protein

nutrient_no: 203

2.5.4 lipid_total

nutrient_no: 204

2.5.5 ash

nutrient_no: 207

2.5.6 carbohydrate

nutrient_no: 205

2.5.7 fiber_total

nutrient_no: 291

2.5.8 sugars_total

nutrient_no: 269

2.5.9 sucrose

nutrient_no: 210

2.5.10 glucose

nutrient_no: 211

2.5.11 fructose

nutrient_no: 212

2.5.12 lactose

nutrient_no: 213

2.5.13 maltose

nutrient_no: 214

2.5.14 galactose

nutrient_no: 287

2.5.15 starch

nutrient_no: 209

2.6 minerals

2.6.1 calcium

nutrient_no: 301

2.6.2 iron

nutrient_no: 303

2.6.3 magnesium

nutrient_no: 304

2.6.4 phosphorus

nutrient_no: 305

2.6.5 potassium

nutrient_no: 306

2.6.6 sodium

nutrient_no: 307

2.6.7 zinc

nutrient_no: 309

2.6.8 copper

nutrient_no: 312

2.6.9 manganese

nutrient_no: 315

2.6.10 selenium

nutrient_no: 317

2.6.11 flouride

nutrient_no: 313

2.7 vitamins

2.7.1 vitamin_c

nutrient_no: 401

2.7.2 thiamin

nutrient_no: 404

2.7.3 riboflavin

nutrient_no: 405

2.7.4 niacin

nutrient_no: 406

2.7.5 pantothenic_acid

nutrient_no: 410

2.7.6 vitamin_b6

nutrient_no: 415

2.7.7 folate_total

nutrient_no: 417

2.7.8 folic_acid

nutrient_no: 431

2.7.9 folate_food

nutrient_no: 432

2.7.10 folate_dfe

nutrient_no: 435

2.7.11 choline_total

nutrient_no: 421

2.7.12 betaine

nutrient_no:454

2.7.13 vitamin_b12

nutrient_no:418

2.7.14 vitamin_b12_added

nutrient_no: 578

2.7.15 vitamin_a_rae

nutrient_no: 320

2.7.16 retinol

nutrient_no: 319

2.7.17 carotene_beta

nutrient_no: 321

2.7.18 carotene_alpha

nutrient_no: 322

2.7.19 cryptoxanthin_beta

nutrient_no: 334

2.7.20 vitamin_a_iu

nutrient_no: 318

2.7.21 lycopene

nutrient_no: 337

2.7.22 lutein_zeaxanthin

nutrient_no: 338

2.7.23 vitamin_e_alpha_tocopherol

nutrient_no: 323

2.7.24 vitamin_e_added

nutrient_no: 573

2.7.25 tocopherol_beta

nutrient_no: 341

2.7.26 tocopherol_gamma

nutrient_no: 342

2.7.27 tocopherol_delta

nutrient_no: 343

2.7.28 vitamin_d2_d3

nutrient_no: 328

2.7.29 vitamin_d_ergocalciferol

nutrient_no: 325

2.7.30 vitamin_d_cholecalciferol

nutrient_no: 326

2.7.31 vitamin_d

nutrient_no: 324

2.7.32 vitamin_k

nutrient_no: 430

2.7.33 menaquinone_4

nutrient_no: 428

2.8 lipids

2.8.1 total_saturated

nutrient_no: 606

4:0 nutrient_no: 607

6:0 nutrient_no: 608

8:0 nutrient_no: 609

10:0 nutrient_no: 610

12:0 nutrient_no: 611

13:0 nutrient_no: 696

14:0 nutrient_no: 612

15:0 nutrient_no: 652

16:0 nutrient_no: 613

17:0 nutrient_no: 653

18:0 nutrient_no: 614

20:0 nutrient_no: 615

22:0 nutrient_no: 624

24:0 nutrient_no: 654

2.8.2 total_monounsaturated

nutrient_no: 645

14:1 nutrient_no: 625

15:1 nutrient_no: 697

16:1 undifferentiated nutrient_no: 626

16:1 nutrient_no: 673

17:1 nutrient_no: 662

18:1 undifferentiated nutrient_no: 617

18:1 c nutrient_no: 674

18:1 1 nutrient_no: 663

18:1-11t (18:1t n-7) nutrient_no: 859

20:1 nutrient_no: 628

22:1 undifferentiated nutrient_no: 630

22:1 c nutrient_no: 676

22:1 t nutrient_no: 664

24:1 c nutrient_no: 671

2.8.3 total_polyunsaturated

nutrient_no: 646

2.8.4 cholesterol

nutrient_no: 601

2.9 other

2.9.1 caffeine

nutrient_no: 262

CHAPTER

THREE

USING ALERTS

The Open Recipe Format allows each step of a recipe to include extra, computer- readable information about that step. For instance, a duration may be stored, such as *25M* (25 minutes). Durations are based on the Unix *date* command, and include:

- S: Seconds
- M: Minutes
- H: Hours
- d: Days
- W: Weeks
- m: Months
- Y: Years

Please note that as with the *date* command, these abbreviations are case- sensitive; *M* refers to minutes, while *m* refers to months.

3.1 Long-Term Durations

The first question in many cooks' minds will be, why worry about anything longer than hours? Bear in mind that this format is intended to be usable in any situation where food is to be prepared. The following are all valid use cases:

- Curing a duck breast prosciutto for several days.
- Dry-curing an Italian sausage for several weeks.
- Allowing vanilla beans to soak in vodka for several months.
- Allowing whiskey to age in barrels for several years.

3.2 Software-Based Alerts

With these considerations in mind, software can be written that helps the cook manage recipes, and alert the cook when an action needs to be taken.

For instance, mobile recipe software can not only display the steps of a recipe, but also allow a cook to press a button when putting a tray of cookies into the oven, which will trigger an alarm 12 minutes later to remove them from the oven. Software designed for cheesemakers can be set up to add an item to a person's calendar, or send an email, when a particular batch of cheese needs to be checked. Such software should assume that multiple batches are being

processed at once, and allow the food producer to store their own internal storage information (bin codes, batch dates, etc) within the software.

3.3 Examples

CHAPTER

FOUR

INTRODUCTION TO GIT

Using a text-based format, especially YAML, provides a powerful platform for recipe management. Recipes are essentially source code, using a human compiler rather than a machine compiler. As such, they can benefit from many of the same tools as computer programming languages. One of these tools that can be especially powerful in kitchen lab environments and recipe development is version control. Unfortunately, most cooks are not used to using version control, especially git.

This document is currently targeted at developers and other technical professionals who are more used to using command line-based software. It assumes that the user has not previously used git, but can follow along with command line instructions. It was written for the Linux environment, but other Unices should not differ too terribly. Another guide, targeted at graphical environments such as Gnome, KDE and Windows, is planned.

4.1 Distributed Revision Control

Classical revision control systems used a centralized repository server, from which a user could check out files, modify them, and check them back in. Some older systems, such as CVS, would maintain version numbers of the repository as a whole; even checking in changes to a single file would update the version of every file in the repository. Some newer systems such as Subversion would maintain individual version numbers on each file. In either case, the central repository server would manage the version numbers, and was the final word on conflicting versions of files that were checked in.

This scenerio was common in the software industry, and tended to work well for small software projects. However, it posed several limitations in larger organizations, which needed to handle multiple, disparate branches of a project in multiple, disparate locations. A distributed solution was needed. Several of these now exist, each with their own strengths. This guide focuses on git, partly because of its current popularity, and partly because of the author's own persional experience.

Git was written by Linus Torvalds, creator of the Linux operating system kernel. It does not inherently assume a central repository, which controls all other checkouts of a project. Each user maintains their own repository, and with it the ability to send patches to another copy of the repository, and merge patches sent from another copy of the repository.

When a repository is copied from another source, it is known as a fork. The process of forking a repository in git uses the *clone* command. Forked repositories can *pull* changes from other forks, or from the repository that they were cloned from. Forks can also send changes upstream to the initial repository using, among other options, a "pull request", in which the initial repository pulls changes from a fork.

Confused yet? Don't worry, we'll take it step by step.

4.2 Initializing a Repo

This guide assumes the use of GitHub, a popular site which maintains Git repositories (known as "repos"), using both free and paid models. There are other stellar hosting options available, but by and large the usage will be the same.

If you haven't already, go to https://github.com/ and create an account. Once you have signed up, you need to create a new repo. In GitHub, the icon to create a new repo is located in the top-right corner of the page, and looks like a book with a plus sign on it. Call your repo "myrecipes", and go ahead and make it public.

Once the repo has been created on GitHub, you need to initialize it on your local machine.

```
localhost> mkdir -p ~/git/myrecipes
localhost> cd ~/git/myrecipes
localhost> git init
Initialized empty Git repository in /home/larry/git/myrecipes/.git/
```

TODO: Push the repository to origin

4.3 Creating a Recipe

We'll create a simple recipe, using your favorite text editor. This recipe will be saved as ~/git/myrecipes/apple.yaml:

```
recipe_name: Giving an Apple to a Friend
ingredients:
    - apple:
        amounts:
            - amount: 1
              unit: each
steps:
    - step: Give an apple to a fiend.
```

You may notice that this recipe contains an inconsistency. Don't worry about it, we'll fix it up in just a moment. For now, we'll save the recipe, and then see what git has to say about it.

```
localhost> git status
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#   apple.yaml
nothing added to commit but untracked files present (use "git add" to track)
```

Git can see that the file is there, but it's not currently tracking any changes to it. Let's add it, and see what git thinks about it.

```
localhost> git add apple.yaml
localhost> git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
```

(continues on next page)

(continued from previous page)

```
#   (use "git rm --cached <file>..." to unstage)
#
#   new file:   apple.yaml
#
```

Git has now been notifed that *apple.yaml* is available to be added to the repo. However, it has not yet been checked in (or "committed", as git calls it), and so git is still not technically tracking changes to it. Let's go ahead and commit it.

```
localhost> git commit -m 'This is my first commit'
[master (root-commit) 1617167] This is my first commit
 1 file changed, 8 insertions(+)
 create mode 100644 apple.yaml
localhost> git status
# On branch master
nothing to commit, working directory clean
```

The *-m* option designates a commit message. This is usually a quick, one-line message giving a brief overview of what changes have occurred between this version and the previous version. With the first commit, it is usually reasonable to just say, "First commit". Any changes after that should be detailed enough that somebody in the future (who may be you) can easily identify when certain changes were made. One way to remind yourself to do this, is to assume that the next person to look at your work is a homocidal axe murderer who knows who you are and where you live.

Git is now officially tracking changes to this file. But as you may have noticed before, there is an error in the recipe. The title of the recipe is, "Giving an Apple to a Friend", but the recipe itself states that the apple is to be given to a fiend. After this typo has been corrected, we can check Git to see how it is tracking our change.

```
localhost> git diff
diff --git a/apple.yaml b/apple.yaml
index 72cd1a1..0ec3011 100644
--- a/apple.yaml
+++ b/apple.yaml
@@ -5,4 +5,4 @@ ingredients:
             - amount: 1
               unit: each
 steps:
-    - step: Give an apple to a fiend.
+    - step: Give an apple to a friend.
```

The *git diff* command shows us the difference between the old version of the file (*a/apple.yaml*) and the new version of the file (*b/apple.yaml*). The format that it uses tells us that any line starting with - shows a line that has been removed from the old version, and any line starting with + is a line that was added to the old version. In this case, the following line:

```
- step: Give an apple to a fiend.
```

Has been changed to this in the new version:

```
- step: Give an apple to a friend.
```

With the change in place, we may now add and commit a new version of this file.

```
localhost> git add apple.yaml
localhost> git commit -m 'Correcting typo: friend, not fiend'
[master 4f59a41] Correcting typo: friend, not fiend
 1 file changed, 1 insertion(+), 1 deletion(-)
```

At this point, two different versions of the file exist in the git repo. This is the important part of revision control: we can go back and look at old versions, and compare how something is done now with how it was done before. This can be especially important when doing recipe testing; somethings it helps to see how a recipe has evolved over time.

Since we've left reasonable commit messages in git, we can go back and see, at a glace, where certain changes were made.

```
localhost> git log
commit 4f59a41a9bc0e06f2858302ce3332d336140ca7f
Author: Larry Fine <larry@stooges.com>
Date:   Sat Jun 22 14:33:29 2013 -0600

    Correcting typo: friend, not fiend

commit 824ba5bc2aa4cb07f33a82c9f1f833debb5fd055
Author: Larry Fine <larry@stooges.com>
Date:   Sat Jun 22 14:22:57 2013 -0600

    This is my first commit
```

You can see that log messages are displayed in reverse order (most recent first). If we had enough entries that this output took up more than a single screen, git would have automatically displayed it using the default paging program on your system.

CHAPTER

FIVE

WRITING RECIPES

A number of years ago, the originator of the ORF composed a number of articles concerning developing and writing recipes. The links to these articles are listed here for reference.

- http://blog.josephhall.com/2007/12/volume-measurements.html
- http://blog.josephhall.com/2007/12/writing-recipes.html
- http://blog.josephhall.com/2007/12/standardizing-recipes.html
- http://blog.josephhall.com/2007/12/haccp.html
- http://blog.josephhall.com/2007/12/adding-haccp-to-recipes.html
- http://blog.josephhall.com/2007/12/same-recipe-multiple-scalings.html
- http://blog.josephhall.com/2007/12/recipes-with-media-content.html
- http://blog.josephhall.com/2008/01/nutritional-information-for-recipes.html
- http://blog.josephhall.com/2008/10/baking-percentages.html
- http://blog.josephhall.com/2008/10/composite-recipe-oatmeal-cookies.html
- http://blog.josephhall.com/2008/10/implementation-recipes.html
- http://blog.josephhall.com/2010/03/interpreting-recipe-input.html

CHAPTER

SIX

OPEN RECIPE FORMAT

In the beginning, there was food. And then there were dishes. And somewhere down the line, people started writing down how to make the dishes. And most of them weren't very good at it. And then they got computers, and they still weren't very good at it.

The Open Recipe Format has been designed to solve some basic needs: accurate and flexible storage of recipes. While there have been attempts in the past to create standardized computer recipe formats, they have so far met with failure. This stems from several problems, all of which stem from a lack of understanding of the tasks at hand (and in most cases, ignorance of which tasks actually need to be addressed).

A significant portion of the population cooks at home, and so most solutions target home cooks. Commercial solutions exist, and address a host of new considerations that are not thought to relate to home cooks. The Open Recipe Format was initially created by a software engineer with a cooking degree and professional kitchen experience. Few, if any, other attempts at writing food software, much less standardized recipe data modeling, can make this claim.

At the heart of the Open Recipe Format is an established file format called YAML. This format was chosen for a variety of reasons:

- It is text-based, and therefore human readable.
- It mimics the internal data structures of several high-level programming languages.
- It can easily be converted to and from other formats, such as the more popular JSON (in fact, JSON is syntactically-correct YAML).
- It is more human readable than JSON.
- It is far more light-weight than other chatty formats, like XML.
- It is far more flexible than legacy formats like EDI.
- As a text-based format, it can be easily managed by revision control software, such as git.

This repository is intended more for programmers than anyone else. As human- readable as the format is, it is likely to be confusing to many in its raw format. The job of the programmer is to use this spec in the development of their own software packages, websites, etc., and provide a friendly interface to the end-user.

Many of the files in this repository are intended to document the Open Recipe Format. Others are examples, to help programmers understand how to best use the format. And finally, sample source code is provided to give programmers a jump start in their coding efforts.

6.1 Getting Started

This walkthrough is made to help individuals get started quickly and gain a foundational knowledge of the Open Recipe Format:

A Basic Recipe

In addition to an Open Recipe Format, an Open Nutrition Format is being established. These two formats are described in more detail here:

Developers References:

- *Open Recipe Format Reference*
- *Open Nutrition Format Reference*