FRUIT Process Documentation

Created 9/24/14 bkv -- Updated 11/5/14 bkv

Table of Contents

Back to TOC

The Big Picture

The Florida Department of Chip Snacks (Dorito) wants a standardized test assessing what the students in grades 3 through 10 have learned. They contract out the creation, administration, and scoring of the test to a company (currently Pear). To ensure accuracy, they contract out a verification process on the test results to an independent, third party (currently Cheerio).

We have four main areas as part of our contract with Dorito: Scoring and Reporting, Demonstration, Retrofitting and School Grades, which encompasses several distinct processes of its own. We will focus right now on Scoring and Reporting. Within Scoring and Reporting, there are two sections. There are the actual FRUIT tests (Raspberry, Melon, Watermelon and Satsuma), and then there is Eggplant (for Apricot, Banana, Cherry, Grape and Honeydew).

Our job, then, is to independently check Pear's work. We achieve this by recreating the scoring and reporting aspects using our own systems and programming language, but with the same supporting documentation and specifications that the primary vendor (Pear) gets. We get these documents from Dorito via e-mail and SFTP server access. We check the files (to ensure that the right type of data is present) and we look for duplicates. We provide feedback to Dorito via e-mail (with certain files stored on an SFTP server).

We do not talk to Pear at all.

Back to TOC

The environment

We use a few programming languages for this task. The assessment-old project folder uses Ruby ver 1.9.3-p545 (as verified by the .ruby-version file) and Python 2.7 (NOT 3.x). The new Assessment project is being rewritten in Objective-C. We use Git for version control and we house our own Git server on premise for this exact task. It is (per the contract), NOT accessible from outside of our network. We also do not run code on machines outside of our network. We also house our own SFTP server (on another machine, other than the Git server), for Dorito access to files mentioned in our feedback.

Back to TOC

The Lifecycle

Fall : Mock Seek Change, Mock Strawberry, Mock Apple, Mock Reports
(only done internally),

Live SeekChange, Live Strawberry, Live Apple, Live Reports, Late Reporting (Strawberry),
(per the contract, only Raspberry/Melon Retake, no Eggplants)

Winter : Mock Seek Change, Mock Strawberry, Mock Apple, Mock Reports
(only done internally),

Live SeekChange, Live Strawberry, Live Apple, Live Reports, Late Reporting (Strawberry), Closeout (Strawberry)

Spring : Mock Seek Change, Mock Strawberry, Mock Apple, Mock Reports
(only done internally),

Live SeekChange, Live Strawberry, Live Apple, Live Reports, Late Reporting (Strawberry), Closeout (Strawberry)
Spring also has Satsuma (5 th and 8th grades).

Summer : Mock Seek Change, Mock Strawberry, Mock Apple, Mock Reports
(only done internally),

Live SeekChange, Live Strawberry, Live Apple, Live Reports, Late Reporting (Strawberry), Closeout (Strawberry)

Back to TOC

The Overall Process

The process starts with Seek and Change files. The paper-based tests are seekned in by Pear and a Seek file is created. That file is then changeed for accuracy and a separate Change file is created. Pear sends a notification to Dorito and to us via e-mail, detailing the files available and the correct path to said files located on their SFTP server. We retrieve the files. The details of how that is done is in a later section (titled "Getting the Data Files").

Once the Seek and Change files are approved by Dorito (after we verify them), Pear creates an Strawberry file (which stands for REDACTED). Pear sends another notification via email for the Strawberry files.

Once the Strawberry files are approved, Pear then appleregates the results into an Apple file (sometimes called an Starfruit file). Then, Pear creates six reports, based on the Strawberry and Apple files.

Lastly, Pear will generate some late reporting. This is the fault of the school districts. This process may occur multiple times, but the checking process differs only slightly from the normal process.

The last thing that Pear will do (for Scoring and Reporting) is where they combine all of the Strawberry's and Late Reporting Strawberry files together (Closeout files) and it is checked one last time. After those files are approved, Dorito asks us for Demonstration files, based upon those closeout files.

Demonstration is a completely separate process and code base from scoring and reporting.

Back to TOC

Cloning our project

1. Open your Terminal window and Finder window.
2. In Finder, create a folder called Projects in your User home directory.
3. In Terminal, type in "git clone USER@SERVER:/home/USER/assessment-old.git"
4. It will prompt you for a password. Find your Team Lead and they will provide it for you if necessary.

Back to TOC

The Setup

Once you clone our project using git, you will notice several folders inside of the assessment-old folder. Each year's administration is contained in a folder. Our current administion is located in the 2014-2015 folder. Within that folder is a single folder, "scoring_reporting". Within that folder are the folders for our software.

They are as follows: /bin, /config, /docs, /layouts, /lib, /maps,

/pylib and /script. Let's take them one at a time.

/bin: This contains the starting point of our program. It instantiates a module called running.

/config: This folder contains the supporting files (in .yml format) that specify which files that particular subject uses for layouts, based on whether or not it is a seek or change file, an strawberry file, or an apple file.

/docs: This one is confusing, as there is another docs folder further up the directory tree. This particular docs folder contains the layouts and maps subfolders that pertain to the CURRENT administration. The /docs folder further up the tree contains documentation that is not administration specific.

/layouts: This is also confusing, as we just saw a layouts folder inside the previous folder. This layouts folder contains the code files (in .yml format) pertaining to the layout of the data files, whereas the /docs/layouts folder contains the Excel spreadsheets documenting which field is in which position. The contents of these two folders work hand in hand.

/lib: This contains a folder called bigdog, which contains the .rb files that we modify, based upon the layouts as well as the rules, which is based upon the specification document. This is where the rules are changeed. It's also where the program files are actually stored.

/maps: This folder is similar to the /layouts folder in terms of confusion. Like the layouts folder, its counterpart is in /docs/maps. The /maps folder contains .csv files, derived from the .xls files found in /docs/maps. Our code uses the .csv files to verify that the answers are correct for each of the test questions.

/pylib: Like /lib, this folder contains the Python code that we run for certain portions of the process. It also contains some layout information as well, so we'll have to verify that it's current, based on the layouts given to us each term.

/script: This is the actual starting point for running our code. Run.sh contains the commands, options, and modes used to check the data files for accuracy and, as an aside, serves as a historical record of which processes have been run on which data files and which versions of said data files. It also contains the sftp.sh script, which allows us access to Pear's SFTP server.

/test: This folder contains .rb files which do testing for our code to double check that we are getting the right results that we expect and to verify that we get error messages when we pass in the wrong information.

There are other folders which are mandatory for the process to work correctly. You will need to create them if they don't exist.

/data: This is where you will store the data files while we are checking them. It should contain two subfolders, "strawberry_durian" and "satsuma_date". These subfolders are used in a very minor capacity during the Reports portion of our process.

/ChangeStrawberryCompare: This folder is used as part of a Python process that compares the Change file to the Strawberry file- the process writes its output to this folder. It is OVERWRITTEN each time you run that process, so you will need to do that aspect of the checking process one by one, saving the output to a separate folder outside of the project. I have a Feedback_Issues folder on my Desktop for this exact purpose.

/SeekChangeCompare: This folder is similar to the above folder, only it's used for comparing the Seek file to the Change file. It behaves in the same fashion, so copy that output before you run the next Python process for the next data file.

/out: This folder is only used for reports. The code for the reports that we generate outputs the pdf to this folder. SOME of the reports will overwrite the output as you process each grade, while others will not. You will have to look at the output section of the code to see whether or not there is a grade field. If it overwrites the .pdf, then you will have to rename it, adding the grade to the filename, before running the next report.

Back to TOC

Getting Started

Typically, we start at the Seek /Change phase. The first step is to look at the config files, found in /config. At this point, lets' look at the naming convention. Currently, there are three files: rmfrt_seek.yml, rfrt2_strawberry.yml and rfrtcbt2_strawberry.yml.

```
R-raspberry
M-melon
M4-Melon 4th Grade
RT-Retake
CBT-Computer Based Test
F-Fall
Wx-Winter (e.g. wa_eggplant.yml - means Winter Apricot)
A-Apricot
B-Banana
C-Cherry
G-Grape
H- Honeydew
S-Satsuma
W-Watermelon (e.g. w_seek.yml- means Seek file, Watermelon)
```

Spring and Summer are wrapped into the normal files (e.g. rmrt_seek.yml).

We want the rmfrt_seek.yml file (Raspberry Melon Fall Retake Seek file configuration). Open that and you'll see that it's broken down into sections. They are: batch header, school header, student and unscorable. There may be some other information there underneath that (like Achieve Table or Dev Score), but we can safely ignore that for the time being.

The batch header is where we'll start. The config file for your particular subject will tell you which layout files to open.

Back to TOC

Changing Layouts

Next is to examine the layouts. They are inside of the /docs/layouts folder. The folder structure is broken down by type of file, then within that, season (Fall, Winter etc.). Open the BatchHDR file. Also open the ruby layout file, in this case, frt_seek_batch_header.yml.

Things to pay attention to are the Start and End positions for each of the fields. Where this makes sense is the understanding that the data files that we are given from Pear are position based. That is, the Record Type field is at Column 1 in the data file and it's value ends at Column 2. In the ruby layout file, verify that record_type has the same values for start and end positions. (e.g. !ruby/sym record_type: [10, 11, !ruby/regexp "/GK/"] and observe the /GK/ value. According to the BATCHHDR.rtf, the value of RECORD TYPE should be GK. This is our regular expression confirming that.

At this point, you would verify 1) the start/end positions match between the ruby code and the layout, 2) the regular expression is correct for the values that we are expecting (based upon the layout), and 3) that each field in the layout is mentioned in our ruby code. On this last point, some of the fields are labelled as filler. Filler is NOT a data record. It creates spaces to make the data file a bit easier to read.

One interesting thing is the use of aliases.

```
Ex.!ruby/sym batch_num: [37, 40, &n_l4 !ruby/regexp "/[0-9]{4}/"] !ruby/sym stack: [42, 55, *n_l4]
```

Here the &n_l4 !ruby/regexp "/[0-9]{4}/" is creating an alias called n_l4 (which means numbers 0-9 with a length (l) of 4. It happily gets used right below. On some ruby layout files, the alias get's used more than once. This is designed to cut down on the amount of code and adheres to Ruby's DRY principle. The alias is defined with an &

and used with an *.

Another would be a bit of Ruby convention that we use:

```
!ruby/sym cat_points_possible: [497, 504, &possible !ruby/regexp "/09|10|21|27|55|62/", 2]
```

This means that for the fields cat_points_possible (of which there are FOUR), we are looking for the values listed in the regular expression. We are starting at position number 497 and looking at the values 2 at a time, until we stop at position 504. That's the significance of the ",2" after the regular expression.

NOTE: You can test your regular expression by typing "irb" in your Terminal window. irb stands for Interactive Ruby.

Then type in something like the following:

```
a="4"
a=~/[1235]/
```

It should return nil.

For the example a=~/[1235]/, it means, does a (which we defined a moment ago) MATCH the regular express 1235?

If the answers is no (or false), it returns nil. If it's yes (or True), then it will return 0.

Back to TOC

Tests

After updating the layouts for a given subject, you will need to go into the corresponding test files (located in /test) and update the test to reflect the changed values from the layouts. Upon completing that step, in your Terminal, run “rake test” to verify that all of the tests work properly. If you only changed one test file, you could simply run that one file instead. e.g. ruby test_seek_expressions.rb

Back to TOC

Test Cases

(our example is from test/test_eligible_ape_rules.rb)

The first part of our test is setting up a simulated datafile. We provide an entry with correct values, and then one with incorrect values.

```
setup do
  @ape_list = {"1234567890" => "1",
               "          " => "1"}
  @preid_barcode = { '001' => { preid_barcode: '1234567890' },
                     '002' => { preid_barcode: '          ' } }
end


Then, we put in values that are a success

should 'return nil when the rule succeeds' do
  assert_equal BigDog::EligibleApeRules.run(@ape_list,
                                            "1",
                                            "1234567890",
                                            "1"), nil
end

Then we provide values that should fail the test
should "return a properly populated results hash when the rule fails" do
  result = BigDog::EligibleApeRules.run(@ape_list,
                                        " ",
                                        "          ",
                                        "1")
```

and finally, we provide a mechanism to show us a result hash when it fails

```
    assert_kind_of Hash, result
    assert_equal 5, result.size
    assert_equal :eligible_ape_rules, result[:name]
    assert_equal "1", result[:calc_eligible_ape]
    assert_equal " ", result[:eligible_ape]
    assert_equal "          ", result[:preid_barcode]
    assert_equal "1", result[:score_status]
end
```

Note that the "assert_equal " ", result[:preid_barcode]" in the result hash matches the " ", in failure section. If not, then it will not fail the test.

After you finish modifying the test cases, test them out by typing in "rake test". If there's a failure, it should spit out something like this:

The following examples are from test/test_eggplant_expressions.rb

```
# Running tests:

.....................................................................................F.....................................................

Finished tests in 0.183657s, 1453.7970 tests/s, 40793.4356 assertions/s.

 1) Failure:
```

```
test: regular expressions in the weggplantpbt layout should match and fail to match properly. (TestEggplantExpressions) [/Users/bvagnini/Prc
Expected "    " to not match /[\d\s]{4}/.
```

This means that the layout changed (something that you changed, but forgot about), but the test case is expecting something else. Change an assert to a refute or vice versa and the problem should be resolved.

If there's an error, it should spit out something like this:

```
# Running tests:

.....................................................................................EF.....................................................

Finished tests in 0.180790s, 1476.8516 tests/s, 38010.9519 assertions/s.

  1) Error:
test: regular expressions in the weggplantcbt layout should match and fail to match properly. (TestEggplantExpressions):
NoMethodError: undefined method `[]' for nil:NilClass
    /Users/bvagnini/Projects/assessment-old/2014-2015/scoring_reporting/test/test_eggplant_expressions.rb:289:in `block (3 levels) in '
    /Users/bvagnini/.rbenv/versions/1.9.3-p545/lib/ruby/gems/1.9.1/gems/shoulda-context-1.1.1/lib/shoulda/context/context.rb:400:in `call'
    /Users/bvagnini/.rbenv/versions/1.9.3-p545/lib/ruby/gems/1.9.1/gems/shoulda-context-1.1.1/lib/shoulda/context/context.rb:400:in `block i

  2) Failure:
test: regular expressions in the weggplantpbt layout should match and fail to match properly. (TestEggplantExpressions) [/Users/bvagnini/Prc
Expected " " to match /Y\s/.
```

The nil:NilClass error simply means that, for the yml file that you are inspecting with this particular test case, the field doesn't exist in the yml file, but you are asking the test case to check it. Comment it out and re-run. The error should disappear.

NOTE: The test case will halt on the FIRST error that it finds; it will not report all errors that it finds. This means that you will have to re-run rake test multiple times to find each issue.

If there's NO errors or failures, it will output this:

```
# Running tests:

.....................................................................................................................................................

Finished tests in 0.185646s, 1438.2211 tests/s, 40388.6968 assertions/s.
```

Back to TOC

Generating Maps

We open the Excel spreadsheets (using LibreOffice) that Dorito provides (from /docs/maps) and Save As as a CSV file, with the tilde (~) as the separator. You will have to physically type in the tilde character, as it is not available from the dropdown list of field separators. The tilde is crucial, as that's how our program knows when the next field starts. We change the name of the file to something like a.csv (where a stands for Apricot, the course name) and save it in the /maps folder (vs. the /docs/maps folder).

Ensure that there are NO spaces before or after the field name and there is not a carriage return within the field name (e.g. someone at Pear hit the Enter key). The easiest way to see this is to open the generated .csv file within Textmate. The field names should occupy a single row, with no spaces (e.g. ~field~field~field~).

After making any changes to the Ruby code layouts, maps, and test cases, you'll need to check in your code.

Back to TOC

Checking in your code

In the Terminal, type in "git status". It should display a list of files that are modified in red.

"git add ." will add any modified files to your local working directory.

"git status" to confirm. The list of modified files should change to green.

"git commit -m "Update Fall RT layouts." " Note that the commit messages are always in present tense and that they are short but descriptive and are a complete sentence.

Git should now show you the number of files changed.

"git status" should now reveal "nothing to commit, working directory clean"

At this point, you need to see what portions of the codebase have been changed by others.

"git fetch origin"

If there are changes committed by others, it will show a new HEAD. If that's the case, you will need do a pull request via "git pull origin develop" and merge your code together. Otherwise, do a push with "git push origin develop" (you are pushing your develop branch to origin's develop branch.)

It will display something like this: To USER@SERVER:/home/USER/assessment-old.git 269dda8..0b718e3 develop -> develop

Confirm that your code is there with "`gitk --all`". NOTE: You will have to close out of Wish (which is the program running `gitk --all`), before you are allowed access to your Terminal command prompt again.

Repeat the process for the Strawberry and Apple files as the layouts become available. Remember, prior to changing any code, execute a "git fetch origin" command first. Someone may have already fixed the issue you were about to work on.

Back to TOC

Verification Process

There are cases where we either have to split the files (e.g. they gave us a combined file for Raspberry and Melon, but we are checking the subjects individually, or more commonly, we split between computer based (cbt) and paper based (pbt)) OR we have to combine files together (e.g. they gave us several different files that make up the Seek file).

We either use an awk script to split or a cat to combine the files together. Examples of these are located in the run lines. Assuming that we now have the file(s) in the right form, we continue on.

Our program goes and extracts certain fields, based upon what our layout file says (which is based upon what the Dorito supplied layout files say), and our program reads in those values.

Then our program compares the value read against the value it's expecting to see (via Regular Expressions) and reports any errors that it finds. In other cases, we are looking to one field for a certain value (a Y, for example), and based upon the rules outlined in the Specification (and defined in the ruby files in /lib/bigdog), we calculate the value of a different field, comparing that calculated value against the actual value in the data file. Again, we report any errors as either output to the terminal window or as a file inside of a folder with the same name as the data file we are checking.

The next step after that is to verify that the errors are NOT in our code, but rather with the data file. This may be done by opening the data file in either Textmate or in Vim. Errors in our code could be the result of a rule change, an incorrect layout position, an incorrect layout regular expression value, a typo...basically a lot of things. Most likely though, it will be some value that changed in the provided layout that didn't get changed in the Ruby layouts.

Once the code is corrected, re-run it and you should see some output in your Terminal session. You should generally see folders in your “scoring_reporting” folder with the filename you ran the program against as the folder name. You will need to examine the contents of those folders for any errors.

Back to TOC

Running an Strawberry file

1- Update the Config file, achievement table / third table, score table/dev score, and passing score and category table (for Melon). [/scoring_reporting/maps vs. /lib/bigdog/maps.rb] (original Dorito provided maps are in scoring_reporting/docs /maps/SUBJECTNAME/SEASON)

2- Check the class map to make sure that all the columns of the map match with our code. Verify that the column names in our code match the spelling of the column names in the Dorito provided map (see above) - Spelling counts: Case doesn't.

3- Update the student class accordingly (e.g. Raspberry Strawberry Student or Banana Eggplant Student).

4- Run the awk script to split the combined course data file into a pbt version and cbt version. This is in run.sh. Do this for ALL items. This will create 2 files: FILENAME_cbt.txt and FILENAME.pbt.txt in the /data folder. Use these files when running the Strawberry Checks, not the original Dorito provided data file. For the Dupes check, use the original Dorito provided data file.

5-Check that the map is being read correct by putting a “pp map” statement in the Eggplant_Checker.rb (line 27) or the Strawberry_Checker.rb file. Run the run.sh file and press CTRL+C shortly afterwards to stop running it. Compare the map files and Sequence numbers etc. This is done the FIRST time we check that particular course map . Subsequent iterations will not need this step, nor step 6.

6-Remove the pp map statement (or comment it out).

7- Run the run.sh file again. Verify that our code is correct prior to reporting to Dorito. If necessary, look at the actual data file to verify that the column/field numbers are correct.

8- After running our program, it will output three folders: FILENAME_cbt.txt, FILENAME_pbt.txt,and FILENAME.txt. This is a result of the two Checks and one Dupes process. These folders contain the output of any errors or issues that you may find. You may want to hang onto these folders until Dorito approves the files in question, just in case they ever come back with a question about what you've discovered. This output is also what you will verify in the actual data file. Occasionally, our code is incorrect and we have to make an adjustment to it, then re-run the files again. Above all else, remember that our goal is NOT to match Pear's results, but to verify (via the specification document and the contract) that Pear is doing THEIR job correctly.

For Strawberry files, we also run a Python process called change_to_strawberry.py ONE AT A TIME, as the process will overwrite the contents of your /ChangeStrawberryCompare folder.

In any case, we report our findings to Dorito via email. In some cases, we will post certain files (e.g. Change-to-Strawberry output, or Seek-to-Change output) to our SFTP site for Dorito to follow up on.

Check in your code.

Back to TOC

Verifying Output

Sometimes you have to open the actual data file within Textmate or Vim. In most cases, our output will supply you withthe PAS in question. If the field is the PreID or perhaps the Lithocode, then the output will supply that information. You can then search for that in the data file.

In Textmate, Apple + F opens the Find dialog box (for when you need to verify that there actually two duplicate PreID Barcodes, for example). You can double-click on the results found and it will take you to that exact spot. In some cases though, you need to be able to verify that the pattern that it found is actually in the correct postion. This is done via opening up the layouts (from the /docs/layouts/TASK folder) OR you can open another instance of Terminal (Apple + T) and do a grep (e.g. "grep preid layouts/*.yml"). This will let you know which position the duplicate field in question is at. At this point, within Textmate, you can go to Navigate > Go to Line (or Apple +L) and specify the exact line number and column number (e.g. 96037:212).

Back to TOC

Gotchas

1- Sometimes, you will get output like this:

```
./script/run.sh Running appleregate file checks... /Users/bvagnini/Projects/assessment-old/2014-2015/scoring_reporting/lib
/bigdog/running.rb:318:in `initialize': No such file or directory - data/IRS1XSBY0a.txt (Errno::ENOENT) from
```

```
/Users/bvagnini/Projects/assessment-old/2014-2015/scoring_reporting/ lib/bigdog/running.rb:318:in `open' from
/Users/bvagnini/Projects/assessment-old/2014-2015/scoring_reporting/ lib/bigdog/running.rb:318:in `each_line_with_progress'
```

etc.

What's happening here is a common, but simple mistake. The file that you are trying to run our program against doesn't exist. This is usually due to a copy/paste error, where you forgot to change the filename, or in the process of changing the filename, you enter a typo (extra space, remove a _). It looks scarier than it is. Just confirm the filename and it will work just fine.

2- When you are viewing the output from a check done on a split file (meaning the Raspberry PBT file), you should verify that output by opening and checking the split Raspberry PBT file, and not the main data file, as the line numbers in the output won't match up at all.

Back to TOC

Eggplant Apple

Look at individual course files (e.g. config/a_eggplant.yml) This will give you the layout files for Apple tasks.

Update the regular expressions in the relevant files first. Use the layout in /docs/layouts/apple/eggplant/spring for guidance.

Next is to update the rules.

CODE

1) /config/SUBJECT LETTER_strawberry.yml

/layouts/eggplant_apple.yml # check fields and reg exp against layout

/layouts/abgh_apple.yml #the new class (e.g. cherry or c_apple.yml) will have it's own layout, not a shared one

2) /lib/appleregator.rb # check rules against spec and layout

LAYOUTS

/scoring_reporting/docs/layouts/apple/eggplant/spring

-Eggplant Strawberry-

CODE

/config/[subject letter]_eggplant.yml

/config/[subject letter]cbt_eggplant.yml

LAYOUTS

/scoring_reporting/docs/layouts/strawberry/eggplant/SEASON

Change test_eggplant_expressions.rb to match changed layout files.

("rake test" will help you chase them down)

Back to TOC

Rulesets

Rulesets get called from other files within our process. For example, we have this snippet from raspberry_retake_ruleset.rb:

```
  @results.clear
  add_result EligibleApeRule.run(@ape_list, student[:eligible_ape],
                                 student[:last_name], student[:first_name],
                                 student[:student_id],
                                 student[:preid_barcode],
                                 student[:score_status])
      @results
```

Observe the number (7) and the order of the arguments for EligibleApeRule.run. This will match up exactly to the "def self.run(ape_list, eligible_ape, student_last_name, student_first_name, student_id, preid_barcode, score_status)" section of eligible_ape_rule.rb. If the arguments aren't correct, you will get a "wrong number of arguments (6 for 7) (ArgumentError)" when you attempt to run the code.

In each portion of the code (that is a ruleset), we want to get certain output. This is accomplished with this section of code below:

```
  result_data = { :name => :eligible_ape_rule,
                  :calc_eligible_ape => calc_eligible_ape,
                  :eligible_ape => eligible_ape,
                  :score_status => score_status,
                  :student_last_name => student_last_name,
                  :student_first_name => student_first_name,
                  :student_id => student_id,} if calc_eligible_ape
                  != eligible_ape
```

The output that it generates looks like this:

```
line, id, [:calc_eligible_ape, :eligible_ape, :score_status, :student_last_name, :student_first_name, :student_id]
2614, C21674768, [" ", "1", "1", "JAMES            ", "JESSE     ", "123456789X"]
2615, C26687520, [" ", "1", "1", "JAMES           ", "FRANK     ", "987654321X"]
```

This output creates a file called eligible_ape_rule.ftl in the FILENAME.txt folder. That's the purpose of the ":name => :eligible_ape_rule," section.

Everthing else below it becomes the column headers in the resulting hash.

Check in your code.

Back to TOC

Mapping Rules to Specific Contract lines

We start at pg 7 of 21 pages. The header to look for is "Verification of Seek Files (original and post-change file)". This lists the tasks for each section (Seek/Change, Strawberry etc.) While the original contract used bullets (not numbers), we will be assigning a number to each of the bulleted items. For example, Strawberry and Durians have 28 rules total (20 on pg 7 and the remaining 8 on pg 8).

Seek and Change have no rules. Everything there (all 4 items) is done through different parts of the program. RE-WRITE ME WITH CORRECT INFO!!

Strawberry/Durian section has the bulk of the rulesets.

```
1 running.rb → check (wa_eggplant) : verify valid values from layout
2 change_to_strawberry_check.py : verify change to Strawberry check, find missing litho, indentify changes
3 eggplant_score_status_rule.rb : verify score flags properly set.
4 McScoredItemRule.rb : very multiple choice, FrScoredItemRule.rb: fillin response
5 eggplant_raw_score_rule.rb ,EggplantCatPointsEarnedRule.rb : verify total raw/sub scores
6 achievement_rule.rb : verify level is correct, based on scale/score dev/scale score
7 eggplant_passing_indicator_rule.rb : verify pass/fail indicator is correct, based on scale/dev score
8 report_status_code.rb : verify report status code / score flag are consistent.
9 school_type_rule.rb : verify school type  ReportReceivedCodeRule.rb : status codes
10 curriculum_group_rule.rb /standard_curriculum_code_rule.rb : verify group code is corret
11 race_code_rule.rb : Verify reported race field is correct
12 accomodation_code_rule.rb : verify large print/Braille records =  have Y accommodation codes
13 pex_section504_code_rule.rb : check braille and large print records pex /= blank or sect 504 = Y
14 running.rb → check dupes: verify no sec barcodes are missing when ans doc = sec doc
15 running.rb → check dupes: verify no print after seek #, litho codes = missing, when doc = seekned
16 weggplantpbt.yml (regex) : dob ruby/sym → verify century in virthdates have been set properly
17 running.rb → check dupes: verify no dupes
18 FillinItemChangesRule.rb : verify gridded / fill-in responses
19 n/a
20 n/a (no calib file)
21 EggplantScaleScoreLowerBoundRule.rb / EggplantScaleScoreUpperBoundrule.rb: Verify up/low scalescore.
22 strawberry_to_durian.sh → run seperately : verify same data in Strawberry to Durian
23 n/a
24n /a
25 n/a
26 n/a
27 n/a
28 n/a
```

For Winter 2014 Eggplant Strawberrys, the only rulesets that come into play are the following:

```
winter_abgh_eggplant_ruleset.rb which calls:
  achievement_rule.rb
  eggplant_passing_indicator_rule.rb

 Winter_Eggplant_Ruleset.rb which calls:
   EggplantScaleScoreLowerBoundRule
   EggplantScaleScoreUpperBoundRule
   ReportStatusCodeRule
   ReportReceivedCodeRule
   StandardCurriculumCodeRule
   RaceCodeRule
   AccomodationCodeRule
   PexSection504CodeRule
   McScoredItemRule
   EggplantScoreStatusRule
   FillinItemChangesRule
   FrScoredItemRule
   SchoolTypeRule
   CurriculumGroupRule
   EggplantRawScoreRule
   EggplantCatPointsEarnedRule
```

At this time, Apple rulesets have NOT been mapped to their contract bullet items (112114).

Back to TOC

Getting the data files

Back to Overall Process

At this point, the layouts have been adjusted. We receive an email from Pear stating:

"Fall 2014 Retake Mock Seek and Change"

The email should contain the path to where the files are located on their SFTP server. It may look something like this:

/test-materials/Test Administration/Scoring and Reporting/Dorito to FSU Cheerio/Fall 2014 Retake

In your Terminal session, type in "./script/sftp.sh". This will open a session with Pear's FTP server using Sweta's credentials. (There are three sets but we only need to use one at the time.) Switch back to your email and copy the path to the files that you want.

Switch back to your Terminal session and type in "cd" and paste the path. Since there are spaces in the path for Pear's FTP and *nix systems don't really allow for spaces, your command should have quotes around the path section.

e.g. cd "/test-materials/Test Administration/Scoring and Reporting/Dorito to FSU Cheerio/Fall 2014 Retake"

Do an "ls -a" to verify the files existence.

At this point, you can type in "get *.*" and it will download the files to your "scoring_reporting" folder. In some cases (particularly during reports, where the files we want are in a folder called "Files" and in "Reports"), it is more advantageous to type this in instead:

"get -r Files" or "get -r Reports"

This will create the "Files" folder in your "scoring_reporting" folder and download the contents into the newly created folder. NOTE: You MUST move the files/folder out of your "scoring_reporting" folder (maybe place them on your Desktop?), prior to attempting to copy the files into our code server (for everyone to have access to.) The reason is, is that Git is tracking those files and it causes an issue while copying, since the file is "in use".

At this point, open a connection to our code server (called "Catkin"). You do this by clicking on Go >Connect to Server in the Finder. The server that you want is "cifs://catkin/files". If you don't have access to this folder, please let the System Administrator know and they will add you to the list of allowed users. The folder structure for this server is broken down by either task (e.g. "Accountability 2014 Files") or by season (e.g. "Fall 2014 Retake"). Within that folder, you should see a folder called "Live Data Files", and within that, folders such as "SeekChange" and "Strawberry-Starfruit". (there's a LOT of drilling down to the folder that you want.) Once you've figured out where to put it, paste the downloaded files. They will now be available for everyone to use.

One point of clarification: You will not see the git repos listed in this view, even though the git repo is stored on Catkin. Not to worry, git can see the repo and that's all that really matters. By the way, the git repo is backed up once a week (even though the code is on multiple machines) and that backup is transferred to an offsite location for safe-keeping. Your hard work will NOT disappear due to a hard disk failure.

Back to TOC

Reports

This is the portion of the project where you will involve other Cheerio staff besides the Programmers. The other members will be comparing the Pear generated report, against the Cheerio generated report. There are six types of reports generated through FRUIT.

They are as follows:

```
Individual Student Reports (ISR) – these go to the schools to be mailed out to the Student's parents.
State Summary (SS)-
State Report of Districts (SRD)-
District Summary (DS)-
District Report of Schools (DRS)-
School Report of Students (SRS)-
```

Everything except for the ISR's are considered to be "Educator Reports", since the students/parents will never see them. The Schools, School Districts and Dorito will use those reports. We get the ISR, or Individual Student Report, via Fedex in boxes.

Back to TOC

Obtaining Pear's Reports

You will get an e-mail and download the reports in the same fashion that you download the other files. The filenames will come in two different flavors:

Eggplant_SUM14_55_SRS_ALG1_DIST_WAVE1.pdf and Eggplant_SUM14_550411_SRS_ALG1_SCHL_WAVE1.pdf

When you print out their reports to check, we only want the first file, not the second. The second is specifically for District 55, School 0411. We only care at the District level.

Place the Reports (and Files- they are used for something else), in Catkin in the approriate Season folder.

Back to TOC

Generating our reports

The run lines for these look like this:

```
#ruby -I lib bin/bigdog --mode rfrtpbt_strawberry --report isr --districts 65,67 --file IRS1XSBY0a.txt
#ruby -I lib bin/bigdog --mode mfrtpbt_strawberry --report isr --districts 65,67 --file IRS1XSBY0a.txt
#ruby -I lib bin/bigdog --mode rfrtpbt_strawberry --report ss --apple_file IRS1XAPL0a.txt
#ruby -I lib bin/bigdog --mode mfrtpbt_strawberry --report ss --apple_file IRS1XAPL0a.txt
#ruby -I lib bin/bigdog --mode rfrtpbt_strawberry --report srd --apple_file IRS1XAPL0a.txt
#ruby -I lib bin/bigdog --mode mfrtpbt_strawberry --report srd --apple_file IRS1XAPL0a.txt
#ruby -I lib bin/bigdog --mode rfrtpbt_strawberry --report ds --apple_file IRS1XAPL0a.txt
#ruby -I lib bin/bigdog --mode mfrtpbt_strawberry --report ds --apple_file IRS1XAPL0a.txt
#ruby -I lib bin/bigdog --mode rfrtpbt_strawberry --report drs --districts 05,17,45,56 --apple_file IRS1XAPL0a.txt
#ruby -I lib bin/bigdog --mode mfrtpbt_strawberry --report drs -->districts 05,17,45,56 --apple_file IRS1XAPL0a.txt
#ruby -I lib bin/bigdog -->mode rfrtpbt_strawberry --report srs --districts 05,17,45,56 --file IRS1XSBY0a.txt
#ruby -I lib bin/bigdog --mode mfrtpbt_strawberry --report srs --districts 05,17,45,56 --file IRS1XSBY0a.txt
```

One thing to notice is that for the ISRs, DRSs and SRSs, you have to specify which districts. For the ISRs, this decision is made by Pear. They will generate reports for all 70(?) districts, but will only ship us two districts to check (usually via UPS or Fedex). You will have to open the boxes to figure out which districts to print out. They usually send us District 65 and 67, but it does vary from time to time.

For the DRS and SRS, we typically choose four districts to check; two large and two small. We usually pick the same four districts to simplify things.

Every once in a while, we have to specify a different district. This is usually due to no reports generated for a particular subject within that district

(e.g. Kids took Raspberry, but not Melon.) Since Raspberry and Melon are usually reported together, we would have to pick a different district.

The other thing to note is that the ISR and SRS reports (both of which pertain to actual student information), use the Strawberry file. All other reports use the Apple file.

Be careful when copying and pasting to remember to change the mode from a_eggplant to b_eggplant when it's for Banana and you copied the Apricot lines.

Back to TOC

Technical note:

In the run line for the district option, (--districts 14,20,30,38) notice that there are NO spaces after the commas. If you place a space after the comma (e.g. 14,20, 30,38), it will fail to generate any reports for districts 30 and 38.

Back to TOC

Modifying the Reports

The code files for reports are located in the lib/bigdog folder and are named similar to this:

banana_ds_reporter.rb

Anything with a _reporter.rb is a report file.

Open apricot_isr_reporter.rb

Things you typically have to change are the @header (line 18) to reflect the current Season (the type of report is found in IsrReporter (line 15) and generally doesn't ever change).

Verify fields in the student_record (line 40) against the Pear report.

Note the output folder (line 237). This is why you created the /out folder within the project. Also note that there are no grades in the filename. For Apricot (and the rest of the Eggplant courses), this isn't a problem. However, for Raspberry and Melon, there are multiple grades (which means multiple run lines to generate our version of the ISR). This means that you will need to run the run lines one at a time, changing the outputted PDF filename to reflect the grade, prior to running the next grade.

One thing to note in the student reports (ISR and SRS). Some of the schools assign the student ID field for each student as a random 10 digit number. This is what they're supposed to do. However, SOME of the districts use the student's SSN as the student ID, adding an X at the end to make it 10 digits. In these cases, we have a function (line 240) that looks for that X, then changes the student ID to mask out the first digits of their SSN to X, in order to protect the students. The ISR is generally the only report that this applies to; however, in the student's best interests, we have modified all the reports to do this process.

Back to TOC

In-depth view of the Report codebase

The report process starts off in run.sh with the following run line:

"ruby -I lib bin/bigdog --mode rfrtpbt_strawberry --report srs --file IRS1XSBY0a.txt"

Run line calls the run method (line 61) in running.rb via the –report srs option (defined in options.rb reports section). Report call the report method (line 239) in running.rb. This calls student_factory.rb or apple_record_factory.rb, depending upon the file (defined in options.rb line 60 or 61). Student_factory.rb builds the student record.

In student_factory.rb, the mode supplied in the run line determines which type of Strawberry student record to build (line 21). In our case, it calls raspberry_fall_retake_strawberry_student.rb.

Back in the report method of running.rb (line 408), there is a call to reporter_factory.rb. This takes the mode from the run line and calls raspberry_fall_retake_srs_reporter.rb (line 180 in reporter_factory.rb).

Next, the approriate hashes get built via the initialize method in raspberry_fall_retake_srs_reporter.rb (line 6).

The report method in running.rb (line 399) starts parsing the record, calling the method (via line 409) each_line_with_progress (defined in line 416). It then calls record.parse, then the collect method in reporter.rb (line 19), and finally the generate_report method in reporter.rb (line 22).

Back to TOC

Gotchas

One of the things to watch out for is when copying/pasting code between reports (say that you fixed something in one report and now you need to make that same change in the others. The gotcha here is that some reports use district_record, while others use state_record. This will throw an error if you forget to change it to match up with the type of records that it is actually using.

Back to TOC

Strawberry to Durian / Starfruit to Date comparison

We also typically run an awk script (/script/satsuma_to_date.sh and /script/strawberry_to_durian.sh) to perform a comparison between ours and theirs. It will require 2 folders inside of /data - /satsuma_date and strawberry_durian. When the awk script finishes, it will produce a file called difference.txt. We report this file to Dorito. It also creates 2 other files (other than the DATEs, Starfruits, and Durians placed there). These are sorted_our_date_file.txt (or durian) and sorted_their_date_file.txt (or durian).

NOTE: You will need to place the Apple or Strawberry file inside of this directory for the awk script to function properly. UNZIP the files first! You will also need to remove district 98 from the DATE by deleting it from the Starfruit_DATE folder.

Back to TOC

Getting Started

The most efficient way to do this process is to be inside of the data/satsuma_date directory when running the script/sftp.sh, then get all of the files (get *.*) from the path provided (you may have to call the sftp script via ../../../script.sftp.sh). This will dump ALL the files inside of the satsuma_date folder. To move just the Durian files, run the following from inside of the satsuma_date folder:

mv *Durian*.txt ../strawberry_durian

This will move the Durian files to the strawberry_durian folder for you. Switch to that folder. Next, run the following to combine the Strawberry files into 1 file (do this for both folders):

cat R04WStrawberry0c.txt R08WStrawberry0c.txt R10WStrawberry0c.txt > our_strawberry_file.txt

Then, verify in the strawberry_to_durian.sh file the length and the correct start and end positions of the pas via the Durian layout, not the Strawberry layout. This step isn't required in the satsuma_date.sh script.

Now, change directories back to the scoring_reporting folder and run the script/strawberry_to_durian.sh our_strawberry_file.txt and script/satsuma_to_date.sh our_apple_file.txt (for the satsuma_date process, you may be able to use the concatenated file they provided (FRUIT_14SPR_Starfruit_WRIT.txt))

This script requires an argument for the Strawberry or Starfruit file that it will run against. In our case, it's our_strawberry_file.txt.

Back to TOC

Finding the cause of Issues:

(e.g. they excluded school type 17 from their Durian files)

To find the count of the schools that are of type 17 (at position 89, per the Durian Layout):

awk 'substr($0, 89, 2)=="17" {print}' data/strawberry_durian/our_strawberry_file.txt > temp

then

"wc -l temp"

to get the line count.

Check in your code.

Back to TOC

Sending Feedback

This is the most important part of the job. This is how we can prove that we did the work, and thus, get paid for it.

The Email Subject header usually contains the Season, Course and Version of the files that we checked.

(e.g. Spring 2014 RM Late Reporting Wave 2 Strawberrys Version b)

The body of the email is best described by examining one of the previous ones.

Ex.

```
R04XStrawberry2b.txt, R06XStrawberry2b.txt, R07XStrawberry2b.txt, R09XStrawberry2b.txt, & R10XStrawberry2b.txt

No issues are found.
No blanks and duplicates are found.

Change to Strawberry

The differences between Change and Strawberry files are posted on the Cheerio sftp site at
cheerio_to_dorito/spring_2014_live/change_to_strawberry/rm_retake/late_reporting
/wave_2/version_b

Thanks,
```

This is one that there were very few issues found throughout the entire process. In this case, we put the filenames across the top. You always report any Issues with the checks process and then any duplicates that are found. Everything is in the present tense.

Then, because this was an Strawberry file, we had access to the Change file as well, so we ran the python process "change_to_strawberry" (found in /pylib).

The output of that process (if any) is dumped into the /ChangeStrawberryCompare folder within the scoring_reporting folder of the project. We take that output and copy it into the appropriate spot on our SFTP server (which differs from Pear's SFTP Server.) The server name is currently called SnapDragon.

Back to TOC

Example Seek/Change Feedback that has some issues but not a lot

Fall 2014 Retake Live Seek Change Version a

Sent 10/21/2014

```
RTKseekFiles.txt

No issues are found.
No blanks and duplicates are found.


IRS1XCNG0a.txt

No issues are found.
No blanks and duplicates are found.


Frequency Response Check

Raspberry
No issues are found.

Melon
Response Item 23 is missing the value “4”
Response Item 48 is missing the value “1”
Response Item 58 is missing the value “4”

Seek to Change

There is one record in Change, but not in Seek.
PAS
8675309

Raspberry

1 record went from 99 to 17.


Melon

0 record went from 99 to 17.


The differences between Seek and Change files are posted on the Cheerio sftp
site at
cheerio_to_dorito/fall_2014_retake_live/seek_to_change/rm_retake/version_a


Thanks,
```

Back to TOC

Example Strawberry Feedback that has issues

Fall Retake 2014 Live RM Strawberry Ver a

Sent: 10/23/2014

```
IRS1XSBY0a.txt

Raspberry

There are 189 records with a different value for field APS than what we
calculated. The “Scrunch logic” is applied for matching records between
Strawberry and PreID files. If no “Scrunch logic” is applied there are 61
mismatches. Do we need to scrunch the fields (last name and first name)
as the specification document says or has that changed?
Example PASs
C23659988
C23644708
C23667536
C23624306

There are 2564 records with incorrect values for the field Theta Score.
In every case, there are four zeros after the decimal point. According to
the contract, that is not allowed but the layout does not specify this.
Which one is correct - the contract or the layout?
Example PASs
846800394
846700407
C23503087
C23503327

There is a record with PAS 847100051 with invalid value for the Student_Id
field.

There are three records (PASs C23628631, C23629095, C23629075) where the
“First Name” and a record (PAS C23669958)where the “Last Name”
start with something unusual. According to the layout it’s not an
invalid value but it does not seem correct.

There are 2 duplicate PreIDs.
PreID
101168649


Melon

There are many records that have different value for changeed gridded items
1, 3, 4, 5, etc. than what we calculated.
```

```
In all of the cases we calculated the value to be blank.
Example PASs
C23673825
C22928965
C23635667
C23350133

There are no blanks and duplicates found.


Change to Strawberry

No issues are found.


Thanks,
```

Back to TOC

Vs. One that does not have any issues

Spring 2014 Eggplant Late Reporting Wave 2 Strawberry b Files

Sent: 08/15/2014

```
RS3HSBY2b.txt

No issues are found.
No blanks and duplicates are found.


Change to Strawberry
Without the Wave 2 Ver B Change file, a good compare of the Change to Strawberry
cannot be done at this time.

Thanks,
```

Back to TOC

Example Apple feedback that has issues

Fall Retake 2014 Live RM Starfruit Ver a

Sent: 10/23/2014

```
IRS1XAPL0a.txt

Raspberry

There are four records with a different value than what we calculated for
Fewer than 10 Suppression flag. In all cases, we calculated it to be
“Y” and not blank. According to the layout , blank is not a valid
value for Modes 1, 2, and 3.
District = 13, School = 7835, Grade = AD
District = 48, School = 0202, Grade = 10
District = 50, School = 3083, Grade = 12
District = 50, School = 3396, Grade = AD

There are four records with a different value than what we calculated for
Number of Students. In all cases, we calculated it to be all zeros and
not blank. According to the layout , blank is not a valid value for
Modes 1, 2, and 3.
District = 13, School = 7835, Grade = AD
District = 48, School = 0202, Grade = 10
District = 50, School = 3083, Grade = 12
District = 50, School = 3396, Grade = AD

There are four records with a different value than what we calculated for
Percentage Passing Suppression flag. In all cases, we calculated it to be
“N”  and not blank. According to the layout , blank is not a valid
value for Modes 1, 2, and 3.
District = 13, School = 7835, Grade = AD
District = 48, School = 0202, Grade = 10
District = 50, School = 3083, Grade = 12
District = 50, School = 3396, Grade = AD

There are four records with a different value than what we calculated for
Same Achievement Level Suppression flag. In all cases, we calculated it to
be “N" and not blank. According to the layout , blank is not a valid
value for Modes 1, 2, and 3.
District = 13, School = 7835, Grade = AD
District = 48, School = 0202, Grade = 10
District = 50, School = 3083, Grade = 12
District = 50, School = 3396, Grade = AD


Melon

There are 2434 records with a different value than what we calculated for
Fewer than 10 Suppression flag. In all cases, we calculated it to be
“Y” and not blank. According to the layout , blank is not a valid
value for Modes 1, 2, and 3.
Example Records
```

```
District = 01, School = 0000, Grade = 10
District = 01, School = 0000, Grade = 13
District = 01, School = 0151, Grade = 10
District = 01, School = 0151, Grade = 11

There are 2434 records with a different value than what we calculated for
Number of Students. In all cases, we calculated it to be all zeros and not
blank. According to the layout , blank is not a valid value for Modes
1, 2, and 3.
Example Records
District = 01, School = 0000, Grade = 10
District = 01, School = 0000, Grade = 13
District = 01, School = 0151, Grade = 10
District = 01, School = 0151, Grade = 11

There are 2434 records with a different value than what we calculated for
Percentage Passing Suppression flag. In all cases, we calculated it to be
“N”  and not blank. According to the layout , blank is not a valid
value for Modes 1, 2, and 3.
Example Records
District = 01, School = 0000, Grade = 10
District = 01, School = 0000, Grade = 13
District = 01, School = 0151, Grade = 10
District = 01, School = 0151, Grade = 11

There are 2434 records with a different value than what we calculated for
Same Achievement Level Suppression flag. In all cases, we calculated it to
be “N" and not blank. According to the layout , blank is not a valid
value for Modes 1, 2, and 3.
Example Records
District = 01, School = 0000, Grade = 10
District = 01, School = 0000, Grade = 13
District = 01, School = 0151, Grade = 10
District = 01, School = 0151, Grade = 11



Thanks,
```

Vs. One that doesn't have any issues.

Sent: 10/23/2014

```
IRS1XAPL0a.txt

There are no issues found.


Thanks,
```

Back to TOC

Vs. This one which has a common problem

Fall Retake 2014 Live RM Starfruit Ver a

Sent: 10/23/2014

```
IRS1XAPL0a.txt


There are many records for both Raspberry and Melon that have blank for the
fields “Number of Students”, “Fewer than 10 Suppression flag”,
“Percentage Passing Suppression flag”, and “Same Achievement
Suppression flag”. According to the layout , blank is not a valid value
for Modes 1, 2, and 3 for these fields. It looks like these fields have
blank for a record when no scorables were returned. This is not specified
in the layout like in the “Spring 2014 RM Apple layout”. So, is it
correct to have blanks for these fields when no scorables are returned?
Other than that, no issues are found.



Thanks,
```

Back to TOC

Connecting to SnapDragon

Open a Finder window and click on Go > Connect to Server. The server address that you want to use is "cifs://snapdragon/FTPData". Look for the cheerio_to_dorito folder.

Back to TOC

Once you click on "Send"

Copy the feedback emails that you sent into a text file onto your Desktop. The text file should have Sent: and the date the email was sent on it. (e.g. Sent: 10/02/2014), followed by an empty line and the body of the email that you sent to the Doritoassessment list in your email client. The Filename of the text file should be whatever the Subject line was. (e.g. Summer 2014 Eggplant Wave 1 Late Reporting Educator Reports.txt)

Open a Finder window and click on Go > Connect to Server. The server address that you want to use is "cifs://violet/production". Look for the fcat_deliverables folder.

Ask your team lead for which invoice_XXXXXX folder to put it in, and paste the text file into the folder.

Back to TOC

Misc. Stuff

AWK scripts

Isolating just the kids with a Y for Eligible_APE

awk 'substr($0,242,1) == "Y" {print}' data/fl_fsa_fcatfa14_int_ extract_preid.dat > data/APE_File.txt

NOTE: AWK is 1 based, not 0 based like Ruby is...

Removing extra whitespace at the end of lines of code

Be inside of your scoring_reporting folder in the terminal window.

```
grep -n -e "\s$" layouts/*.yml
```

Code map

VERY rough draft of this

running.rb - Kickstarts the whole program

- options.rb - Defines modes used in command line arguments
- config.rb - Loads config files from /config
- student_factory.rb - Creates a student object, based upon mode presented
- checker_factory.rb - Based upon modes presented, starts checkers (e.g. eggplant_checker)
- field_writer.rb - >Writes the program output to the Terminal window and writes the error files (e.g. mi.ftl)
- appleregator_factory.rb -
- apple_record_factory.rb -
- reporter_factory.rb - Based upon mode presented, starts creating reports

Back to TOC

What next?

Rinse and repeat as you move through the FRUIT process (Mock Seek Change, Mock Strawberry, Mock Apple, Mock Reports, Live SeekChange, Live Strawberry, Live Apple, Live Reports, Late Reporting (Strawberry) Closeout (Strawberry)).

Best of luck...