CART User Guide

Table of Contents

When running CART, do not just run “./cart” !! Please run instead “./run_cart” !!! why? Because run_cart first clears out the database. UNLESS you are running CART in “Current” mode, in that case then just run “./cart”, otherwise, to make sure you have a CLEAN system, run “run_cart” because it does a ./clearKB -YES before running CONALAUN.

1.0 Overview

CART (Complete Automated Regression Tester) is an independent project from conalaun. CART is used to test Conalaun.

The official backup for CART source code is Git Hub. (@ https://github.com/victortimothyshulist/cart). It is on Github unencrypted.

conalaun however, is backed up on Google Drive and IS encrypted.

Since CART is its own project but it is required to exist inside conalaun home directory, copy cart.py from its own directory into conalaun directory. (Make sure that ./conalaun/cart.py is updated and matches /cart/cart.py).

cp /home/victor/cart/cart.py /home/victor/conalaun/

2.0 How it works

conalaun can be invoked and executed in two modes:

- Mode 1 : NOT running under CART
- Mode 2: running under CART.

In Mode 1, you run conalaun.py by just calling it as:

```
cd conalaun
cd ./conalaun.py
```

When it is run this way, TESTING is set to False. When it is False, conalaun.py reads input from the console (keyboard) and responds to the screen. But in Mode 2, CART creates a temporary version of conalaun, and calls it. Then TESTING is True. When true, conalaun.py does not read from keyboard input, but from a file (a file that CART creates).

For a given version of conalaun, say version 1.000, you create a configuration file.

As an example, for version 1.000, you would create a file in...

./testpackages/v1.000.conf

This file contains the list of input strings to process. It also contains the names of the compressed tar ball files that contain 'state' to process each input string under. The 'state' is the directories and files for conalaun to operate under, for each input. These directories for example are...

./conversation-history/

./sessions/

./tls_csv/

./interpretations/

./compiled-classes/

So that, if you test the processing of input string 'x' a year ago, with specific compiled-classes and conversation-history, you know that every time you run it in the future, it will run under the same circumstances ('state' – files in the above directories).

So, if you run CART for say v1.000, CART will decompress the files in

testpackages/files/v1.000/

and run each input listed in testpackages/v1.000.conf.

There are 2 modes you run CART in...

- Mode 1 : 'create 'mode. In this mode you are establishing what the expected results are for processing each line in your *.conf file ('v1.000.conf' in this example). In mode 1, conalaun creates /home/victor/cart/testpackages/files/v1.000/v1.000.tar.gz. Let's say you just coded version 1.000, and all test cases pass, let's say this is June 01, 2020. In

mode 1, testpackages/files/v1.000/v1.000.tar.gz is created.
(in mode 2, it is not created, but compared to the results produced in results_cart_tests).

- Mode 2 : 'test' mode. In mode 1 your expected results file was created. In Mode 2, let's say you have just coded version 2.000. Well, before you even start developing the test cases for version 2.000, you want to make sure you didn't break any v1.000 functionality. So, first you re-run the tests for v1.000. You do this in Mode 2 ... where CART runs conalaun with all tests for v1.000 and the output goes into
/home/victor/conalaun/results_cart_tests
Let's say you just coded version 2.000 and it is June 01, 2022. Well, you coded v1.000 two years ago !! SO, did the new code you added to conalaun.py (which provides version 2.000 functionality), did that new code of v2.000 break the version 1.000 code? Well, the output produced in /home/victor/conalaun/results_cart_tests (from test results in 2022 ('just now') on June 01, 2022, is compared with the expected results for v1.000 from 3 years ago (the file /home/victor/cart/testpackages/files/v1.000/v1.000.tar.gz – test results from 3 years ago). CART decompresses that tar.gz file and compares it to results in
/home/victor/conalaun/results_cart_tests
any mismatch at all in contents results in 'regression test failed' ! Error message appearing on screen.

3.0 Example of using cart for version 1.000 and 2.000 of conalaun

Let's say we just coded version 1.000. So, we develop the test cases, and then create the v1.000.conf file. You create testpackages/v1.000.conf
let's say it has this for contents....

New feature : added March 7, 2021 (SD 503.76) – if you put the text "/PAUSE" (no quotes, but make sure to have the leading forward slash and all uppercase 'PAUSE') like shown below, then CART will force conalaun to pause with a "press enter to continue" message after processing input "input A" and processing "input B" as shown below. You can have as many "/PAUSE" commands as you want. You can use it to suspend CART/conalaun while you run another instance of conalaun that you are debugging.

(((check this – the code seems to support it.. but yet doesn't LOL))))-->>> *Instead of having "/PAUSE" on a line by itself, you have the option of putting your string to process after it on same line. So you can say "/PAUSE:hell there". Conalaun will then pause waiting for*

user to press enter then process “hello there”.

```
# CART test package file for version 1.000.
areas = red, green, blue

[state.1.tar.gz]

“input A”
/PAUSE
“input B”

[state.2.tar.gz]

“my line X”
“my line Y”

[state.3.tar.gz]

"my line A”
“my line B”
```

After creating all your 'state' files/folders (any state files that conalaun depends on, example, ./conversation-history/, ./sessions/, ./tls_csv/, ./interpretations/ and ./compiled-classes/) you would figure out the inputs for conalaun to process while it is in that 'directory state'. If all tests pass, then create a tar.gz file for all those state files, and call it (in this example, "state.1.tar.gz").

So state.1.tar.gz will contains all 'state' files/folders (./interpretations/* ,etc).

Then put that file (state.1.tar.gz) in ./testpackages/files/v1.000/ directory.

Now, CART (in this example) will have conalaun process the LUI, for each input (in this example (“input A” and “input B”).

conalaun's output (from 'cartlog()' output, discussed later), will be produced in results_cart_tests (and, if in 'create' mode, ./cart/testpackages/files/v1.000/v1.000.tar.gz).

If all tests pass, testpackages/files/v1.000/v1.000.tar.gz will not ever change (and it will be used to compare to, years in future). So we ran in 'create' mode and that tar.gz file was created, say it's the year 2020.

To have testpackages/files/v1.000/v1.000.tar.gz created, run this...

./cart.py v1.000 create

after running that, testpackages/files/v1.000/v1.000.tar.gz is created.

Now, years later, say you're on version 15.0 of conalaun. Well, we want to re-test all previous versions (under the exact same directory state as we did years ago).

Now, we run CART not in 'create' mode, but 'test' mode.
In 'test' mode ./testpackages/files/v1.000/v1.000.tar.gz is NOT created, but it is assumed to already exist (and CART will error out if it does not exist).

In test mode, CART will run conalaun and the output (“now” - say we are in the year 2035). Now, the output from conalaun is produced in results_cart_tests. So, now, in test mode in year 2035 (test mode for v1.000), CART decompresses testpackages/files/v1.000/v1.000.tar.gz (that was made 15 years ago), with the latest output (that is in results_cart_tests).
Any differences result in 'regression test ' failed ! Message appearing.

The command to do this is...

./cart.py v1.000 test

so when CART is run as in the above, it does **NOT** create the testpackages/files/v1.000/v1.000.tar.gz file, but instead just compares the output (that conalaun produced) in results_cart_tests with the results that are stored in the tar.gz file (from 15 years ago,), in testpackages/files/v1.000/v1.000.tar.gz.

When CART is run in 'create' mode, the file testpackages/files/v1.000/v1.000.tar.gz is created. In CREATE mode, the contents of that tar.gz file are identical to the non-compressed data in results_cart_tests directory so that you can easily look at the output without having to manually decompress the tar.gz. (They are the same because right after data is produced in results_cart_tests, CART creates v1.000.tar.gz from the contents in results_cart_tests).

But, in 'test' mode, the data in results_cart_tests may or may not be the same as what is in v1.000.tar.gz. In test mode, v1.000.tar.gz is NOT created (but instead just consulted – in test mode, v1.000.tar.gz was created years ago), but results_cart_tests was

produced 'just now' (the current test run).
If the two compare perfectly, then regression test pass, if they differ at all even in slightest way, regression test is reported as failing.

> *calls to "cartlog()". The first argument to cartlog has to be a string. That first string argument MUST BE exactly, one of the "areas" given in your configuration file.*
> *The second argument is also a string – and it is whatever test results you want to record.*

Looking at the configuration file again, (for a specific version –one *.conf file per version).

[state.1.tar.gz]

"input A"
"input B"

[state.2.tar.gz]

"my line X"
"my line Y"

This tells CART.... (IF run with "./cart.py v1.000 create')....

1. Decompress the tar.gz in /home/victor/conalaun/testpackages/files/v1.000/state.1.tar.gz into the root of ./conalaun directory.

2. then, have conalaun execute the inputs

"input A" and then "input B" (set "lui" variable to each one of those, one at a time).

Then conalaun.py calls "cartlog()" to report testing output (which ends up in results_cart_tests).

Then, CART cleans up the directory (gets rid of all files from decompressing state.1.tar.gz.

Then CART decompresses state.2.tar.gz, and then directs conalaun to process the "last user input" (lui) values of... "my line X" and "my line Y"

again, output going into results_cart_tests. (output goes into results_cart_tests when conalaun.py calls cartlog() – discussed below.

Question --- How does output from conalaun make it into results_cart_tests ? (and, IF in 'create' mode, into v1.000.tar.gz)?

Answer – by conalaun calling cartlog()

OK, so , how does conalaun call cartlog() ?

Question --- what is cartlog() for ?
Cartlog() --- is the means for conalaun to report the test output.

Question – how to call cartlog() ? (areas-- what is that?)

** NO actual executable Python code for CART testing is ever actually put into conalaun.py.

The only thing that is done is insertions of comments into conalaun.py.
Example...

#CART_INCLUDE_v1.000_file1.py

if you have that comment in your conalaun.py source code, it would cause CART to (as part of its creation of a temporary version of conalaun.py)... to...

look into (IF you were running for version 1.000)...

look into... testpackages/files/v1.000 to find a file “file1.py”.

The above comment line would be replaced with all the lines in testpackages/files/v1.000/file1.py.

This is very clean – because the only change made to your actual conalaun source code is the addition of Python comments !!!

If you were testing another version, say v2.500, and you wanted to include testing file 'myfile.py', then you would create

testpackages/files/v2.500/myfile.py

then, at the appropriate place inside conalaun.py, have a comment....

#CART_INCLUDE_v2.500_myfile.py

then, ONLY IF you ran CART with...

./cart.py v2.500 create

then and only then would the comment line

#CART_INCLUDE_v2.500_myfile.py

be replaced with all lines from

testpackages/files/v2.500/myfile.py

if you ran for another version (example “./cart.py v3.500 create”), then it would leave the comment line

#CART_INCLUDE_v2.500_myfile.py

alone.

So, in the above, the file “myfile.py” is called a “CART Include File”.

What does a CART Include File contain?

It contains calls to “cartlog()”.

Let's say you wanted to produce the results of doing a “pick highest” functionality. So you want conalaun to report the results of it doing that. So, first you'd create a file, maybe called “pickh.py”. (short for pick highest). Maybe there's a python function “def pick_highest” that we are testing.

So, inside conalaun.py, right after calling the function, we put a include....

```
pick_highest()
```

and then add your comment...

```
pick_highest()
#CART_INCLUDE_v5.690_pickh.py
```

this example says we are testing version 5.690 of conalaun. (so in order for this include to actually be replaced by contents of pickh.py file, you'd have to run “./cart.py v5.690 create”).

Now, create the file

```
./testpackages/files/v5.690/pickh.py
```

now, the contents of that file would be, for example...

```
for thevalue in somelist:
    cartlog(pickhighest, thevalue)
```

So we are saving all the values in the Python list variable “somelist” to the CART results directory, in a file called “ pickhighest”.

However, in order for this to work, you have to Define the “area” inside

```
./testpackages/v5.690.conf.
```

Inside the /testpackages/v5.690.conf you would have to have a line....

```
areas = pickhighest
```

(and whatever other areas you want, comma separated).

What are areas?? they are functionalities you want tested. The idea is to logically separate (into files), all results for one feature or perhaps one function, in the same file.

The two modes conalaun is called...

Mode 1:

CART calls conalaun.py (well, it makes a copy of conalaun.py and creates a temporary version of it (called ./conalaun-cart-temp.py) that is same as conalaun.py except with the CART include comments replaced by the actual cart include files (which live in testpackages/files/v1.000 directory – or whatever version).

>> Example calling from CART...

./conalaun-cart-temp.py -T 0:temp_cart_input.txt

When called this way, conalaun only gets input to process (user input strings) from a file that CART dynamically creates for that test run. conalaun does NOT read in keyboard input for strings to process. This is when conalaun is run in "Testing mode"

Mode 2:

when called as in...

./conalaun.py

Then conalaun reads strings to process from keyboard input only. NOT from a file. This is the "normal mode" , or "production mode".

5.0 Important Note about calling cartlog()

When calling cartlog()... do NOT write the output like this....

```
For  x in  var_1:
       cartlog(somearea,  "x = " + x)
       for y in var_2[x]:
            cartlog(somearea, "\t" + "y = " + y)
```

Instead, have EVERY LINE COMPLETELY qualified....

```
For  x in  var_1:
```

```
    for y in var_2[x]:
        cartlog(somearea, “x = “ + x + “, y = “ + y)
```

This is so that, after CART sorts all output files in results_cart_tests, that the diff will work.

So again, we will NOT require that you sort before calling cartlog() --- nothing in your testpackages NOR conalaun.py should care about sorting.. CART will do all sorting before comparing last run with previous expected results file.

Calls to cartlog() should, EVERY TIME, have ALL relevant values of variables that pertain to that test, right from the “main call” (from code that is in “main” – that is, outside any other function). Every call to cartlog() should include ALL relevant variable values and never exclude the value of any parent variables. The only exceptions are: (1) the directory state database ID and (2) index into “LUI” number... in other words, the 2 directories right under results_cart_tests.

The results_cart_tests directory

If you had the following in your *.conf file. (testpackage configuration file)... (only one area defined but can be more, comma separated)...

areas = myarea_1

[state1.tar.gz]
“xxx”
“yyy”

[state2.tar.gz]
“jjj”
“kkk”

Then if you ran CART in create mode, you look inside our results_cart_tests directory...

first directory under results_cart_tests is the directory state ID, and second directory is the input number.....

results_cart_tests/0/0/myarea_1.res

this is data that is sent via “cartlog(' myarea_1', ...)
This is for when the directory state is as inside the state1.tar.gz file (after CART uncompresses this file into the root of conalaun, then runs input # 0, “xxx”.

results_cart_tests/0/1/myarea_1.res
same as above but when “yyy” is processed (the “1” in “/0/1/” of the path... index 1... or 2nd input line (“yyy”)...

results_cart_tests/1/0/myarea_1.res

here the first directory under results_cart_tests is “1” .. which means index 1.. or 2nd directory state (state2.tar.gz)... and /0/ in path means index 0.. or 1st input line (“jjj”). So this is all the output of 'cartlog('myarea_1',.....)' when under that state of directories and files, and that input. (first.. or index 0.. 'jjj')

results_cart_tests/1/1/myarea_1.res
same as above... first '/1/' in path means under directory state given by contents of state2.tar.gz, and Second /1/ means input index 1 (== 2nd ---> “kkk”).

Current mode

What if you just want to run conalaun from within CART but do not want to disturb the current directory ? Well, you run in “current” mode ! If you call cart.py with 2nd parameter of “<any integer>” (no quotes) then it runs in current mode. What happens in current mode?

1) CART does not touch the directories – (the conalaun directories – tls_csv, compiled-classes, etc),

So, if you don't want to have your conalaun directories changed, you just want to run conalaun from within CART (so that you can use “include” files and use “cartlog” to output test results), but do not want to touch/change the state directories, run in current mode.

Let's say your v1.000 configuration file contained..

Line contents	comments
# testing file	All lines are ignored that start with “#” (no quotes).

[state_A.tar.gz]	Index 0 file state db. – This gives name of a “File State Database”
“input A” “input B”	So, CART will send to conalaun (via setting variable “lui”) the two lines “input A” and “input B” (but first establishing the directory state as given in state_A.tar.gz.
[state_B.tar.gz]	Index 1 file state db.
“input C” “input D”	CART deletes all state files (tls_csv, compiled-classes/* , etc), then unzips state_B.tar.gz and sends “input C” and “input D” to conalaun.
[state_C.tar.gz]	Index 2 file state db.
“input E”	

So, if CART is called via..

./cart.py v1.000 test

Then CART will test conalaun under all 3 of the above File/directory states... state_A.tar.gz, state_B.tar.gz and state_C.tar.gz.
Then, when done, it will clean the directory (delete tls_csv/* , compiled-classes/* ,etc)

But if you run CART as...

./cart.py v1.000 2

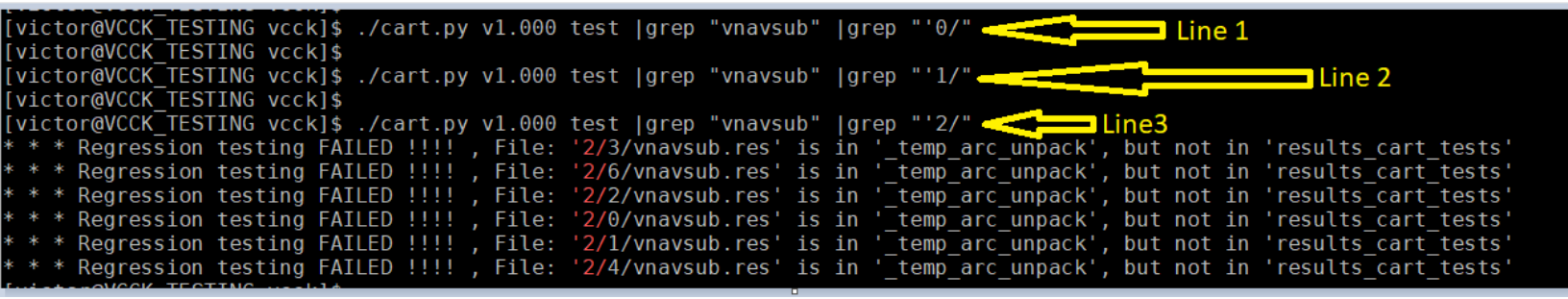
so, above – uses index 2 file state db in v1.000.conf

Then CART will run conalaun with _CART_FILEDB == 2, and NOT unzip state_C.tar.gz... instead of using state_A.tar.gz, state_B.tar.gz, or state_C.tar.gz, it just uses the current directory/file state files as they are right now, And does not do any “cleanup” after... it just uses the tls_csv, compiled-classes, etc AS they are, and does not remove them (unlike in 'test' mode where it cleans up the directory after each file state db).

Any logic you have in your responder (INCLUDE files) that look at _CART_FILEDB will work the same whether run in 'test' mode or 'current (\d+)' mode.

Checking which T.C. (Test cases) passed and which failed

Say if you wanted to run all test cases for version 1.000 and view all results for FILE STATE DATABASE index 0 (so the first File state database in v1.000.conf)...



So in the above…

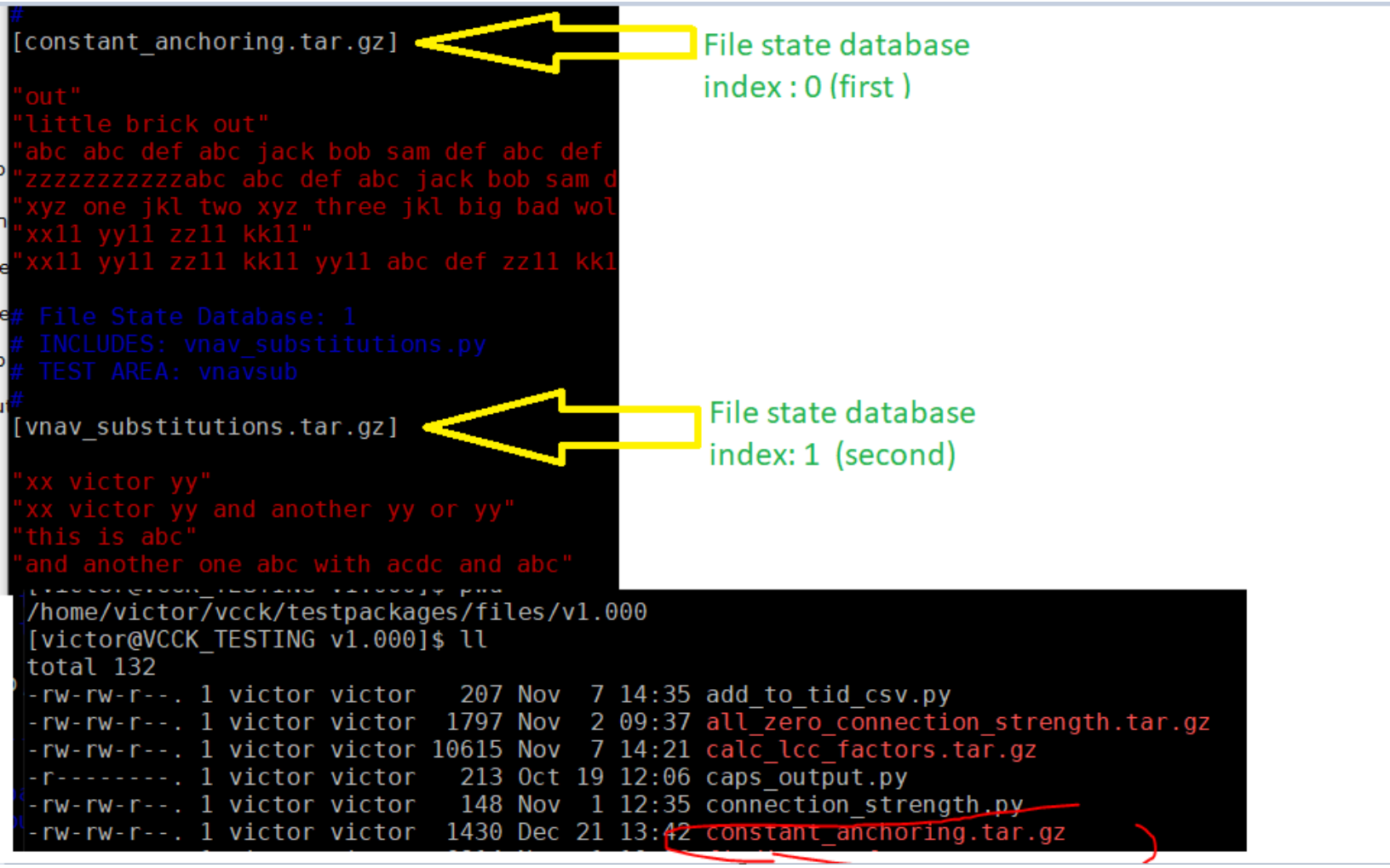
Line "1" – we are viewing all results for failures for TEST AREA " vnavsub" (So all lines that call cartlog('vnavsub', '.....')).

We see no output for Line 1 – so we know that all TC (test cases) for File state database index 0 (so the first file state gz file –first square bracketed filename in v1.000.conf) ALL PASSED.

We also see that File state database index 1 (line 2 .. grep "'1/")… so the second file state database also has all passed tests for test area vnavsub).

So for all inputs for file state database 1 and file state database 2 – for test area " vnavsub" ALL PASS.

But there are FAILED test results for file state database index 2 (3rd file state database given in v1.000.conf).



File state database index 0 (first) means... system will unzip and untar “constant_anchoring.tar.gz” into ./conalaun so that all files “come into effect” when testing the corresponding inputs (“out”, “little brick out”, etc).

6. FILES_TO_SCAN feature

This feature was added in January 2023. So, originally, CART would only scan conalaun.py for lines that had "#CART_INCLUDE_<version>_<filename>.py. CART replaces that "include line" with the actual file contents from ./testpackages/* directory. Now CART can do this for any other Python script file, not just conalaun.py. In order for this to work, you must tell CART. Right now we don't do this in a configuration file, but instead we just open the cart python script and simply locate, towards top of script, the list variable called FILESS_TO_SCAN. For any file that you want CART to process the "include" files for, just add to this variable. Example below for adding "term_manager.py".

```
FILES_TO_SCAN = ['term_manager',]
```

This is a Python "list" type variable, so add as many files you want, seperated by commas. In this example, CART will create an temporary alternative version of this file (called "term_manager_temp.py") and replace all refernces to "term_manager" with "term_manager_temp" in conalaun.py. An important thing to remember, is for each file you must add a line:

```
cartlog = None
```

to the top. So in this example above, since we have added "term_manager" to this FILES_TO_SCAN variable, we must edit term_manager.py and add "cartlog = None" to it. Reason being is that that variable must be there to receive the reference to the cartlog() function in conalaun.py - so that imported file can call cartlog. All names of files in FILES_TO_SCAN should contain only lowercase/uppercase letters and underscore.

```
[victor@fedora cart]$ more term_manager.py
import os
import re

cartlog = None
.
..
..... rest of the file....
```

Above - showing the first few lines in term_manager.py - notice we have "cartlog = None" - do this for all files that you mention in FILES_TO_SCAN.

CART now includes a directory called "-1" (no quotes), in the results_cart_tests/. What is the purpose of this directory ? This directory contains all deposits of "cartlog()" that are done before ANY of the "ico" files are processed.

For example, if a given FSDB contains only one ICO...

```
[victor@fedora cart]$ ll cart_raw_ico/
total 4
-rw-r--r--. 1 victor victor 116 Jan  8 13:41 ico0-a
```

Then, when CART runs, if you get the results below...

```
[victor@fedora cart]$ find results_cart_tests/
results_cart_tests/
results_cart_tests/lexicon_dict
results_cart_tests/lexicon_dict/-1
results_cart_tests/lexicon_dict/-1/lexicon.res
results_cart_tests/lexicon_dict/0
results_cart_tests/lexicon_dict/0/lexicon.res
```

Then, the red line, (with "-1") is the directory where all output from "cartlog()" calls have gone, BEFORE processing any "ico" files from cart_raw_ico/. And the green line (with "/0/") is where all output from cartlog() calls have gone after processng first ICO (zero, [0] is first ICO). And of course results_cart_tests/lexicon_dict/2/lexicon.res would be where all cartlog() output goes after CART has CONALAUN process the THIRD input (third is index [2]).

7. The "-D" switch

If you have NO idea what the heck is going on, and want to Single step through the modified version of CONALAUN (./conalaun-cart-temp.py) you can do this with "-D".
So, if you run like this...

```
cart test -D
```

Then, CART will do all the setup, including installing the FSDB, and will create ./conalaun-cart-temp.py (and all other modified versions of files in FILES_TO_SCAN), but it will NOT actually invoke conalaun-cart-temp.py. It will then stop and tell you to open /conalaun-cart-temp.py in your favorite single-step debugger (currently Visual Code). This way you can debug what's going on. Make sure, in your debugger, to specify command line arguments to the script.

Example, CART will display something like this...

```
        --- PAUSED ---

        OK- Please open './conalaun-cart-temp.py -T lexicon_dict:temp_cart_input.txt' in your debugger (VS Code, or other), with the
        specified arguments, do your debugging, and press enter to continue. . .
```

So you can now use your favorite IDE single-step debugger to single step through ./conalaun-cart-temp.py. CART is showing you above the arguments that it should be called with:

Argument #1	-T
Argument #2	lexicon_dict:temp_cart_input.txt

So setup your debugger to supply those two arguments to the script and happy debugging !!