-
- 

-

Hybrid VFS ENV-VAR Library Documentation 1.2

::

::

https://github.com/lastforkbender/staqtapp

*Staqtapp 1.2 is a all-in-one stand alone vfs env-var module. No third-party libraries are needed to use staqtapp.py module or modules it makes for other functional return features.

rcttcr5@gmail.com

UPDATED: SAT, OCT5 2024, updated 'next.omega.*' ctpk-span bracket use info

---

STAQTAPP V1.2 IMPORT FUNCTIONS USAGE:

----> **makevfs(vfsFileName, directoryName, folderName)**, builds a .sqtpp vfs file
in current ../staqtapp/* directory; @vfsFileName cannot include ':' chars,
tqpt file auto inserted to sub-folder naming of @folderName & is set path.

----> **setpath(vfsFileName, directoryName, folderName)**, sets the working path for
the .sqtpp vfs file @vfsFileName, inline with @directoryName + @folderName
paths given to a env-var stack/tqpt file. This can also be done by opening
the .stg file in the current working module dir .../staqtapp1_2/ where for
a path example: 'vfs-A:dir-A:folder-A:subFolder-A:tqpt-A' can be editted.
A subfolder name required and only dash chars - allowed, not underscores _.
By use of this function is assumed @folderName is same for subfolder name.
(This wouldn't be true for a rev9 built vfs file having many dirs+folders.)

----> **addvar(varName, varData)**, adds variable to path set vfs tqpt file. @varData must be tagged with '@qp(...):' for staqtapps unique read parsing routines. Can be as many @qp(...): tags as needed, with commenting in between. Read as comma separation for a list return when more than one. This function does not change variable data, see changevar to edit stacked variable datas. Is suggested to use lockvar after adding a variable with this function. *****If storing a lambda function as a env-var runnable, only use one @qp(...): tag for that lambda function use, including the lambda name & equal sign for the function. @varName is ignored in this case. You can also skip qp tagging for this type env-var storing of which gets @q|p tagged anyway. See lambdavar(). Example: @qp(my_lambda = lambda x,y,z,r: (x*z/y)/(r)): *****

----> **corevar(mode, varName, booleanList)**, for storing & reading boolean type list. If @mode == 1 is write, 2 is returned boolean list or 3 returns the special -- delta RLE encoding that was saved to tqpt stack, a tuple(s) list encoding. (The parity bit[last index value of stored boolean list] is always flipped on read.)

----> **appvar(varNames, varDatas, varLocks)**, adds env-vars by lists only. All params must be list type except for @varLocks, which can be None if not needed. If the list lengths between @varNames & @varDatas is unbalanced then adds @qp(null): to the tqpt stack as a value for env-var; and also if no @varLocks list element found then no lock block is made in the tpqt file for env-var. @varLocks correct example index ..., 'myvar2:fnc1,fnc2,fnc3', ...] whereof this type :, format to lock names must be to properly add a lock block for env-var. When lock block needs one fnc name or other name, no comma use is needed. If you do not need a value or lock block for a env-var added, simply None as that pair [element] bypasses any checking needed, otherwise can be errors. (Recognition of lambda function strings is not done with appvar, only addvar.)

----> **listvars()**, returns stack list names of current env-vars in tqpt vfs source.

----> **joinvars(newVarName, varNames)**, as a list of str names @varNames, this will join together those env-var datas at the qp(...): taggings merged into one new env-var @newVarName. Precaution assumed, if doesn't find a valid name listed in tqpt env-var source then does not add to joining. If @newVarName is a stalked env-var then will merge all spawned vars from a stalkvar use. (No spawned vars are removed from svvs listing or the tqpt vars stack.) No join for any env-var data having chars ☆ in it, will raise exception that it is a smv-tree type error that cannot be merged with other env-vars. This is powerful feature in connect of stalkvar function env-var spawned data use, of which is possible to do multiple layered recursion from just a singular event loop. However in some adv cases, would need the raw @qp(...): tagging to combine ast converted smv-tree dict type for this mathematically, via some other dict type conversions or abstract reductions of paired branching, see loadvar_stx.

----> **changevar(varName, newVarData)**, change @varName data to @newVarData if @newVarData has proper @qp(...): data tag(s) found. To change the names of env-vars in a vfs tqpt source, see renamevar. Duplicates only allowed for the stalkvar() feature. *This function should be used with lockvar & keyvar. If the @varName is a stalked env-var, the original stalked and not the other incremented spawned vars from it, then this function will check if any of --

the incremented spawned var data matches the @newVarData string. If it does then any spawned vars matching @newVarData are removed; and any spawns left are renamed of a new incremented naming, re-named to a number order. This changes no env-var data. Becomes a find & remove operation allowing a programmer more advanced control over stalkvar spawned env-vars, in a larger scope use of stalkvar env-var spawnings in the tqpt source. If any spawned env-var is removed from tqpt and svvs sub file listings, is also removed @ a lockvar tpqt listing if found. Will not do any of this if the stalked var is read as having only one spawned env-var. Raises a non-allowed svvs error. If this function switches to this type find & del, returns named list of the spawned env-var(s) it deleted or else returns None.

----> **addtree_stx(treeName, initialTreePathList)**, adds a initial dict type tree structure to vfs tqpt stacks for deep spahk-mv type tree edits. @initial-TreePathList is simply a list elements of strings or int values. Chars not allowed @initialTreePathList string element(s) are \n,':{} and ☆ The --- other functions that edit this type, all stalkvar() type similiar actions.

----> **addbranch_stx(treeName, branchId, newBranchId, newBranchValue)**, shifts, replaces, or spawns merged branch for a mv-tree. Behavior of mv-tree branch edits are in exotic relation to stalkvar. The examples below show this. The first dict key cannot be changed, will raise exception. Is 'a' in examples:

Example A:
smvt-state {'a':{'b':{'c':'vl1','d':'vl2'}, 'e':'vl3'}}
.addbranch_stx('smvt_tree_a', 'c', 2035, None)
results ---> {'a':{'b':{'c':{'2035':'knt'}, 'd':'vl2'}, 'e':'vl3'}}
'vl1' now a nested branch, not a separate value

Example B:
smvt-state {'a':{'b':{'c':'vl1'},'e':'vl3'}}
.addbranch_stx('smvt_tree_b', 'b', 2056, None)
results ---> {'a':{'b':{'2056':'knt'},'b_2056':{'c':'vl1'},'e':'vl3'}}
'b' hoplinked to branch&value, however not full link replace parse @'b_2056', feature serves as replace options and other features

Example C:
smvt-state {'a':{'b':{'c'},'e':'vl3'}}
.addbranch_stx('smvt_tree_c', 'c', 2032, 2051)
results ---> {'a':{'b':{'2032':'2051'},'e':'vl3'}}
'c' replaced by {'2032':'2051'} branch

----> **getbranch_stx(isAlf, treeName, branchId)**, returns a found branch value or None. @branchId can be either int or str for param usage. @isAlf is for a return list of all original branch key(s) + value(s) to an @brnchId merge. The return list format as ['br_br:br:bv',...], whereof first after :orig: Avoid using alf as much as possible on large mv-trees. If use, @branchId= None. A tip to this type dict tree type: Record env-var any adds done last to a branch, that can be then known replaced or merged with other branch. The staqtapp mv-tree type can be difficult to use at first, however opens a whole other world for *indirect & *adapt recursion unions with stalkvar.

----> **lockvar(varName, fncName)**, adds a function name or else associated to a var

name in a tpqt type file. This can then be used with keyvar to gate what is allowed to edit the env-var. There no real security/math to it like the Koch version Staqtapp, however is very useful in preventing any env-vars mishaps. @fncName can be either a string; or, list parameter of string names properly. To delete vfs entries from a lockvar use, see lockdel to do that effectively.

----> **locklist(varName)**, returns listed fnc/etc. names for a lockvar tpqt vfs entry.

----> **keyvar(varName, fncName)**, returns True @fncNm if allowed to edit @varNm, is a listed key to @varNm. Other very useful conditions of this include --- @fncNm being a env-var name of a spawned var set, from use of the stalkvar function, with multi-processing. Unlike Staqtapp Koch version has no vfs -- tpqt directory restriction features via a directory split route linkers.

----> **revar(isNewVfsPth, newVfsFileName, newVfsDirName, newVfsFldrName)**, replicates a new .sqtpp vfs file in the .../staqtapp/ folder, having only the env-var(s) from the current set vfs file that are linked to a tpqt lockvar entry. @isNewSetVfsPath as True will reset current set vfs file path to the now new vfs file and it's tqpt path. (If this function is called and the ---- current set vfs file has no tpqt lockvar entries found a exception is raised.) **This function is used for advanced rev9 staqtapp features in building custom env-var library modules from within a staqtapp1.2 vfs thru self-replications. Can respawn any missing stalkvar() env-vars to a new self-made svvs sub file. The svvs file from the current set vfs file is lost on the new transfer.**

----> **lambdavar(lambdaName, inputVars)**, function is an original of Staqtapp1.2. See addvar() for storing a lambda in tqpt stack first. Of the instruction sets connected to lambdavar: Makes sqtpp1_2_LMB.py mdl, converts lambda function in tqpt stack to a parameterless lambda; params made to slots attrs sqtpp1_2_LMB.py. Once the parameterless lambda there is then directly ran if called again, no param/body conversions or edit of the module. With inputVars as str element list only. Example: ['x=4','y=2,3,4'] If is use of commas then is converted to a list when the slots attrs are set before run of slots lambda. If string elements after the = are all numbers is converted to int, both for a list value(s) or single value. The created module in the working current dir of staqtapp.py, with a import math declaring in the module. Because this function leads to all lambda parameters added to the slots attrs, keeping the parameter names generic to a lambda when use of addvar() bypass edits & slots lengths.

----> **lambdalist(isComplete)**, returns stored lambda data from a vfs tqpt stack. If is @isComplete=True then returns a list of the full lambda functions. Other option as @isComplete=False returns a special formatted string of the lambda func name(s) and param(s) so is much easier to work alongside a lambdavar() call

Example:

'@.:my_lambda1,i=,p=,l=,@.:my_lambda2,x=,n=' whereof separated ",@.:"

----> **removevar(varName)**, remove env-var of tqpt stack & any tpqt lockvar block. Will not delete a stalked env-var or a dark env-var, raises a exception that the @varNm parameter choice is connected to a vfs dependency function only. This is a dangerous function; well not as dangerous as us, to rely on for env

var use. It is much better to just rename a env-var and go from there, when then use of changevar function. Keeps your env-var stack smaller and more easily controlled instead of wild adding until you don't know what to remove. Which could then result in bad collisions of a spawned or dark env-var group if using those to protect a hidden database of passwords or drone config file.

----> **darkvar()**, mirrors any removal of a stalked env-var, placing the then removed spawned vars datas to a assigned address, that is all the incremented nums of the spawned var namings to be removed. After this assigns a pointer to the address by dash, example 56791218-3095903, a rand 7-digit palindrome num that will then be of a vssv sub-file content. A darkvar, or dark env-var does not repeat any char when saving the data of the removed vars to a @qp(...): tagging. It only interacts with other env-vars by any palindrome sequences of those chars, if is visible/matching @ the new address to env-var(s) data. This type env-var uses menorah base numbering, non-zero by shifting the pointer to a most probable next match in chars of a recent env-var data viewed. The slots attributes specific to this env-var are _sf_sLstX, _sf_sStrX and _sf_sIntX. To call on this non-parameter method it will react either two ways: One, _sf_sLstX is of a visible relevant list of env-var(s) in the tqpt source and is spawned set of env-var(s) with at least one tpqt lock block visible. This will cause a assignment probable str to @_sf_sStrX and a new palindrome pntr @_sf_sIntX. Or two, then uses what is done by the first to build a new env-var having --- data of a probable set of decisions chars relevant to a new spawned var add. The rev9 features of this library to be added later, uses this to vastly alter any module building it does from another, by decisions in conditionals choice; by lengthing a klf id depend string or splicing it to lesser address probables. (The Staqtapp-Koch does this similar type env-var actions by numerous mdls classed Seten modules, yet for random length key padding in vfs-dir routing.) The type mathematical looping of this function with palindrome address shifts is tri-half looping, just as seen on the Menorah. Where a pairing loop is not matched to a static center necessary pairing(Base10), however of LR.R or RL.L rotation, of which either larger loop with middle exact to 1 | smaller " to 1. (UNDER CONSTRUCTION)

----> **renamevar_stx(varName newVarName)**, function is important one to the data structure of this new staqtapp one version. It actions a normal renaming of a env-var if does not conflict with other namings and can also change the ---- spawned var(s) naming(s) from a stalkvar() use just as staqtapp koch version. To do this there must be more than one spawned env-var from a stalked var. By @newVarName being a int & @varName being the stalked var, the renames of the spawned vars take order increasing from the int value; but if negative number|zero for @newVarName will remove the spawned var(s) until reach of int one or is only one spawned var left..if negative int, replaced positive. Explained removes spawned vars by nearest increment of spawned names. With negative param doesn't rename individual spawned of env-var namings, the renaming is central of svvs list, shifted left after any number of removes. This is tricky at first to use with negative del order, however a very powerful use when matching spawned env-vars to a replaced branching of smv-tree use. Returns 1 if renamed env-var, 2 if renamed spawned var(s) and there is only -- one left spawned var of the stalked var and 3 if renamed spawned env-var(s).

----> **findvar(varName)**, returns True if variable present to set vfs tqpt file.

----> **findvar_stx(varNameList, stalkVarName)**, findvar function for multiple found result list return. If @stalkVarName not None then returns the list of the spawned incremented var names, with each s-var name of a additional equal detail = <if @varNameList[ ]'s data ?= @incremented_var's data> which will be either =1 or =-1. List length @varNameList determines return length for both search options. This new feature is used for recursive analysis or chained events hidden from pivot conclusion or escape type params forming. [If stalked var will have thousands of spawned naming see modvar for lambda spawned res-type slots attribute class caching before calling this method.]

----> **vardata_stx(isRegex, varNameList, search)**, returns list type from search on the @varNameList if @search terms found in a env-var data; returns names of env-var(s) from @varNameList, not the env-var data. @isRegEx instructs the @search is a regex type pattern search. Can search mv-tree stx var types of their keys & values for a listed name return also if @search pattern found. Either way, regular find or regex of no found results returns a empty list.

----> **loadvar(isAllNumbers, varName, mode)**, returns env-var data from @varName. @mode either 'd' for deque type return or 's' for a string type return. All tqpt env-var data parsing is smart on if is a list return, has numbers with decimals for decimal type returns and etc. via @qp(...): env-var data tags. Be aware if @isAllNumbers set to True then will return list within list if reads multiple qp tags having commas. This will also convert to decimal if tqpt_spdr() finds use of '.' with numbers for a deque type returns. (By use of char * only @varName, load most recent added spawned var from stalkvar() function use. This spawned var tracking found in the tpqt lockvar stacks -- with the special header naming as '___SQTPP___MRSV___' and can be editted.)

----> **stalkvar(varName, varData)**, keeps the original env-var as a static read only var and makes a new but same @varName, with extension '_ ' as a numbered increment _1~ . If the @varData matches the original, is no longer a stalked env-var and all the incremented spawns are removed from both the tqpt & svvs set vfs path files contents. This feature is used for recursive analysis or chained events hidden from pivot conclusion or escape type params forming.

----> **lockdel(isRemoveAll, varName, fncName)**, removes lockvar entries from a tpqt block of names that are associated to a env-var name. The @fncName param can be str or a list for the entries to remove. The previous Staqtapp 1 ver. had a log function name when a keyvar is called option. That is no longer in use. If @isRemoveAll then the entire tpqt block is removed for @varName. If you need a list return of what @fncName exist of a tpqt block, see getlocks. Once again, there is no security measures added of this function. To have full password & tor like vfs dir encryption security for this, you must have the Staqtapp-Koch version. The full ver is not open freeware and is the most secure env-var library on earth, that does not use any online third party to to store any env-var data or passwords.

----> **registry(isRead, keyName, keyData, harpSchema)**, stores registry keys in the auto-generated slots perform module sqtpp1_2_REG.py for integrated lambda functions use on registry value(s) normalizations. It does this by a schema of custom coding, see REGISTRY_CALLS.TXT example on write of harp-schema. All registry keys added|changed must have a assigned schema, that is already

stored in the registry:

EXAMPLE ---> Adding or edit of a registry schema:
sqtpp.registry(False, None, None, '.../my_files/my_reg_schema.txt')

EXAMPLE ---> Adding or edit of a registry key:
sqtpp.registry(False, my_reg_key, [1,'212',434,'11.1'], 'my_reg_schema')

*@keyData can be of list, str, int, float, complex, set and bool but no nested list are supported with this type registry system or bytes/bytearray

Staqtapp's registry system is quantum like. When a registry key is called on to be read and return a value; the static stored value of that registry key, processed inline down the schema it is assigned to. If a lambda function that is currently being applied to the now current value of the registry key is not applicable to the current value, errors or is non-observed then the previous now state of the registry key is not changed. Is filtered thru the harp-schema that may not change the registry key's value at all, however of the more adv pojishon optional features, this doesn't conclude is a same identical. Because the harp-schema observed the registry key and of that now schema is changed to reflect the current expansion of any new observe of the registry key. (Only specific pojishon optionals do this, normal use of registry will bypass this)

As such staqtapp first checks @harpSchema is a path, having '/' chars or not. If adding of a registry schema or key then returns True, else an exception. The registry schema is fully checked before added to the module registry. Of this type registry system the schema is in control of how the registry key's data will be returned when @isRead=True, meaning registry key can be made to take on a more analog occurence when compared to static file data or other.

EXAMPLE ---> Return of a registry key's data:
key_return = **sqtpp.registry**(True, my_reg_key, None, None)

---OR---

EXAMPLE ---> Return of a registry key's state:
key_state = **sqtpp.registry**(True, my_reg_key, my_data, None)

Of the later example, a direct compare check on the outcome state of the key in match of @my_data. If match returns True else False. This option on read of a registry key is of staqtapp's pojishon quantum env mv-tree state features and heavily used of some pojishon optionals, when called on with a stored vfs file.

*In use of adding a registry key as a set() or complex() declare as a string to avoid certain issues of these types. The registry module sqtpp1_2_REG.py will recognize & process those to proper convert type without any ast problems. As a 'set()' for convert does not support complex elements in the conversion and no string quote declaring for text elements in a wrapped string as shown:

EXAMPLE ---> 'set(5.4, True, 8, txt)' ---or--- 'complex(3.141592653, 8)'

----> **pojishon(mode, varData, varName, dirList)**, gives access to Staqtapp's vfs dir/embedded file system located at the top directory of a sqtpp file. Includes -- 28 functional calls in 6 categories: next, jump, tqpt, tpqt, watch and record.

Below are the functional aspects of these categories and optional parameters. Argument *mode* is a str that includes the dot refrences and any optionals. Of @varData and @varName can be either str or list type; @dirList the folder tree path to file(s). Env-vars are as files whereof @dirList = list = [mount, dir] and @varName the filename as positional read/write file in vfs mount/dir seq. *In a single mount only can a env-var/file be moved, edited, copied or removed. These positional file features allow vast options with env-vars within the vfs use across many vfs directories & files for any circumstances need making - Staqtapp 1.2 the most versatile env-vars py module publicly available. Some of these dot functional modes include integration of normal sqtpp env-var uses.

							NEXT																																																																		

(1) **("next", required, required, required)**, makes new mount, new directory and a new file. *Of no dot refrence instructions and has no optionals.

(2) **("next.cast.*", non-required, non-required, required)**, copies last env-var(s) file added to a new directory. *dirList[0] argument string must match the last mount name used or throws a mount placement exception; and argument dirList[1] does not match last directory used. If new dir @dirList[1] is not currently a dir in mnt then will add the new dir.

(3) **("next.ghost.*", non-required, non-required, required)**, from last file added compares it's data to any file @dirList path. If a entire line in the most recent added file matches a entire line in @dirList file, then an add all matching lines to a new file in same directory of @dirList[1] whereof index[0] is mount; index[1] is directory. The @varName parameter is ignored and the new file's name is of a number of lines #. Is a 1rst match basis on any file then scans the same file for more line matches.

EXAMPLE: lastfileaddedname_#numberoflines

As example above takes on last added file name and number of lines that will be a number, not #numberoflines. If at any time this action repeat and the naming above matches for a new file to added in same directory, the file is replaced. Has to complete match. Will not overwrite a file having a different number of lines. In other words the original *host* file of this add file feature is a unknown in Staqtapp's procedures -- when next.refill or next.remove is used on that same mount/directory. (The setting for the last file added in vfs settings is not updated to the mnt/dir/file of the new match lines file just added, is ghost file) One to be careful with. If the @dirLst is pointing to directory where there is a file of a same naming, however not a lines match file then gets over written just a same anyway. Also to remember the last added file in the vfs settings is not there for this type file added. Any optionals used after this command sees same unchanged m/d/f setting.

(4) **("next.omega.*", required, required, non-required)**, from a last dir use this pojishon next call works. It's main emphasis is a category pack or "catpack" built to staqtapp's registry, from a parsing of the files in the last directory used. The @varData argument is used in a time append parsing when compared with the files in the dir. Below a list of special character sequences that are used with @varData, a str or list str type. If using as a list then list will be joined together for file searches.

Should be noted when Staqtapp processes the search tokens for @varData, any mistypes/errors are ignored, continues to assemble an searching seq. Searches can be as complex as you need it, however if is complex search use a list approach for your matches and it will be less error prone.

CTPK-SPAN SEARCH BRACKETS(next.omega):

1. {...] read as a forward scanned data group for comparisons
2. [...} read as a reverse scanned data group for comparisons
3. }...{ read as a neutral scanned data group for comparisons

4.]...[read as a neutral scanned data group for comparisons
   *use of this now/neutral matching is numerical seeks
    and ignores all other chars besides digits/numbers,
    is not the same as number 8.]#[of these char uses
    the added number(s) are as string to a list element

5. |:{: read as special numerical block assign for reg-key(s)
   *can/will assign float point numbers if passes format
    otherwise assigns int as a list element to reg-key

6. |:[: read as special numerical lists assign for reg-key(s)
   *numbers can be separated from (...) of when searched
    meaning data in an file has such formatting applied
    and as long as the found number is in each set (...)
    at the begin then comma use as example (match,#,#,#)
    as same above, can assign floating point numberings
    and will be added to the registry as a separate key
    with it's key value a list type converted from a set

   *if no found (...) set format then ignores the find
    altogether, found number not added to any reg-key

7. }#{ read as a skip next # of times for any scanned data
   *not a gray area, is simply mirrored back to number 3
    of these char uses if is other chars besides numbers
    (NOT FULLY IMPLEMENTED)

8.]#[read as a reverse next # of times for any sequence(s)
   *is gray area, if other chars besides numbers then is
    number 4. of these listed chars uses | is vice versa
    in between from specific optionals that use darkvars
    | has merged more than one stalkvar to an locked key
    (NOT FULLY IMPLEMENTED)

EXAMPLE: {find txt]0[find txt, add rev}|:{:12]1[34 is txt}1{skip txt]

*{, [, |:{: and |:[: are as key symbols when @varData is recorded into a list for matching against all files in last used dir; when any of these begin bracket uses encountered then expects and end of] or } for closing the search parameter wether txt | numbers

**neutral scan/found data groups are added first, forward next and any reverse scans then added behind a last forward. If there is no forward scan group added yet then acts as a forward, or limbo registry key element which will begin with char-seq '^;^'*

The pojishon omega call will write a new env-var to the tqpt stack and create any necessary registry key(s) in process of scanning the file(s). Type reg-keys take on the same name @varName used for assignment @ the env-var added. If multiple registry keys(catpack keys) are stored in the registry module then @varName is followed by _catType then an _# of order to which added. @varName's env-var added to tqpt stack is the same for @varData, it's value not changed. When this next call is done it changes the last file used setting, in the last used directory, to the file it last found matching and made a registry key. The schema of each registry key(s) added are include of this generic harp-schema:

```
__sqtpp__default__omega__key__ = (
    __dok__type: (
    type: ( list,
    validator [
        lambda x: if len(x) > 0
    ]
    )
    )
)
```

**this harp-schema can be edited, see registry() & REGISTRY_CALLS.TXT; the name of this schema will be '__sqtpp__default__omega__key__' and if Staqtapp doesn't find it, stores the same schema as above on read of any pojishon omega search category keys it finds in registry mdl when it's executing a omega call with optionals or other later calls*

As you can see is very generic with only one lambda validator, checking if list is of any length. The registry key(s), if any added by this next call are all list basis via scans for comma separations. Meaning with of above char sequences uses, in comparisons of each file in the directory, if any comma uses found in between brackets then is a separate element added to the registry key's value. This pojishon next feature is geared toward embedded positional traversing a file read or write with other registry involved pojishon features like watch and record, however is of other important uses similiar to time crucial added/edited reg-keys in parallel to a env-var that added it there from a set mnt/dir/...

								OPTIONALS																																																									

****using a optional will be the final return, either True or False applied; or for the quantumf optional, a discrete math based lambda function str*

****arguments separator must be ";" not "," or will result in exceptions and any use of ; in a 'string' also results in exceptions, see below*

****optional's params all string and requiring normal type declarations, if list required param then list=[..., ...] with the brackets and all else also string for #, str or #[index] a number list needed; if is a string or string list then use of ' is needed between " meaning cannot use ; or ' in a 'str', is formed by ast literals*

EXAMPLE:

("**next.cast.open**(['@qp(data1,1X):', '@qp(data2,2X):']; [3,4,5,6])", ...

****if a raised exception happens with parsing or instructions from an optional, the dot commands before it already done and saved to the .sqtpp vfs file*

(1) **.open(varDt=list; occurence=#[index]) | .open(varNm=list; occurence=#[index])**

If @varDt then requires @qp(...): normal staqtapp env-var data tagging. This will cause staqtapp to search for qp tag env-var data from @varDt and then compare it to the env-var data being set in a directory. If there is a match in data seq, anywhere of the new set dir/file data compared against the @varDt found in stored qp tag env-var(s), then those qp tagged env-var(s) take on any further compared/found data of a same. Will add to qp tagged env-var(s), from the file, starting at the end index of find up to a newline encountered or the char scanning method finds a char set of ":|:". If there is commas found from scanning then is separated into @qp(...): taggings. Char sequence ":|:" marks the end of a file and if no newline encountered is an add all data until end of file. See 3rd paragraph for occurence detail.

The same @varNm choice env-var search, however search & find method is based upon match a name category: Is first find env-var name match from varNm element then on first match of data sequence comparing that is of a len matching @occurence element, of comparison between newly added file data & the tqpt env-var data then/when the new data from file's data is add to the env-var @occurence param a same for @varNm param choice though doubles as scan find & scan cut. If lack of a pairing occurence index value for matching, a raised exception. With @varDt use, single element str of "*" for that list ignores @occurence length cut-offs. Works same for this param choice yet a add all data only of newly added file; if scans data with newlines or commas, splits data into separate added qp(...): taggings. The param choice varNm tricky since @occurence elements both len scan for a match and the length cut-offs after match. You'll need to balance your file type data being added before open optional request, specific of where the newlines arrange the data or adv char sequence ":|:" cuts from counts list.

This option instruct from a pojishon next command & is a omega pojishon next dot comnd involved then occurence ignored. Occurence argument tells staqtapp of a specific super length order and must be present, as a list. The index value(s) as int, whereof any search & found happens, index value is max length new data will be added to the qp tagged env-var(s). If you are to bypass this argument then occurence=['*'] will do so but not same as a omega pojishon next comnd that auto involves certain lock properties for stored qp tagged env-vars, outside the embedded mnt/directory/files edits. *This option will bypass scanning qp tag data on a globe var, KLF__DSG var, smv tree var, darkvar, st1_atlaspice var(lambdavar()) and any corevar.
All six are special env-vars of Staqtapp's parsing methods and are skipped.

(2) **.closed(varNm=str; start=#; end=#) | .closed(varNm=list; start=#; end=#)**

With param @varNm can be either closed phase string or single env-var nm if param is a string. If a phase string then char '=' must be present and char sequence '::' for separation of each closed phase. A phase string use

is same as @varNm being a list of env-var names but not a single s & e for all var names. This allows closed-off data from a file in multiple phases if a match is found from the env-var's data.

EXAMPLE:
("**next.cast.closed**('varNmA=64:133::varNmB=85:456'; None; None)", ...

This optional quantum like between file being compared with the env-var. If any match found between the s & e indices in the file, the observation is removed from the file and is now of the env-var's data. The env-var's values or qp(...): tagged data, individual is the comparison check of the s & e str. If is a dual observation match, meaning the entire length of the qp tagged data matches the entire length in chars of s & e from the file, both will be removed.

The match is removed either way; not entire s & e--->unless is a full length enviro-match both ways met, a completed dual identical len matching. Closed optional works with optionals cache, seal, setfreq and removefreq of a similiar instructions bounded to env-vars' names and env-vars' datas in a dynamic observe, pass, collect, merge or remove context search further.

(3) .cache(varNm=str; occurence=#) | .cache(varNm=list; occurence=#[index])
(NOT YET IMPLEMENTED)

(4) .seal(dirLst=list; match=str; freq=str; type=unlocked=0|locked=1)
(NOT YET IMPLEMENTED)

.setfreq(varNm=str; dirLst=list; occurence=#) | .setfreq(varNm=list; dirLst=list; occurence=#[index]) --> (NOT YET IMPLEMENTED)
.removefreq(varNm=str; dirLst=list; freq=str) | .removefreq(varNm=list; dirLst=list; freq=str) --> (NOT YET IMPLEMENTED)
.retag(varDt=str; freq=str; span=#; removeat=#) | .retag(varDt=list; freq=str; span=#[index]; removeat=#[index]) --> (NOT YET IMPLEMENTED)
.quantumf(varNm=str; dirLst=list; span=#) | .quantumf(varNm=list; dirLst=list; span=#[index]) --> (NOT YET IMPLEMENTED)