]DTest

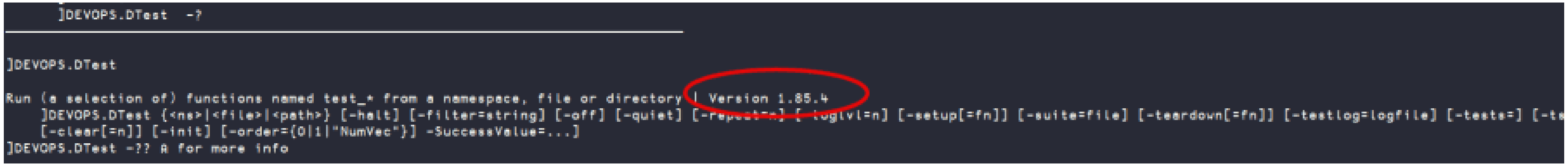
Michael Baas

Motivation

- Research
  - "Errare Humanum est"
    Seneca, 62
  - "Anything that can go wrong will go wrong [at the worst possible moment]."
    Murphy's Law or "Finagle's Law of Dynamic Negatives", 1970s; also: Sod's Law
  - "Shit happens"
    Forrest Gump, 1994
  - "We all know our code doesn't fail."
    Brian Becker, 2022
- and experience
- make it obvious that software needs to be tested before leaving the house!

Are you ready?

- Version 18.2
- https://github.com/Dyalog/DBuildTest ("devt" branch if main isn't updated yet)
- https://github.com/dyalog-training/2023-TP1
- Start Dyalog
- Same version?



```
      ]DEVOPS.DTest  -?
────────────────────────────────────────────

]DEVOPS.DTest

Run (a selection of) functions named test_* from a namespace, file or directory | Version 1.85.4
    ]DEVOPS.DTest {<ns>|<file>|<path>} [-halt] [-filter=string] [-off] [-quiet] [-repeat=n] [-loglvl=n] [-setup[=fn]] [-suite=file] [-teardown[=fn]] [-testlog=logfile] [-tests=] [-te
    [-clear[=n]] [-init] [-order={0|1|"NumVec"}] -SuccessValue=...]
]DEVOPS.DTest -?? ⍝ for more info
```

- :If not ⋄ :Andif v18 ⋄ :Then

```
]set cmddir ",[USERPROFILE]\Documents\My UCMDs" -p
```

Scope of the workshop

- Unit testing with DTest

  ...verify the functionality
  of a specific section of code...

  (for APLers: "a function")

- there's more...
- Leave inspired! 😉

Organisation of files & tests

```
┌─ demo/
│ ...
├─────┬─ src/
│     │ coolStat.apln
├─────┼─ Tests/
│     │ coolStat.dyalogtest
│     │ setup_coolStat.aplf
│     │ test_Avg.aplf
│     │ test_Median.aplf
```

- tests live in a dedicated folder
- optional .dyalogtest files define a "test suite" and are advantegous when you have multiple test suites ("basic " and "overnight") etc. or additional parameters (CodeCoverage or SuccessValue)
- files with prefix setup_ define setups that set the stage
- the files with prefix test_ do the real work...
- and you can also have teardown_ fn that remove ~~the mess that the test created~~ any leftovers

Writing tests

{res}←a Check b

a≡b: returns 0
a≢b: returns 1

a←a Because b
returns a and appends b to global r

{res}←a Assert b

a≡b: returns 0
a≢b: returns 1, **logs failed Assertion**

comment can also be in separate line
or
`var ⊢ a Assert b`
"var" has explanation of failure

```
2 Assert 1+1  ⍝ doc doc
∇
```

]DTest

-]demo

Writing tests

.dyalogtest:

```
DyalogTest: 1.84
[SuccessValue: …]
[Setup: …]
Test: test_1
Test: test_foo
...
[Teardown: …]
```

Test function

```
∇ r←mytest sink
  ⍝ test stuff
  x←testSubj arg
  expct Assert x  ⍝ doc
∇
```

OR

```
{
   y←testSubj arg
   [MsgVar]⊢expct Assert y: ⍝
   ...
}
```

Test functions need to return an empty string to indicate success. If you want to use 0 or other values, have a look at the "`SuccessValue`" modifier or add it to the .dyalogtest suite.

Test DSL

`[docvar]⊢x Assert y OR x Check y`

Returns 1 if the assertion that x=y is wrong, 0 otherwise.
If –halt modifier is set, halts execution if check fails.
Additional comments on line or immediately before or after. If comments are computed, use `docvar⊢x Assert y`

`x IsNotElement y`

test `~x∊y` and halts execution if it isn't.

`x Because y`

concatenates y to global "r" and returns x.
=> "Syntax sugar" to enable statements like:
`:if 1 Check 2 ⋄ →0 Because'1≠2!' ⋄ :endif`

`n←[id] ##.RandomVal x [y]`

generates y (default=1) random values identified by „id“ (like `[y]?x`).

`('Type' 'I|W|E')Log txt`

Adds txt to specified log (Info / Warning / Error)

Running tests

]DTest {.dyalogtest | .aplf | .dyalog | path}   -modifiers

Modifiers:

`-halt`: halts execution when Check or Assert fails (so that you can examine the ws)
`-trace:` trace into setup(s) and tests()
`-verbose:` show text logged with Log. (test fns should access `##.verbose` if they want to support this for ⎕←..output!)
`-quiet[=0|1]`: only shows error messages (1) or all messages (0)
`-filter=`*aaa*: select tests to execute (supports `*` and `?`)
`-loglvl=`*n*: controls the log files DTest creates. Value is a sum of the values.
  1={base.log} - Errors
  2={base}.warn.log - Warnings
  4={base}.info.log - Informations
  8={base}.session.log - Session log
  16={base}.session.log - Session log ONLY for failing tests
  32={base}.log.json - machine-readable results ("rc"=20: Success, 21=Failure)
`-off[=0|1]`: do (1) or do not (0) exit APL after running tests (also writes logfiles if required)
`-order[=0|1|"numvec"]`: order of tests. (0=random, 1=alphabetical, numvec specifies alternate order)
`-SuccessValue=nnn`: the value that successful tests need to return

Excercise

- Implement a test for the coolStat.Count function!
- Bonus points if you find a way to improve the implementation.
  (Is there a way to improve this (is that even possible?))

Test automation

-]demo

Automating tests

- Classic or Unicode?
- Unicode
  - LX="⎕SE.DTest ...."
  - LOAD=".../Tests" with Run.[aplf|dyalog]
- Classic
  - needs a .dws to start things
  - keep it small: ⎕LX←'⎕FIX"file:...Run.aplf"'
- loglvl=32 to get a .log.json

Code Coverage

- Careful: 100% Coverage does not mean 100% Correctness!
- 100% Coverage means that all code was executed, all possible branches were excuted.
- So IF your test cases were designed to be be wide and general (and cover ALL requirements), chances are that your code is good ;)
-]demo