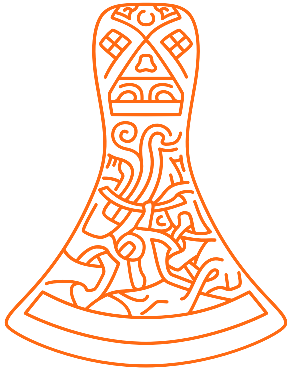
Glasgow 2024

Setting and Getting Variable Values Mk II



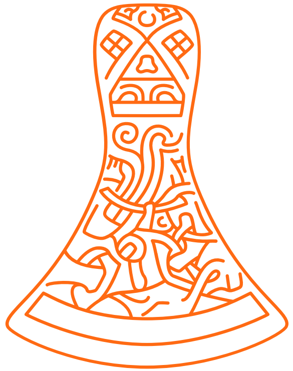
Adám Brudzewsky

DYALOG

Glasgow 2024

- APL language
- Teaching APL
- APL in text files
- Online communities and services

… *plus much more*



Adám Brudzewsky — just call me *Adam*

In 2023, I wanted to…

- get the values of variables using an array of variable names
- set variables using arrays containing names and values
- set a default left argument for an ambivalent tradfn
- base a new namespace on two source namespaces
- query data objects, but some have missing values
- construct a namespace from names and values
- populate class fields from name–value pairs
- convert between tables and namespaces
- check the value of an optional global

Set

Separate name list and value list:

```
target{⍺.{⍎⍺,'←⍵'}¨/⍵}names vals
```

List of name–value pairs:

```
target{⍺.{⍎⍺,'←⍵'}/¨⍵}('name1' val1)('name2' val2)
```

Set

Separate name list and value list:

```
target{⍺.{⍎⍺,'←⍵'}¨/⍵}names vals
target     ⎕VSET    (↑names)vals
```

List of name–value pairs:

```
target{⍺.{⍎⍺,'←⍵'}/¨⍵}('name1' val1)('name2' val2)
target      ⎕VSET     ('name1' val1)('name2' val2)
```

Set: performance

```
target{⍺.{⍎⍺,'←⍵'}/¨⍵}('name1' val1)('name2' val2)

target      ⎕VSET      ('name1' val1)('name2' val2)
```

```
      nvs←100⍴⊂'Data' 42
      ]runtime -c "⎕SE{⍺.{⍎⍺,'←⍵'}/¨⍵}nvs" "⎕SE ⎕VSET nvs"

  ⎕SE{⍺.{⍎⍺,'←⍵'}/¨⍵}nvs → 2.0E¯4 |   0% ⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕
* ⎕SE ⎕VSET nvs          → 7.3E¯5 | -63% ⎕⎕⎕⎕⎕⎕⎕⎕
```

Set: performance

```
target{α.{⍎α,'←ω'}/¨ω}('name1' val1)('name2' val2)

target     ⎕VSET      ('name1' val1)('name2' val2)
```

```
      nvs←100ρ⊂'Data' 42
      ]runtime -c "⎕SE{α.{⍎α,'                nvs" "⎕SE ⎕VSET nvs"

  ⎕SE{α.{⍎α,'←ω'}/¨ω}nvs → 2.         % ⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕
* ⎕SE ⎕VSET nvs          → 7.         % ⎕⎕⎕⎕⎕⎕⎕⎕
```

Get

Separate name list and value list:

```
target{⍺.{6::⍵ ⋄ ⍎⍺}¨/⍵}names vals
```

List of name–value pairs:

```
target{⍺.{6::⍵ ⋄ ⍎⍺}/¨⍵}('name1' val1)('name2' val2)
```

Get

Separate name list and value list:

```
target{⍺.{6::⍵ ⋄ ⍎⍺}¨/⍵}names vals
target         ⎕VGET      (↑names)vals
```

List of name–value pairs:

```
target{⍺.{6::⍵ ⋄ ⍎⍺}/¨⍵}('name1' val1)('name2' val2)
target          ⎕VGET         ('name1' val1)('name2' val2)
```

Get: Performance

```
target{⍺{6::⍵ ⋄ ⍺⍺⍎⍺}/¨⍵}('name1' val1)('name2' val2)

target       ⎕VGET       ('name1' val1)('name2' val2)


      nvs←100⍴('Data' 0)('Miss' 0)
      ]runtime -c "⎕SE{⍺{6::⍵ ⋄ ⍺⍺⍎⍺}/¨⍵}nvs" "⎕SE ⎕VGET nvs"

  ⎕SE{⍺{6::⍵ ⋄ ⍺⍺⍎⍺}/¨⍵}nvs → 2.0E¯3 |   0% ⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕
  ⎕SE ⎕VGET nvs             → 7.4E¯4 | -64% ⎕⎕⎕⎕⎕⎕⎕⎕
```

Get: Performance

```
target{α{6::ω ⋄ αα⍎α}/¨ω}('name1' val1)('name2' val2)

target        ⎕VGET        ('name1' val1)('name2' val2)


      nvs←100⍴('Data' 0)('Miss' 0)
      ]runtime -c "⎕SE{α{6::ω                ¨ω}nvs" "⎕SE ⎕VGET nvs"

  ⎕SE{α{6::ω ⋄ αα⍎α}/¨ω}nvs -                0% ⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕
  ⎕SE ⎕VGET nvs             →              -64% ⎕⎕⎕⎕⎕⎕⎕⎕
```

DYALOG

⎕VSET: Value Set

`ref←target ⎕VSET ('name1' val1) ('name2' val2)`	Add vars with values
`ref←target ⎕VSET ⊂'name1' 'val1'`	Set single variable
`… ⎕VSET (↑'name1' 'name2') (val1 val2)`	Two separate lists

⎕VGET: Value Get

By name:

`vals←source ⎕VGET ('name1' val1) ('name2' val2)`	Values w/ fallbacks
`vals←source ⎕VGET 'name1' 'name2'`	Values w/o fallbacks
`vals←source ⎕VGET ⊂'name1' val1`	Single name w/ fallback
`vals←source ⎕VGET 'name1'`	Single name w/o fallback

By nameclass:

`(name1 val1)(name2 val2)←source ⎕VGET ¯2`	Name–value pairs
`(nameMatrix valueVector)←source ⎕VGET 2`	Two separate lists

source & target: flexibility

`ref`	Namespace reference
`ref1 ref2 …`	Several references
`'name'`	Namespace name
`'name1' 'name2' …`	Several names
`ref1 'name1' ref2 …`	Any mixture of the above

everything on right to each on left

result structure from left argument

⎕NS extension

```
ref←        ⎕NS ns1 ns2
ref←target ⎕NS ns1 ns2
```

`ref← ⎕NS ns1 ns2`	Merge into new ns
`ref←target ⎕NS ns1 ns2`	Merge into existing ns

Let’s see that in context!

Get the values of variables using an array of variable names

```
vals←namespace ⎕VGET namesVector
```

Get the values of variables using an array of variable names

```
vals←namespace ⎕VGET namesMatrix
```

Set variables using arrays containing name–value pairs

```
namespace ⎕VSET nameValuePairs
```

Set variables using arrays containing names and values

```
namespace ⎕VSET namesMatrix valuesVector
```

Set a default left argument for an ambivalent tradfn

```
∇ r←{x} Foo y
  :If 42=⎕VGET⊂'x' 42
      answer
  :Else
      …
∇
```

Base a new namespace on two source namespaces

```
new←⎕NS defaults input
```

- All names from both namespaces included
- `input`'s values prevail

```
new←⎕NS namespaces
```

- All names from all namespaces included
- Rightmost values prevail

Query data objects,
but some have missing values

```
myFamily ⎕VGET ⊂'kidAges'⍬
families ⎕VGET ⊂'kidAges'⍬

myFamily ⎕VGET ('kidAges'⍬)('kidNames'(0⍴⊂''))
families ⎕VGET ('kidAges'⍬)('kidNames'(0⍴⊂''))
```

Construct a namespace from names and values

```
myns←(⎕NS⍬)⎕VSET (↑names) values

myns←(⎕NS⍬)⎕VSET↓⍉↑names  values
```

Construct a namespace from names and values

```
myns← () ⎕VSET (↑names) values

myns← () ⎕VSET↓⍉↑names  values
```

Populate class fields from name–value pairs

```
myInstance ⎕VSET('name1' val1)('name2' val2)
```

Convert
table to namespace

```
(data header)←⎕CSV path ⍬ 4 1
namespace←() ⎕VSET (↑header) (↓⍉data)


(data header)←⎕CSV(⎕OPT'Invert' 2) path ⍬ 4 1
namespace←() ⎕VSET (↑header) data
```

Convert
table to namespace

```
(data header)←⎕CSV path ⍬ 4 1
namespace←() ⎕VSET (↑header) (↓⍉data)


(data header)←⎕CSV(⎕OPT'Invert'(2 1)) path ⍬ 4 1
namespace←() ⎕VSET   header   data
```

Convert
table to namespace

```
(data header)←⎕CSV path ⍬ 4 1
namespace←() ⎕VSET (↑0(7162⌶)header) (↓⍉data)


(data header)←⎕CSV(⎕OPT'Invert'(2 1)) path ⍬ 4 1
namespace←() ⎕VSET (↑0(7162⌶)header) data
```

Convert namespace to table

```
(header data)←namespace ⎕VGET 2
data header ⎕CSV path


pairs←namespace ⎕VGET ¯2
(header data)←↓⍉↑pairs
]disp header⍪⍉↑data
```

Convert
namespace to table

```
(header data)←namespace ⎕VGET 2
data (1(7162⌶)header) ⎕CSV path


pairs←namespace ⎕VGET ¯2
(header data)←↓⍉↑pairs
]disp (1(7162⌶)header)⍪⍉↑data
```

Convert namespace to table

```
      ns←0⎕JSON'{"NUM":[1,2,3,4],
                 "DA" :["En","To","Tre","Fire"],
                 "EN" :["One","Two","Three","Fou
      (names vals)←ns ⎕VGET 2 ⋄ (↓names),↑vals
┌───┬───┬───┬─────┬────┐
│DA │En │To │Tre  │Fire│
├───┼───┼───┼─────┼────┤
│EN │One│Two│Three│Four│
├───┼───┼───┼─────┼────┤
│NUM│1  │2  │3    │4   │
└───┴───┴───┴─────┴────┘
```

Convert
namespace to table

```
ns←0⎕JSON'{"NUM":[1,2,3,4],
           "DA" :["En","To","Tre","Fire"],
           "EN" :["One","Two","Three","Fou
(names vals)←ns ⎕VGET 2 ⋄ ⍉(↓names),↑vals
```

DA	EN	NUM
En	One	1
To	Two	2
Tre	Three	3

DYALOG

Check the value of an optional global

```
⎕VGET⊂'DEBUG' 0

:Trap (⎕VGET⊂'DEBUG' 0)↓0

:Trap ⎕VGET⊂'DEBUG' 0
```

Value Set

```
⎕VSET name name…
⎕VSET (name val)…
⎕VSET names vals

'target'⎕VSET …
ref ref… ⎕VSET …
```

Value Get

```
⎕VGET name name…
⎕VGET (name val)…
⎕VGET names vals

⎕VGET type type…

'target'⎕VGET …
ref ref… ⎕VGET …
```

Namespace

```
⎕NS ns ns…

'target'⎕NS …
ref ref… ⎕NS …
```

Call a function when given its name?

```
(⎕VGET fnName) argument
(⍎fnName) argument
(⎕OR fnName){⍺⍺ ⍵} argument
```

NEW IDEA

DYALOG

Call a function when given its name?

```
(namespace ⎕VGET fnName) argument
(namespace⍎fnName) argument
(namespace.⎕OR fnName){⍺⍺ ⍵} argument
```

NEW IDEA

Call a function when given its name?

```
(namespace ⎕VGET fnName) argument
(namespace⍎fnName) a          t
(namespace.⎕OR fnNa          ⍵} argument
```

NEW IDEA

Value Set

```
⎕VSET name name…
⎕VSET (name val)…
⎕VSET names vals

'target'⎕VSET …
ref ref… ⎕VSET …
```

Value Get

```
⎕VGET name name…
⎕VGET (name val)…
⎕VGET names vals

⎕VGET type type…

'target'⎕VGET …
ref ref… ⎕VGET …

(⎕VGET fnName) arg
```

Namespace

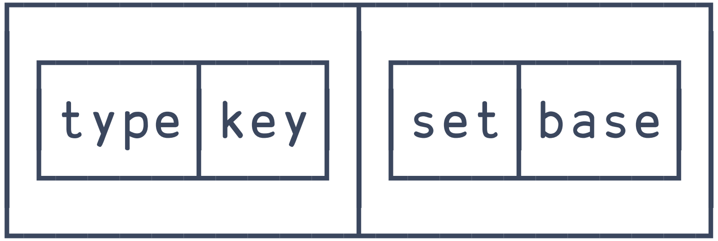
```
⎕NS ns ns…

'target'⎕NS …
ref ref… ⎕NS …
```

Why a matrix name list?

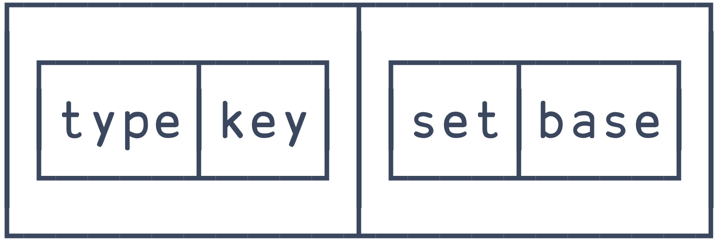
```
nv1←'type' 'key'
nv2←'set' 'base'
⎕VSET nv1 nv2

nv1 nv2
```



```
names←'type' 'key'
values←'set' 'base'
⎕VSET names value

names values
```



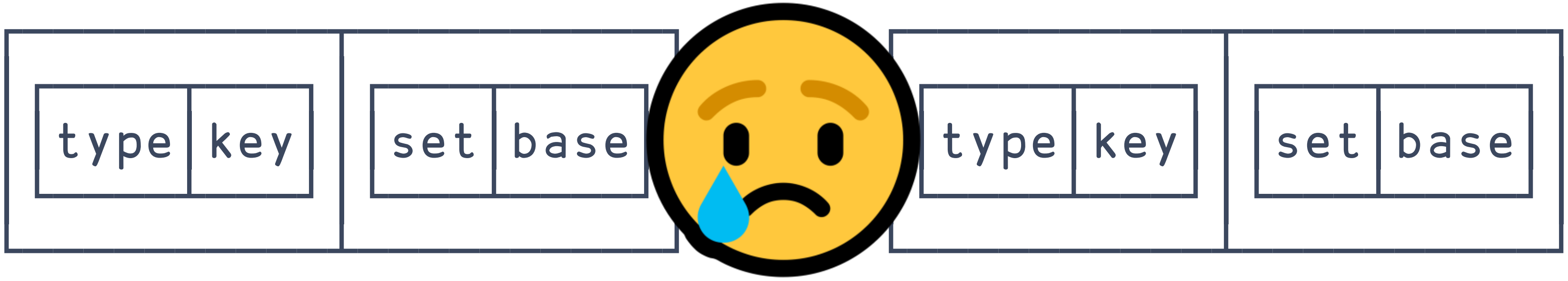
Why a matrix name list?

```
nv1←'type' 'key'
nv2←'set' 'base'
⎕VSET nv1 nv2

nv1 nv2
```

```
names←'type' 'key'
values←'set' 'base'
⎕VSET names value

names values
```


type
key
set
base
type
key
set
base