Literal Notation for Arrays and Namespaces

DYALOG

Elsinore 2017

Adám Brudzewsky

adam@dyalog.com

Literal Notation for Arrays and Namespaces

We have good notations for

- simple scalars and vectors
- small, depth-2 nested arrays

We need notations for

- higher rank arrays
- more complex nested arrays
- namespaces

The need to maintain application definitions and data
in text source files is making it urgent!

Literal Notation for Arrays and Namespaces

We have good notations for

- simple scalars and vectors
- small, depth-2 nested arrays

We need notations for

- higher rank arrays
- more complex nested arrays
- namespaces

The need to maintain application definitions and data in text source files is making it urgent!

Not final – please provide thoughts and feedback!

Why do we need literal notations?

- More readable
- Use any text editor to edit
  - any array
  - namespaces that are not scripted
  - tacit functions
- Collaborative editing
- Transfer across versions and systems
- Allow other languages to generate APL data
- Version tracking (GitHub, et al.)

Literal Notation for Arrays

```
]Boxing on -style=max
```

What we have good notation for

What we have good notation for

- Scalars

```
42
'a'
```

What we have good notation for

- Scalars `42`

  `'a'`

- Simple vectors `1 2 3`

  `'Hello'`

What we have good notation for

- Scalars `42`

  `'a'`

- Simple vectors `1 2 3`

  `'Hello'`

- Small vectors of vectors `(1 2 3)(4 5 6)`

  `'Hello' 'World'`

What we need notations for

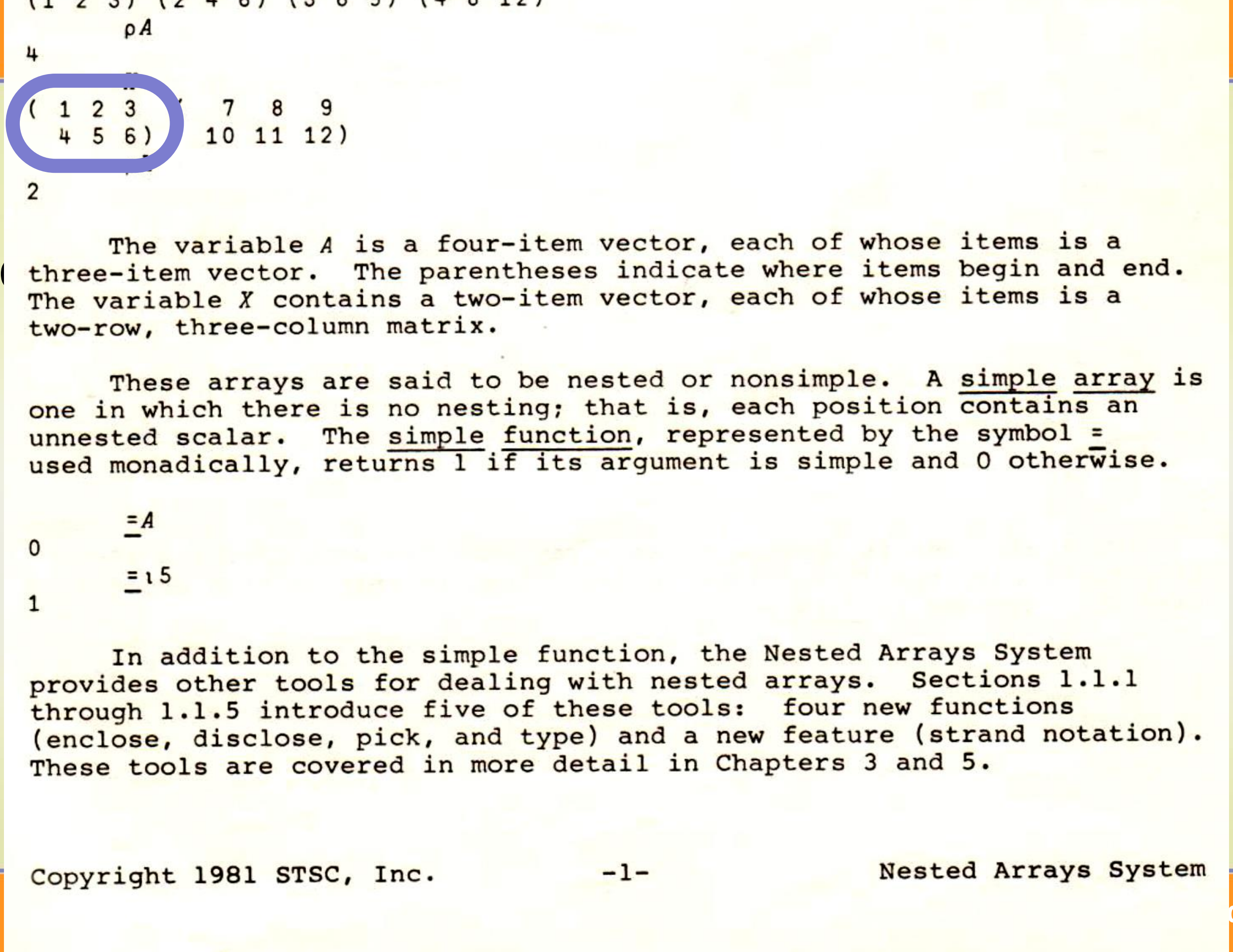
What we need notations for

- Higher rank arrays

```
(1 2 3
 4 5 6)
```

What

- High



```
(1 2 3) (2 4 6) (3 6 9) (4 8 12)
      ρA
4

( 1 2 3      7  8  9
  4 5 6)    10 11 12)

2
```

The variable *A* is a four-item vector, each of whose items is a three-item vector. The parentheses indicate where items begin and end. The variable *X* contains a two-item vector, each of whose items is a two-row, three-column matrix.

These arrays are said to be nested or nonsimple. A <u>simple array</u> is one in which there is no nesting; that is, each position contains an unnested scalar. The <u>simple function</u>, represented by the symbol ≡ used monadically, returns 1 if its argument is simple and 0 otherwise.

```
      ≡A
0
      ≡⍳5
1
```

In addition to the simple function, the Nested Arrays System provides other tools for dealing with nested arrays. Sections 1.1.1 through 1.1.5 introduce five of these tools: four new functions (enclose, disclose, pick, and type) and a new feature (strand notation). These tools are covered in more detail in Chapters 3 and 5.

What we need notations for

- Higher rank arrays

```
(1 2 3
 4 5 6)
```

What we need notations for

- Higher rank arrays

```
(1 2 3
 4 5 6)
```

- More complex nested arrays

```
(⊂1 2 3 'Hello'
 ⊂4 5 6 'World')
```

What we need notations for

- Higher rank arrays

```
(1 2 3
 4 5 6)
```

- More complex nested arrays

```
(⊂1 2 3 'Hello'
 ⊂4 5 6 'World')
```

- Namespaces

```
(greeting:'Hello'
 target:'World')
```

From vector to matrix

```
┌→──┐
│1 2│
└~──┘
```

```
(1 2)
```

From vector to matrix

```
┌→────┐
↓1  2 │
│3  4 │
│5  6 │
│7  8 │
└~────┘
```

```
(1  2
 3  4
 5  6
 7  8)
```

Simple numeric matrix

Current

```
m←1 2⍴34 ¯12
m⍪←   43  ¯3
m⍪←    0   1.5
```

Proposed

```
m←(34 ¯12
   43  ¯3
    0   1.5)
```

```
┌→──────┐
↓34 ¯12 │
│43  ¯3 │
│ 0  1.5│
└~──────┘
```

Simple character matrix

Current

```
r←1 5⍴'Three'
r⍪←   'Blind'
r⍪←   'Mice '
```

Proposed

```
r←('Three'
   'Blind'
   'Mice')
```

```
┌→────┐
↓Three│
│Blind│
│Mice │
└─────┘
```

Simple numeric 3D array

Current

```
d←1 2 3⍴3 1 4 1 5 9
d⍪← 2 3⍴2 7 1 2 8 2
```

```
┌┌→────┐
↓↓3 1 4│
││1 5 9│
││     │
││2 7 1│
││2 8 2│
└└~────┘
```

Proposed

```
d←((3 1 4
    1 5 9)

   (2 7 1
    2 8 2))
```

Nested table

Current

```
e←⍉⍪0 'OK'
e⍪←1 'WS FULL'
e⍪←2 'SYNTAX ERROR'
e⍪←3 'INDEX ERROR'
e⍪←4 'RANK ERROR'
```

```
┌→─────────────────────┐
↓                      │
│ 0  ┌→─┐              │
│    │OK│              │
│    └──┘              │
│ 1  ┌→──────┐         │
│    │WS FULL│         │
│    └───────┘         │
│ 2  ┌→───────────┐    │
│    │SYNTAX ERROR│    │
│    └────────────┘    │
│ 3  ┌→──────────┐     │
│    │INDEX ERROR│     │
│    └───────────┘     │
└∊─────────────────────┘
```

Proposed

```
e←(0 'OK'
   1 'WS FULL'
   2 'SYNTAX ERROR'
   3 'INDEX ERROR'
   4 'RANK ERROR')
```

Deeply nested vector

Current

```
l←⊂'Fiona' 'Gitte' 'Helene'
l,←⊂'Jay' 'Karen'
```

Proposed

```
l←(⊂'Fiona' 'Gitte' 'Helene'
   ⊂'Jay' 'Karen')
```

```
┌→──────────────────────────────────────────────┐
│ ┌→────────────────────────┐ ┌→─────────────┐ │
│ │ ┌→────┐ ┌→────┐ ┌→─────┐ │ │ ┌→──┐ ┌→────┐ │ │
│ │ │Fiona│ │Gitte│ │Helene│ │ │ │Jay│ │Karen│ │ │
│ │ └─────┘ └─────┘ └──────┘ │ │ └───┘ └─────┘ │ │
│ └∊────────────────────────┘ └∊─────────────┘ │
└∊──────────────────────────────────────────────┘
```

Deeply nested vector

Current

```
l←⊂'Fiona' 'Gitte' 'Helene'
l,←⊂'Jay' 'Karen'
```

Proposed

```
l←(⊂(⊂'Fiona'
     ⊂'Gitte'
     ⊂'Helene')

   ⊂(⊂'Jay'
     ⊂'Karen'))
```

```
┌→───────────────────────────────────────────────┐
│ ┌→─────────────────────────┐ ┌→──────────────┐ │
│ │ ┌→────┐ ┌→────┐ ┌→─────┐ │ │ ┌→──┐ ┌→────┐ │ │
│ │ │Fiona│ │Gitte│ │Helene│ │ │ │Jay│ │Karen│ │ │
│ │ └─────┘ └─────┘ └──────┘ │ │ └───┘ └─────┘ │ │
│ └∊─────────────────────────┘ └∊──────────────┘ │
└∊───────────────────────────────────────────────┘
```

How is the array assembled?

1. The result of each *statement* is collected into a list
2. Mix is applied to the list, producing an array of rank one higher than the highest rank item
3. Thus, each item of the list becomes a *major cell* of the array which is represented by the nearest surrounding parentheses
4. Any *embedded parentheses* are resolved first; each result becomes an item of the list

```
(⊂'Fiona'
 ⊂'Gitte'
 ⊂'Helene')

(0 'OK'
 1 'WS FULL'
 2 'SYNTAX ERROR'
 3 'INDEX ERROR'
 4 'RANK ERROR')

((3 1 4
  1 5 9)
 (2 7 1
  2 8 2))
```

More examples

```
('Jan'             'Feb'             'Mar'
 (101 102 103 104) (201 202 203 204) (301 302 303 304))
```

```
┌→──────────────────────────────────────────────────────┐
↓ ┌→──┐             ┌→──┐             ┌→──┐             │
│ │jan│             │feb│             │mar│             │
│ └───┘             └───┘             └───┘             │
│ ┌→──────────────┐ ┌→──────────────┐ ┌→──────────────┐ │
│ │101 102 103 104│ │201 202 203 204│ │301 302 303 304│ │
│ └~──────────────┘ └~──────────────┘ └~──────────────┘ │
└∊──────────────────────────────────────────────────────┘
```

```
('Jan' (101 102 103 104)
 'Feb' (201 202 203 204)
 'Mar' (301 302 303 304))
```

```
┌→────────────────────────┐
↓ ┌→──┐ ┌→──────────────┐ │
│ │jan│ │101 102 103 104│ │
│ └───┘ └~──────────────┘ │
│ ┌→──┐ ┌→──────────────┐ │
│ │feb│ │201 202 203 204│ │
│ └───┘ └~──────────────┘ │
│ ┌→──┐ ┌→──────────────┐ │
│ │mar│ │301 302 303 304│ │
│ └───┘ └~──────────────┘ │
└∊────────────────────────┘
```

Literal Notation for Namespaces

```
]box -s=min
```

Literal Notation for Namespaces

```
]box -s=min
```

```
⎕JSON
```

Namespace

Current

```
ns←⎕NS ''
ns.life←42
ns.name←'Andy'
```

Proposed

```
ns←(life:42
    name:'Andy')
```

JSON

```
{"life":42,
 "name":"Andy"}
```

Inline namespace

Current `(⎕NS '').(life name)←42 'Andy'`

Proposed `(life:42 ⋄ name:'Andy')`

JSON `{"life":42, "name":"Andy"}`

Example utility namespace

```
utils←(
    ∇ res←avg nums;count
      total←+/nums
      count←≢nums
      res←total÷count
    ∇
    identity3:(1 0 0
               0 1 0
               0 0 1)
    product:'Dyalog APL'
    link:{(⊂⍺),⊆⍵}
    primes:(⊢~∘.×⍨)1↓⍳100
)
```

Empty namespace

Current	`⎕NS ''`
Proposed	`( )`
JSON	`{ }`

Populating namespaces in a program

```
names←'life' 'lang'
vals←42 'APL'
```

Populating namespaces in a program

```
names←'life' 'lang'
vals←42 'APL'
```

Current

```
ns←⎕NS ''
names ns.{⍎⍺,'←⍵'}¨ vals
```

Populating namespaces in a program

```
names←'life' 'lang'
vals←42 'APL'
```

Current

```
ns←⎕NS ''
names ns.{⍎⍺,'←⍵'}¨ vals
```

Proposed

```
ns←names ⎕NS vals
```

Vectors of Text Vectors (VTVs)

```
r←'Snap [path] (default current workdir)'
r,←⊂''
r,←⊂'Save all new or modified SALT objects in path'
r,←⊂''
r,←⊂'Modifiers:'
r,←⊂'-loadfn[=] the path (default =arg)
r,←⊂'-nosource don''t bring in the source
r,←⊂'-∆⍙= characters to use in filenames
r,←⊂'* Note: to exclude simply prefix with "~"'
```

Vectors of Text Vectors (VTVs)

```
r←'Snap [path] (default current workdir)

    Save all new or modified SALT objects in path

    Modifiers:
    -loadfn[=] the path (default =arg)
    -nosource don''t bring in the source
    -∆⍙= characters to use in filenames
    * Note: to exclude simply prefix with "~"'
```

Vectors of Text Vectors (VTVs)

```
r←'Snap [path] (default current workdir)

   Save all new or modified SALT objects in path

   Modifiers:
   -loadfn[=] the path (default =arg)
   -nosource don''t bring in the source
   -Δ⍙= characters to use in filenames
   * Note: to exclude simply prefix with "~"'
```

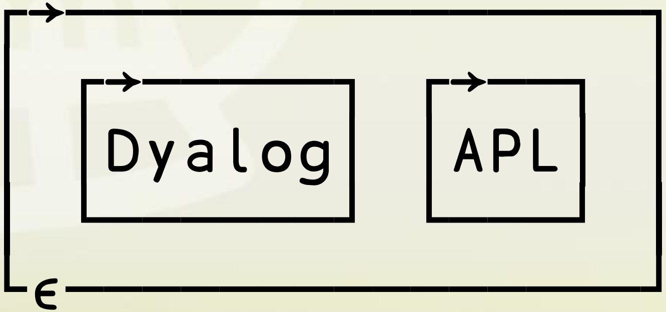
Vectors of Text Vectors (VTVs)

```
r←'Snap [path] (default current workdir)

   Save all new or modified SALT objects in path

   Modifiers:
   -loadfn[=] the path (default =arg)
   -nosource don''t bring in the source
   -∆⍙= characters to use in filenames
   * Note: to exclude simply prefix with "~"'
```

Vectors of Text Vectors (VTVs)

1. String with line-breaks are vectors of text vectors.
2. Each line constitutes an element in the overall vector.
3. Leading and trailing spaces are stripped to allow code indentation and alignment.

```
'Dyalog
 APL'
```



```
┌→─────────────────────┐
│ ┌→─────┐   ┌→──┐     │
│ │Dyalog│   │APL│     │
│ └──────┘   └───┘     │
└∊─────────────────────┘
```

Summary of notations

```
VTV     'Dyalog
         APL'

Array   (1 2 3
         4 5 6)    (1 2 3 ⋄ 4 5 6)

NS      (Greeting:'hello' ⋄ target:'World')
        'Greeting' 'target' ⎕NS 'hello' 'World'
```

Should items always have minimum rank 1?

NO, what looks like a scalar is a scalar!

Single column matrices

```
(,1          (,'a'
  2            'b'
  3)           'c')
```

Simple vectors

```
(1          ('a'
 2           'b'
 3)          'c')
```

Vectors of vectors

```
(⊂1 2       (⊂'aA'
 ⊂2 3        ⊂'bB'
 ⊂3 4)       ⊂'cC')
```

YES, what looks like matrix is a matrix!

Single column matrices

```
(1          ('a'
 2            'b'
 3)           'c')
```

Simple vectors

```
1`          'a'`
2`          'b'`
3           'c'
```

Vectors of vectors

```
1 2`        'aA'`
2 3`        'bB'`
3 4         'cC'
```