DYALOG

DYALOG
Belfast 2018

Namespace workshop

Nicolas Delcros
nicolas@dyalog.com

One typical problem

Workspace 1	Workspace 2
Run DATA	Go DATA

One typical problem

Workspace

Run
DATA

Go
DATA

One typical problem

Workspace

Run
WS1_DATA

Go
WS2_DATA

One typical problem

Workspace

WS1_Run
WS1_DATA

WS2_Go
WS2_DATA

One typical problem

Workspace

WS1 namespace

WS1.Run
WS1.DATA

WS2 namespace

WS2.Go
WS2.DATA

One typical problem

Workspace

WS1 namespace

Run
DATA

WS2 namespace

Go
DATA

One typical problem

Workspace 1

Run
DATA

Workspace 2

Go
DATA

Using namespaces

Tree	Workspace	File system
Node	Namespace	Directory
Leaf	Name	File
Roots	# ⎕SE	/ or C: D: E:
Separator	.	/ or \
Parent Node	##	..
Current Node	⎕THIS	.
Create Node	⎕NS	mkdir
Change Node	⎕CS	cd
Create Leaf	←	>

Using namespaces

```
    ns←⎕NS''                   ⍝ create a namespace

    ]display ns (2 2⍴ns)       ⍝ it's a new kind of scalar
┌→────────────────────────────────────────────────────────┐
│                ┌→────────────────────────────────────┐  │
│ #.[Namespace] ↓ #.[Namespace]  #.[Namespace] │  │
│                │ #.[Namespace]  #.[Namespace] │  │
│                └#────────────────────────────────────┘  │
└∊────────────────────────────────────────────────────────┘
```

Using namespaces

```
      data←7 8 9
      ns←⎕NS''
      ns.data←'hello'
      ⎕←(data)(ns.data)
7 8 9  hello
      ⎕CS ns         ⍝ change current space
      ⎕←data
hello
      ⎕CS #          ⍝ come back to root
      ⎕←data
7 8 9
```

Using namespaces

Exercise (easy)

Write a function that tells you whether a namespace is a root

```
{⍵∊# ⎕SE}
{⍵≡⍵.##}
```

Using namespaces

Exercise (medium)

Write a function that returns the root of a namespace

```
{p←⍵.## ⋄ ⍵≡p:⍵ ⋄ ∇p}
{⍵.##}⍣≡
```

Dotted expressions

Left side must be an array of namespace(s)

```
      (ns1 ns2)←(⎕NS'')(⎕NS'')
      (ns1.num ns2.num)←77 99
      ⎕←(ns1 ns2).num
77 99
      ⎕←(2 3⍴ns1 ns2).num
77 99 77
99 77 99
      (ns1 ns2).num←55 ⋄ ⎕←(ns1 ns2).num
55 55                          ⍝ scalar extension
```

Dotted expressions

Right side may be any APL expression (without ⋄)
executed in the namespace(s) on the left side

```
      (ns1 ns2)←(⎕NS'')(⎕NS'')
      ns1.(a b)←'hello' 'world' ⋄ ⎕←ns1.(a,b)
helloworld
      ns2.a←{2/⍵} ⋄ ns2.b←7 8 9 ⋄ ⎕←ns2.(a,b)
7 7 8 8 9 9
      ⎕←(2 3⍴ns1 ns2).(a,b) ⍝ which languages allow this?
 helloworld   7 7 8 8 9 9  helloworld
 7 7 8 8 9 9  helloworld   7 7 8 8 9 9
```

Dotted expressions

Exercise (easy)

What's the difference between `NS.A←B` and `(NS.A)←B` ?

None

What's the difference between `NS.A←B` and `NS.(A←B)` ?

B is looked up in NS (even if it is local!)

The whole expression `(A←B)` is evaluated in NS

Dotted expressions

Exercise (easy)

Write a function that returns the roots of an arbitrary array of namespaces

The previous answer still works

```
{⍵.##}⍣≡
```

By-value VS by-reference

Arrays are interpreted by value

```
      n←7 8 9 ⋄ m←n ⋄ ⎕←n m
7 8 9  7 8 9      ⍝ arrays have the same value

      n[2]←80 ⋄ ⎕←n m
7 80 9  7 8 9     ⍝ m still has the same value

      m[3]←90 ⋄ ⎕←n m
7 80 9  7 8 90    ⍝ n still has the same value
```

By-value VS by-reference

Arrays are interpreted by value

```
      n←3 ⋄ f←n∘+ ⋄ ⎕←f 100
103

      n←4 ⋄ ⎕←f 100
103         ⍝ n was passed by value - f hasn't changed

      f←n∘+ ⋄ ⎕←f 100
104         ⍝ f has changed - not n
```

By-value VS by-reference

Arrays are interpreted by value

```
    ∇ foo arg
      arg[2]←10
    ∇
    n←7 8 9 ⋄ foo n ⋄ ⎕←n
7 8 9    ⍝ n hasn't changed
    ∇ arg←foo arg
      arg[2]←10
    ∇
    n←foo n
```

By-value VS by-reference

Exercise (medium)

Is there a difference between `(⎕NS'')(⎕NS'')` and `(2⍴⎕NS'')` ?

By-value: no, there is only one empty namespace: `⍬⍬≡2⍴⊂⍬`

By-reference: yes, in the first case we create two entities, in the second case only one

When should two namespaces compare equal?

By-value: when they happen to have the exact same content (slow)

By-reference: when they originate from the same call to `⎕NS` (fast)

By-value VS by-reference

Namespaces are interpreted by reference

A namespace is an identifiable container, irrespectively of the content
An array is a conceptual value, irrespectively of the actual instance

In the context of Dyalog APL, namespaces ARE called “references” or “refs”
They have a different name classes (2=array ; 3=function ; 9=reference)

By-value VS by-reference

Namespaces are interpreted by reference

```
      ns1←⎕NS''
      ns2←ns1     ⍝ both names designate the same namespace
      ⎕←ns2.vec
VALUE ERROR   ⍝ name undefined yet
      ns1.vec←7 8 9 ⋄ ⎕←ns2.vec
7 8 9         ⍝ name defined
      ns2.vec[2]←10 ⋄ ⎕←ns1.vec
7 10 9        ⍝ value has changed
```

By-value VS by-reference

Namespaces are interpreted by reference

```
    ∇ new goo arg
      arg.vec[2]←new
    ∇
    (ns1←⎕NS'').vec←7 8 9 ⋄ 10 goo ns1 ⋄ ⎕←ns1.vec
7 10 9                     ⍝ namespace has been modified
    ns2←ns1 ⋄ 100 goo ns1 ⋄ ⎕←ns2.vec
7 100 9                    ⍝ namespace has been modified
    1000 goo ns3←⎕NS'' ⋄ ⎕←ns3.vec
VALUE ERROR                ⍝ vec is not defined in ns3
```

By-value VS by-reference

Namespaces are interpreted by reference

```
    ∇ foo arg
      arg[2]←10
    ∇
    (ns1←⎕NS'').vec←7 8 9
    foo ns1.vec ⋄ ⎕←ns1.vec
7 8 9           ⍝ ns1.vec is an array passed by value
```

By-value VS by-reference

Namespaces are interpreted by reference

```
      (ns1←⎕NS'').vec←7 8 9
      ns3←⎕NS ns1    ⍝ take a deep copy
      ns3.vec[2]←10 ⋄ ⎕←(ns1 ns3).vec
 7 8 9  7 10 9       ⍝ only the new copy has changed
```

Namespaces by value are rare but possible ☺

By-value VS by-reference

Namespaces semantics are by reference, and not by value (assignment, comparison, argument passing…) unless you really want to.

This allows

- tracking individual entities independently of their content
- pass modifiable arguments to functions (use with care)

Because of the different semantics, it is worth distinguishing names of references. Depth-0 references are of name class 9, depth≥1 arrays of refs are of name class 2

Recognise dottable names

By-value VS by-reference

Exercise (easy)

Write a function that copies a workspace into a namespace

```
{⍺.⎕CY ⍵}
```

NB. No need to return a result - caller already knows ⍺

By-value VS by-reference

Exercise (medium)

Make it work on an arbitrary array of namespaces
Requires care to distribute the one workspace name to multiple namespaces

```
{⍺.⎕CY ⊂⍣(≡⍵)⊢⍵}            ⍝ for positive depth
{0=≡⍺:⍺.⎕CY ⍵ ⋄ ⍺∇¨⊂⍵}      ⍝ for negative depth

{⍺.(⎕CY ⍵)}                 ⍝ Brenner’s trick
```

By-value VS by-reference

Exercise (medium)

Write a boolean-returning function that detects namespaces

Tip: The name class of a scalar namespace is 9. The name class of non-scalar arrays and of non-namespace scalars is 2.

```
IsScalarRef←{9=⎕NC'⍵'}
RefMask←{0=≡⍵:9=⎕NC'⍵' ⋄ ∇¨⍵}
```

Homework : RefMask could use ⎕DR for performance

Parent hierarchy (optional)

Namespaces have a single immutable parent, fixed at creation time

```
      ns←⎕NS'' ⋄ ⎕←#=ns.##
1                            ⍝ created in #
      ns.sub1←ns.⎕NS'' ⋄ ⎕←ns=ns.sub1.##
1                            ⍝ created in #.ns
      ns.sub1.(sub2←⎕NS'') ⋄ ⎕←ns.sub1=ns.sub1.sub2.##
1                            ⍝ created in #.ns.sub1
      ⎕←ns.sub1.sub2.##.##.##
#
```

Parent hierarchy (optional)

Names of a namespace are NOT (necessarily) its children
Children are NOT (necessarily) named from their parent

```
      ns0←⎕NS'' ⋄ ns1←⎕NS'' ⋄ ns1.ns2←ns0
      ⎕←ns1=ns1.ns2.##
0 ⍝ ns1.ns2 is ns0 which was created in #
      ns3←ns1.ns2.⎕NS'' ⋄ ⎕←ns0=ns3.##
1 ⍝ yet ns0 contains no name designating its child ns3
```

Just like arrays, namespaces do not know their names
(they only know their parent)

Parent hierarchy (optional)

Exercise (hard) :

Write a function that lists the children of a namespace

Tip 1. ⎕NL gives the list of names, not the list of children
Tip 2. impossible without crawling through the whole workspace

Parent hierarchy (optional)

```
∇ children←{arg}ListChildren target;args;next;parents;visited
  :If 0=⎕NC'arg' ⋄ arg←(# ⎕SE target)(0⍴#)(0⍴#) ⋄ :EndIf
  (parents children visited)←arg
  next←∪∊parents.(⍎¨'##' '⎕THIS',⎕NL ¯9)⍝ visit all reachable ns
  next~←visited                         ⍝ excepted visited ones
  :If 0∊⍴next ⋄ :Return ⋄ :EndIf        ⍝ (0⍴#).## is NONCE ERROR
  children∪←(target=next.##)/next       ⍝ append children of target
  visited,←next                         ⍝ no next has been visited
  :If 0∊⍴next ⋄ :Return ⋄ :EndIf        ⍝ F¨0⍴# is NONCE ERROR
  args←next children visited
  children∪←args ListChildren target    ⍝ recur on unvisited ns
∇
```

Display form (optional)

Since namespaces may or may not have a name,
their display form is ARBITRARY.

```
    ns←⎕NS'' ⋄ ⎕←2⍴ns
#.[Namespace]  #.[Namespace]
    ns.⎕DF'<My Namespace>' ⋄ ⎕←2⍴ns
<My Namespace>  <My Namespace>
```

Just like arrays, namespaces do not know their names.

Display form (optional)

Crafting namespaces where (⍎∘⍕) is identity requires care.
All parents must be correctly named.

```
      'named'⎕NS''
      'named.sub'⎕NS''  ⍝ better than named.('sub'⎕NS'')
      ns1←named ⋄ ns2←named.sub ⋄ ⎕←ns1 ns2
#.named  #.named.sub
      ⎕←{⍵≡⍎⍕⍵}¨ns1 ns2
1 1
```

Namespaces are then a proper tree where each node knows its path.

Display form (optional)

Exercise (medium)

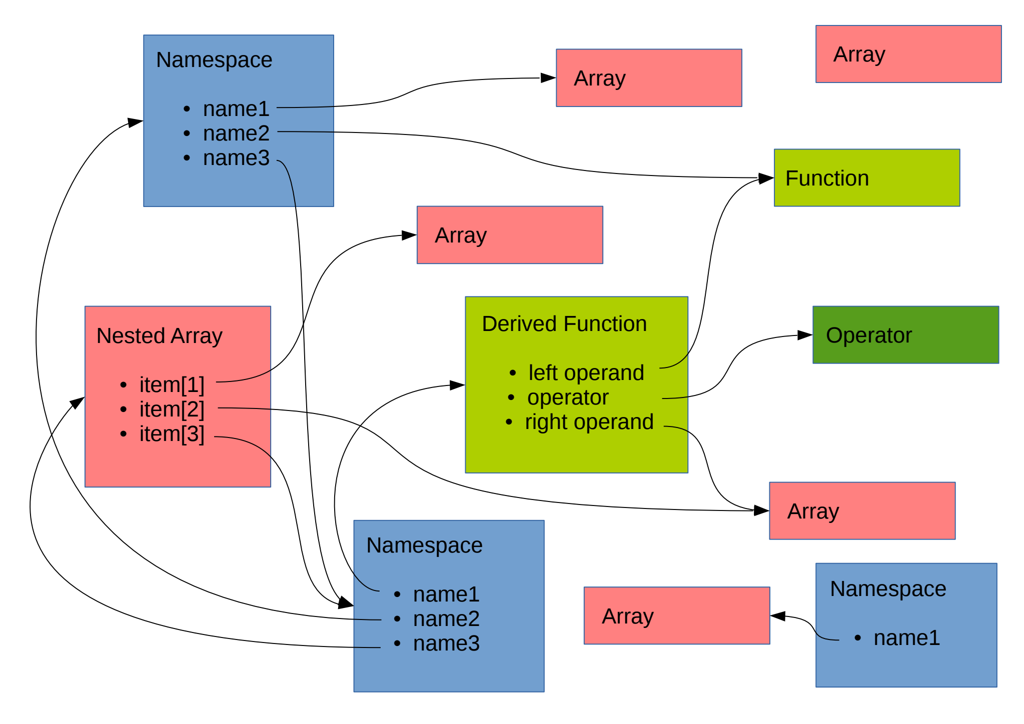
Write a function that lists the children of a namespace, assuming they're all named

```
{⍵.(⍎¨⎕NL ¯9)}
```

Homework: supplementary filtering with `⎕STATE` might be required if the namespace may have a function on the stack with locals of name class 9

Hack of the day: one can turn a name class 9 (scalar reference) into a name class 2 by ravelling it

The workspace fauna



The workspace fauna

A namespace contains nothing but

- a list of names: `ω.⎕NL-⍳10`
- a reference to its parent: `ω.##`
- a local copy of system variables: `⎕IO ⎕CT ⎕RL...`
- a display form: `ω.⎕DF →→ ⍕ω`

A name can designate any entity (array, function, operator, namespace)

Just like arrays, namespaces do NOT know their names.

Namespaces do NOT contain entities

Names of a namespace are NOT children of the namespace

Two approaches to namespaces

Usage	Creation	Display Form	Parent Hierarchy
Grounded	Named	Full Path	Tree
Floating	Unnamed	Description	Flat

Each approach is easy, mixing them is tough !

Scripted namespaces

Store a namespace as a single piece of text
Helpful for text-based version control

```
)ed ⍟NS              ⍝ equivalent to '⍟'⎕ED'NS'
⎕FIX  ':Namespace NS'  'VAR←123'  ':EndNamespace'
⎕←⎕SRC NS
```

Every time the script is fixed, then the namespace is cleared and the script is re-run. This happens only once at run-time, but many times at development time.

At the doorstep of Object Orientation

Objects ARE namespaces, with a few constraints:

- Can only call functions and modify variables
- Cannot create variables or change code
- New tree hierarchy called “derived classes” to avoid duplicating code
- Possibility to “hide” internal code

This is the fashionable way to provide an API

Belfast 2018

Namespace workshop

Nicolas Delcros
nicolas@dyalog.com