

# Error Handling

## Adám Brudzewsky



# Error Handling

## Adám Brudzewsky



# Error Handling – Part 1

*Number for given message*  
aplcart.info?q=:: message

Tradfn structure : Trap errnos (0 means all)

Dfn error guard errnos : : code

Error Number of last error □ EN

Error Message for number □ EM

## Error Handling – Part 2

Execute there on error

Cut back and execute

Ext'd Diagnostic Message

*Construct error message*  
`aplcart.info?q=dmx 1`

□ TRAP ← errnos ' E ' code

□ TRAP ← errnos ' C ' code

□ DMX

# Error Handling – Part 2

Execute there on error

Cut back and execute

Ext'd Diagnostic Message

*Construct error message*  
aplcart.info?q=dmx 1

□ TRAP ← errnos ' E ' code

□ TRAP ← errnos ' C ' code

□ DMX

dyalog.tv

# Overview

- ◆ Signalling an Error

# Overview

- ◆ Signalling an Error
- ◆ Assertions

# Overview

- ◆ Signalling an Error
- ◆ Assertions
- ◆ Resignalling an Error



# Signalling an Event

$M \leftarrow \{$   
     $(+\neq\omega) \div \neq\omega$   
 $\}$

M 3 1 4 1 5

2.8

# Signalling an Event

$M \leftarrow \{$   
     $(+\neq\omega) \div \neq\omega$   
 $\}$

M 3 1 4 1 5

2.8

M  $\theta$

1

# Signalling an Event

$$M \leftarrow \left\{ \begin{array}{l} (+/\omega) \div \neq \omega \\ \end{array} \right\}$$

M 3 1 4 1 5

2.8

M  $\emptyset$

DOMAIN ERROR

M  $\emptyset$

^

# Signalling an Event

```
M ← {
      (+ / ω) ÷ ≠ ω
    }
```

```
M 3 1 4 1 5
```

2.8

```
M 0
```

```
DOMAIN ERROR: Divide by zero
```

```
M 0
```

```
^
```

# Signalling an Event

```
M ← {
      (+ ≠ ω) ÷ ≠ ω
    }
```

```
M 3 1 4 1 5
```

2.8

```
M θ
```

```
MEAN ERROR: Empty 1st axis
```

```
M θ
```

```
^
```

# Signal Error

system function

□ SIGNAL

# Signal Error

□ SIGNAL

scalar

# Signal Error

□ SIGNAL — scalar —





# Signal Error

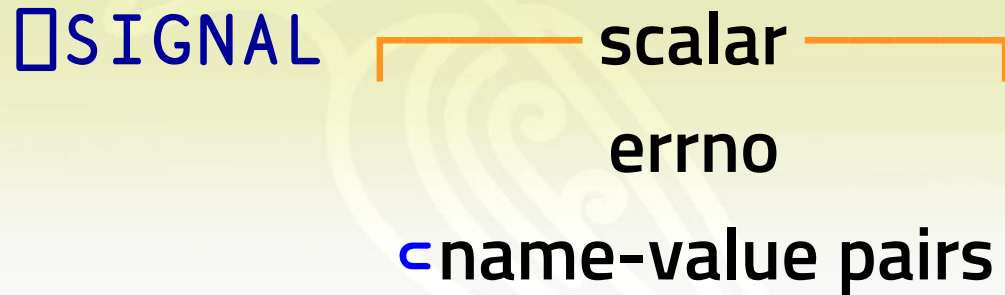
□ SIGNAL

scalar

errno


# Signal Error

□ SIGNAL — scalar —  
    errno  
    c name-value pairs



# Signal Error

□ SIGNAL — scalar —  
      errno  
      c name-value pairs



# Signal Error

```

M ← {
  0 ≠ ω : □ SIGNAL 11
  (+ ≠ ω) ÷ ≠ ω
}

```

```

M 3 1 4 1 5

```

2.8

```

M θ

```

DOMAIN ERROR

```

M θ

```

```

^

```

# Signal Error

```
M ← {
  0 = ≠ ω : □ SIGNAL 11
  (+ ≠ ω) ÷ ≠ ω
}
```

```
M 3 1 4 1 5
```

2.8

```
M θ
```

```
DOMAIN ERROR
```

```
M θ
```

```
^
```

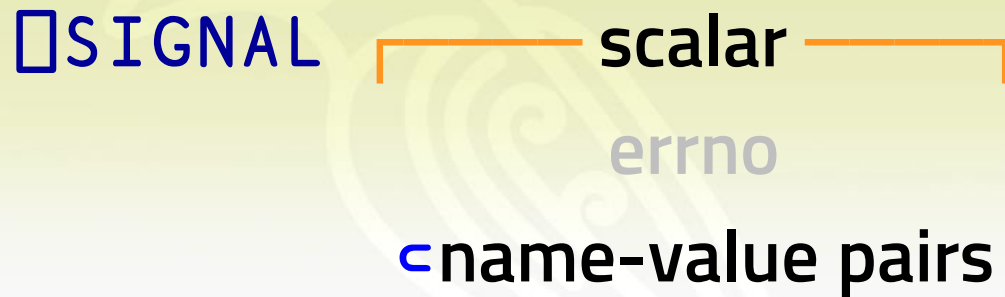
17::  
DOMAIN  
ERROR

# Signal Error

□ SIGNAL — scalar —  
      errno  
      c name-value pairs

# Signal Error

□ SIGNAL — scalar —  
    errno  
    c name-value pairs



# Signal Error

**name-value pairs**





# Signal Error

name-value pairs

# Extended Diagnostic Message

JSON Compact DMX

name-value pairs

# Extended Diagnostic Message

JSON Compact DMX

```
{
  "Category": "General",
  "DM": [
    "DOMAIN ERROR",
    "      1÷0",
    "      ^"
  ],
  "EM": "DOMAIN ERROR",
  "EN": 11,
  "ENX": 1,
  "HelpURL": "https://help.dyalog.com/dm
```

name-value pairs

# Extended Diagnostic Message

```
JSON Compact DMX
{
  "Category": "General",
  "DM": [
    "DOMAIN ERROR",
    "    1÷0",
    "    ^"
  ],
  "EM": "DOMAIN ERROR",
  "EN": 11,
  "ENX": 1,
  "HelpURL": "https://help.dyalog.com/dm"
}
```

name-value pairs

# Extended Diagnostic Message

```
{}JSON{} Compact '0' DMX
{
  names      values
  "Category" : "General",
  "DM" : [
    "DOMAIN ERROR",
    "    1÷0",
    "    ^"
  ],
  "EM" : "DOMAIN ERROR",
  "EN" : 11,
  "ENX" : 1,
  "HelpURL" : "https://help.dyalog.com/d
```

name-value pairs

# Signal Error

name-value pairs

# Signal Error

name-value pairs

# Signal Error

name-value pairs

The background of the slide features two stylized white line-art birds, possibly parrots or toucans, facing each other. They are rendered in a simple, clean style with prominent beaks and curved wings. The birds are set against a light green gradient background that transitions from a darker green at the top to a lighter green at the bottom.



# Signal Error

□ SIGNAL scalar

errno

⊂ name-value pairs

'EN'

11

'EM'

'MEAN ERROR'

'Message'

'Empty 1st axis'

# Signal Error

□ SIGNAL scalar

errno

⊂ name-value pairs

'EN'

11

'EM'

'MEAN ERROR'

'Message' 'Empty 1st axis'

# Signal Error

```
□ SIGNAL ← ('EN' 11) ('EM' 'MEAN ERROR') ('Mes
```

# Signal Error

```

M ← {
  0 ≠ ω : □ SIGNAL ← ('EN' 11) ('EM' 'MEAN ERROR
    (+ / ω) ÷ ≠ ω
}

```

# Signal Error

```
M ← {
  0 = ≠ω : □ SIGNAL = ('EN' 11) ('EM' 'MEAN ERROR
  (+ ≠ω) ÷ ≠ω
}
```

```
M 3 1 4 1 5
```


2.8

```
M θ
```

```
MEAN ERROR: Empty 1st axis
```

```
M θ
```

```
^
```



**use-cases**  
for  
**□ SIGNAL**

# Assert

utility function

```
{ ... □ SIGNAL ... }
```

# Assert

utility function

```
{ ... □ SIGNAL ... }
```



[aplcart.info](http://aplcart.info)



# Assert

utility function

{ ... **□ SIGNAL** ... }

aplcart.info

18.1:  
]APLcart

X,Y,Z:any M,N:num I,J:int A,B:Bool C,D:char f,g,h:fn ax:axis s:scal v:vec m:mat

- ▶  $\emptyset$  ? Empty Numeric Vector
- ▶  $\vdash Y$  ? Same: Y
- ▶  $X \text{ dop } Y \vdash Z$  ? Separate dyadic operator's right operand from its right argument (same as  $(X \text{ dop } Y)Z$ )
- ▶  $X \vdash Y$  ? Right: Y
- ▶  $X \vdash Y$  ? Church Boolean false (X if false, else Y)
- ▶  $\neg Y$  ? Same: Y
- ▶  $X \rightarrow Y$  ? Left: X
- ▶  $X \rightarrow Y$  ? Church Boolean true (X if true, else Y)
- ▶  $+Y$  ? Conjugate ('Identity' if Y not complex)
- ▶  $+N$  ? Mirror complex N across x-axis
- ▶  $M+N$  ? Adding N to M



Tell me about: `assert`



X,Y,Z:any M,N:num I,J:int A,B:Bool C,D:char f,g,h:fn ax:axis s:scal v:vec m:mat

▶ {α←'assertion failure' ◊ 0εω:α □ SIGNAL 8 ◊ shy←0}B

Signal an error if any condition is not met (optional left argument: message on failure)



X,Y,Z:any M,N:num I,J:int A,B:Bool C,D:char f,g,h:fn ax:axis s:scal v:vec m:mat

```
{α←'assertion failure' ◊ 0εω:α □ SIGNAL 8 ◊ shy←0}B
```

Signal an error if any condition is not met (optional left argument: message on failure)

# Assert

utility function

```
{ $\alpha$  ← 'assertion failure' ◇  
  0 ∈  $\omega$  :  $\alpha$  □ SIGNAL 8 ◇  
  shy ← 0 }
```

# Assert

utility function

```
{ $\alpha$  ← 'assertion failure'  
  0 ∈  $\omega$  :  $\alpha$  □ SIGNAL 8  
  shy ← 0 }
```

```
2 × {shy ← ω} 3
```

```
shy ← 0 }
```

```
2 × {shy ← ω} 3
```

```
6
```

```
shy ← 0 }
```



```
2 × {shy ← ω} 3  
6  
2 × {shy ← ω  ♦ } 3
```

```
shy ← 0 }
```

```
2×{shy←ω}3
```

```
6
```

```
2×{shy←ω ◊ }3
```

```
VALUE ERROR: No result was provided  
when the context expected one
```

```
2×{shy←ω ◊ }3
```

```
^
```

```
shy←0}
```

# Assert

utility function

```
{α←'assertion failure'  
  0∈ω:α □ SIGNAL 8  
  shy←0}
```

# Assert

utility function

```
{α←'assertion failure'  
  0∈ω:α □ SIGNAL 8  
  shy←0}
```



α □ SIGNAL 8

# Signal Error

```
□ SIGNAL ← ( 'EN' 11 ) ( 'EM' 'MEAN ERROR' )
```

α □ SIGNAL 8

# Signal Error

```
'MEAN ERROR' □ SIGNAL 11  
α □ SIGNAL 8
```

# Assert

utility function

```
{α←'assertion failure'  
  0∈ω:α □ SIGNAL 8  
  shy←0}
```



# Assert

utility function

```
{α←'assertion failure'  
  0∈ω:α □ SIGNAL 8  
  shy←0}
```

```
2 {  $\alpha \leftarrow 10$   $\diamond$   $\alpha \times \omega$  } 3
```

```
{  $\alpha \leftarrow$  'assertion failure'  
  0  $\in$   $\omega$  :  $\alpha$   $\square$  SIGNAL 8  
  shy  $\leftarrow$  0 }
```

```
2 {α←10 ♦ α×ω} 3
6
```

```
{α←'assertion failure'
 0∈ω:α □ SIGNAL 8
 shy←0}
```

$$2\{\alpha \leftarrow 10 \ \diamond \ \alpha \times \omega\} 3$$

6

$$\{\alpha \leftarrow 10 \ \diamond \ \alpha \times \omega\} 3$$

{ $\alpha \leftarrow$  'assertion failure'  
 $0 \in \omega : \alpha \ \square \ \text{SIGNAL} \ 8$   
 $\text{shy} \leftarrow 0$ }

$2\{\alpha \leftarrow 10 \ \diamond \ \alpha \times \omega\} 3$   
 6  
 $\{\alpha \leftarrow 10 \ \diamond \ \alpha \times \omega\} 3$   
 30

$\{\alpha \leftarrow \text{'assertion failure'}$   
 $0 \in \omega : \alpha \ \square \ \text{SIGNAL} \ 8$   
 $\text{shy} \leftarrow 0\}$

# Assert

utility function

```
{ $\alpha$  ← 'assertion failure'  
  0 ∈  $\omega$  :  $\alpha$  □ SIGNAL 8  
  shy ← 0 }
```

# Assert

utility function

```
{α←'assertion failure'  
  0∈ω:α □ SIGNAL 8  
  shy←0}
```

```
{2×ω}3
```

```
ert
```

```
nction
```

```
failure'
```

```
8
```

```
shy←0}
```



$\{2 \times \omega\} 3$ 

6

ert

nction

failure'

8

shy ← 0 }

```
6      {2×ω}3  
      {r←2×ω}3
```

```
shy←0}
```

ert

nction

failure'

8

6

 $\{2 \times \omega\} 3$  $\{r \leftarrow 2 \times \omega\} 3$  $\{1 + r \leftarrow 2 \times \omega\} 3$  $\text{shy} \leftarrow 0\}$ 

ert

nction

failure'

8

```
6      {2×ω}3  
      {r←2×ω}3  
7      {1+r←2×ω}3
```

```
shy←0}
```

ert

nction

failure'

8

```
6      {2×ω}3  
      {r←2×ω}3  
      {1+r←2×ω}3  
7  
      1+{r←2×ω}3
```

```
shy←0}
```

ert

nction

failure'

8

```

        {2×ω}3
6
        {r←2×ω}3
        {1+r←2×ω}3
7
        1+{r←2×ω}3
7

```

```
shy←0}
```

ert

nction

failure'

8

# Assert

utility function

```
{α←'assertion failure'  
  0∈ω:α □ SIGNAL 8  
  shy←0}
```

# Assert

utility function

```
{ $\alpha$  ← 'assertion failure' ◇  
  0 ∈  $\omega$  :  $\alpha$  □ SIGNAL 8 ◇  
  shy ← 0 }
```



# Assert: Use-case

```
M ← {
    'MEAN ERROR' Assert 0 <≠ ω:
    (+ / ω) ÷ ≠ ω
}
```

```
M 3 1 4 1 5
```

2.8

```
M θ
```

```
MEAN ERROR
```

```
M θ
```

```
^
```

# Assert: Use-case

```
M ← {  
    'MEAN ERROR' Assert 0 <≠ ω:  
    {0 ♦ ω} 3
```

2.8

MEAN

^

# Assert: Use-case

$M \leftarrow \{$   
 'MEAN ERROR' Assert  $0 < \neq \omega :$

$\{0 \diamond \omega\} 3$

0

2.8

MEAN

^

# Assert: Use-case

```
M ← {
    'MEAN ERROR' Assert 0 <≠ ω:
```

```
    {0 ♦ ω} 3
```

```
0
```

```
    {0: ♦ ω} 3
```

2.8

MEAN

^

# Assert: Use-case

```
M ← {
    'MEAN ERROR' Assert 0 <≠ ω:
```

```
    {0 ♦ ω} 3
```

```
0
```

```
    {0: ♦ ω} 3
```

```
2.8
```

```
3
```

```
MEAN
```

```
^
```

# Assert: Use-case

```
M ← {
    'MEAN ERROR' Assert 0 <≠ ω:
```

```
    {0 ♦ ω} 3
```

```
0
```

```
    {0: ♦ ω} 3
```

```
2.8
```

```
3
```

```
    {_ ← 2 ♦ ω} 3
```

```
MEAN
```

```
^
```

# Assert: Use-case

```
M ← {
    'MEAN ERROR' Assert 0 <≠ ω:
```

```
    {0 ♦ ω} 3
```

```
0
```

```
    {0: ♦ ω} 3
```

```
2.8
```

```
3
```

```
MEAN
```

```
    {_ ← 2 ♦ ω} 3
```

```
3
```

```
^
```

# Assert: Use-case

```
M ← {
    'MEAN ERROR' Assert 0 <≠ ω:
    (+ / ω) ÷ ≠ ω
}
```

```
M 3 1 4 1 5
```

2.8

```
M θ
```

```
MEAN ERROR
```

```
M θ
```

```
^
```



# Assert: Use-case

▽ `z ← M y`  
    `'MEAN ERROR' Assert 0 <≠ y`  
    `(+ / y) ÷ ≠ y`

▽  
 M 3 1 4 1 5

2.8

M θ  
 MEAN ERROR

M θ

^

# Assert: Use-case

```
SsetTest ← {  
  Assert 8 = sset 3:  
  Assert 551872 = sset 857:  
  Assert 935424 = sset 870:  
  'Test passed'  
}
```

# Assert: Use-case

[is.gd/apl2020](https://is.gd/apl2020)

```
SsetTest ← {  
  Assert 8 = sset 3:  
  Assert 551872 = sset 857:  
  Assert 935424 = sset 870:  
  'Test passed'  
}
```

# Resignal Error

utility function

```
□ SIGNAL { ... }
```

# Resignal

```
Lines ← {  
  text ← □NGET ω 1  
  ≠ text  
}
```

# Resignal

Lines '/tmp/my.txt'

# Resignal

```
Lines '/tmp/my.txt'
```

```
42
```

# Resignal

```
Lines '/tmp/my.txt'
```

42

```
Lines '/tmp/no.txt'
```



# Resignal

```
Lines '/tmp/my.txt'
```

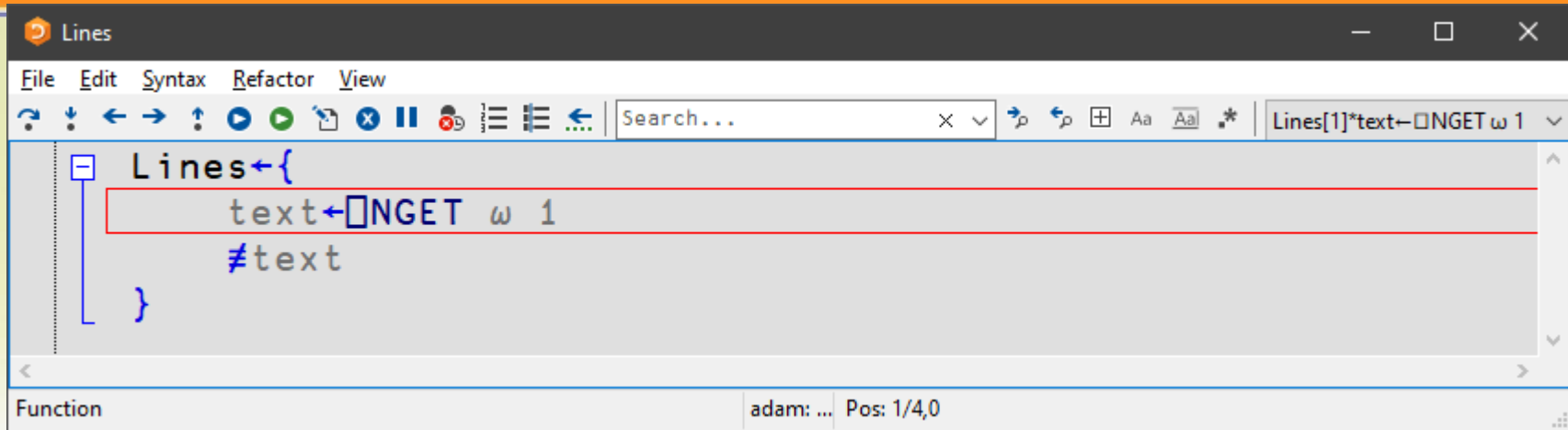
```
42
```

```
Lines '/tmp/no.txt'
```

```
FILE NAME ERROR: /tmp/no.txt: Unable to open ("T
```

```
Lines[1] text←NGET ω 1
```

```
^
```



The screenshot shows the Dyalog Lines editor window. The title bar reads "Lines". The menu bar includes "File", "Edit", "Syntax", "Refactor", and "View". The toolbar contains various icons for navigation and editing. The search bar is empty. The main editor area displays the following code:

```
Lines←{  
  text←⎕NGET ω 1  
  ≠text  
}
```

The line `text←⎕NGET ω 1` is highlighted with a red box. The status bar at the bottom shows "Function" and "adam: ... Pos: 1/4,0".

FILE NAME ERROR: /tmp/no.txt: Unable to open ("T

```
Lines[1] text←⎕NGET ω 1
```

^

# Resignal

```
Lines ← {  
  0 :: □ SIGNAL ...  
  text ← □ NGET ω 1  
  ≠ text  
}
```

# Resignal

```
Lines ← {  
  0:: □ SIGNAL ...  
  text ← □ NGET ω 1  
  ≠ text
```

0::  
ALL  
ERRORS

# Resignal

```
Lines ← {  
  0:: □ SIGNAL ...  
  text ↔ □ NGET ω 1  
  ≠ text
```

0::  
ALL  
ERRORS



aplcart.info

X,Y,Z:any M,N:num I,J:int A,B:Bool C,D:char f,g,h:fn ax:axis s:scal v:vec m:mat

- ▶  $\emptyset$  ? Empty Numeric Vector
- ▶  $\vdash Y$  ? Same: Y
- ▶  $X \text{ dop } Y \vdash Z$  ? Separate dyadic operator's right operand from its right argument (same as  $(X \text{ dop } Y)Z$ )
- ▶  $X \vdash Y$  ? Right: Y
- ▶  $X \vdash Y$  ? Church Boolean false (X if false, else Y)
- ▶  $\neg Y$  ? Same: Y
- ▶  $X \rightarrow Y$  ? Left: X
- ▶  $X \rightarrow Y$  ? Church Boolean true (X if true, else Y)
- ▶  $+Y$  ? Conjugate ('Identity' if Y not complex)
- ▶  $+N$  ? Mirror complex N across x-axis
- ▶  $M+N$  ? Adding N to M

X,Y,Z:any M,N:num I,J:int A,B:Bool C,D:char f,g,h:fn ax:axis s:scal v:vec m:mat

```
▶ resignal <= DMX. (( 'EN' EN) ('EM' EM)
('Message' (OSError{ω, 2φ(×≠⇒θρ2φα, c'') / '' )
('', o↔⇒θρ2φα}Message)))
```

Re-signal last caught error to caller (works with any IO and ML)



Tell me about: **resignal**



X,Y,Z:any M,N:num I,J:int A,B:Bool C,D:char f,g,h:fn ax:axis s:scal v:vec m:mat

```

▶ SIGNAL ← DMX. (( 'EN' EN) ('EM' EM)
('Message' (OSError{ω, 2φ(x≠⇒θρ2φα, c'') / ''}
('', o↔⇒θρ2φα}Message)))

```

Re-signal last caught error to caller (works with any IO and ML)





# Resignal

```
Lines ← {  
  0 :: □ SIGNAL ← □ DMX . ( ( ' EN ' EN ) ( ' EN ' EM ) ( ' Mes  
  text ← □ NGET ω 1  
  ≠ text  
}
```

# Resignal

```
Lines '/tmp/my.txt'
```

```
42
```

```
Lines '/tmp/no.txt'
```

```
FILE NAME ERROR: /tmp/no.txt: Unable to open file
```

```
Lines '/tmp/no.txt'
```

```
^
```

# Resignal

```
Lines ← {  
  0 :: resignal  
  text ← □NGET ω 1  
  ≠ text  
}  
▽ resignal  
  □SIGNAL ← □DMX. ( ( 'EN' EN )  
▽
```

# Resignal

```
Lines '/tmp/my.txt'
```

42

```
Lines '/tmp/no.txt'
```

# Resignal

```
Lines '/tmp/my.txt'
```

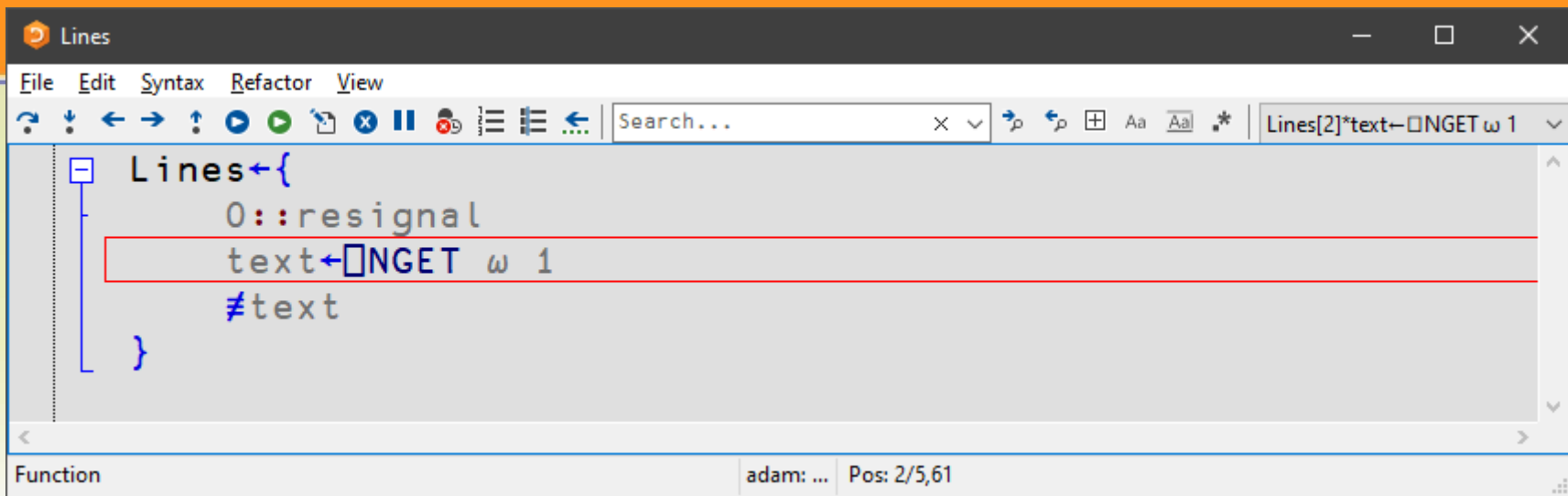
```
42
```

```
Lines '/tmp/no.txt'
```

```
FILE NAME ERROR: /tmp/no.txt: Unable to open file
```

```
Lines[1] 0::resignal
```

```
^
```



The screenshot shows the Dyalog Lines editor window. The title bar reads "Lines". The menu bar includes "File", "Edit", "Syntax", "Refactor", and "View". The toolbar contains various icons for navigation and editing. A search bar is visible with the text "Search...". The main editor area displays the following code:

```
Lines←{  
  0::resignal  
  text←□NGET ω 1  
  ≠text  
}
```

The line `text←□NGET ω 1` is highlighted with a red rectangular box. The status bar at the bottom shows "Function", "adam: ...", and "Pos: 2/5,61".

FILE NAME ERROR: /tmp/no.txt: Unable to open file  
Lines[1] 0::resignal

^

# Resignal

```
Lines ← {  
  0 :: resignal  
  text ← □NGET ω 1  
  ≠ text  
}  
▽ resignal  
  □SIGNAL ← □DMX. ( ( 'EN' EN )  
▽
```

# Resignal

```
Lines ← {  
  0 :: Resignal θ  
  text ← □NGET ω 1  
  ≠ text  
}  
Resignal ← { □SIGNAL ← □DMX . ( ( 'EN' EN )
```



# Resignal

```
Lines '/tmp/my.txt'
```

42

```
Lines '/tmp/no.txt'
```

# Resignal

```
Lines '/tmp/my.txt'
```

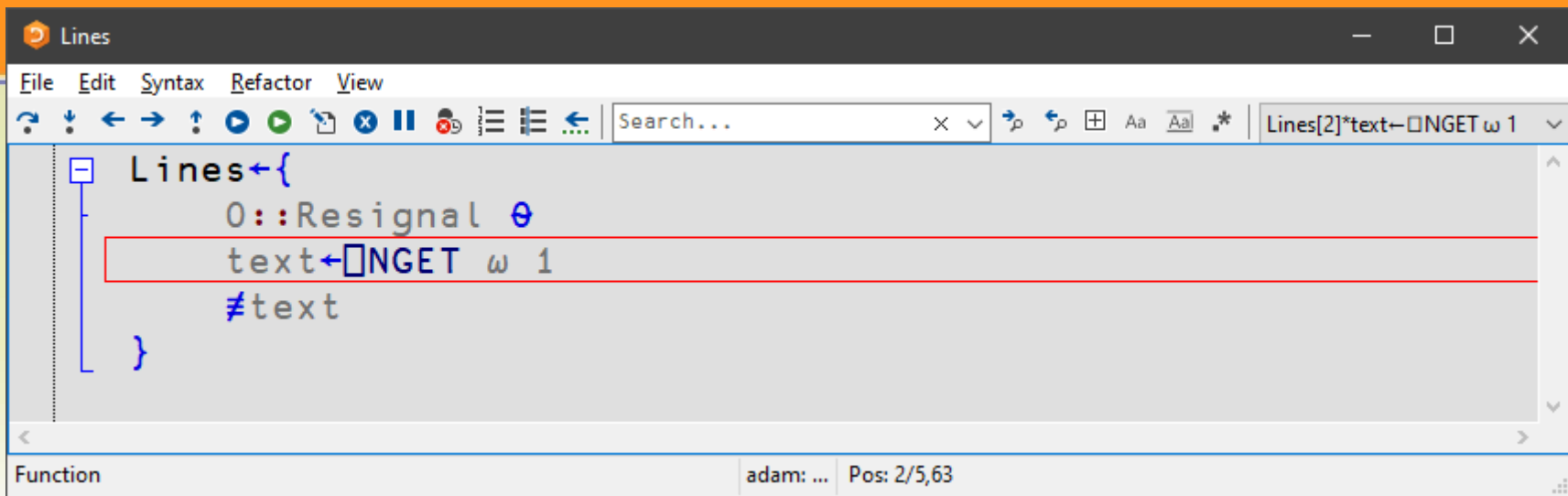
```
42
```

```
Lines '/tmp/no.txt'
```

```
FILE NAME ERROR: /tmp/no.txt: Unable to open file
```

```
Lines[1] 0::Resignal 0
```

```
^
```



The screenshot shows the Dyalog APL IDE interface. The title bar reads "Lines". The menu bar includes "File", "Edit", "Syntax", "Refactor", and "View". The toolbar contains various icons for navigation and editing. A search bar is visible with the text "Search...". The main editor area displays the following APL code:

```
Lines←{  
  0::Resignal 0  
  text←⊞NGET ω 1  
  ≠text  
}
```

The line `text←⊞NGET ω 1` is highlighted with a red box. The status bar at the bottom shows "Function" and "adam: ... Pos: 2/5,63".

FILE NAME ERROR: /tmp/no.txt: Unable to open file  
Lines[1] 0::Resignal 0

^

# Resignal

```
Lines ← {  
  0 :: Resignal θ  
  text ← □NGET ω 1  
  ≠ text  
}  
Resignal ← { □SIGNAL ← □DMX . (
```

# Resignal

```
Lines ← {  
  0 :: Resignal θ  
  text ← □ NGET ω 1  
  ≠ text  
}  
Resignal ← □ SIGNAL { c □ DMX . (
```

# Resignal

```
Foo ← { Goo θ }
```

1

```
Resignal ← □ SIGNAL { c □ DMX . (
```

# Resignal

```
Foo ← { Goo θ }  
Goo ← { □SI }
```

1

```
Resignal ← □ SIGNAL { c □ DMX . (
```

# Resignal

```
Foo ← { Goo 0 }  
Goo ← { [SI] }
```

*Stack  
Indicator*

1

```
Resignal ← [ SIGNAL { c [ DMX . (
```

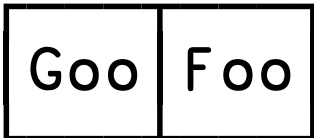


# Resignal

```
Foo ← { Goo θ }
```

```
Goo ← { □ SI }
```

```
Foo θ
```



*Stack Indicator*

1

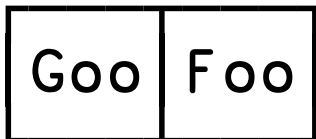
```
Resignal ← □ SIGNAL { c □ DMX . (
```

# Resignal

```

Foo ← { Goo θ }
Goo ← { □SI }
Foo θ

```



*Stack  
Indicator*

```

Foo ← { Goo θ }
Goo ← ◻ ◦ '□SI' ◻ ◦ #

```

```

Resignal ← □ SIGNAL { ◻ DMX . (

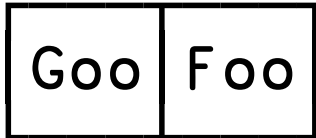
```

# Resignal

```

Foo ← { Goo θ }
Goo ← { □ SI }
Foo θ

```



*Stack Indicator*

```

Foo ← { Goo θ }
Goo ← ◻ ◦ '□ SI' ◻ ◦ #
Foo θ

```



```

Resignal ← □ SIGNAL { ◻ DMX . (

```

# Resignal

```
Lines '/tmp/my.txt'
```

42

```
Lines '/tmp/no.txt'
```

# Resignal

```
Lines '/tmp/my.txt'
```

```
42
```

```
Lines '/tmp/no.txt'
```

```
FILE NAME ERROR: /tmp/no.txt: Unable to open file
```

```
Lines '/tmp/no.txt'
```

```
^
```

## Error Handling – Part 3

Signal an error: `□ SIGNAL errno`

Signal a custom error: `□ SIGNAL <name-value pairs`

Assert: `aplcart.info?q=assert`

Resignal: `aplcart.info?q=resignal`

## Error Handling – Part 3

Press **F1** for full docs!

Signal an error: `□ SIGNAL` `errno`

Signal a custom error: `□ SIGNAL` `name-value pairs`

Assert: `aplcart.info?q=assert`

Resignal: `aplcart.info?q=resignal`

# Upcoming webinars

Jun	17	BAA	open session
Jun	24	---	
July	1	BAA	open session
July	8	Dyalog	TBD
July	15	BAA	open session
July	22	---	
July	29	BAA	open session
Aug	5	Dyalog	Error handling – pt. 4



# Upcoming webinars

Jun	17	BAA	open session
Jun	24	---	
July	1	BAA	open session
July	8	Dyalog	TBD
July	15	BAA	open session
July	22	---	
July	29	BAA	open session
Aug	5	Dyalog	Error handling – pt. 4

One open session will be AGM  
(Annual General Meeting)  
For details, check BAA's site:  
[britishaplassociation.org](http://britishaplassociation.org)