



- ▶ Multi Threading
- ▶ Synchronisation

Wait for threads to terminate:

`{R}←□TSYNC Y`

Y must be a simple array of thread numbers.

If *Y* is a simple scalar, *R* is an array; the result (if any) of the thread.

If *Y* is a simple non-scalar, *R* has the same shape as *Y*, and result is an array of enclosed thread results.

The interpreter detects a potential deadlock if a number of threads wait for each other in a cyclic dependency. In this case, the thread that attempts to cause the deadlock issues error number 1008: *DEADLOCK*.

Examples:

```

dup←{w w}           A Duplicate
□←dup&88           A Show thread number
11
88 88
□←□TSYNC dup&88   A Wait for result
88 88
□TSYNC,dup&88
88 88
□TSYNC dup&1 2 3
1 2 3 1 2 3
□TSYNC dup&"1 2 3
1 1 2 2 3 3
□TSYNC □TID       A Wait for self
DEADLOCK
□TSYNC □TID
^
□EN
1008
    
```

Dyalog Ltd
 South Barn
 Minchens Court
 Minchens Lane
 Bramley
 Hampshire
 RG26 5BH
 United Kingdom

Phone:
 + 44 (0) 1256 830 030

Fax:
 + 44 (0) 1256 830 031

e-mail:
 sales@dyalog.com



▶ Multi Threading

▶ Synchronisation

Dyalog Ltd

South Barn
Minchens Court
Minchens Lane
Bramley
Hampshire
RG26 5BH
United Kingdom

Phone:

+ 44 (0) 1256 830 030

Fax:

+ 44 (0) 1256 830 031

e-mail:

sales@dyalog.com

Hold:

:Hold Y

Whenever more than one thread tries to access the same piece of data or shared resource at the same time, you need some type of synchronisation to control access to that data. This is provided by *:Hold*.

:Hold provides a mechanism to control thread entry into a critical section of code. *Y* must be a simple character vector or scalar, or a vector of character vectors. *Y* represents a set of 'tokens', all of which must be acquired before the thread can continue into the control structure. *:Hold* is analogous to the component file system `□FHOLD`.

Syntax

```
:Hold token(s)
|
| code
|
|-----|
|           |
|           | :Else
|           |
|           | code
|           |
|<-----|
|
| :End[Hold]
|
```

Within the whole active workspace, a token with a particular value may be held only once. If the hold succeeds, the current thread *acquires* the tokens and execution continues with the first phrase in the control structure. On exit from the structure, the tokens are released for use by other threads. If the hold fails, because one or more of the tokens is already in use:

If there is no *:Else* clause in the control structure, execution of the thread is blocked until the requested tokens become available.



- ▶ Multi Threading
- ▶ Synchronisation

Otherwise, acquisition of the tokens is abandoned and execution resumed immediately at the first phrase in the `:Else` clause.

`Y` can be either a single token:

```
'a'
'Red'
'#.Util'
''
'Program Files'
```

... Or a number of tokens:

```
'red' 'green' 'blue'
'doe' 'a' 'deer'
, '' 'abc'
↓ nl 9
```

Pre-processing removes trailing blanks from each token before comparison, so that, for example, the following two statements are equivalent:

```
:Hold 'Red' 'Green'
:Hold ↓2 5p'Red Green'
```

Unlike `□FHOLD`, a thread does not release all existing tokens before attempting to acquire new ones. This enables the nesting of holds, which can be useful when multiple threads are concurrently updating parts of a complex data structure.

In the following example, a thread updates a critical structure in a child namespace, and then updates a structure in its parent space. The holds will allow all 'sibling' namespaces to update concurrently, but will constrain updates to the parent structure to be executed one at a time.

```
:Hold □cs'' a Hold child space
... a Update child space
:Hold ##.□cs'' a Hold parent space
... a Update Parent space
:EndHold

...
:EndHold
```

Dyalog Ltd

South Barn
Minchens Court
Minchens Lane
Bramley
Hampshire
RG26 5BH
United Kingdom

Phone:

+ 44 (0) 1256 830 030

Fax:

+ 44 (0) 1256 830 031

e-mail:

sales@dyalog.com



- ▶ Multi Threading
- ▶ Synchronisation

However, with the nesting of holds comes the possibility of a 'deadlock'. For example, consider the two threads:

```

Thread 1           Thread 2
-----
:Hold 'red'       :Hold 'green'
...
:Hold 'green'     :Hold 'red'
...
:EndHold          :EndHold
:EndHold          :EndHold
    
```

In this case if both threads succeed in acquiring their first hold, they will both block waiting for the other to release its token. Fortunately, the interpreter detects such cases and issues an error (1008) *DEADLOCK*. You can avoid deadlock by ensuring that threads always attempt to acquire tokens in the same chronological order, and that threads never attempt to acquire tokens that they already own.

Note that token acquisition for any particular *:Hold* is atomic, that is, either *all* of the tokens or *none* of them are acquired. The following example *cannot* deadlock:

```

Thread 1           Thread 2
-----
:Hold 'red'       ...
:Hold 'green' 'red' :Hold 'green'
...
:EndHold          :EndHold
:EndHold
    
```

Examples:

:Hold could be used for example, during the update of a complex data structure that might take several lines of code. In this case, an appropriate value for the token would be the name of the data structure variable itself, although this is just a programming convention: the interpreter does not associate the token value with the data variable.

```

:Hold 'Struct'
...
Struct ← ...
:EndHold
    
```

* Update Struct

Dyalog Ltd

South Barn
Minchens Court
Minchens Lane
Bramley
Hampshire
RG26 5BH
United Kingdom

Phone:

+ 44 (0) 1256 830 030

Fax:

+ 44 (0) 1256 830 031

e-mail:

sales@dyalog.com



- ▶ Multi Threading
- ▶ Synchronisation

The next example guarantees exclusive use of the current namespace:

```
:Hold [CS]'      * Hold current space
...
:EndHold
```

The following example shows code that holds two positions in a vector while the contents are exchanged.

```
:Hold *to fm
  :If >/vec[fm to]
    vec[fm to]←vec[to fm]
  :End
:End
```

Between obtaining the next available file tie number and using it:

```
:Hold '[FNUMS'
  tie←1+[ /0,[FNUMS
  fname [FSTIE tie
:End
```

The above hold is not necessary if the code is combined into a single line:

```
fname [FSTIE tie←1+[ /0,[FNUMS
or,
tie←fname [FSTIE 0
```

Note that *:Hold*, like its component file system counterpart: *[FHOLD*, is a device to enable *co-operating* threads to synchronise their operation.

:Hold does not *prevent* threads from updating the same data structures concurrently, it prevents threads only from *:Hold*-ing the same tokens.

Dyalog Ltd

South Barn
Minchens Court
Minchens Lane
Bramley
Hampshire
RG26 5BH
United Kingdom

Phone:

+ 44 (0) 1256 830 030

Fax:

+ 44 (0) 1256 830 031

e-mail:

sales@dyalog.com



- ▶ Multi Threading
- ▶ Synchronisation

Display held tokens:

```
)HOLDS
```

System command `)HOLDS` displays a list of tokens which have been acquired or requested by the `:Hold` control structure. Each line of the display is of the form:

```
token:   acq      req      req ...
```

Where `acq` is the number of *the* thread that has acquired the token, and `req` is the number of *a* thread which is requesting it. For a token to appear in the display, a thread (and only one thread) must have acquired it, whereas any number of threads can be requesting it.

Example:

Thread 300's attempt to acquire token '`blue`' results in a deadlock:

```
300:DEADLOCK
      Sema4[1] :Hold 'blue'
      ^
      )HOLDS
blue:   100
green:  200      100
red:    300      200      100
```

- Blue has been acquired by thread 100.
- Green has been acquired by 200 and requested by 100.
- Red has been acquired by 300 and requested by 200 and 100.

The following cycle of dependencies has caused the deadlock:

- Thread 300 attempts to acquire `blue`,
- which is owned by 100,
- which is waiting for `red`,
- which is owned by 300.

```
300 → blue
      ↑      ↓
red ← 100
```

Dyalog Ltd

South Barn
Minchens Court
Minchens Lane
Bramley
Hampshire
RG26 5BH
United Kingdom

Phone:

+ 44 (0) 1256 830 030

Fax:

+ 44 (0) 1256 830 031

e-mail:

sales@dyalog.com