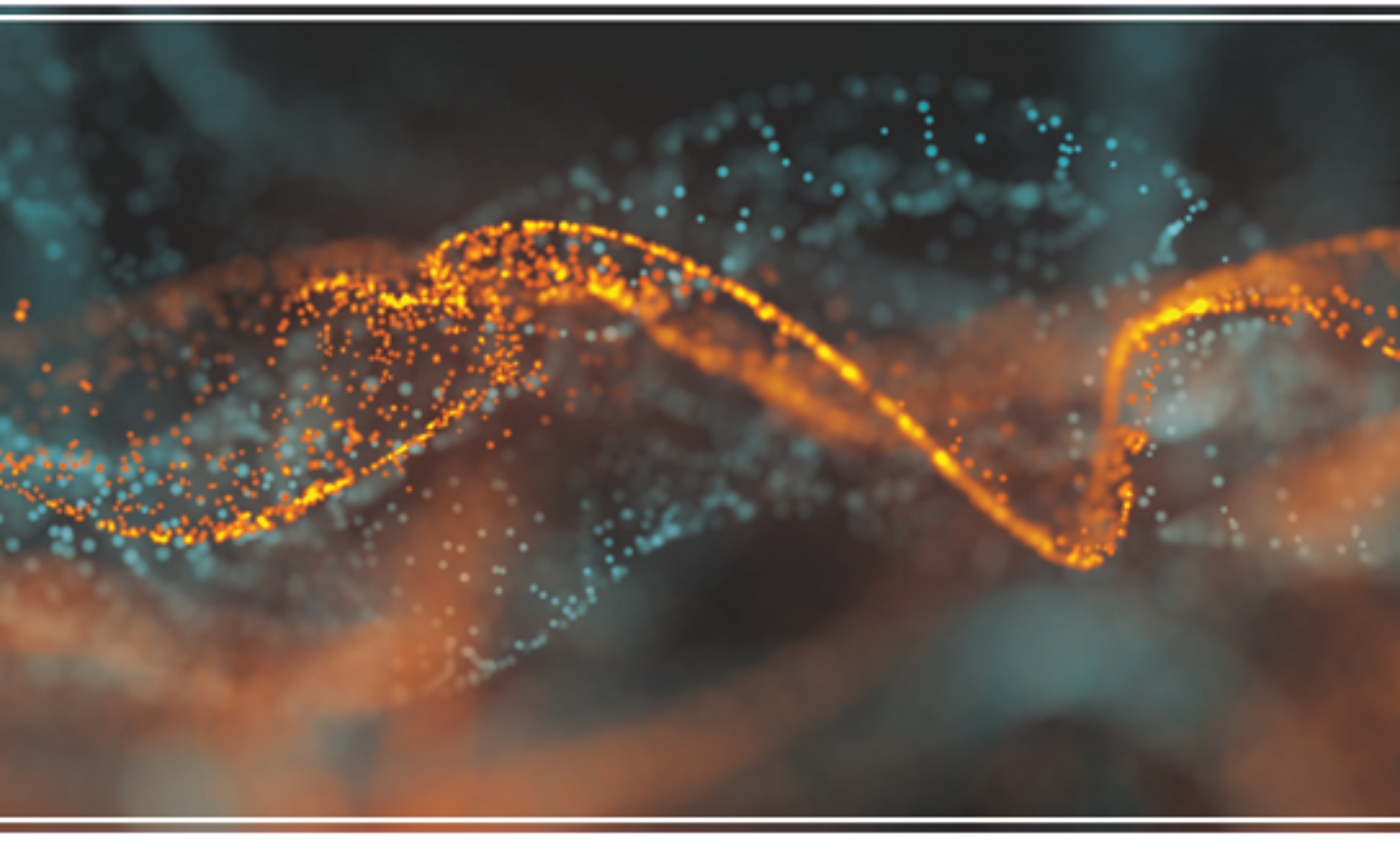
# Dyalog
# Release Notes

## Dyalog version 17.0



DYALOG

The tool of thought for software solutions

*Version: 17.0*

*Revision: 2561 dated 20181002*

**Please note that unless otherwise stated, all the examples in this document assume that ⎕IO is 1, and ⎕ML is 1.**



*email: support@dyalog.com*
*https://www.dyalog.com*

# Contents

# Chapter 1:

# Introduction

## Key Features

Dyalog APL Version 17.0 provides the following new features, enhancements and changes:

### Performance Improvements

As part of the ongoing Performance Quality Assurance project[1], Version 17.0 includes a considerable amount of research and development work designed to substantially improve speed of execution[2].

The performance of 128-bit Decimal floating-point calculations on all platforms which do not have hardware support for such numbers has been significantly improved by changing the internal format for such numbers. See *128-Bit Decimal Numbers* on page 17.

- Editing scripts is significantly faster. The improvement is especially noticeable on large scripts.
- `⎕NINFO` is faster when the Wildcard option is enabled and the name does not contain any wildcard characters.

#### New Idioms

The following new idioms provide the fastest way to sort arrays of any rank:

| Expression | Description |
|---|---|
| `{(⊂⍋⍵)⌷⍵}⍤⍋⍤1⊢A` | `A` sorted into ascending order |
| `{(⊂⍒⍵)⌷⍵}⍤⍋⍤1⊢A` | `A` sorted into descending order |

[1] https://www.dyalog.com/blog/2016/03/pqa/

[2] https://www.dyalog.com/dyalog/dyalog-versions/170/performance.htm

# Language Enhancements

## New Language Features

- Monadic Grade Up (⍋), Grade Down (⍒) and Interval Index (⍳) have been extended to apply to all arrays. See *Total Array Ordering* on page 4.
- New system function ⎕NCOPY. See *Native File Copy* on page 57.
- New system function ⎕NMOVE. See *Native File Move* on page 75.
- The localisation of names in the header of a defined function and operator has been augmented by *Locals Lines* which may appear between line `[0]` and the first executable statement. See *Locals Lines* on page 13.

## Enhancements

A number of the native file functions have been extended to handle multiple files. Instead of accepting just a single file name, they now accept zero, one, or multiple file names. Where appropriate these functions also accept wildcards in names.

- The Unique primitive function (monadic ∪) has been extended to work with all arrays, not just vectors. See *Unique* on page 40.
- ⎕CSV provides a new column type (numeric/empty) and 3 new variant options to handle metacharacters. The **Overwrite** variant option is replaced by **IfExists**. See *Comma Separated Values* on page 41.
- ⎕MKDIR accepts multiple file names. See *Make Directory* on page 55.
- ⎕NDELETE accepts multiple file names, wildcards, and has an option to delete non-empty directories. See *Native File Delete* on page 64.
- ⎕NEXISTS accepts multiple file names and wildcards. See *Native File Exists* on page 66.
- ⎕NINFO accepts multiple file names, reports additional properties and has a **Recurse** variant option. See *Native File Information* on page 70.
- ⎕NPARTS accepts multiple file names. See *File Name Parts* on page 79.
- ⎕NPUT now supports an option to append data to a file. See *Write Text File* on page 81.
- ⎕NCREATE is extended to allow semi-automatic naming of files and to allow existing files to be overwritten. See *Native File Create* on page 62.
- System functions ⎕CSV, ⎕NGET, ⎕NPUT, ⎕R, ⎕S which accept an encoding name (e.g. `'Windows-1252'`) have been extended to support any user-definable 1-byte character set. See *Comma Separated Values* on page 41, *Read Text File* on page 67, and *Write Text File* on page 81.
- To facilitate the use of all system functions in dfns, those that previously did not return a result now return a shy result. See *Shy Results for System Functions* on page 34.

- Global Triggers have been enhanced to identify indexed assignment. See *Global Triggers* on page 15.

### New I-beam Features

- A new I-beam function provides the list of syntax colour tokens. See *Syntax Colour Tokens* on page 84
- A new I-beam function obtains the name of a system temporary directory. See *Temporary Directory* on page 85.
- Two new I-beam functions have been provided to better manage the information pertaining to objects associated with files. See *Loaded File Object Info* on page 86 and *Remove Loaded File Object Info* on page 85. The information reported has been supplemented by the file checksum and modification date.

### New Command-Line Options

- New **-apl** and **-cef** options are provided to control how Command-Line parameters are processed. See *Command-Line Options* on page 25.

## IDE Enhancements

- The Find Objects Tool provides a simpler, cleaner user-interface and is no longer a tabbed dialog. See *Find Objects Tool* on page 27.
- The Backtick keyboard provided by the RIDE may now be used natively. See *Backtick Keyboard* on page 26.

## GUI Enhancements

- The CellMove event message contains 2 new elements to resolve a validation anomaly that occurred when CellMove and CellChange events activated callbacks on the same Grid. See *CellMove* on page 88.

# Total Array Ordering

Total Array Ordering (TAO) extends the sort functions Grade Up (⍋), Grade Down (⍒) and Interval Index (⍳) to handle all APL arrays, not just simple numeric and character arrays. In practice, TAO extends to complex numbers, nested arrays and ⎕NULL but not to namespaces. For an introduction to the concept of total order in mathematics see https://en.wikipedia.org/wiki/Total_order.

In order to implement TAO, according to which any APL array must be comparable to any other APL array, it has been necessary to extend the set of rules that govern these comparisons. The new set of rules is as follows.

### Rules for comparing simple scalars

- Numeric comparisons are exact, as if ⎕CT←⎕DCT←0 and ⎕FR←1287
- Two real numbers are compared numerically, thus 1.2 precedes 3.
- In the Unicode Edition two characters are compared numerically according to their position in the Unicode table. Thus 'a' (⎕UCS 97) precedes 'b' (⎕UCS 98). In the Classic Edition characters are compared according to their index in ⎕AV.
- Complex numbers are ordered by first comparing their real parts. If these are equal, the order is determined by comparing their imaginary parts. Thus 1J¯2 precedes 1 which precedes 1J2.
- ⎕NULL (which represents a null item obtained from an external source) precedes all numbers, and all numbers precede all characters. Thus ⎕NULL precedes 100, and 100 precedes 'A'.

### Rules for comparing non-scalar arrays

- Arrays are compared item by item in ravel order.
- For arrays of equal shape, the order is determined by the first pair of items which differ, thus (1949 4 29) precedes (1949 4 30). Similarly ('April' 29) precedes ('April' 30).
- Arrays with the same rank but different shape are ordered as if the shorter array were padded with items that precede all other types of item (negative infinity) including ⎕NULL. Thus 'car' precedes 'carpet' and (1949 4) precedes (1949 4 30). An alternative model is to say that shorter arrays precede longer ones that begin the same way. For character vectors this is described as lexicographical ordering, which is the order that words appear in a dictionary.
- Arrays with differing rank are ordered by first extending the shape of the lower-ranked array with 1s at the beginning, and then comparing the resultant equal-rank arrays as described above. So, to compare a vector (rank 1) with a matrix (rank 2), the vector is reshaped into a 1-row matrix.

- Empty arrays are compared first by type alone, so an empty numeric array precedes an empty character array, regardless of rank or shape.
  Thus `((0 3 2)⍴0)` precedes `''`. If the empty arrays are of the same type, they are sorted in order of their shape vector, working right to left.
  So `((0 5 2)⍴99)` precedes `((0 3 4)⍴0)` and
  `((0 3 4)⍴'')` precedes `((1 0 5 4)⍴'')`.

## Nested Array Example (phone book)

In our office, certain people have two phone extensions. We want to sort our phone book first by name and then by extension number. With TAO, this just falls out.

```
      ⍴pb
6 3
      pb
```

| Rivers | Jason | 554 |
|---|---|---|
| Daintree | John | 532 |
| Rivers | Jason | 543 |
| Foad | Jay | 558 |
| Scholes | John | 547 |
| Scholes | John | 535 |

```
      Sort←{(⊂⍋⍵)⌷⍵}

      ⊢pb←Sort pb
```

| Daintree | John | 532 |
|---|---|---|
| Foad | Jay | 558 |
| Rivers | Jason | 543 |
| Rivers | Jason | 554 |
| Scholes | John | 535 |
| Scholes | John | 547 |

Interval Index ⍸ is also extended to support TAO, so we can use it to find the position for a new member of staff:

```
      new←'Kromberg' 'Morten' 584
      pb ⍸ new
2
```

Then we can insert the new row into the appropriate position using ↑ and ↓:

```
      (2↑pb),⊂new,(2↓pb)
```

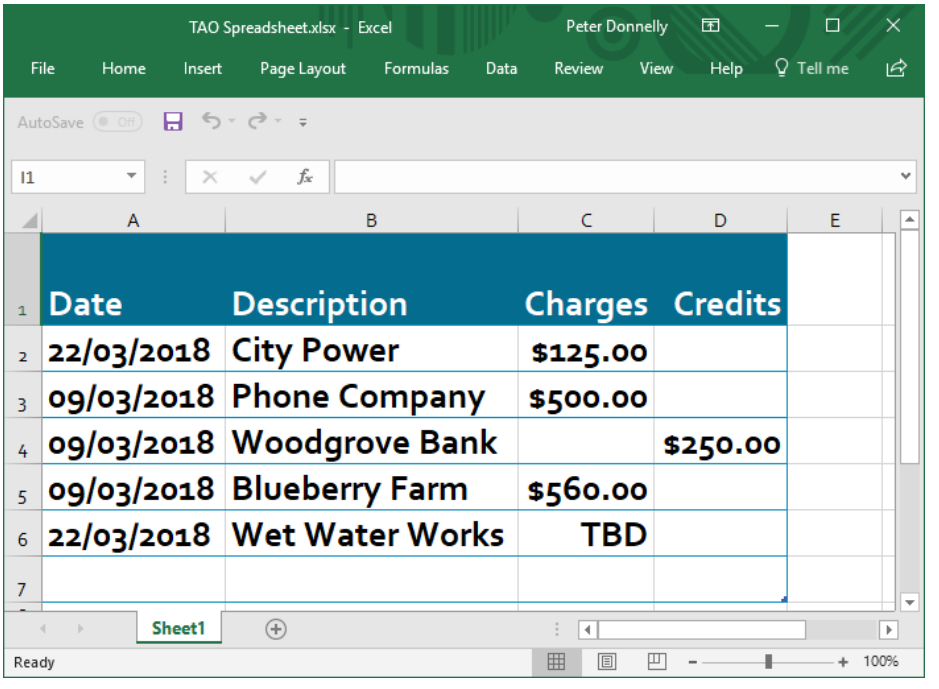
| | | |
|---|---|---|
| Daintree | John | 532 |
| Foad | Jay | 558 |
| Kromberg | Morten | 584 |
| Rivers | Jason | 543 |
| Rivers | Jason | 554 |
| Scholes | John | 535 |
| Scholes | John | 547 |

... although a somewhat neater solution is to use a function train:

```
      Into←{(ω⍸α)(↑,⊂α,↓)ω}
      new Into pb
```

| | | |
|---|---|---|
| Daintree | John | 532 |
| Foad | Jay | 558 |
| Kromberg | Morten | 584 |
| Rivers | Jason | 543 |
| Rivers | Jason | 554 |
| Scholes | John | 535 |
| Scholes | John | 547 |

### Spreadsheet Example




A common requirement is to sort a table by a particular column, or by first one column and then another, and so forth ...

Here is a Dyalog matrix resulting from importing the Excel spreadsheet shown above using the Clipboard object. Note that the first column is an array of dates in ⎕TS format.

```
      ⊢ss←(⎕NEW'Clipboard').Array
```

| Date | Description | Charges | Credits |
|---|---|---|---|
| 2018 3 22 0 0 0 0 | City Power | 125 | [Null] |
| 2018 3 9 0 0 0 0 | Phone Company | 500 | [Null] |
| 2018 3 9 0 0 0 0 | Woodgrove Bank | [Null] | 250 |
| 2018 3 9 0 0 0 0 | Blueberry Farm | 560 | [Null] |
| 2018 3 22 0 0 0 0 | Wet Water Works | TBD | [Null] |
| [Null] | [Null] | [Null] | [Null] |

When we sort the array, ⍋ will begin by comparing the items, one by one, in the first column.

According to the TAO rule, ⎕NULL precedes all the numbers 2018, so the last row in the table sorts first. Similarly, the numbers 2018 all precede the 'D' in Date, so the first row sorts last.

Then amongst the 7-element numeric vectors, (2018 3 9 0 0 0 0) precedes (2018 3 22 0 0 0 0) because 9 precedes 22. Finally, those rows with the same date are ordered by the next item which is the character vector in column 2.

```
      Sort ss
```

| [Null] | [Null] | [Null] | [Null] |
|---|---|---|---|
| 2018 3 9 0 0 0 0 | Blueberry Farm | 560 | [Null] |
| 2018 3 9 0 0 0 0 | Phone Company | 500 | [Null] |
| 2018 3 9 0 0 0 0 | Woodgrove Bank | [Null] | 250 |
| 2018 3 22 0 0 0 0 | City Power | 125 | [Null] |
| 2018 3 22 0 0 0 0 | Wet Water Works | TBD | [Null] |
| Date | Description | Charges | Credits |

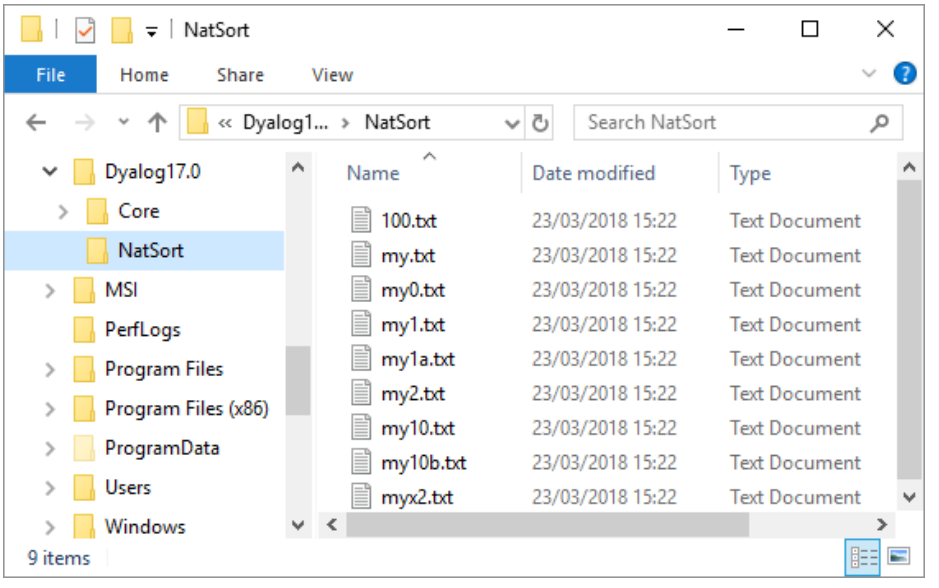
We can sort by Charges within Date by ordering the first and third columns:

```
      ss[⍋ss[;1 3];]
```

| [Null] | [Null] | [Null] | [Null] |
|---|---|---|---|
| 2018 3 9 0 0 0 0 | Woodgrove Bank | [Null] | 250 |
| 2018 3 9 0 0 0 0 | Phone Company | 500 | [Null] |
| 2018 3 9 0 0 0 0 | Blueberry Farm | 560 | [Null] |
| 2018 3 22 0 0 0 0 | City Power | 125 | [Null] |
| 2018 3 22 0 0 0 0 | Wet Water Works | TBD | [Null] |
| Date | Description | Charges | Credits |

### Natural Sort Example

Windows Explorer sorts file names in *Natural Order* as illustrated below. Where file names contain numbers, the numeric parts are ordered in numerical order, so for example, `my2` precedes `my10` because 2 precedes 10.



```
      names
```

| myx2 | my10b | my | 100 | my0 | my2 | my1 | my1a | my10 |
|---|---|---|---|---|---|---|---|---|

`names` is a vector of character vectors containing the file names shown above. If we sort `names` using ⍋ we will get a lexicographical ordering:

```
      Sort names
```

| 100 | my | my0 | my1 | my10 | my10b | my1a | my2 | myx2 |
|---|---|---|---|---|---|---|---|---|

Note that in alphabetic order, `my10` precedes `my2` because the character `'1'` precedes the character `'2'`. In order to sort the array into *Natural Order* we need to split each name into its character and numeric parts. First we need to identify the numeric digits:

```
      Digits ← {ω∈⎕D}
      Digits ¨names
```

| 0 0 0 1 | 0 0 1 1 0 | 0 0 | 1 1 1 | 0 0 1 | 0 0 1 | 0 0 1 | 0 0 1 0 | 0 0 1 1 |
|---|---|---|---|---|---|---|---|---|

Next, we create a partition vector to identify the start positions of each of the parts:

```
      CutOffs ← {1,2≠/Digits ω}
      CutOffs¨names
```

| 1 0 0 1 | 1 0 1 0 1 | 1 0 | 1 0 0 | 1 0 1 | 1 0 1 | 1 0 1 | 1 0 1 1 | 1 0 1 0 |
|---|---|---|---|---|---|---|---|---|

and use this to partition each of the strings into its character and numeric parts.

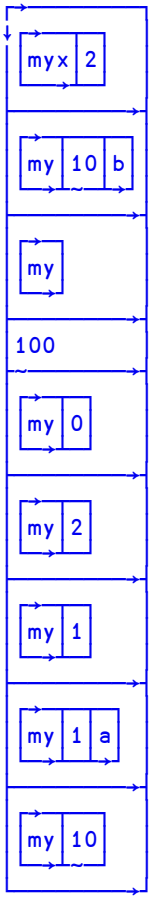
```
      Parts ← {(CutOffs ω)⊂ω}
      Parts¨¯1↓names  ⍝ Drop last element to fit page
```

| myx \| 2 | my \| 10 \| b | my | 100 | my \| 0 | my \| 2 | my \| 1 | my \| 1 \| a |
|---|---|---|---|---|---|---|---|

Then we must convert the numeric strings into numbers. Here it is safe to use ⍎.

```
      ExecNums ← {∧/Digits ⍵:⍎⍵ ◇ ⍵}

      ⍪ExecNums¨∘Parts¨ names
```



Then we can define the function `Order` using ⍋.

```
Order ← {⍋ExecNums¨∘Parts¨⍵}
```

and finally the function `NatSort` to order the array.

```
NatSort ← {(⊂Order ⍵)⌷⍵}
NatSort names
```

| 100 | my | my0 | my1 | my1a | my2 | my10 | my10b | myx2 |
|---|---|---|---|---|---|---|---|---|

```
⍪NatSort names
```

| 100 |
|---|
| my |
| my0 |
| my1 |
| my1a |
| my2 |
| my10 |
| my10b |
| myx2 |

# Locals Lines

*Locals Lines* are lines in a defined function or operator that serve only to define local names.

A Locals Line may appear anywhere between line [0] and the first executable statement in the function or operator. Locals lines may be interspersed with blank lines and comments. A Locals Line is identified by starting with a semicolon, prefixed optionally by whitespace. It may contain a comment at the end.

A Locals Line must be of the form `;name;name;name` where name is any valid APL name or localisable system variable. The names are localised on entry to the function exactly as if they were specified as locals on line `[0]`.

### Example

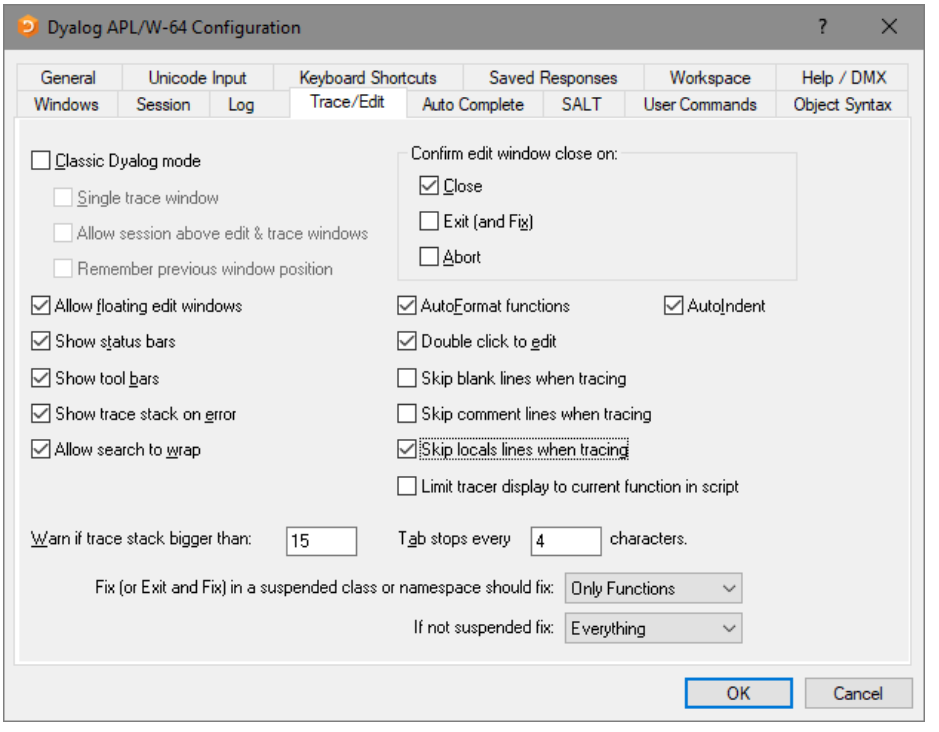
```
∇ r←foo y;a;b        ⍝ some locals
          ;c;d       ⍝ some more locals
  (a b c d)←y
  r←a+b-c×d
∇
```

The function `foo` shown above localises names `a`, `b`, `c` and `d` (the indentation on line `[1]` in this example is entirely optional)

Syntactical errors on Locals Lines are detected when the user attempts to fix the function using the Editor or `⎕FX` and will causes the operation to fail.

The local names defined in Locals Lines are syntax-coloured as locals.

You may skip Locals Lines when Tracing by checking the option labelled *Skip Locals Lines when Tracing* on the *Trace/Edit* tab of the *Options/Configure Dialog*,


Dyalog APL/W-64 Configuration
General
Unicode Input
Keyboard Shortcuts
Saved Responses
Workspace
Help / DMX
Windows
Session
Log
Trace/Edit
Auto Complete
SALT
User Commands
Object Syntax
Classic Dyalog mode
Single trace window
Allow session above edit & trace windows
Remember previous window position
Confirm edit window close on:
Close
Exit (and Fix)
Abort
Allow floating edit windows
AutoFormat functions
AutoIndent
Show status bars
Double click to edit
Show tool bars
Skip blank lines when tracing
Show trace stack on error
Skip comment lines when tracing
Allow search to wrap
Skip locals lines when tracing
Limit tracer display to current function in script
Warn if trace stack bigger than:
15
Tab stops every
4
characters.
Fix (or Exit and Fix) in a suspended class or namespace should fix:
Only Functions
If not suspended fix:
Everything
OK
Cancel

# Global Triggers

A global Trigger is a function that triggers on any assignment to a global variable in the same namespace.

This is implemented by the function declaration statement:

```
:Implements Trigger *
```

The argument to the trigger function is an instance of the internal class `TriggerArguments` which contains the following members:

| Member | Description |
|---|---|
| `Name` | The name of the global variable that is about to be changed. |
| `Indexers` | If the assignment is some form of indexed assignment, `Indexers` is an array with the same shape as the sub-array that was assigned and contains the ravel-order, ⎕IO-sensitive, indices of the changed elements. Otherwise, `Indexers` is undefined. |

**Example:**

```
     ∇ foo args
[1]    :Implements Trigger *
[2]    args.Name'has changed'
[3]    :If 2=args.⎕NC'Indexers'
[4]        'ρIndexers'(ρargs.Indexers)
[5]        'Indexers'(,args.Indexers)
[6]    :EndIf
     ∇

      vec←⍳5
 vec  has changed

      a b←10 'Pete'
 a  has changed
 b  has changed

      vec[2 4]←99
 vec  has changed
 ρIndexers  2
 Indexers  2 4
```

```
      array←2 3 4⍴⍳12
 array  has changed

      (2 1 3⍉array)←42
 array  has changed
 ⍴Indexers  2 1 3
 Indexers  1 2 3 13 14 15
```

Notes:

- like other Triggers, only the most recently fixed global trigger function will apply and be called on assignment to a global variable.
- global triggers do not apply to local names nor to semi-globals (names which are localised further up the stack).
- an assignment to a global variable will fire both its specific trigger (if defined) and the global trigger. However, the order of execution is undefined.
- do not use an argument name for your trigger function that may conflict with a global variable name in the namespace.

### Further Example

A potential use for a global trigger is to detect the unintended creation of global variables due to localisation omissions. Note however that the timing of the activation of the Trigger is unpredictable. In this example, the trigger for the assignment to `b` activates after function `hoo` has exited. When Threads are involved, timing becomes even less predictable.

```
      ∇ CatchGlobals arg
[1]     ⍝ Displays a warning when a global is assigned
[2]     :Implements Trigger *
[3]     '*** assignment to global variable: ',
         arg.Name,' from ',1↓⎕SI
      ∇
      ∇ foo
[1]     goo
      ∇
      ∇ goo
[1]     hoo
      ∇
      ∇ hoo
[1]     a←10
[2]     b←a
      ∇
       foo
*** assignment to global variable: a from  hoo  goo  foo
*** assignment to global variable: b from  goo  foo
```

# 128-Bit Decimal Numbers

On all platforms except IBM's AIX on POWER 128-bit decimal numbers are now represented using Binary Integer Decimal (BID) encoding format. Previous versions of Dyalog use Densely Packed Decimal (DPD) format.

This change has been made to speed up 128-bit floating-point calculations under all non-AIX/POWER platforms when `⎕FR` is 1287. AIX/POWER continues to use DPD since DPD is supported by the hardware; there is no performance degradation on AIX as a result of the change on all other platforms.

In 17.0 128-bit decimal numbers are automatically converted from BID to DPD format and vice versa when necessary; workspaces and component files can be shared between AIX and non-AIX versions of Dyalog APL.

Arrays containing 128-bit decimal values written to component files by 17.0 interpreters on non-AIX platforms cannot be read by earlier interpreters on any platform; the error `DOMAIN ERROR: Array is from a later version of APL` will be signalled. Arrays containing 128-bit decimal values written to component files by 17.0 interpreters on AIX/POWER can be read by earlier interpreters on all platforms (subject to other limitations which are detailed in the interoperability section of these release notes). The same is true of TCPSockets, Conga communications and arrays that are serialised and unserialised using `219⌶` and `220⌶`.

The `⎕NA` type code `D` will continue to mean DECF DPD on AIX/POWER and will mean BID on all other platforms.

# System Requirements

## Microsoft Windows

Dyalog APL Version 17.0 is supported on versions of Microsoft Windows from Windows 7 up to and including Windows 10 and Windows Server 2016.

## Microsoft .NET Interface

Dyalog APL Version 17.0 .NET Interface requires Version 4.0 or greater of the Microsoft .NET Framework. It does *not* operate with earlier versions of .NET.

For full Data Binding support (including support for the `INotifyCollectionChanged` interface[1]), and Syncfusion, Version 17.0 requires .NET Version 4.5.

The examples provided in the sub-directory `Samples/asp.net` require that IIS is installed. If IIS and ASP.NET are not present, the `asp.net` sub-directory will not be installed during the Dyalog installation.

## AIX

For AIX, Version 17.0 requires AIX 7.2 or higher, and a POWER7 chip or higher.

## Raspberry Pi

On the Raspberry Pi, Dyalog 32-bit Unicode supports Raspbian Jessie or later.

## Non-Pi Linux

For non-Pi Linux, Version 17.0 only exists as 64-bit interpreters - there are no 32-bit versions. It is built on Debian 7, and QAed on RedHat 6; it runs on all recent distributions, including Ubuntu 14.04 and openSUSE Leap 42.3. Contact Dyalog for information about other distributions.

## macOS/Mac OS X

Version 17.0 requires Mac OS X Yosemite or El Capitan or macOS Sierra or later. The target Mac must have been introduced in 2010 or later.

[1]This interface is used by Dyalog to notify a data consumer when the contents of a variable, that is data bound as a list of items, changes.

# Interoperability

## Introduction

Workspaces and component files are stored on disk in a binary format (illegible to text editors). This format differs between machine architectures and among versions of Dyalog. For example, a file component written by a PC may well have an internal format that is different from one written by a UNIX machine. Similarly, a workspace saved from Dyalog Version 17.0 will differ internally from one saved by a previous version of Dyalog APL.

It is convenient for versions of Dyalog APL running on different platforms to be able to *interoperate* by sharing workspaces and component files. From Version 11.0, component files and workspaces can generally be shared between Dyalog interpreters running on different platforms. However, this is not always possible and the following sections describe limitations in interoperability:

## Code and ⎕ORs

Code that is saved in workspaces, or embedded within ⎕ORs stored in component files, can only be read by the Dyalog version which saved them and later versions of the interpreter. In the case of workspaces, a load (or copy) into an older version would fail with the message:

```
this WS requires a later version of the interpreter.
```

Every time a ⎕OR object is read by a version later than that which created it, time may be spent in converting the internal representation into the latest form. Dyalog recommends that ⎕ORs should not be used as a mechanism for sharing code or objects between different versions of APL.

## "Ordinary" Arrays

With the exception of the Unicode restrictions described in the following paragraphs, Dyalog APL provides interoperability for arrays that only contain (nested) character and numeric data. Such arrays can be stored in component files - or transmitted using `TCPSocket` objects and Conga connections, and shared between all versions and across all platforms.

Full cross-platform interoperability of component files is only available for large-span component files.

## Null Items (⎕NULL) and Compressed Components

⎕NULLs and components from compressed component files, which were created in Version 17.0, can be brought into Versions 14.1, 15.0 and 16.0 provided that the interpreters have been patched to revision 33394 or higher. Attempts to bring ⎕NULL or compressed component into earlier versions of Dyalog APL or lower revisions of the aforementioned versions will fail with:

```
DOMAIN ERROR: Array is from a later version of APL.
```

## Object Representations (⎕OR)

An attempt to ⎕FREAD a component containing a ⎕OR that was created by a later version of Dyalog APL will generate `DOMAIN ERROR: Array is from a later version of APL`. This also applies to APL objects passed via Conga or TCPSockets, or objects that have been serialised using `220⌶`.

## 32 vs. 64-bit Component Files

It is no longer possible to *create* or write to small-span (32-bit) files; however it is still currently possible to *read* from small span files. Setting the second item of the right argument of ⎕FCREATE to anything other than 64 will generate a `DOMAIN ERROR`.

Note that *small-span* (32-bit-addressing) component files cannot contain Unicode data. Unicode editions of Dyalog APL can only write character data which would be readable by a Classic edition (consisting of elements of ⎕AV).

## External Variables

External variables are subject to the same restrictions as small-span component files regarding Unicode data. External variables are unlikely to be developed further; Dyalog recommends that applications which use them should switch to using mapped files or traditional component files. Please contact Dyalog if you need further advice on this topic.

## 32 vs. 64-bit Interpreters

There is complete interoperability between 32- and 64-bit interpreters, except that 32-bit interpreters are unable to work with arrays or workspaces greater than 2GB in size.

Note however that under Windows a 32-bit version of Dyalog APL may only access 32-bit DLLs, and a 64-bit version of Dyalog APL may only access 64-bit DLLs. This is a Windows restriction.

## Unicode vs. Classic Editions

Two editions are available on some platforms. Unicode editions work with the entire Unicode character set. Classic editions (a term which includes versions prior to 12.0) are limited to the 256 characters defined in the atomic vector, `⎕AV`).

Component files have a Unicode property. When this is enabled, all characters will be written as Unicode data to the file. The Unicode property is always off for small-span (32-bit addressing) files, as these cannot contain Unicode data. For large-span (64-bit addressing) component files, the Unicode property is set *on* by Unicode Editions and *off* by Classic Editions, by default. The Unicode property can subsequently be toggled on and off using `⎕FPROPS`.

When a Unicode edition writes to a component file that cannot contain Unicode data, character data is mapped using `⎕AVU`; it can therefore be read without problems by Classic editions.

A `TRANSLATION ERROR` will occur if a Unicode edition writes to a non-Unicode component file (that is either a 32-bit file, or a 64-bit file when the Unicode property is currently off) if the data being written contains characters that are not in `⎕AVU`.

Likewise, a Classic edition will issue a `TRANSLATION ERROR` if it attempts to read a component containing Unicode data that is not in `⎕AVU` from a component file.

A `TRANSLATION ERROR` will also be issued when a Classic edition attempts to `)LOAD` or `)COPY` a workspace containing Unicode data that cannot be mapped to `⎕AV` using the `⎕AVU` in the recipient workspace.

`TCPSocket` objects have an `APL` property that corresponds to the Unicode property of a file, if this is set to `Classic` (the default) the data in the socket will be restricted to `⎕AV`, if Unicode it will contain Unicode character data. As a result, `TRANSLATION ERROR`s can occur on transmission or reception in the same way as when updating or reading a file component.

The symbols ⊆, ⍳, ⍤, ⌸, ⌸ and ⌺ used for the Nest (Interval Index) and Where (Partition) functions, the Rank, Variant, Key and Stencil operators respectively are available only in the Unicode edition. In the Classic edition, these symbols are replaced by `⎕U2286`, `⎕U2378`, `⎕U2364`, `⎕U2360`, `⎕U2338` and `⎕U233A` respectively. In both Unicode and Classic editions Variant may be represented by `⎕OPT`.

## Very large array components

An attempt to read a component greater than 2GB in 32-bit interpreters will result in a `WS FULL`.

## TCPSockets and Conga

TCPSockets and Conga can be used to communicate between differing versions of Dyalog APL and are subject to similar limitations to those described above for component files.

## Auxiliary Processors

A Dyalog APL process is restricted to starting an AP of exactly the same architecture from the same operating system. In other words, the AP must share the same word-width and byte-ordering as its interpreter process.

## Session Files

Session (.dse) files can only be used on the platform on which they were created and saved. Under Microsoft Windows, Session files may only be used by the architecture (32-bit-or 64-bit) of the Version of Dyalog that saved them.

# Announcements

## Tools and Interfaces

In addition to the software provided as part of the Dyalog installation package, there are a growing number of tools and interfaces available for download from GitHub. For details, see https://www.dyalog.com/tools/tools-and-code-libraries.htm.

## Withdrawal of Support for Version 14.1

The supported Versions of Dyalog APL are now Version 17.0, 16.0 and 15.0. Version 14.1 and earlier versions are no longer supported.

## Planned Operating System Requirements for the next version

Dyalog Ltd expects that the next version of Dyalog will require the following minimum platform requirements:

| Operating System | Version |
|---|---|
| Microsoft Windows | Windows 7 or Server 2008 R2 |
| AIX | 7.2 on POWER 7 |
| Linux | RedHat/Centos 7 or equivalent |
| OS X | OS X Yosemite 10.10.x |
| Raspberry Pi | Raspbian Jessie or above |

Further updates to this information will appear on the Forums as and when available.

## Planned Hardware Requirements for next version

The same as Dyalog Version 17.0.

## Deprecation of small-span component files

As announced in the Version 16.0 Release Notes, the facility to write to small-span component files has been removed. You may continue to tie and read small-span component files.

However, if an attempt is made to execute any system function that would write to or amend the file in any way, the operation will fail as follows:

```
FILE ACCESS ERROR: Small-span files are read-only; ⎕FCOPY
can create a writable large-span copy
```

The system functions that are affected are: ⎕FAPPEND, ⎕FREPLACE, ⎕FDROP, dyadic ⎕FPROPS, ⎕FRENAME, ⎕FRESIZE, ⎕FSTAC. In addition, ⎕FCHK cannot write to small-span files, so cannot perform Repair or Rebuild operations.

### CHM Help

The CHM help files currently shipped with Dyalog will be removed in a forthcoming release.

# Bug Fixes

A number of bug fixes implemented in Version 17.0 may change the way that existing code operates and are therefore documented in this section.

### Incorrect Identity Element (9849)

When ⎕FR is 645, the expressions ⌈/2*1E400 and ⌈/2*1E400 now correctly generate DOMAIN ERROR. Previously these expressions returned a (wrong) result.

### Correction to System Function ⎕R

In earlier versions in Line mode (the default) ⎕R applied to an argument which ends with multiple empty lines would return one fewer element than it was passed. In Version 17.0 all the empty elements are preserved.

#### Examples

```
      ≢('a' ⎕r'bb') 'Andy' '' 'Pete'
3

      ≢('a' ⎕r'bb') 'Andy' '' 'Pete' ''
4

      ≢('a' ⎕r'bb') 'Andy' '' 'Pete'  '' ''
5
```

In previous versions, the results were 3, 3, 4 respectively.

### APL_COMPLEX_AS_V12 parameter

If **APL_COMPLEX_AS_V12** is set to 1, objects containing complex numbers cannot be transferred to or from component files, TCP/IP (CONGA), or auxiliary processors and may not be used as an argument to Serialise/Deserialise Array (220⌶). Instead, a DOMAIN ERROR will be issued.

# Chapter 2: Miscellaneous

## Command-Line Options

The command-line to run the Dyalog program or a bound Dyalog executable may now contain options which control how parameters are handled by the Dyalog component.

### -apl Option

The **-apl** option is designed to be used for bound executables. It causes the following parameter to be passed to the Dyalog engine for processing. If multiple parameters need to be passed to the Dyalog engine each must be preceded with **-apl**. In versions prior to Version 17.0 all command-line parameters were ignored by the Dyalog DLL.

### -cef Option

The **-cef** option is designed to be used with the Development EXE and the Run-Time EXE. It causes the following parameter to be ignored by the Dyalog executable. In versions prior to Version 17.0 **all** command-line parameters are passed to the Dyalog executable, whether or not they are relevant to it.

This option is intended to isolate parameters intended for the built-in Chromium Embedded Framework (CEF) which provides the HTMLRenderer object. See HTML Renderer Documentation.

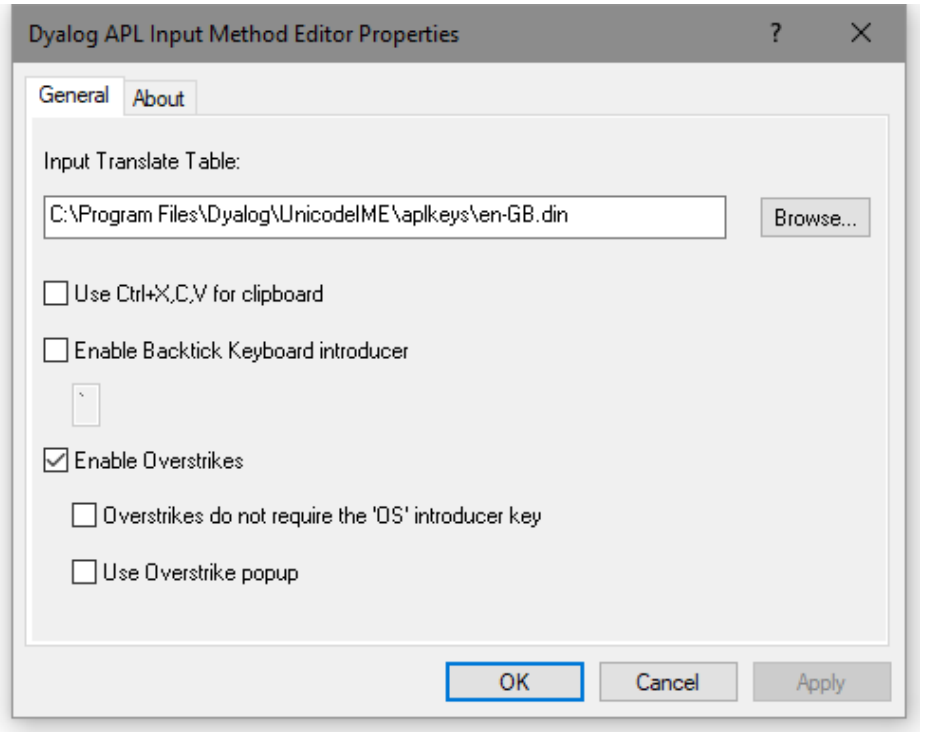
.

# IDE Enhancements

## Backtick Keyboard

The *Backtick* keyboard provided by the RIDE may now be used natively. For information on using this keyboard interface, see http://docs.dyalog.com/17.0/RIDE User Guide.pdf *Section 7.4*.

By default, the *Backtick* keyboard is disabled . To enable it, select *Options/Configure* from the Session menu, click the *Unicode Input* tab, then click *Configure Layout ...*.



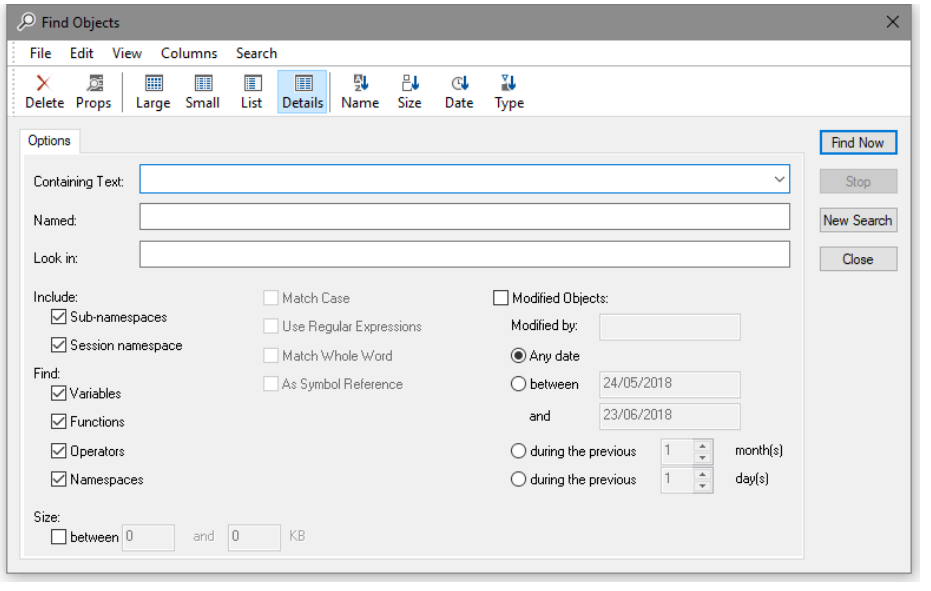

Now check the option button labelled *Enable Backtick Keyboard introducer*. You may replace the backtick character (`) with an alternative character to act as the introducer for APL glyphs, but take care to choose a character that is otherwise unused.

Finally, click *OK*. Despite the general warning message that appears, this change will take effect and you may use the *Backtick* keyboard layout immediately.

# Find Objects Tool

The *Find Objects* tool is a modeless dialog box that may be toggled on and off by the system action `[WSSearch]`. In a default Session, this action is attached to a MenuItem in the Tools menu and a Button on the session toolbar.

The Find Objects tool allows you to search the active workspace for objects that satisfy various criteria.




## Name

The *Named* field is used to search for objects with a particular name and is case-insensitive.

## Containing Text

The *Containing Text* field is used to search for objects that contain a particular text string. The string search is controlled by the fields *Match Case*, *Use Regular Expressions*, *Match Whole Word* and *As Symbol Reference*.

*Match Case* specifies whether or not the string search (for name and/or contents) is case sensitive.

*Use Regular Expressions* specifies whether or not regular expressions are applicable. For example, if you enter `FOO*` into the field labelled *Containing Text* and check this box, the system will find objects that contain any text string starting with the 3 characters `FOO`.

If this box is not checked, the system will find objects that contain the 4 characters `FOO*`.

Text searches are performed using PCRE. If the *Use Regular Expressions* box is checked, the full range of regular expressions provided by PCRE are available for use. See *Language Reference Guide: Appendix A*.

*Match Whole Word* specifies whether or not the search is restricted to entire words.

*As Symbol Reference* specifies whether or not the search is restricted to APL symbols. If so, matching text in comments and other strings is ignored.

## Object Criteria

Four check boxes are provided for you to specify the types of objects you wish to locate. For example, if you clear *Variables*, *Operators* and *Namespaces*, the system will only search for functions.

To make the search dependent upon modification, you must check the *Modified Objects* check box.

To locate objects modified by a particular user, enter the user name in the field labelled *Modified by*. Otherwise leave this blank.

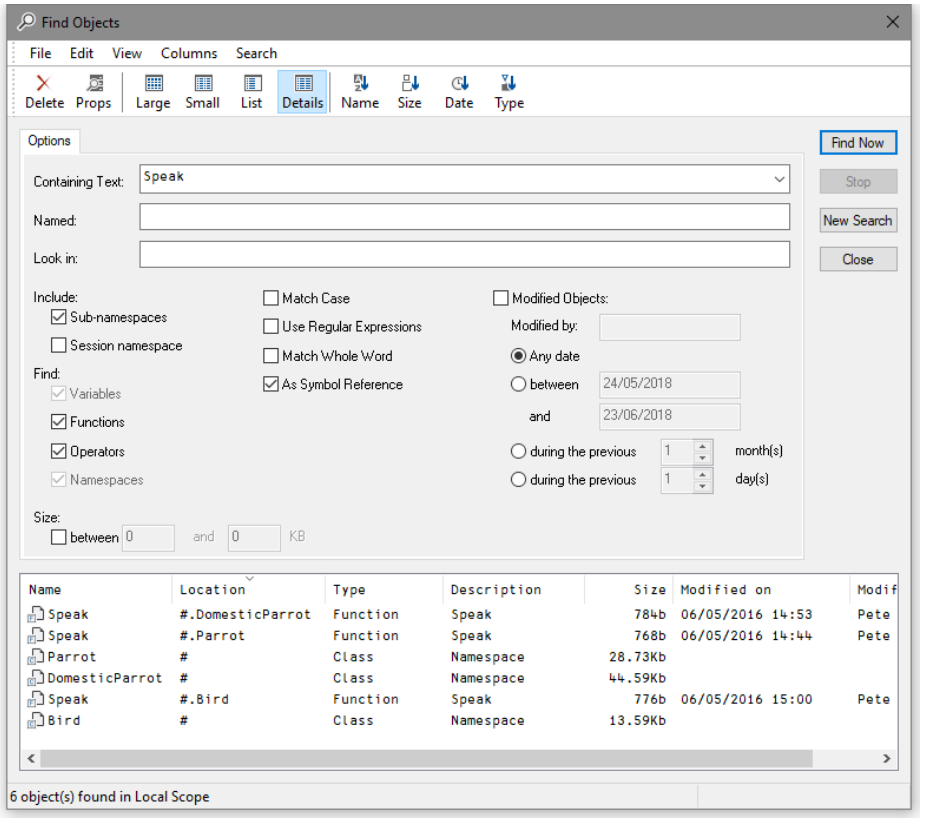
To find objects which have been modified at a certain time or within a specified period of time, check the appropriate radio button and enter the appropriate dates or time spans.

If you wish to restrict the search to find only objects whose size is within a given range, check the box labelled *Size* is between and enter values into the fields provided.

## Location Criteria

You can restrict the search to a particular namespace by typing its name into the field labelled *Look in*. You can further restrict the search by clearing the *Include sub-namespaces* and *Include Session namespace* check boxes. Clearing the former restricts the search to the root namespace or to the namespace that you have specified in *Look in*, and does not search within any sub-namespaces contained therein. Clearing the latter causes the system to ignore `⎕SE` in its search.

When you press the *Find Now* button, the system searches for objects that satisfy all of the criteria that you have specified on all 3 pages of the dialog box and displays them in a ListView. The example below illustrates the result of searching the workspace for all objects containing references to the symbol `Speak`.




You may change the way in which the objects are displayed in the ListView using the *View* menu or the tool buttons, in the same manner as for objects displayed in the Workspace Explorer. You may also edit, delete and rename objects in the same way. Furthermore, objects can be copied or moved by dragging from the ListView in the Search tool to the TreeView in the Explorer.

If you wish to specify a completely new set of criteria, press the *New Search* button. This will reset all of the various controls of the dialog box to their default values.

# New Tool Buttons

## Editor Toolbar

Two new buttons have been added to the Editor toolbar.

| Button | Description |
|---|---|
|  Toggle tree view | Toggles the treeview on/off. |
|  Previous Location | Certain operations (such as selecting an item in the treeview) reposition the caret in the Editor window. This button moves the caret back to its previous location. |

## Session Tools

A new button has been added to the Session toolbar.

| | |
|---|---|
|  Boxing On/Off | Executes the user command `]boxing` to toggle boxing on/off. |

# Other Changes

- The left argument to ⎕CY may now be a vector of character vectors.
- 2000⌶ accepts a new argument 19 which reports the number of calls to ⎕WA and 2002⌶.
- For a VALUE ERROR, the error message and ⎕DMX now report the undefined name.
- For a LIMIT ERROR caused by an insufficiency of GUI resources, ⎕DMX now identifies the cause.
- ⎕FX has been extended to allow ∇s around functions.
- If a file contains a single function surrounded by ∇s the Editor treats it as a single function. Previously this was not the case.
- Version 17.0 supports late binding of Base Classes and included namespaces. This means that the Editor will fix a Class even though its Base Class is not present in the workspace. The Base Class is only required to be present when the Class is instantiated. The same is true for included namespaces (namespaces declared using the :Include statement).
- If the system cannot infer the type of a file from its content, it treats it as a function.
- ⎕STACK has changed so that when there is an inner dfn on the stack, instead of reporting the ⎕OR of that inner dfn (which might be anonymous) it reports the ⎕OR of the capsule (which is always named). A capsule is defined as the outermost dfn in a set of nested dfns. This makes it more consistent with ⎕LC and ⎕XSI and other stack-related functions, which always work with capsules and capsule-relative line numbers.
- The left argument to ⎕ED is now a character scalar that specifies the type for all the names in the right argument Y, or a character vector whose elements specify the types of each of the corresponding names in Y.
- In earlier versions of Dyalog attempting to ⎕FIX a malformed scripted object would result in the generic message:
  `DOMAIN ERROR: There were errors processing the script`. In Version 17.0 the error reported is the same as the first error which appears in the status window; for example:
  `DOMAIN ERROR: invalid base class or interface declaration`.
  A side effect of this is that ⎕DMX.ENX will be different, and the text and error numbers that appear in the status window may differ compared with earlier versions.
- The native-file related functions now include the file or directory name in the error message.
- The position of the caret in an error report is more informative than in previous versions and now identifies more precisely the operation that failed.

# Chapter 3:

# Language Reference Changes

## Language Changes

The following table summarises the main changes to language features in Version 17.0.

| Function | Description | Change |
|---|---|---|
| □CSV | Comma Separated Values | Supports user-defined character set, a new column type (numeric/empty) and 3 new variant options to handle metacharacters. |
| □MKDIR | Make Directory | Supports multiple files |
| □NCOPY | Native File Copy | New system function |
| □NDELETE | Native File Delete | Supports multiple files, wildcards and recursion |
| □NEXISTS | Native File Exists | Supports multiple files and wildcards |
| □NGET | Read Text File | Supports user-defined character set |
| □NINFO | Native File Information | Supports multiple files, recursion and additional file properties |
| □NPARTS | Native File Parts | Supports multiple files |
| □NMOVE | Native File Move | New system function |
| □NPUT | Write Text File | Supports user-defined character set and append operation |
| □R and □S | Search/Replace | Supports user-defined character set |

# Shy Results for System Functions

To facilitate the use of system functions in dfns, it has been decided to extend those that (in previous versions) do not return a result, to return a shy result. This decision affects the following system functions which now return shy results.

| System Function | Shy result |
|---|---|
| `{X} ⎕ARBOUT Y` | `0` |
| `X ⎕CMD Y` | Process ID of created process |
| `{X} ⎕CY Y` | `0⍴''` |
| `{X} ⎕LOCK Y` | New *lock level* of `Y` |
| `X ⎕SH Y` | Process ID of created process |
| `⎕SHADOW Y` | Boolean vector |

Due to the minor nature of the effect on the Version 17.0 documentation, the full descriptions of these system functions are not included in these Release Notes.

## Grade Down (Monadic) R←⍒Y

Y may be any array of rank greater than 0 but may not contain namespaces. R is an integer vector being the permutation of ⍳1↑⍴Y that places the sub-arrays along the first axis in descending order. For the rules for comparing items of Y with one another, see *Grade Up (Monadic)* on page 37.

⎕IO is an implicit argument of Grade Down.

### Examples

```
      ⍒22.5 1 15 3 ¯4
1 3 4 2 5

      M
2 3 5
1 4 7

2 3 4
5 2 4

2 3 5
1 2 6
      ⍒M
1 3 2
```

**Note that character arrays sort differently in the Unicode and Classic Editions.**

```
      M
Goldilocks
porridge
Porridge
3 bears
```

| Unicode Edition | Classic Edition |
|---|---|
| `      ⍒M`<br>`2 3 1 4` | `      ⍒M`<br>`3 1 4 2` |
| `      M[⍒M;]`<br>`porridge`<br>`Porridge`<br>`Goldilocks`<br>`3 bears` | `      M[⍒M;]`<br>`Porridge`<br>`Goldilocks`<br>`3 bears`<br>`porridge` |

```
      ⍴pb
6 3
      pb
```

| | | |
|---|---|---|
| Rivers | Jason | 554 |
| Daintree | John | 532 |
| Rivers | Jason | 543 |
| Foad | Jay | 558 |
| Scholes | John | 547 |
| Scholes | John | 535 |

```
      ⍒pb
5 6 1 3 4 2
```

# Grade Up (Monadic) R←⍋Y

Y may be any array of rank greater than 0 but may not contain namespaces. R is an integer vector being the permutation of ⍳1↑⍴Y that places the sub-arrays along the first axis in ascending order. The rules for comparing items of Y with one another are as follows:

### Rules for comparing simple scalars

- Numeric comparisons are exact, as if ⎕CT←⎕DCT←0 and ⎕FR←1287
- Two real numbers are compared numerically, thus 1.2 precedes 3.
- In the Unicode Edition two characters are compared numerically according to their position in the Unicode table. Thus 'a' (⎕UCS 97) precedes 'b' (⎕UCS 98). In the Classic Edition characters are compared according to their index in ⎕AV.
- Complex numbers are ordered by first comparing their real parts. If these are equal, the order is determined by comparing their imaginary parts. Thus 1J¯2 precedes 1 which precedes 1J2.
- ⎕NULL (which represents a null item obtained from an external source) precedes all numbers, and all numbers precede all characters. Thus ⎕NULL precedes 100, and 100 precedes 'A'.

### Rules for comparing non-scalar arrays

- Arrays are compared item by item in ravel order.
- For arrays of equal shape, the order is determined by the first pair of items which differ, thus (1949 4 29) precedes (1949 4 30). Similarly ('April' 29) precedes ('April' 30).
- Arrays with the same rank but different shape are ordered as if the shorter array were padded with items that precede all other types of item (negative infinity) including ⎕NULL. Thus 'car' precedes 'carpet' and (1949 4) precedes (1949 4 30). An alternative model is to say that shorter arrays precede longer ones that begin the same way. For character vectors this is described as Lexicographical ordering, which is the order that words appear in a dictionary.
- Arrays with differing rank are ordered by first extending the shape of the lower-ranked array with 1s at the beginning, and then comparing the resultant equal-rank arrays as described above. So, to compare a vector (rank 1) with a matrix (rank 2), the vector is reshaped into a 1-row matrix.

- Empty arrays are compared first by type alone, so an empty numeric array precedes an empty character array, regardless of rank or shape.
  Thus `((0 3 2)⍴0)` precedes `''`. If the empty arrays are of the same type, they are sorted in order of their shape vector, working right to left.
  So `((0 5 2)⍴99)` precedes `((0 3 4)⍴0)` and `((0 3 4)⍴'')` precedes `((1 0 5 4)⍴'')`.

`⎕IO` is an implicit argument of Grade Up

### Examples

```
      ⍋22.5 1 15 3 ¯4
5 2 4 3 1

      M
2 3 5
1 4 7

2 3 4
5 2 4

2 3 5
1 2 6
      ⍋M
2 3 1
```

**Note that character arrays sort differently in the Unicode and Classic Editions.**

```
      M
Goldilocks
porridge
Porridge
3 bears
```

| Unicode Edition | Classic Edition |
|---|---|
| `      ⍋M`<br>`4 1 3 2` | `      ⍋M`<br>`2 4 1 3` |
| `      M[⍋M;]`<br>`3 bears`<br>`Goldilocks`<br>`Porridge`<br>`porridge` | `      M[⍋M;]`<br>`porridge`<br>`3 bears`<br>`Goldilocks`<br>`Porridge` |

```
      ⍴pb
6 3
      pb
```

| | | |
|---|---|---|
| Rivers | Jason | 554 |
| Daintree | John | 532 |
| Rivers | Jason | 543 |
| Foad | Jay | 558 |
| Scholes | John | 547 |
| Scholes | John | 535 |

```
      ⍋pb
2 4 3 1 6 5
```

# Unique R←∪Y

Y may be any array. R is a vector of the unique major cells of Y (the unique items of a vector, the unique rows of a matrix and so forth), in the order in which they first appear in Y. For further information, see *Programming Reference Guide: Cells and Subarrays*.

⎕CT is an implicit argument of Unique.

## Examples

```
      ∪ 22 10 22 22 21 10 5 10
22 10 21 5

      ∪ v←'CAT' 'DOG' 'CAT' 'DUCK' 'DOG' 'DUCK'
```

| CAT | DOG | DUCK |
|---|---|---|

```
      ⊢mat←↑v
CAT
DOG
CAT
DUCK
DOG
DUCK
      ∪mat
CAT
DOG
DUCK

      a←3 4 5⍴⍳20
      a
 1  2  3  4  5
 6  7  8  9 10
11 12 13 14 15
16 17 18 19 20

 1  2  3  4  5
 6  7  8  9 10
11 12 13 14 15
16 17 18 19 20

 1  2  3  4  5
 6  7  8  9 10
11 12 13 14 15
16 17 18 19 20
      ∪a
 1  2  3  4  5
 6  7  8  9 10
11 12 13 14 15
16 17 18 19 20
```

# Comma Separated Values {R}←{X} ⎕CSV Y

This function imports and exports Comma Separated Value (CSV) data.

Monadic ⎕CSV imports data from a CSV file or converts data from CSV format to an internal format. Dyadic ⎕CSV exports data to a CSV file or converts data from internal format to a CSV format.

## Internal Format

Arrays that result from importing CSV data or arrays that are suitable for exporting as CSV data are represented by 3 possible structures:

- A table (a matrix whose elements are character vectors or scalars, or numbers).
- A vector, each of whose items contain field (column) values. Character field values are character matrices; numeric field values are numeric vectors.
- A vector, each of whose items contain field (column) values. Character field values are vectors of character vectors; numeric field values are numeric vectors.

Note that when importing CSV data, all fields are assumed to be character fields unless otherwise specified (see *Column Types* below). A field that contains only "numbers" will not be converted to numeric data unless specified as being numeric.

## MetaCharacters

Some characters in a CSV file are metacharacters which define the structure of the data; for example, the field separator character between fields. Characters which are not metacharacters are literal characters. The variant options **QuoteChar**, **EscapeChar** and **DoubleQuote** make it possible to interpret metacharacters as literal characters and thus permit fields to contain field separator characters, leading and trailing spaces, and line-endings.

Fixed-width fields do not require these options and they are ignored if fixed-width fields are being processed.

## Monadic ⎕CSV

```
R←⎕CSV Y
```

`Y` is an array that specifies just the source of the CSV data (see below) or a 1,2,3 or 4-element vector containing:

| | |
|---|---|
| `[1]` | Source of CSV Data |
| `[2]` | Description of the CSV data |
| `[3]` | Column Types |
| `[4]` | Header Row Indicator |

*Source* may be one of:

- a character vector or scalar containing a file name
- a native tie number
- a character vector or scalar containing CSV data with embedded newline characters. To avoid this source being interpreted as a file name, `Y[2]` must be specified as `'S'`.
- a vector of character vectors and/or scalars containing CSV data with implicit newlines after each character vector or scalar

*Description*

If `Y[1]` is a file name or tie number *Description* may be one of:

- a character vector specifying the file encoding such as `'UTF-8'`.
- a 256-element numeric vector that maps each possible byte value (0-255) to a Unicode code point (1st element = Unicode code point corresponding to byte value 0, and so on). ¯1 indicates that the corresponding byte value is not mapped to any character. Apart from ¯1, no value may appear in the table more than once.

If omitted or empty, the file encoding is deduced (see below).

If `Y[1]` is a character array containing CSV data *Description* is a character scalar `'S'` (simple) or `'N'` (nested). The default is `'N'`

*Column Types*

This is a scalar numeric code or vector of numeric codes that specifies the field types from the list below. If *Column Types* is zilde or omitted, the default is 1 (all fields are character).

| | |
|---|---|
| 0 | The field is ignored. |
| 1 | The field contains character data. |
| 2 | The field is to be interpreted as being numeric. Empty cells and cells which cannot be converted to numeric values are not tolerated and cause an error to be signalled. |
| 3 | The field is to be interpreted as being numeric but invalid numeric vales are tolerated. Empty fields and fields which cannot be converted to numeric values are replaced with the Fill variant option (default 0). |
| 4 | The field is to be interpreted numeric data but invalid numeric data is tolerated. Empty fields and fields which cannot be converted to numeric values are returned instead as character data; this type is disallowed when variant option Invert is set to 1. |
| 5 | The field is to be interpreted as being numeric but empty fields are tolerated and are replaced with the Fill variant option (default 0). Non-empty cells which cannot be converted to numeric values are not tolerated and cause an error to be signalled. |

Note that if *Column Types* is specified by a scalar 4, all numeric data in all fields will be converted to numbers.

*Header Row Indicator*

This is a Boolean value (default 0) to specify whether or not the first record in a CSV file is a list of column labels. If *Header Row Indicator* is 1, the first record (the *header row*) is treated differently from other records. It is assumed to contain character data (labels) regardless of `Y[3]` and is returned separately in the result.

## Variant options

Monadic ⎕CSV may be applied using the Variant operator with the following options. The Principal option is **Invert**.

| Name | Meaning | Default |
|---|---|---|
| **Invert** | 0, 1 or 2 (see below) | 0 |
| **Separator** | The field separator, any single character. If Widths is other than ⍬, Separator is ignored. | ',' |
| **Widths** | A vector of numeric values describing the width (in characters) of the corresponding columns in the CSV source, or ⍬ for variable width delimited fields | ⍬ |
| **Decimal** | The decimal mark in numeric fields - one of '.' or ',' | '.' |
| **Thousands** | The thousands separator in numeric fields, which may be specified as an empty character vector (meaning no separator is defined) or a character scalar | '' |
| **Trim** | A Boolean specifying whether undelimited/unescaped whitespace is trimmed at the beginning and end of fields | 1 |
| **Ragged** | A Boolean specifying whether records with varying numbers of fields are allowed; see notes below | 0 |
| **Fill** | The numeric value substituted for invalid numeric data in columns of type 3 | 0 |
| **Records** | The maximum number of records to process or 0 for no limit. This applies only to a file specified by a tie number. | 0 |
| **QuoteChar** | The field quote character (delimiter), which may be specified as an empty character vector (meaning none is defined) or a character scalar | " |
| **EscapeChar** | The escape character, which may be specified as an empty character vector (meaning none is defined) or a character scalar | '' |
| **DoubleQuote** | A Boolean which indicates whether (1) or not (0) a quote character within a quoted field is represented by two consecutive quote characters | 1 |

The **Separator**, **QuoteChar** and **EscapeChar** characters, when defined, must be different.

Other options defined for export are also accepted but ignored.

## Invert Option

This option specifies how the CSV data should be returned as follows:

| | |
|---|---|
| `0` | A table (a matrix whose elements are character vectors or scalars or numbers). |
| `1` | A vector, each of whose items contain field (column) values. Character field values are character matrices; numeric field values are numeric vectors. |
| `2` | A vector, each of whose items contain field (column) values. Character field values are vectors of character vectors; numeric field values are numeric vectors. |

## QuoteChar, EscapeChar and DoubleQuote Options

If **EscapeChar** is set then any character may be prefixed by the escape character. The escape character is typically defined as `'\'`. The escape character immediately followed by the character `c` is the literal character `c` even if `c` alone would have been a metacharacter.

If **QuoteChar** is set then fields may be delimited by the specified quote character. Within quoted fields all characters except the quote character, and the escape character if defined, are literal characters.

If **DoubleQuote** is set to 1 then two consecutive quote characters within a quoted field are interpreted as the single literal quote character.

## Result

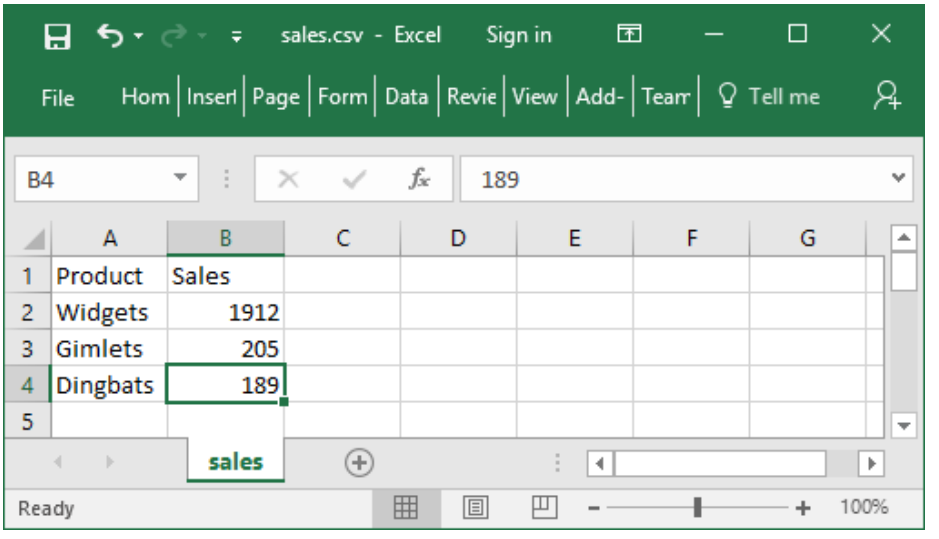
The result `R` contains the imported data.

If `Y[4]` does not specify that the data contains a header then `R` contains the entire data in the form specified by the Invert variant option.

If `Y[4]` does specify that the data contains a header then `R` is a 2-element vector where:

- `R[1]` is the imported data excluding the header.
- `R[2]` is a vector of character vectors containing the header record.

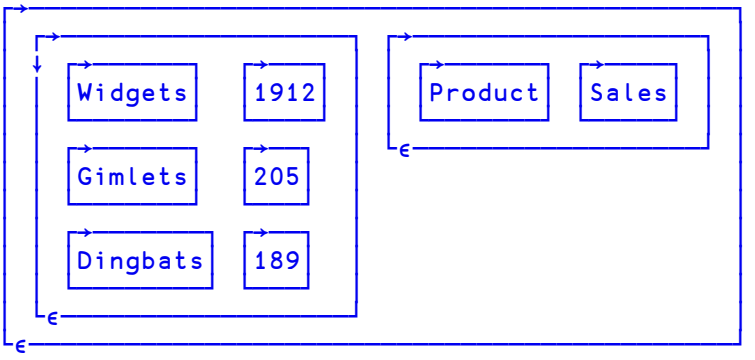
### Examples



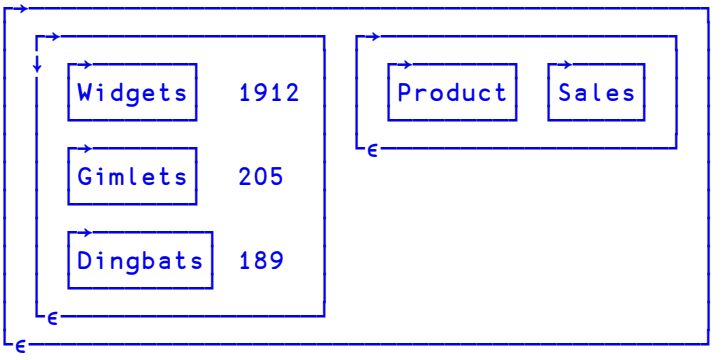

```
      ⎕NGET CSVFile←'c:\Dyalog16.0\sales.csv'
┌──────────────────────────────────────────┐
│Product,Sales                             │
│             Widgets,1912                 │
│                        Gimlets,205       │
│                                  Dingbats,189
└──────────────────────────────────────────┘

      ⎕CSV CSVFile
┌──────────────────┐
↓┌─────┐ ┌─────┐   │
 │Product│ │Sales│ │
 └─────┘ └─────┘   │
 ┌─────┐ ┌─────┐   │
 │Widgets│ │1912│ │
 └─────┘ └─────┘   │
 ┌─────┐ ┌─────┐   │
 │Gimlets│ │205│  │
 └─────┘ └─────┘   │
 ┌─────┐ ┌─────┐   │
 │Dingbats│ │189│ │
 └─────┘ └─────┘   │
└──────────────────┘
```

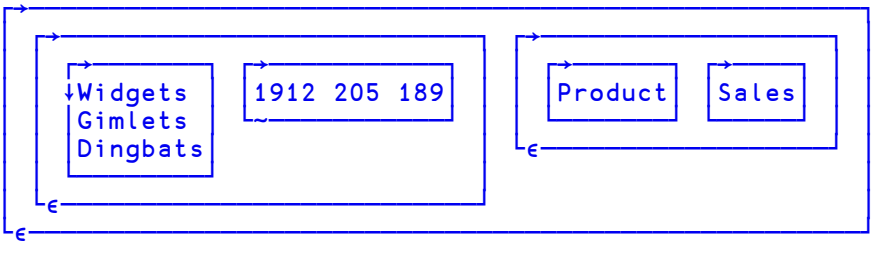
```
      □CSV CSVFile'' ⍬ 1 ⍝ Header row
```




```
      □CSV CSVFile''(1 2)1 ⍝ Fields are Char, Num
```




```
      (□CSV⍠'Invert' 1)CSVFile'' (1 2) 1    ⍝ Invert 1
```




```
      ⍝(□CSV⍠'Invert' 2)CSVFile'' (1 2) 1    ⍝ Invert 2
```

### Notes

- When `Y` specifies just the source of the CSV data, it does not need to be enclosed or ravelled to create a 1-element vector.
- `Y[2]`, the description of the source, distinguishes an otherwise ambiguous character vector source (which could contain either CSV data or a file name). The other source forms are unambiguous but the description, when given, must still match the given source type.
- Tab-separated fields may be imported by specifying `'Separator' (⎕UCS 9)`.
- Fields containing embedded new lines are supported (they must, of course, appear in quotes or be prefixed by the escape character). On import, line endings are always converted to a single line feed character.
- If **Ragged** is not set then all records must have the same number of fields (character delimited format) or same number of characters (fixed width field format).
- If **Ragged** is set:
  - The expected number of columns must be specified using the **Widths** variant option and/or the column types in `Y[3]`.
  - In character delimited format, all processed records are implicitly extended or truncated as required so that they contain the expected number of fields; implicitly added fields will be empty.
  - In fixed width format, all processed records are implicitly extended with spaces or truncated as required so that they contain as many characters as are specified in the **Widths** option declaration.

### File handling

Data may be read from a named file or a tied native file. A tied native file may be read in sections by repeatedly invoking ⎕CSV for a specified maximum number of records (specified by the **Records** variant) until no more data is read.

In all cases the files must contain text using one of the supported encodings. The method used to determine the file encoding is as follows:

- If a Byte Order Mark (BOM) is encountered at the start of the file, it is used regardless of Y[2] (if specified). Note, however, that the BOM can only be encountered if the file is read from the start - specifically, if a native file is read in sections, any BOM present will only be encountered when the first section is read.
- Otherwise, the file will be read and decoded according to the file encoding in Y[2] if specified.
- Otherwise:
  - Native files will be decoded as if 'UTF-8' had been specified.
  - Files specified by name will be examined and the likely file encoding will be deduced using the same heuristics performed by ⎕NGET.

### Note also:

- Native files are read from the current file position. On successful completion, the file position will be at the first unprocessed character (end of file if the **Records** variant option is not specified). If an error is signalled the file position is undefined.
- The result does not report the file encoding or line ending type as it does with ⎕NGET. If this information is required then it must be obtained by other means.

## Dyadic ⎕CSV

```
{R}←X ⎕CSV Y
```

The left argument `X` is either:

- a matrix or a vector of vectors/matrices containing the data to be converted to CSV format.
- or a 2-element vector containing a matrix or vector of vectors/matrices containing the data to be converted to CSV format, and a vector of character vectors containing the header record.

`Y` is a 1 or 2-element vector containing:

| | |
|---|---|
| `[1]` | Destination of CSV Data (see below) |
| `[2]` | Description of the CSV data (see below) |

*Destination* - may be one of:

- a character vector or scalar containing a file name
- a native tie number
- an empty character vector, indicating that the CSV data is to be returned in the result `R`

*Description*

If `Y[1]` is a file name or tie number, *Description* may be:

- a character vector specifying the file encoding such as `'UTF-8'`.
- a 256-element numeric vector that maps each possible byte value (0-255) to a Unicode code point (1st element = Unicode code point corresponding to byte value 0, and so on). ¯1 indicates that the corresponding byte value is not mapped to any character. Apart from ¯1, no value may appear in the table more than once.

If `Y[1]` is empty, *Description* may be a character scalar `'S'` (simple) or `'N'` (nested). If omitted, the default is `'S'`

## Variant options

Dyadic ⎕CSV may be applied using the Variant operator with the following options.

| Name | Meaning | Default |
|---|---|---|
| **IfExists** | a character vector `'Error'` or `'Replace'` which specifies, when creating a named file which already exists, whether to overwrite it (`'Replace'`) or signal an error (`'Error'`) | `'Error'` |
| **Separator** | the field separator, any single character. If Widths is other than `0`, Separator is ignored. | `','` |
| **Widths** | a vector of numeric values describing the width (in characters) of the corresponding columns in the CSV source, or `0` for variable width delimited fields | `0` |
| **Decimal** | the decimal mark in numeric fields - one of `'.'` or `','` | `'.'` |
| **Thousands** | the thousands separator in numeric fields, which may be specified as an empty character vector (meaning no separator is defined) or a character scalar | `''` |
| **Trim** | a Boolean specifying whether whitespace is trimmed at the beginning and end of character fields | `1` |
| **LineEnding** | the line ending sequence | (13 10) on Windows; 10 on other platforms |
| **QuoteChar** | The field quote character (delimiter), which may be specified as an empty character vector (meaning none is defined) or a character scalar | `"` |
| **EscapeChar** | The escape character, which may be specified as an empty character vector (meaning none is defined) or a character scalar | `''` |
| **DoubleQuote** | A Boolean which indicates whether (1) or not (0) a quote character within a quoted field is represented by two consecutive quote characters | `1` |

The **Separator**, **QuoteChar** and **EscapeChar** characters, when defined, must be different. Other options defined for import are also accepted but ignored.

The **Overwrite** variant option (Boolean) from Version 16.0 remains supported but is deprecated in favour of **IfExists**.

## QuoteChar, EscapeChar and DoubleQuote options

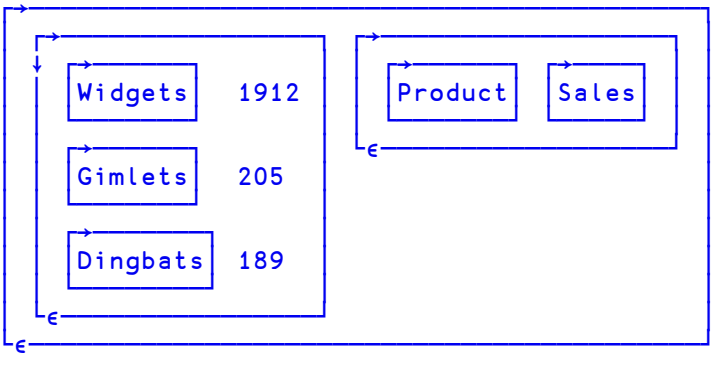
- The CSV text will be generated such that it can be read back according to the corresponding rules for import.
- If these options do not permit this (for example, a field contains the quote character and neither **DoubleQuote** or **EscapeChar** are set) an error is signalled.
- Quoting and Escaping is used as conservatively as possible.
- If both **QuoteChar** and **EscapeChar** are set, quoting is favoured.

If `Y` specifies that the CSV data is written to a file then `R` is the number of bytes (not characters) written, and is shy.

Otherwise, `R` is the CSV data in the format specified in Y, and is not shy.

### Examples

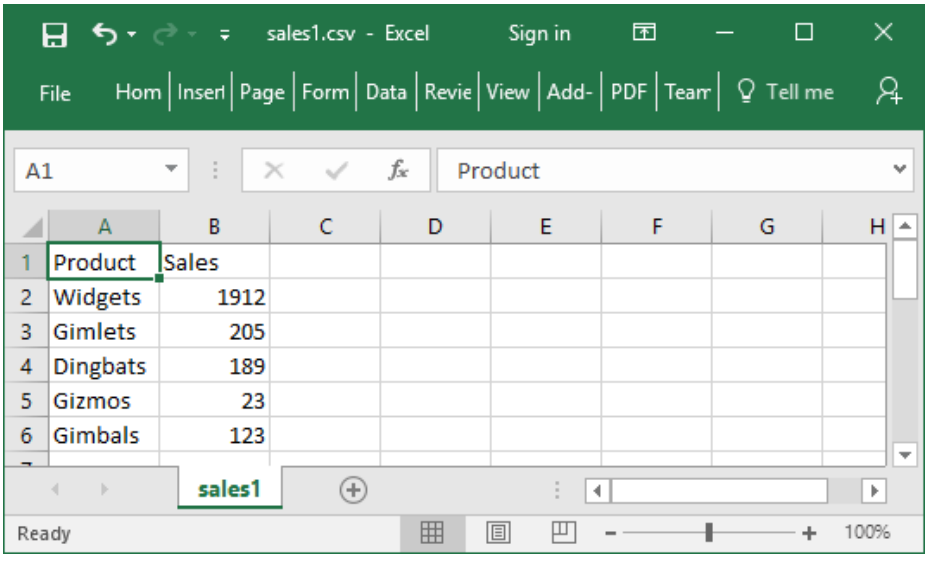
```
      CSVFile←'c:\Dyalog16.0\sales.csv'
      ⎕←DATA HDR←⎕CSV CSVFile''(1 2)1
```



```
      DATA←'Gizmos' 23
       DATA HDR ⎕CSV''
┌→────────────┐
│Product,Sales│
│             │
│Widgets,1912 │
│             │
│Gimlets,205  │
│             │
│Dingbats,189 │
│             │
│Gizmos,23    │
│             │
└─────────────┘
       CSVFile1←'c:\Dyalog16.0\sales1.csv'
       ⎕←DATA HDR ⎕CSV CSVFile1

67
       DATA←'Gimbals' 123
       ⎕←DATA HDR ⎕CSV CSVFile1
FILE NAME ERROR: Unable to create file ("The file
exists.")
       ⎕←DATA HDR ⎕CSV CSVFile1
      ^
       ⎕←DATA HDR(⎕CSV⍠'IfExists' 'Replace')CSVFile1

80
```



### Notes

- When `Y` contains only the destination of the CSV data (i.e. omits the description in its second element) it does not have to be enclosed to form a single element vector.
- Native files are written from the current file position. On successful completion, the file position will be at the end of the written data. If an error is signalled the amount of data written is undefined.
- If the file encoding specifies that a BOM is required and output is to a native file, it will only be written if the file position is initially at 0 - that is, the start of the file is being written.
- When fixed width fields are written, character data shorter than the specified width is padded with spaces to the right and character data longer than the specified width signals an error. Numeric data is converted to character data as far as possible so that it fits into the specified width. If this is not possible, an error is signalled.
- Tab-separated fields may be exported by specifying `'Separator' (⎕UCS 9)`.
- Fields containing a single embedded new line are supported. On export, line feed characters are mapped back to the defined line ending sequence.

## Make Directory {R}←{X}⎕MKDIR Y

This function creates new directories.

Y is a character vector or scalar containing a single directory name, or a vector of character vectors containing zero or more directory names. Names must conform to the naming rules of the host Operating System.

By default, for each file in Y the path must exist and the base name must not exist (see *File Name Parts* on page 79), otherwise an error is signalled. The optional left argument X is the numeric scalar 0, 1, 2 or 3 which amends this behaviour as shown in the following table. If omitted, it is assumed to be 0.

| | |
|---|---|
| 0 | Default behaviour. |
| 1 | No action is taken if a directory specified by Y already exists. The return value may be used to determine whether a new directory was created or not. |
| 2 | Any part of the *paths* specified in Y which does not already exist will be created in preparation of creating the corresponding directory. |
| 3 | Combination of 1 and 2. |

If Y specifies a single name, the shy result R is a scalar 1 if a directory was created or 0 if not. If Y is a vector of character vectors, R is a vector of 1s and 0s with the same length as Y.

### Examples

```
      ⎕NEXISTS '/Users/Pete/Documents/temp'
0
      ⎕←⎕MKDIR '/Users/Pete/Documents/temp'
1
      ⎕←⎕MKDIR '/Users/Pete/Documents/temp'
FILE NAME ERROR: Directory exists
      ⎕←⎕MKDIR'/Users/Pete/Documents/temp'
     ^

      ⎕←⎕MKDIR'/Users/Pete/Documents/temp/t1/t2'
FILE NAME ERROR: Unable to create directory ("The system
cannot find the path specified.")
      ⎕←⎕MKDIR'/Users/Pete/Documents/temp/t1/t2'
     ^

      ⎕←2 ⎕MKDIR'/Users/Pete/Documents/temp/t1/t2'
1
      ⊢⎕MKDIR'temp1' 'temp2'
1 1
```

### Note

When multiple names are specified they are processed in the order given. If an error occurs at any point whilst creating directories, processing will immediately stop and an error will be signalled. The operation is not atomic; some directories may be created before this happens. In the event of an error there will be no result and therefore no indication of how many directories were created before the error occurred.

# Native File Copy {R}←X ⎕NCOPY Y

This function copies native files and directories from one or more sources specified by Y to a destination specified by X. ⎕NCOPY is similar to ⎕NMOVE (see *Native File Move* on page 75).

X is a character vector that specifies the name of the destination.

Y is a character vector that specifies the name of the source, or a vector of character vectors containing zero or more sources.

Source and destination path names may be full or relative (to the current working directory) path names which adhere to the operating system conventions.

If X specifies an existent directory then each source in Y is copied into that directory, otherwise X specifies the name of the copy. X must specify an existent directory if the source contains multiple names or if the Wildcard option is set.

The shy result R contains count(s) of top-level items copied. If Y is a single source name, R is a scalar otherwise it is a vector of the same length as Y.

## Variant Options

⎕NCOPY may be applied using the Variant operator with the options **Wildcard** (the Principal option), **IfExists** and **PreserveAttributes**.

## Wildcard Option (Boolean)

| | |
|---|---|
| 0 | the name or names in Y identifies a specific file name. |
| 1 | the name or names in Y that specify the *base name* and *extension* (see *File Name Parts* on page 79), may also contain the wildcard characters "?" and "*". An asterisk is a substitute for any 0 or more characters in a file name or extension; a question-mark is a substitute for any single character. |

Note that when **Wildcard** is 1, element(s) of R can be 0, 1 or >1. If **Wildcard** is 0, elements of R are always 1.

## IfExists Option

The **IfExists** variant option determines what happens when a source file is to be copied to a target file that already exists. It does not apply to directories, only to the files within them.

| Value | Description |
|---|---|
| `'Error'` | Existing files will not be overwritten and an error will be signalled. This is the default |
| `'Skip'` | Existing files will not be overwritten but the corresponding copy operation will be skipped (ignored). |
| `'Replace'` | Existing files will be overwritten. |
| `'ReplaceIfNewer'` | Existing files may be overwritten if, and only if, the corresponding source file is newer (more recently modified) than the existing one, otherwise it is skipped. |

The following cases cause an error to be signalled regardless of the value of the **IfExists** variant.

- If the source specifies a directory and the destination specifies an existing file.
- If the source specifies a file and the same base name exists as a sub-directory in the destination.

## PreserveAttributes Option (Boolean)

The **PreserveAttributes** variant option determines whether or not file attributes are preserved. It does not apply to directories, only to files.

| | |
|---|---|
| 0 | file attributes are not preserved. |
| `1` | where possible, copied files will be given at least the same modification time as the source. Other file attributes will be preserved as permitted by the operating system and file system. |

Note also that when files are copied across file systems, the different file systems may have different timestamp granularity and the timestamps may not be exactly the same.

## Examples

There are a number of possibilities which are illustrated below. In all cases, if the source is a file, a copy of the file is created. If the source is a directory, a copy of the directory and all its contents is created.

### Examples (single source, Wildcard is 0)

- The source name must be an existent file or directory.
- If the destination name does not exist but its path name does exist, the source is copied to the destination name.
- If the destination name is an existing directory the copy is created within that directory with the base name of the source.

```
      ⍴1 ⎕NPARTS ''
i:/Documents/Dyalog APL-64 17.0 Unicode Files/

⍝ Make a named back-up of the Session file
      ⊢'session.bak' ⎕NCOPY 'default.dlf'

1
      ⊢ ⎕MKDIR 'backups' ⍝ Make a backups directory
1
 ⍝ Copy the Session file to backups directory
      ⊢'backups'⎕NCOPY'default.dlf'
1
      ↑⍴0 (⎕NINFO⍠1) 'backups\*'
backups/default.dlf
```

### Examples (single source, Wildcard is 1)

- The source name may include wildcard characters which matches a number of existing files and/or directories. The destination name must be an existing directory.
- The files and/or directories that match the pattern specified by the source name are copied into the destination directory. If there are no matches, zero copies are made.

```
      ⍴1 ⎕NPARTS ''
i:/Documents/Dyalog APL-64 17.0 Unicode Files/

      ⊢ ⎕MKDIR 'backups' ⍝ Make a backups directory
1
⍝ Copy all files to backups directory
      ⊢'backups'(⎕NCOPY⍠'Wildcard' 1)'*.*'
3
```

```
      ↑⊃0 (⎕NINFO⍠1) 'backups\*'
backups/default.dlf
backups/def_uk.dse
backups/UserCommand20.cache
```

### Examples (multiple sources, Wildcard is 0)

- Each source name must specify a single file or directory which must exist. The destination name must be an existing directory.
- Copies of each of the files and/or directories specified by the source base names are made in the destination directory.

```
       ⊃1 ⎕NPARTS ''
i:/Documents/Dyalog APL-64 17.0 Unicode Files/

      ⊢ ⎕MKDIR 'backups' ⍝ Make a backups directory
1
⍝ Copy 2 files to backups directory
      ⊢'backups'⎕NCOPY'default.dlf' 'def_uk.dse'
1 1
      ↑⊃0 (⎕NINFO⍠1) 'backups\*'
backups/default.dlf
backups/def_uk.dse
```

### Examples (multiple sources, Wildcard is 1)

- The destination name must be an existing directory.
- Copies of each of the files and/or directories that match the patterns specified by the source names (if any) are made in the destination directory.

```
      ⊃1 ⎕NPARTS ''
i:/Documents/Dyalog APL-64 17.0 Unicode Files/

      ⊢ ⎕MKDIR 'backups' ⍝ Make a backups directory
1
⍝ Copy files to backups directory
      ⊢'backups'(⎕NCOPY⍠1)'d*' 'UserCommand20.cache'
2 1
      ↑⊃0 (⎕NINFO⍠1) 'backups\*'
backups/default.dlf
backups/def_uk.dse
backups/UserCommand20.cache
```

### Notes

- The special directories `.` and `..` can never be copied into an existing directory.
- If any source name is a symbolic link it is dereferenced; that is, the source or directory it references is copied rather than the link itself.
- In the result `R`, a directory together with all its contents is counted once. A directory may be partially copied if the IfExists option is set to `'Replace'` or `'ReplaceIfNewer'`).
- If an error occurs during the copy process then processing will immediately stop and an error will be signalled. The operation is not atomic; some items may be copied before this happens. In the event of an error there will be no result and therefore no indication of how many names were copied before the error occurred.

# Native File Create {R}←X ⎕NCREATE Y

This function creates a new file. Under Windows the file is opened with mode 66 . Under non-Windows operating systems the current umask will specify the file permissions. The name of the new file is specified by the left argument `X` which must be a simple character vector or scalar containing a valid pathname for the file.

`Y` is 0 or a negative integer value that specifies an (unused) tie number by which the file may subsequently be referred. If `Y` is 0, the system allocates the first (closest to zero) available tie number which is returned as the result.

The shy result of `⎕NCREATE` is the tie number of the new file.

## Variant Options

`⎕NCREATE` may be applied using the Variant operator with the options **Unique** and **IfExists**. There is no primary option.

## Unique Option (Boolean)

| | |
|---|---|
| 0 | the file named by `X` will be created |
| `1` | a uniquely named file will be created by extending the base name (see *File Name Parts* on page 79) with random characters. If a unique name cannot be created then an error will be signalled. The actual name of the file can be determined from `⎕NNAMES` or `⎕NINFO`. |

## IfExists Option (character vector)

| | |
|---|---|
| `Error` | `⎕NCREATE` will generate a `FILE NAME ERROR` if the file already exists |
| `Replace` | `⎕NCREATE` will replace an existing file with an empty one of the same name. |

### Examples

```
      ⊢'myfile' ⎕NCREATE 0
¯1
      ⎕NUNTIE ¯1
      ⊢'myfile' ⎕NCREATE 0
FILE NAME ERROR: myfile: Unable to create file ("The file
exists.")
      ⊢'myfile'⎕NCREATE 0
               ^

      ⊢'myfile' (⎕NCREATE⍠'IfExists' 'Replace') 0

¯1    ⍝ Note that it uses same tie number as before

      ⊢'myfile' (⎕NCREATE⍠('Unique' 1)) 0
¯2
      ⎕NNUMS,⎕NNAMES
¯1 myfile
¯2 myfile52c36z
```

### Notes:

- Setting **IfExists** to `Replace` has no effect when **Unique** is 1, because the file cannot already exist.
- The **IfExists** option does not affect the operation of *slippery ties*.

## Native File Delete {R}←{X}⎕NDELETE Y

This function deletes files and directories.

Y is a character vector or scalar containing a single file or directory name, or a vector of character vectors containing zero or more file or directory names. Names must conform to the naming rules of the host Operating System.

The optional left argument X is a numeric scalar; valid values are shown in the following table. If omitted, its default value is 0.

| | |
|---|---|
| 0 | Each file or directory with the given name must exist. |
| 1 | If the file or directory with the given name does not exist then no action is taken. The result R may be used to determine whether the file or directory was deleted or not. |
| 2 | If a name identifies a non-empty directory it, and all its contents, are to be deleted. |
| 3 | Combination of 1 and 2. |

R is a numeric count of top-level entities deleted when processing the corresponding name in Y. If Y specifies a single name, R is a scalar. If Y is a vector of character vectors R is a vector with the same length as Y.

### Variant Options

⎕NDELETE may be applied using the Variant operator with the **Wildcard** option.

### Wildcard Option (Boolean)

| | |
|---|---|
| 0 | the name or names in Y identifies a specific file name. |
| 1 | the name or names in Y that specify the *base name* and *extension* (see *File Name Parts* on page 79), may also contain the wildcard characters "?" and "*". An asterisk is a substitute for any 0 or more characters in a file name or extension; a question-mark is a substitute for any single character. |

Note that when Wildcard is 1, element(s) of R can be 0 or >1. If Wildcard is 0, elements of R are always 1.

If Y specifies the name of a symbolic link, ⎕NDELETE deletes that symbolic link; the target of the symbolic link is unaffected.

### Examples

```
      ⎕NEXISTS'/Users/Pete/Documents/temp/t1/t2'
1
      ⊢⎕NDELETE'/Users/Pete/Documents/temp/t1/t2'
1
      ⊢⎕NDELETE'/Users/Pete/Documents/temp/t1/t2'
FILE NAME ERROR: Invalid file or directory name ("The
system cannot find the file specified.")
      ⊢⎕NDELETE'/Users/Pete/Documents/temp/t1/t2'
     ^
      ⊢1 ⎕NDELETE'/Users/Pete/Documents/temp/t1/t2'
0
      ⊢⎕NDELETE 'temp1' 'temp2'
1 1
      ⊢⎕MKDIR'temp1' 'temp2'
1 1
       ⊢(⎕NDELETE⍠1)'t*'
2

      ⊢⎕MKDIR'temp1'
1
      ⊢'Hello World' ⎕NPUT 'temp1/hw.txt'
13
      ⊢⎕NDELETE 'temp1'
FILE ACCESS ERROR: temp1: Unable to delete directory
("The directory is not empty.")
      ⊢⎕NDELETE'temp1'
       ^
      ⊢2 ⎕NDELETE 'temp1'
1
```

If the file is in use or the current user is not authorised to delete it, `⎕NDELETE` will not succeed but will instead generate a `FILE ACCESS ERROR`.

### Note

When multiple names are specified they are processed in the order given. If an error occurs at any point whilst deleting files or directories, processing will immediately stop and an error will be signalled. The operation is not atomic; the directory contents may be partially deleted before this happens. In the event of an error there will be no result and therefore no indication of how many files were deleted before the error occurred.

# Native File Exists R←⎕NEXISTS Y

This function reports whether or not file and directories exist.

Y is a character vector or scalar containing a single directory name, or a vector of character vectors containing zero or more directory names. Names must conform to the naming rules of the host Operating System.

If Y specifies a single name, the result R is a scalar 1 if a file or directory exists or 0 if not. If Y is a vector of character vectors, R is a vector of 1s and 0s with the same length as Y.0

## Variant Options

⎕NEXISTS may be applied using the Variant operator with the **Wildcard** option.

## Wildcard Option (Boolean)

| | |
|---|---|
| 0 | the name or names in Y identifies a specific file name. |
| 1 | the name or names in Y that specify the *base name* and *extension* (see *File Name Parts* on page 79), may also contain the wildcard characters "?" and "*". An asterisk is a substitute for any 0 or more characters in a file name or extension; a question-mark is a substitute for any single character. |

If the Wildcard option is 1, R indicates whether or not one or more matches to the corresponding pattern in Y exist.

### Example

```
      ⎕←⎕MKDIR'/Users/Pete/Documents/temp/t1/t2'
1
      ⎕NEXISTS'/Users/Pete/Documents/temp/t1/t2'
1
      ⎕NEXISTS'/Users/Pete/Documents/temp/t1/t2/pd'
0

      ⊢⎕MKDIR'temp1' 'temp2'
1 1
      ⎕NEXISTS 'temp1' 'temp2' 'temp3'
1 1 0
      (⎕NEXISTS⍠1) 't*'
1
```

### Note

If Y is a symbolic link, ⎕NEXISTS will return 1 whether or not the target of the symbolic link exists.

## Read Text File `R←{X} ⎕NGET Y`

This function reads the contents of the specified text file. See also *Write Text File* on page 81.

`Y` is either a character vector/scalar containing the name of the file to be read, or a 2-item vector whose first item is the file name and whose second is an integer scalar specifying `flags` for the operation.

If `flags` is 0 (the default value if omitted) the content in the result `R` is a character vector. If `flags` is 1 the result is a nested array of character vectors corresponding to the lines in the file.

The optional left-argument `X` is either

- a character vector that specifies the file-encoding as shown in the table below.
- a 256-element numeric vector that maps each possible byte value (0-255) to a Unicode code point (1st element = Unicode code point corresponding to byte value 0, and so on). ¯1 indicates that the corresponding byte value is not mapped to any character. Apart from ¯1, no value may appear in the table more than once.

**Table 1: File Encodings**

| Encoding | Description |
|---|---|
| `UTF-8` | The data is encoded as UTF-8 format. |
| `UTF-16LE` | The data is encoded as UTF-16 little-endian format. |
| `UTF-16BE` | The data is encoded as UTF-16 big-endian format. |
| `UTF-16` | The data is encoded as UTF-16 with the endianness of the host system (currently BE on AIX platforms, LE on all others). |
| `UTF-32LE` | The data is encoded as UTF-32 little-endian format. |
| `UTF-32BE` | The data is encoded as UTF-32 big-endian format. |
| `UTF-32` | The data is encoded as UTF-32 with the endianness of the host system (currently BE on AIX platforms, LE on all others). |
| `ASCII` | The data is encoded as 7-bit ASCII format. |
| `Windows-1252` | The data is encoded as 8-bit Windows-1252 format. |
| `ANSI` | ANSI is a synonym of Windows-1252. |

The above UTF formats may be qualified with -BOM or -NOBOM (e.g. UTF-8-BOM). See *Write Text File* on page 81.

Whether or not `X` is specified, if the start of the file contains a recognised Byte Order Mark (BOM), the file is decoded according to the BOM. Otherwise, if `X` is specified the file is decoded according to the value of `X`. Otherwise, the file is examined to try to decide its encoding and is decoded accordingly.

The result `R` is a 3-element vector comprising `(content) (encoding) (newline)` where:

| | |
|---|---|
| `content` | A simple character vector, or a vector of character vectors, according to the value of `flags`. |
| `encoding` | The encoding that was actually used to read the file. If this is a UTF format, it will always include the appropriate endianness (except for UTF-8 to which endianness doesn't apply) and a -BOM or -NOBOM suffix to indicate whether or not a BOM is actually present in the file. For example, UTF-16LE-BOM.<br><br>If `X` specified a user-defined encoding as a 256-element numeric vector, `encoding` will be that same vector. |
| `newline` | Determined by the first occurrence in the file of one of the newline characters identified in the line separator table, or `⍬` if no such line separator is found. |

If `content` is simple then all its line separators (listed in the table below) are replaced by (normalised to) `⎕UCS 10`, which in the Classic Edition must be in `⎕AVU` (else `TRANSLATION ERROR`).

If `content` is nested, it is formed by splitting the contents of the file on the occurrence of *any* of the line separators shown in the table below. These line separators are removed.

The 3rd element of the result `newline` is a numeric vector from the *Value* column of the table below corresponding to the first occurrence of any of the **newline characters** in the file. If none of these characters are present, the value is `⍬`.

**Table 2: Line separators:**

| Value | Code | Description |
|---|---|---|
| **newline characters** | | |
| 13 | `CR` | Carriage Return (U+000D) |
| 10 | `LF` | Line Feed (U+000A) |
| 13 10 | `CRLF` | Carriage Return followed by Line Feed |
| 133 | `NEL` | New Line (U+0085) |
| **other line separator characters** | | |
| 11 | `VT` | Vertical Tab (U+000B) |
| 12 | `FF` | Form Feed (U+000C) |
| 8232 | `LS` | Line Separator (U+2028) |
| 8233 | `PS` | Paragraph Separator (U+2029) |

# Native File Information R←{X}⎕NINFO Y

This function returns information about one or more files or directories.

Y may be:

- a numeric scalar containing the tie number of a native file
- a character vector or scalar containing a file or directory name that conforms to the naming rules of the host Operating System.
- a vector of character vectors and/or tie numbers

## Variant Options

⎕NINFO may be applied using the Variant operator with the options **Wildcard** (the Principal option), **Recurse** and **Follow**.

## Wildcard Option (Boolean)

| | |
|---|---|
| 0 | the name or names in Y identifies a specific file name. |
| 1 | the name or names in Y that specify the *base name* and *extension* (see *File Name Parts* on page 79), may also contain the wildcard characters "?" and "*". An asterisk is a substitute for any 0 or more characters in a file name or extension; a question-mark is a substitute for any single character. |

## Recurse Option (Boolean)

| | |
|---|---|
| 0 | the name(s) in Y are searched for only in the corresponding specified directory |
| 1 | the name(s) in Y are searched for in the corresponding specified directory as well as all sub-directories. If **Wildcard** is also 1, the wild card search is performed recursively. |

The optional left argument X is a simple numeric array containing values shown in the following table.

## Follow Option (Boolean)

| | |
|---|---|
| 0 | the properties reported are those of the symbolic link itself |
| 1 | the properties reported for a symbolic link are those of the target of the symbolic link |

The optional left argument `X` is a simple numeric array containing values shown in the following table.

| X | Property | Default |
|---|---|---|
| 0 | Name of the file or directory, as a character vector. If `Y` is a tie number then this is the name which the file was tied. | |
| 1 | Type, as a numeric scalar:<br>0=Not known<br>1=Directory<br>2=Regular file<br>3=Character device<br>4=Symbolic link (only when Follow is 0)<br>5=Block device<br>6=FIFO (not Windows)<br>7=Socket (not Windows) | 0 |
| 2 | Size in bytes, as a numeric scalar | 0 |
| 3 | Last modification time, as a timestamp in `⎕TS` format | 7⍴0 |
| 4 | Owner user id, as a character vector – on Windows a SID, on other platforms a numeric userid converted to character format | '' |
| 5 | Owner name, as a character vector | '' |
| 6 | Whether the file or directory is hidden (1) or not (0), as a numeric scalar. On Windows, file properties include a "hidden" attribute; on non-Windows platforms a file or directory is implicitly considered to be hidden if its name begins with a "." | ¯1 |
| 7 | Target of symbolic link (when Type is 4) | '' |
| 8 | Current file position | 0 |
| 9 | Last access time in `⎕TS` format, when available | 7⍴0 |
| 10 | Creation time in `⎕TS` format, when available | 7⍴0 |
| 11 | Whether the file can (1) or cannot (0) be read (`¯1` if unknown) | ¯1 |
| 12 | Whether the file can (1) or cannot (0) be written (`¯1` if unknown) | ¯1 |

Note that the current file position identifies where ⎕NREAD will next read from or ⎕NAPPEND will next write to and is only pertinent when the corresponding value in Y is a tie number rather than a name. It will be reported as 0 for named files.

Each value in X identifies a property of the file(s) or directory(ies) identified by Y whose value is to be returned in the result R. If omitted, the default value of X is 0. Values in X may be specified in any order and duplicates are allowed. A value in X which is not defined in the table above will not generate an error but results in a ⍬ (Zilde) in the corresponding element of R.

R is the same shape as X and each element contains value(s) determined by the property specified in the corresponding element in X. The depth of R depends upon whether or not the Wildcard option is enabled. If, for any reason, the function is unable to obtain a property value, (for example, if the file is in use exclusively by another process) the default value shown in the last column is returned instead.

If the **Wildcard** option is not enabled (the default) then Y specifies exactly one file or directory and must exist. In this case each element in R is a single property value for that file. If the name in Y does not exist, the function signals an error. On non-Windows platforms "*" and "?" are treated as normal characters. On Windows an error will be signalled since neither "*" nor "?" are valid characters for file or directory names.

If the **Wildcard** option is enabled, zero or more files and/or directories may match the pattern in Y. In this case each element in R is a vector of property values for each of the files. Note that no error will be signalled if no files match the pattern.

When using the **Wildcard** option, matching of names is done case insensitively on Windows and macOS, and case sensitively on other platforms. The names '.' and '..' are excluded from any matches. The order in which the names match is not defined.

### Examples

```
      (0 1 2) ⎕NINFO 'c:/Users/Pete/Documents'
┌→──────────────────────────────────┐
│ ┌→──────────────────────┐ 1 163840 │
│ │c:/Users/Pete/Documents│          │
└∊┴───────────────────────┴──────────┘

      ⎕CY'dfns' ⍝ current working directory
c:/Users/Pete/
      (⎕NINFO⍠1)'D*'
┌──────────┬─────────┬─────────┬───────┐
│Desktop   │Documents│Downloads│Dropbox│
└──────────┴─────────┴─────────┴───────┘
```

```
      (⎕NINFO⍠1)'Documents/*.zip'
```

| Documents/dyalog.zip |
|---|

```
      ⍪ (0,⍳6) ⎕NINFO 'Documents/dyalog.zip'
```

| Documents/dyalog.zip |
|---|
| 2 |
| 3429284 |
| 2016 1 22 16 43 58 0 |
| S-1-5-21-2756282986-1198856910-2233986399-1001 |
| HP/Pete |
| 0 |

```
      ⊃1⎕NPARTS '' ⍝ current working directory
C:/Users/Pete/Documents/Dyalog APL-64 16.0 Unicode Files/
      (⎕NINFO⍠1)'*.*'
```

| default.dlf | def_uk.dse | jsonx.dws | UserCommand20.cache |
|---|---|---|---|

```
      ⊢ ⎕MKDIR 'd1' 'd2'
1 1
      'a'⍪⎕NPUT¨'find' 'd1/find' 'd1/nofind' 'd2/find'
      (⎕ninfo⍠'Recurse' 1)'find'
```

| d1/find | d2/find | find |
|---|---|---|

The following expression will return all Word document (`.docx` and `.doc`) in the current directory, searching recursively through any sub-directories:

```
      (⎕NINFO⍠('Recurse' 1)('Wildcard' 1))'*.docx' '*.doc'
```

### Note

Of the file timestamps, only the last modification time should be considered reliable and portable. Neither the access time or creation time are well supported across all platforms.

# Native File Move {R}←X ⎕NMOVE Y

This function moves native files and directories from one or more sources specified by Y to a destination specified by X. ⎕NMOVE is similar to ⎕NCOPY (see *Native File Copy* on page 57).

When possible ⎕NMOVE *renames* files and directories, which effects a fast move when the source and destination are on the same file system. By default (see **RenameOnly** option below), if ⎕NMOVE is unable to rename files or directories, it instead copies them and deletes the originals.

X is a character vector that specifies the name of the destination.

Y is a character vector that specifies the name of the source, or a vector of character vectors containing zero or more sources.

Sources and destinations may be full or relative (to the current working directory) path names adhering to the operating system convention.

If Y specifies more than one source, X must be a character vector that specifies an existent directory to which each of the sources in Y is to be moved.

The shy result R contains count(s) of top-level items moved. If Y is a single source name, R is a scalar otherwise it is a vector of the same length as Y.

## Variant Options

⎕NMOVE may be applied using the Variant operator with the options **Wildcard** (the Principal option), **IfExists** and **RenameOnly**.

## Wildcard Option (Boolean)

| | |
|---|---|
| 0 | the name or names in Y identifies a specific file name. |
| 1 | the name or names in Y that specify the *base name* and *extension* (see *File Name Parts* on page 79), may also contain the wildcard characters "?" and "*". An asterisk is a substitute for any 0 or more characters in a file name or extension; a question-mark is a substitute for any single character. |

Note that when **Wildcard** is 1, element(s) of R can be 0 or >1. If **Wildcard** is 0, elements of R are always 1.

## IfExists Option

The **IfExists** variant option determines what happens when a source file is to be copied to a target file that already exists. It does not apply to directories, only to the files within them.

| Value | Description |
|---|---|
| `'Error'` | Existing files will not be overwritten and an error will be signalled. This is the default |
| `'Skip'` | Existing files will not be overwritten but the corresponding copy operation will be skipped (ignored). |

The following cases cause an error to be signalled regardless of the value of the **IfExists** variant.

- If the source specifies a directory and the destination specifies an existing file.
- If the source specifies a file and the same base name exists as a sub-directory in the destination.

## RenameOnly Option (Boolean)

The **RenameOnly** option determines what happens when it is not possible to rename the source.

| | |
|---|---|
| 0 | the source will be copied and the original deleted |
| 1 | the move will fail |

## Examples

A number of possibilities exist, illustrated by the following examples. In all cases, if the source is a file, the file is moved. If the source is a directory, the directory and all of its contents are moved.

### Examples (single source, Wildcard is 0)

- The source name must be an existent file or directory.
- If the destination name does not exist but its path name does exist, the source is moved to the destination name.
- If the destination name is an existing directory the source name is moved to that directory.

```
      ⍝1 ⎕NPARTS ''
i:/Documents/Dyalog APL-64 17.0 Unicode Files/

⍝ Rename the Session file
      ⊢'session.dlf' ⎕NMOVE 'default.dlf'

1
      ⊢ ⎕MKDIR 'backups' ⍝ Make a backups directory
1
 ⍝ Move the Session file to backups directory
      ⊢'backups'⎕NMOVE'default.dlf'
1
      ↑⍝0 (⎕NINFO⍠1) 'backups\*'
backups/default.dlf
```

### Examples (single source, Wildcard is 1)

- The source name may include wildcard characters which matches a number of existing files and/or directories. The destination name must be an existing directory.
- The files and/or directories that match the pattern specified by the source name are moved into the destination directory. If there are no matches, zero copies are made.

```
      ⍝1 ⎕NPARTS ''
i:/Documents/Dyalog APL-64 17.0 Unicode Files/

      ⊢ ⎕MKDIR 'backups' ⍝ Make a backups directory
1
⍝ Move all files to backups directory
      ⊢'backups'(⎕NMOVE⍠'Wildcard' 1)'*.*'
3
      ↑⍝0 (⎕NINFO⍠1) 'backups\*'
backups/default.dlf
backups/def_uk.dse
backups/UserCommand20.cache
```

### Examples (multiple sources, Wildcard is 0)

- Each source name must specify a single file or directory which must exist. The destination name must be an existing directory.
- Each of the files and/or directories specified by the source base names are moved to the destination directory.

```
      ⍝1 ⎕NPARTS ''
i:/Documents/Dyalog APL-64 17.0 Unicode Files/

      ⊢ ⎕MKDIR 'backups' ⍝ Make a backups directory
1
```

```
⍝ Move 2 files to backups directory
      ⊢'backups'⎕NMOVE'default.dlf' 'def_uk.dse'
1 1
      ↑⊃0 (⎕NINFO⍠1) 'backups\*'
backups/default.dlf
backups/def_uk.dse
```

### Examples (multiple sources, Wildcard is 1)

- The destination name must be an existing directory.
- Each of the files and/or directories that match the patterns specified by the source names (if any) are moved to the destination directory.

```
      ⊃1 ⎕NPARTS ''
i:/Documents/Dyalog APL-64 17.0 Unicode Files/

      ⊢ ⎕MKDIR 'backups' ⍝ Make a backups directory
1
⍝ Move files to backups directory
      ⊢'backups'(⎕NMOVE⍠1)'d*' 'UserCommand20.cache'
2 1
      ↑⊃0 (⎕NINFO⍠1) 'backups\*'
backups/default.dlf
backups/def_uk.dse
backups/UserCommand20.cache
```

### Note

When ⎕NMOVE copies and deletes files:

- The operation will take longer to complete.
- File modification times will be preserved but other attributes such as file ownership may be changed.
- Read permissions will be needed on all files within a directory which is moved.
- If the operation fails at any point and an error is signalled it is possible that there may be files and/or directories left duplicated in both the source and destination. It is not possible that a file or directory may be removed from the source without having been copied to the destination.

## File Name Parts R←{X} ⎕NPARTS Y

Splits a file name into its constituent parts.

Y is a character vector or scalar containing a single file name, or a vector of character vectors containing zero or more file names. Names must conform to the naming rules of the host Operating System.

The file(s) need not exist; indeed this system function makes no attempt to identify or locate it/them.

The optional left-argument X specifies whether or not the file name or names specified by Y are *normalised* before being processed. The default value 0 means no normalisation; 1 means normalise as follows:

- Pathnames are made absolute.
- On Windows, all "\" directory separators are changed to "/".
- The resultant name is simplified by removing extraneous directory separators etc. On Windows, this includes resolving occurrences of "." and ".." within the name. On non-Windows platforms single "." are removed. Note that ".." and symbolic links interact differently on Windows to other platforms; on other platforms they cannot be removed without reference to the file system itself and are left in place.

If Y is a scalar or vector, the result R is a 3-element vector of character vectors as follows:

| | |
|---|---|
| [1] | *path* |
| [2] | *base name* |
| [3] | *extension* |

The *path* identifies the directory in which the file exists.

The *base name* is the name of the file stripped of its path and extension, if any.

The *extension* is the file extension including the leading ".".

If Y is a vector of character vectors, R is a vector of 3-element character vectors and is the same length as Y.

## Examples

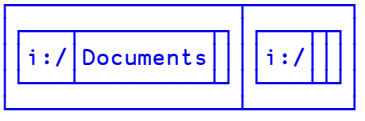
```
      ⎕CMD 'CD'⍵ Current working directory
c:\Users\Pete

      1 ⎕NPARTS 'a'
```


```
      1 ⎕NPARTS '\Users\Pete\Documents\dyalog.zip'
```


```
      ⍕'.'⎕wg'APLVersion'
AIX-64
      1 ⎕nparts'/home/andys/./..'
```


```
      1 ⎕NPARTS '.' '..'
```




Note that `⍕1 ⎕NPARTS ''` returns the current working directory.

```
      ⍕1 ⎕NPARTS ''
```

## Write Text File {R}←X ⎕NPUT Y

This function writes character data to a text file. See also *Read Text File* on page 67.

`Y` is either a simple character vector or scalar containing the name of the file to be written, or a 2-item vector whose first item is the file name and whose second is an integer scalar specifying `flags` for the operation.

If `flags` is 0 (the default value if omitted) the file will not be overwritten if it already exists and `⎕NPUT` will signal an error. If `flags` is 1 the file will be overwritten. If flags is 2 the file will be appended to; i.e.

| `flags` | file does not exist | file exists |
|---|---|---|
| `0` | data is written to new file | error signalled, file is unchanged |
| `1` | data is written to new file | file is overwritten |
| `2` | data is written to new file | data is appended to file |

The left-argument `X` is comprised of 1, 2 or 3 items which identify `(content) (encoding) (newline)` respectively.

`content` is either a vector of character vectors, each of which represents a line in the file to be written, or a simple character vector.

If specified, `encoding` is either:

- a character vector from the first column in the table *File Encodings* on page 67. If `encoding` specifies a UTF format, it may be qualified with -BOM (e.g. UTF-8-BOM), which causes a Byte Order Mark (BOM) to be written at the beginning of the file or -NOBOM which does not. If the -BOM or -NOBOM suffix is omitted, UTF-8 defaults to UTF-8-NOBOM, while the other UTF formats default to -BOM.
- a 256-element numeric vector that maps each possible byte value (0-255) to a Unicode code point (1st element = Unicode code point corresponding to byte value 0, and so on). ¯1 indicates that the corresponding byte value is not mapped to any character. Apart from ¯1, no value may appear in the table more than once.

If omitted, `encoding` defaults to UTF-8-NOBOM.

Note: If a non-empty file is appended to:

- No BOM will be written, even if encoding specifies it.
- No check is made that the existing file content is text in the same encoding format.

If specified, `newline` is numeric and is either `⍬` or a scalar or vector from the column labelled *Value* in the **newline characters** section of the table *Line separators:* on page 69. Any other value causes `DOMAIN ERROR`. If `newline` is omitted it defaults to `(13 10)` on Windows and `10` on other platforms.

If `content` is nested, each element is considered to be to a logical line in the file, and when the file is written, a line separator character corresponding to `newline` is appended to each and every element, i.e. the data written to the file (excluding the BOM) is:

```
      ∈content,¨⊂⎕UCS newline
```

If `content` is simple each and every LF (`⎕UCS 10`) character that it contains is first replaced by the character corresponding to `newline`. If not present, one LF character is added to the end of the array prior to these replacements.

In both cases, any other line separator characters are written *as is* to the file. This allows the APL programmer to insert other line endings if so desired.

If content contains anything other than a character vector or scalar (or these, nested) then a `DOMAIN ERROR` is signalled.

The shy result `R` is the number of bytes written to the file.

Note that when `content` is a vector of character vectors and `encoding` is omitted; it is necessary to enclose the left argument.

**Example:**

```
      txt←'mene' 'mene' 'tekel' 'upharsin'
      ⎕←(⊂txt) ⎕NPUT 'writing.txt'
25
      ⊢(⊂'adding' '3' 'lines')⎕NPUT'writing.txt' 2
18
```

# Chapter 4:

# I-Beam Reference Changes

## I-beam Changes

I-beam functionality changed from Version 16.0.

| A | Description | Change |
|---|---|---|
| 201 | Syntax Colour Tokens | New function |
| 739 | Temporary Directory | New function |
| 5177 | List Loaded File Objects | Now reports file check sum and last modification time |
| 5178 | Remove Loaded File Object Info | New function |
| 5179 | Loaded File Object Info | New function |
| 7159 | JSON Import | Removed (replaced by ]JSON) |
| 7160 | JSON Export | Removed (replaced by ]JSON) |
| 7161 | JSON TrueFalse | Removed (replaced by ]JSON) |

# Syntax Colour Tokens R←201⌶Y

This function provides a description of the syntax colour tokens reported by 200⌶. See *Syntax Colouring* on page 1.

Y is ⍬ (zilde).

R is a 3-column matrix that describes the syntax colouring tokens as follows:

| | |
|---|---|
| R[;1] | Token type |
| R[;2] | Token Value |
| R[;3] | Internal description |

## Example

```
      ⍴201⌶⍬
207 3
      3 3↑201⌶⍬
```

| | | |
|---|---|---|
| Global token | 0 | MINI_NULL |
| Global token | 1 | MINI_COMMENT |
| Global token | 2 | MINI_UCC |

## Temporary Directory R←739⌶Y

Returns the name of a system temporary directory suitable for user files, as a character vector. The name reported does not include a trailing directory separator

`Y` is 0.

The result `R` is a character vector.

**Example (Windows)**

```
      739⌶0
C:/Users/Pete/AppData/Local/Temp
```

**Example (non-Windows)**

```
      739⌶0
/tmp
```

## Remove Loaded File Object Info R←5178⌶Y

The editor may be used to edit Dyalog script files (*.dyalog* files) and general text files and to save the contents in the workspace. Additionally `⎕FIX` can be used to fix scripts held in files. This I-Beam removes the information held about an object in the workspace specified by `Y` that is associated with such a file.

`Y` is a character vector that specifies the name of a workspace object or a ref to an object.

`R` is Boolean. 1 means that the information was removed; 0 that it wasn't.

Note that the workspace object itself remains in the workspace; just the information about its associated file is removed.

**Examples:**

```
      dyalog←2 ⎕NQ '.' 'GetEnvironment' 'DYALOG'
      aedit←'/SALT/spice/aedit.dyalog'
      ⊢⎕FIX 'file://',dyalog,aedit
#.arrayeditor
      5178⌶'arrayeditor'
1
      5178⌶'xyz' ⍝ unused name
0
```

# Loaded File Object Info R←5179⌶Y

The editor may be used to edit Dyalog script files (*.dyalog* files) and general text files and to save the contents in the workspace. Additionally ⎕FIX can be used to fix scripts held in files. This I-Beam returns details about an object in the workspace specified by Y that is associated with such a file.

Y is a character vector that specifies the name of a workspace object or a ref to an object.

R is an 8-element vector containing the following information pertaining to the object and

| Element | Contains |
|---|---|
| 1 | Object name or ref (Y) |
| 2 | Parent namespace |
| 3 | Name class (see ⎕NC) |
| 4 | File name |
| 5 | Start line (first line in file, 0 origin, of the object) |
| 6 | Line count (number of lines in file occupied by the object) |
| 7 | File Checksum |
| 8 | File modification time (⎕TS format) |

If an object occupies a file in its entirety, both *Start line* and *Line count* are 0.

**Examples:**

```
      dyalog←2 ⎕NQ '.' 'GetEnvironment' 'DYALOG'
      aedit←'/SALT/spice/aedit.dyalog'
      ⊢⎕FIX 'file://',dyalog,aedit
#.arrayeditor
      1 1 1 0 1 1 1 1/ 5179⌶'arrayeditor'
```

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| #.arrayeditor | # | 9 | 0 | 0 | 008fe4ed | 2018 | 5 | 11 | 8 | 56 | 10 | 0 |

```
      1 1 1 0 1 1 1 1/ 5179⌶'arrayeditor.List'
```

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| List | #.arrayeditor | 3 | 22 | 5 | 008fe4ed | 2018 | 5 | 11 | 8 | 56 | 10 | 0 |

```
      5179⌶'xyz' ⍝ unused name
```

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | [Null] | 0 | | 0 | 0 | 00000000 | 1970 | 1 | 1 | 0 | 0 | 0 | 0 |

# Chapter 5:

# Object Reference Changes

# CellMove Event 151

**Applies To:** Grid

**Description**

If enabled, this event is reported when the user attempts to position the cursor over a cell in a Grid by clicking the left mouse button or by pressing a cursor movement key. The purpose of this event is to allow an application to perform some action prior to the user entering a cell, to inhibit entry to a cell, or to deny exit from the current cell.

The default action is to position the user on the new cell. This action can be prevented by returning a 0 from the callback function attached to the event.

The event message reported as the result of `⎕DQ`, or supplied as the right argument to your callback function, is an 8 element vector as follows :

| | | |
|---|---|---|
| `[1]` | Object | ref or character vector |
| `[2]` | Event | `'CellMove'` or 151 |
| `[3]` | New cell row | integer |
| `[4]` | New cell column | integer |
| `[5]` | Scroll flag | 0 or 1 |
| `[6]` | Selection flag | 0, 1 or 2 |
| `[7]` | Mouse flag | 0 or 1 |
| `[8]` | Changed flag | 0 or 1 (relates to **current** cell) |
| `[9]` | New value | new value of **current** cell or `⍬` |

The 5th element of the event message is 1 if switching to the new cell would cause the Grid to scroll.

The 6th element of the event message is 1 if the user is moving to the new cell by extending the selection. It is 2 if the user selects an entire row or column (by clicking on a title), which moves the current cell to the first one in the selection.

The 7th element of the event message is 1 if the mouse is used to switch to a new cell.

The 8th element of the event message is 1 if the user is attempting to move to the new cell from another cell in the Grid having typed in (as if to alter) the current cell.

The 9th element of the event message is the intended new value in the current cell or ø (zilde) if *Changed flag* is 0.

The CellMove event may be used to validate and refuse changes as the user navigates between cells.

An application can position the user on a particular cell in a Grid by calling CellMove as a method. If so, the argument need contain only the *New cell row* and *New cell column* parameters.

# Chapter 6:

# Non-Windows Specific Features

## Summary

This section summarises the changes specific to Dyalog APL Version 17.0 on non-Windows platforms. This list currently consists of:

- AIX
- Linux (including the Raspberry Pi)
- macOS/ Mac OS X

## Hardware Requirements

### AIX

For AIX, Version 17.0 requires AIX 7.2 or higher, and a POWER7 chip or higher.

### Raspberry Pi

On the Raspberry Pi, Dyalog 32-bit Unicode supports Raspbian Jessie or later.

### Non-Pi Linux

For non-Pi Linux, Version 17.0 only exists as 64-bit interpreters - there are no 32-bit versions. It is built on Debian 7, and QAed on RedHat 6; it runs on all recent distributions, including Ubuntu 14.04 and openSUSE Leap 42.3. Contact Dyalog for information about other distributions.

### macOS/Mac OS X

Version 17.0 requires Mac OS X Yosemite or El Capitan or macOS Sierra or later. The target Mac must have been introduced in 2010 or later.

# RIDE and Dyalog APL 17.0

Dyalog Version 17.0 supports RIDE 3 and RIDE 4 only; RIDE 2 is not supported. Dyalog recommends that RIDE 4 is used in preference to RIDE 3. RIDE 4 can be used with Version 15.0 too.

RIDE 4 is supported on Raspberry Pi models 2 and 3 only; models Zero and 1 are not supported (the underlying libraries which RIDE is built on are not available for the Pi Zero and 1). The *Dyalog RIDE Reference Guide* details how to configure the APL session to support the underscored alphabet; contact support@dyalog.com if you wish to be able to generate key-chords which result in the underscored alphabet being entered into APL.

Note that on Linux and Pi, if RIDE 4 is installed after Dyalog 16.0 an extra icon will be added to the window manager's start menu which will start Dyalog with a RIDE front end.

# Location of configuration and log files

In Dyalog 17.0 the location of various configuration and log files has been changed so that they are all put in one directory. See the *UNIX Installation and Configuration Guide* for more information.

# SQAPL on macOS

Dyalog 17.0 for macOS includes support for SQAPL. However, it is necessary to install iODBC and suitable drivers for your database before SQAPL can work. *The SQL Interface Guide* describes the steps that are typically necessary to get SQAPL connected to a MySQL database.

# 4000I and 4002I

4000I (Fork process) and 4002I (Reap processes) have been withdrawn on all platforms except AIX. This is due to limitations imposed by the HTMLRenderer, and due to problems in the interaction of forking processes and using RIDE.

# Index