VAX APL

Reference Manual

AA–GV09C–TE

June 1991

This reference manual describes the VAX APL functions, operators, variables, and system commands.

Revision/Update Information: This revised document supersedes the *VAX APL Reference Manual Vols. I&II*

AA-P142D-TE
AA-GV09B-TE

Operating System: VMS Version 5.4

Software Version: VAX APL 4.0

Digital Equipment Corporation
Maynard, Massachusetts

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by Digital Equipment Corporation or its affiliated companies.

The Reader's Comments form at the end of this document requests your critical evaluation to assist in preparing future documentation.

Contents

2 VAX APL System Variables and Functions

3 VAX APL System Commands

Figures

Tables

Preface

This manual describes the VAX APL interpreter, including VAX APL language and programming elements, facilities for controlling the VAX APL environment, the interaction between VAX APL and the VMS operating system, and VAX APL's I/O capabilities.

Intended Audience

This manual is intended for experienced APL programmers. This manual is not a tutorial and is inappropriate for novice users. Programmers experienced with other languages such as FORTRAN or BASIC can learn VAX APL from this manual, but are advised to study it in conjunction with an APL language primer.

Related Documents

The *VAX APL User's Guide* describes the VAX APL interpreter and the environment in which it operates. The *VAX APL Installation Guide* contains instructions for installing VAX APL on the VMS operating system. The *VAX APL Installation Guide* also explains how to install QAPL, the license-free, execute only version of VAX APL.

To find out more about the VMS system, refer to the VMS system documents listed in the *Introduction to VMS* or use the Help utility by entering HELP at the system prompt ($). The *VMS DCL Dictionary* and the *Introduction to VMS System Management* provide detailed information you may need to know to use some of the features of VAX APL.

Product References

In this document, VAX APL is referred to as APL.

Conventions

The following conventions are used in this manual.

Table 1 Documentation Conventions Table

Conventions	Meaning
Default values used in examples	The default value for the index origin (⎕*IO*) is 1, unless explicitly stated to be 0. Numeric print precision (⎕*PP*) is 10 digits. Enclosed arrays are displayed with boxes around enclosed items and with all values in the top left corner of the display areas. This is done using: `⎕DC ← (¯1 ¯1 2 3)'++++││--'`
Delimiting pairs	This manual uses ⍝ *text* ⍝ ; other delimiting pairs may be any of the following pairs: ⍞ ⍞ ¨ ¨ ⎕ ⎕ < > ⊂ ⊃
UPPERCASE	Uppercase words and letters, used in format examples, indicate that you should type the word or letter exactly as shown.
A B K	The APL characters *A*, *B*, and *K* are used in generic descriptions of command formats. *A* represents a left argument, *B* represents a right argument, and *K* represents an axis argument.
italics	Italicized lowercase words and letters, used in format examples, indicate that you are to substitute a word or value of your choice.
Quotation mark (')	The term quotation mark refers to the APL single quotation mark (').
←→	The equivalence symbol means "is equivalent to".
⟦ ⟧	The double square brackets indicate that the item or string of items inside the brackets is optional. Individual items within a string of items are delimited by the .ab character, which indicates that you may choose only one item from the string.
[ ]	Single square brackets that appear in the format specification for a language element are required syntax for the element being described.

(continued on next page)

Table 1 (Cont.) Documentation Conventions Table

Conventions	Meaning
{ }	Braces are used to enclose lists from which one item must be chosen. The items in such a list are delimited by the | character. For some user-defined operation headers, the braces are required syntax (this requirement is described in Chapter 3 of the *VAX APL User's Guide*).
n/a and n/s	These abbreviations indicate that something is Not Applicable or Not Supported in the context being discussed.
. . .	A horizontal ellipsis indicates that the preceding items can be repeated one or more times. A comma preceding the ellipsis indicates that successive items must be separated by commas.
. . .	A vertical ellipsis indicates that not all of the statements in an example or figure are shown.
`Color`	Color in examples shows user input.
<CR><LF>	The <CR><LF> symbol indicates the presence of a control sequence representing a Carriage Return and a Line Feed.
Ctrl/X	The Ctrl/X symbol indicates that you must press the key labeled Ctrl while you simultaneously press another key, for example, Ctrl/C, Ctrl/Y, Ctrl/O.
xxx	A symbol such as xxx indicates that you press a key on the terminal. For example, the Return symbol represents a single stroke of the Return key on a terminal.

Unless otherwise noted:

- All numeric values are represented in decimal notation.
- You terminate commands by pressing the Return key.
- All examples in the manual are executable, and comments beginning with the lamp (⍝) symbol are part of the examples; comments surrounded by parentheses are not part of the examples.

1

VAX APL Primitive Functions and Operators

VAX APL provides functions that allow you to perform various operations with arrays. These functions are termed primitive because they represent the basic capabilities of the language. You do not have to write programs to perform these operations; they are built in. That is, the APL interpreter already knows how to perform them.

Primitive functions may be classified by the characteristics of their arguments and results. One distinction is whether a function is scalar or mixed. The APL primitive scalar functions perform scalar (or scalar-like) operations; the APL primitive mixed functions perform mixed-rank operations.

Primitive functions are either monadic or dyadic. Monadic functions require only one argument, which is placed immediately to the right of the function. Dyadic functions require two arguments, one on either side of the function.

Primitive functions also have a domain and a range. The domain of a function is the permissible type, shape, and values of its argument arrays; the range is the permissible type, shape, and values of its result array.

In addition to describing the APL primitive functions, this chapter describes the APL primitive operators (operations that produce functions as results), and the specification function (a function used to associate values with identifiers).

APL also provides functions for system communication and for I/O. These are explained in Chapter 2, Chapter 3, and in Chapter 5 of the *VAX APL User's Guide*.

1.1 Primitive Scalar Functions

The primitive scalar functions include the arithmetic, relational, and logical functions that almost everyone is familiar with—addition, subtraction, equality, and, or, and so on—plus a few operations that are less familiar, such as residue and roll. These functions are called scalar functions because they take scalar arguments and return scalar results. For example:

```
      !3                ⍝FACTORIAL OF 3
6
      2 + 2
4
```

The primitive scalar functions are extended on an item-by-item basis when the argument array is not a scalar (the argument can be any shape, simple or enclosed). In effect, APL operates on a sequence of scalar arguments and returns one value for each argument. This process is known as scalar product. For example:

```
      !3 4 5            ⍝EACH ITEM IS TREATED AS A SCALAR
6 24 120
      4 9 + 3 12        ⍝EACH PAIR OF ITEMS IS ADDED
7 21
```

Here, APL applies the factorial (!) and addition (+) functions as if each item were a scalar argument. For factorial, each of the three items in the argument (a vector) returns a value. For addition, each corresponding pair of items is added. The results are just as if five statements had been entered as follows:

```
      !3
6
      !4
24
      !5
120
      4 + 3
7
      9 + 12
21
```

Monadic scalar functions take only one argument, which is placed immediately to the right of the function. The shape of the argument determines the shape of the result. For example, a scalar argument returns a scalar result, and a vector argument returns a vector result.

Dyadic scalar functions have two arguments that must conform to each other. They conform if one of the following is true:

- Their shapes match.
- At least one of the arguments is a singleton.

When the shapes match, the function is applied a number of times equal to the number of items in the arguments, and the resulting array has the same shape as the argument arrays.

Each item in the left argument array is associated with the item that has the same position in the right argument array, and the result is placed in that same position in the resulting array. For example:

```
      1 2 3 + 1 2 3     ⍝SHAPES OF BOTH ARGUMENTS CONFORM
2 4 6
                        ⍝SHAPES DO NOT CONFORM
      1 2 3 + 1 2 3 4
10 LENGTH ERROR
      1 2 3 + 1 2 3 4
      ^
```

When one of the arguments is a singleton, the shape of the result is the same as the shape of the nonsingleton argument. Again, the function is applied on an item-by-item basis, but either the right or left argument (whichever is the singleton) is the same each time the function is applied. For example:

```
      1 + 1 2 3          ⍝SINGLETON EXTENSION LEFT ARGUMENT
2 3 4
      4 5 6 + 2          ⍝SINGLETON EXTENSION RIGHT ARGUMENT
6 7 8
      ⎕ ← A ← 10 (15 18 (8 4) 21) 30
10 +--------------+ 30
   |15 18 +---+ 21|
   |      |8 4|   |
   |      +---+   |
   +--------------+
      5 + A
15 +---------------+ 35
   |20 23 +----+ 26|
   |      |13 9|   |
   |      +----+   |
   +---------------+
```

When both arguments are singletons, the shape of the result is the same as the shape of the argument with the higher rank. For example:

```
      B←(1 1 1 ρ2)        ⍝B IS A RANK 3 SINGLETON
      C←(1 1 ρ3)          ⍝C IS A RANK 2 SINGLETON
      D←B + C             ⍝SMALLER RANK WILL CONFORM TO LARGER
      D                   ⍝DISPLAY D, A SINGLETON OF SHAPE 1 1 1
5
```

The primitive scalar functions are pervasive functions; that is, their operations extend pervasively throughout the depth of enclosed arrays:

```
                           ⍝BOTH ARGUMENTS HAVE DEPTH = 3
      ⎕ ← A ← 10 (15 18 (8 4) 21) 30
10 +--------------+ 30
   |15 18 +---+ 21|
   |      |8 4|   |
   |      +---+   |
   +--------------+
      ⎕ ← B ← 5 (12 11 (3 3) 2) 25
5 +-------------+ 25
  |12 11 +---+ 2|
  |      |3 3|  |
  |      +---+  |
  +-------------+
      A - B
5 +------------+ 5
  |3 7 +---+ 19|
  |    |5 1|   |
  |    +---+   |
  +------------+
```

The conformance rules for the arguments of the primitive scalar functions are also pervasive; APL does a conformance check at each level of enclosed arrays. During the check, APL performs singleton extension when necessary. The following example uses the dyadic minimum function (⌊), which returns the smaller of two arguments:

```
      4 (5 3) ⌊ (2 6) 1
+---+ +---+
|2 4| |1 1|
+---+ +---+
```

In the preceding example, APL first pairs the corresponding items (through the process of scalar product). The pairs are `4⌊(2 6)` and `(5 3)⌊1`. Second, APL pairs the singleton argument with each element inside the enclosed arguments (through the process of singleton extension). These pairs are `((4⌊2) (4⌊6))` and `((5⌊1) (3⌊1))`. Finally, APL evaluates each pair of scalar arguments.

The following example shows two arguments that conform at the top level of their nesting, but do not conform at a lower level. This example uses the monadic enclose function (⊂), which encloses its argument, as well as the dyadic minimum function (⌊).

```
      4 (5 3) ⌊ ⊂ 2 6 1
10 LENGTH ERROR
      4 (5 3) ⌊ ⊂ 2 6 1
      ^
```

In the preceding example, APL first pairs the corresponding items. The pairs are `4⌊(2 6 1)` and `(5 3)⌊(2 6 1)`. Second, APL pairs the singleton argument with each element inside the enclosed argument. These pairs are `((4⌊2) (4⌊6) (4⌊1))`. Third, APL recognizes the length error in the pair of enclosed arguments (`5 3` and `2 6 1`) and signals the error. (If one of these enclosed arguments had been a singleton, APL would have applied singleton extension.)

Primitive scalar functions generally take numeric arguments. The argument domain for relational functions (≤, ≥, <, >, =, ≠), however, includes both character and numeric arguments. The equal (=) and not equal (≠) functions can take both character and numeric arguments in the same expression. The result domain for all primitive scalar functions is a scalar numeric array.

Primitive scalar functions return empty arrays when there is an empty argument (provided that APL does not detect an error before evaluating the result). For example:

```
      1+2+3+4+⍳0          ⍝⍳0 ALWAYS GENERATES AN EMPTY ARRAY
                                    (APL outputs a blank line)
      'A'<''
                                    (APL outputs a blank line)
                          ⍝ARGUMENT SHAPES DO NOT CONFORM
      (1 0 3⍴1) + ⍳5
  9 RANK ERROR
      (1 0 3⍴1) + ⍳5
      ^
```

You can specify an axis (`[K]`) with dyadic scalar functions. For example, this allows you to apply a vector to each row or each column of a matrix. The general form of axis is as follows: *Af*[*K*]*B*, where *A* and *B* are the arguments to *f* (a scalar function), and *K* is the axis argument. Note that *K* specifies the axes of subarrays constructed from whichever argument has the larger rank. The argument of smaller rank is combined with these subarrays.

For example, if you specify axes [1 3], then the shape of the subarrays of the larger rank argument is the lengths of that argument's first and third axes, and the smaller rank argument has the same shape as these subarrays. When APL combines the two arguments, it does so along the second axis of the larger rank argument of scalar extension. The length of the second axis in this case is the number of subarrays involved.

In all cases, the axis argument must be near-integer in the vector domain. The length of *K* must be equal to the smaller of the ranks of the arguments, and the values in *K* must be between the index origin and the larger of the ranks of the arguments (you cannot specify an axis that does not exist). The order of the items in the axis argument makes no difference; however, *K* may not contain duplicates. The arguments to the function *f* must conform by having their shapes match along the axes specified by *K*. The shape of the result is the same as the argument with larger rank.

For an enclosed argument, the application of the axis does not pervade, but works only at the top levels of nesting. See the following examples:

```
      A ← 10 100 1000     ⍝CREATE A
      ⎕ ← B ← 3 4⍴⍳12    ⍝CREATE AND DISPLAY B
1  2  3  4
5  6  7  8
9 10 11 12
      A +[1] B            ⍝A CONFORMS TO AXIS 1 OF B
  11   12   13   14
 105  106  107  108
1009 1010 1011 1012
      A ← 1 10 100 1000   ⍝CREATE NEW A
      A ×[2] B            ⍝A CONFORMS TO AXIS 2 OF B
1  20  300  4000
5  60  700  8000
9 100 1100 12000
      ⎕ ← A ← 2 3⍴0.1×⍳6 ⍝CREATE NEW A
0.1 0.2 0.3
0.4 0.5 0.6
      ⎕ ← B ← 2 4 3⍴⍳24   ⍝CREATE NEW B
 1  2  3
 4  5  6
 7  8  9
10 11 12

13 14 15
16 17 18
19 20 21
22 23 24
```

```
                         ⍝ORDER OF AXIS ARGUMENT UNIMPORTANT
      ⎕←Z←A+[3 1]B       ⍝A CONFORMS TO AXES 1 AND 3 OF B
 1.1  2.2  3.3
 4.1  5.2  6.3
 7.1  8.2  9.3
10.1 11.2 12.3

13.4 14.5 15.6
16.4 17.5 18.6
19.4 20.5 21.6
22.4 23.5 24.6
                         ⍝THE FOLLOWING SUBSCRIPTS DEMONSTRATE
                         ⍝ SUBARRAY COMBINATIONS USED BY APL
                         ⍝MATCH RETURNS 1 WHEN TRUE
      Z[;1;] ≡ A + B[;1;]
1
      Z[;2;] ≡ A + B[;2;]
1
      Z[;3;] ≡ A + B[;3;]
1
      Z[;4;] ≡ A + B[;4;]
1
      ⎕ ← A ← 2 4⍴0.1×⍳8 ⍝CREATE NEW A
0.1 0.2 0.3 0.4
0.5 0.6 0.7 0.8
      ⎕ ← Z ← A +[1 2] B ⍝A CONFORMS TO AXES 1 AND 2 OF B
 1.1  2.1  3.1
 4.2  5.2  6.2
 7.3  8.3  9.3
10.4 11.4 12.4

13.5 14.5 15.5
16.6 17.6 18.6
19.7 20.7 21.7
22.8 23.8 24.8
                         ⍝THE FOLLOWING SUBSCRIPTS DEMONSTRATE
                         ⍝ SUBARRAY COMBINATIONS USED BY APL
      Z[;;1] ≡ A + B[;;1]
1
      Z[;;2] ≡ A + B[;;2]
1
      Z[;;3] ≡ A + B[;;3]
1
      ⎕ ← A ← 4 3⍴0.1×⍳12 ⍝CREATE NEW A
0.1 0.2 0.3
0.4 0.5 0.6
0.7 0.8 0.9
1   1.1 1.2
```

```
      ⎕ ← Z ← A +[3 2] B  ⍝A CONFORMS TO AXES 2 AND 3 OF B
 1.1  2.2  3.3
 4.4  5.5  6.6
 7.7  8.8  9.9
11   12.1 13.2

13.1 14.2 15.3
16.4 17.5 18.6
19.7 20.8 21.9
23   24.1 25.2
                          ⍝THE FOLLOWING SUBSCRIPTS DEMONSTRATE
                          ⍝ SUBARRAY COMBINATIONS USED BY APL
      Z[1;;] ≡ A + B[1;;]
1
      Z[2;;] ≡ A + B[2;;]
1
      PC←2 3⍴(⍳3) (⍳3) (⍳3) (2 3⍴⍳6) (2 3⍴⍳6) (2 3⍴⍳6)
      PC
+-----+ +-----+ +-----+
|1 2 3| |1 2 3| |1 2 3|
+-----+ +-----+ +-----+
+-----+ +-----+ +-----+
|1 2 3| |1 2 3| |1 2 3|
|4 5 6| |4 5 6| |4 5 6|
+-----+ +-----+ +-----+
      PC+1
+-----+ +-----+ +-----+
|2 3 4| |2 3 4| |2 3 4|
+-----+ +-----+ +-----+
+-----+ +-----+ +-----+
|2 3 4| |2 3 4| |2 3 4|
|5 6 7| |5 6 7| |5 6 7|
+-----+ +-----+ +-----+
      PC + [2] 1 2 3
+-----+ +-----+ +-----+
|2 3 4| |3 4 5| |4 5 6|
+-----+ +-----+ +-----+
+-----+ +-----+ +-----+
|2 3 4| |3 4 5| |4 5 6|
|5 6 7| |6 7 8| |7 8 9|
+-----+ +-----+ +-----+
      ⎕←ER←1 2 (⍳3)
1 2 +-----+
    |1 2 3|
    +-----+
                       ⍝AXIS IS NOT PERVASIVE SO PLUS (+)
                       ⍝ WITH AXIS APPLIES BETWEEN
                       ⍝ ↑PC[2;3] AND ↑ER[3] WHICH
                       ⍝ IS A RANK ERROR
    PC+[2]ER
9 RANK ERROR
     PC+[2]ER
     ^
```

The individual descriptions of the primitive scalar functions are presented in three sections. Section 1.1.1 describes arithmetic functions, Section 1.1.2 describes logical functions, and Section 1.1.3 describes relational functions. Most of the individual descriptions include examples of how the functions work.

1.1.1 Arithmetic Functions

The arithmetic functions, which are summarized in Table 1–1, perform well-known mathematical operations. All of them take numeric scalar arguments and return numeric scalar results.

Table 1–1 Arithmetic Scalar Functions

Monadic		Dyadic	
Function	**Meaning**	**Function**	**Meaning**
+B	B	A+B	Add A to B
-B	Negative of B	A-B	Subtract B from A
×B	Sign of B	A×B	Multiply A and B
÷B	Reciprocal of B	A÷B	Divide A by B
⋆B	e to the B th power	A⋆B	A to the B th power
|B	Magnitude of B	A|B	A residue of B
⌈B	Ceiling of B	A⌈B	Maximum of A and B
⌊B	Floor of B	A⌊B	Minimum of A and B
⍟B	Natural logarithm of B	A⍟B	Logarithm of B to the base A
!B	Factorial of B	A!B	Binomial coefficient (number of combinations of B things taken A at a time)
○B	Pi times B	A○B	Trigonometric functions (B is in radians; see Table 1–2)
?B	Random integer from ⍳B		

1.1.1.1 + Conjugate

The monadic + function returns a result that is the same as its argument; thus, +B is identical to B. For example:

```
      +5
5
      +¯5
¯5
      +¯1 2 ¯3 4 ¯5
¯1 2 ¯3 4 ¯5
```

1.1.1.2 - Negative

The monadic - function returns the negative of its argument; thus -*B* is the negative of *B*. Be careful not to confuse the negative function with the high minus sign (¯) used to denote a negative number. For example:

```
      -6
¯6
      -1 2 3 ¯4
¯1 ¯2 ¯3 4
      -¯3
3
```

1.1.1.3 × Signum

The monadic × function identifies the sign of its argument; thus, ×*B* is the sign of *B*. The signum function returns ¯1 if the argument is less than 0, 1 if the argument is greater than 0, and 0 if the argument is equal to 0. For example:

```
      ×99
1
      ×0
0
      ×¯5
¯1
```

1.1.1.4 ÷ Reciprocal

The monadic ÷ function returns the reciprocal of its argument; thus, ÷*B* is the reciprocal of *B*. For example:

```
      ÷5
0.2
      ÷2
0.5
      ÷0
 15 DOMAIN ERROR (DIVISION BY ZERO)
      ÷0
       ^
```

1.1.1.5 ⋆ Exponential

The monadic ⋆ function raises the value of e (2.71828182845904523536...) to the power specified by its argument; thus, ⋆*B* is e to the *B* th power. For example:

```
      *0
1
      *1
2.718281828
      *10
22026.46579
      *50
5.184705529E21
```

1.1.1.6 ⍟ Natural Logarithm

The monadic ⍟ function returns the natural logarithm of its argument; thus, ⍟*B* is the natural logarithm (base e) of *B*. For example:

```
      ⍟1
0
      ⍟2.718281828459
1
      ⍟22026.46579
10
      ⍟5.184705529E21
50
```

The ⍟ symbol is formed with the ○ and * symbols.

1.1.1.7 ○ Pi Times

The monadic ○ function returns the product of its argument and the value of π (3.14159265358979323846264...). For example:

```
      ○1
3.141592654
      ○3
9.424777961
```

1.1.1.8 ⌊ Floor

The monadic ⌊ function returns the greatest integer not greater than its argument, within a tolerance defined by ⎕*CT*. For example:

```
      ⌊¯2.5
¯3
      ⌊4.111
4
      ⌊4.999
4
```

Note that the ⎕CT setting may affect the result of ⌊. For example:

```
      ⎕CT←0
      ⌊4.9999999999
4
      ⎕CT←1E¯10
      ⌊4.9999999999
5
```

The following is a formal description of how the floor function is implemented:

```
      ∇Z←FLOOR B ;⎕CT ;BXCT ;N
[1]   BXCT←⎕CT ⋄ ⎕CT←0
[2]   N←(×B)×⌊0.5 + |B
[3]   Z←N-(N-B)>BXCT × ↑⌈|N
[4]   ∇
```

1.1.1.9 ⌈ Ceiling

The monadic ⌈ function returns the smallest integer not less than its argument, within a tolerance defined by ⎕CT. For example:

```
      ⌈¯2.5
¯2
      ⌈4.111
5
      ⌈4.999
5
```

Note that the ⎕CT setting may affect the result of ⌈. For example:

```
      ⎕CT←0
      ⌈4.0000000001
5
      ⎕CT←1E¯10
      ⌈4.0000000001
4
```

The ⌈ and ⌊ functions are related in the following manner: ⌈B←→-⌊-B. For example:

```
      ⌈4.111
5
      -⌈-4.111
4
```

1.1.1.10 | Magnitude

The monadic | function returns the absolute value of its argument; thus, |B is the absolute value of B (that is, B=|B, if B≥0 and (-B)=|B, if B<0). For example:

```
       |9
9
       |¯9
9
```

1.1.1.11 ! Factorial

The ! of B (for integer arguments) is the product of the first B positive integers. For example:

```
       !2
2
       !3
6
       !5
120
```

If the argument to the factorial function is 0, the result is 1. If the argument is a negative integer, APL signals *DOMAIN ERROR*. If the argument is not an integer, !B is defined in terms of the mathematical function *GAMMA* as follows:

```
!B ←→ GAMMA(B+1)
```

The ! symbol is formed with the quote (') and period (.) symbols.

For more information on the Gamma function, see Milton Abramowitz and Irene A. Stegun, eds., *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables* (National Bureau of Standards, November 1964), pp. 255-293; or John F. Hart, et al., *Computer Approximations* (Robert E. Krieger Publishing Company, 1978), pp. 130-136, 243-254.

1.1.1.12 ? Roll

When applied to an argument B, the monadic ? function generates an integer randomly selected from the integers ⍳B (for a near-integer argument). For example:

```
      ?5 10 15 20 25
4 7 8 6 25
      ?5 10 15 20 25
2 9 3 2 10
      A←2 3⍴⍳ 6
      A
1 2 3
4 5 6
      ?A
1 2 2
3 5 5
      ?A
1 1 1
3 3 6
```

At the completion of the roll function, the value of ⎕RL changes:

```
      ⎕RL
695197565
      ?5
4
      ⎕RL
¯1133752294
```

If the argument is not a near-integer, or if a near-integer argument is less than the value of ⎕IO, APL signals *DOMAIN ERROR*:

```
      ⎕IO←1
      ?0
 15 DOMAIN ERROR
       ?0
       ^
      ⎕IO←0
      ?0
0
```

Note that the roll function is ⎕IO-dependent: ?B when ⎕IO is 1, is equivalent to (for the same value of ⎕RL) 1+ ?B when ⎕IO is 0.

The roll function is analogous to the rolling of several dice. Roll may generate duplicate values; thus, it differs from the dyadic deal function (?), which generates a set of unique random numbers.

1.1.1.13 +, -, ×, and ÷ Addition, Subtraction, Multiplication, and Division

The dyadic +, -, ×, and ÷ functions return the sum, difference, product, and quotient of their arguments, respectively.

The right argument for the division function may not be 0 unless the left argument is also 0. For example:

```
      0÷0
1
```

1.1.1.14 * Power

The dyadic * function raises the value of its left argument to the power specified by its right argument. For example:

```
      5*3
125
      ¯5*3
¯125
      3*2.5
15.58845727
      ¯3*2.5
 15 DOMAIN ERROR
       ¯3*2.5
       ^
```

The power function's domain is restricted to the following combinations of arguments:

Left	Right
Any	0
0	≥ 0
> 0	Any
< 0	Integer

Note that 0*0 is 1.

If the right argument of the * function is exactly 0.5, APL returns the square root of the left argument.

1.1.1.15 ⍟ Logarithm

The dyadic ⍟ function returns the logarithm of its right argument in the base of its left argument; thus, *A*⍟*B* is the logarithm of *B* in base *A*. For example:

```
      10⍟1
0
      10⍟10
1
      5⍟10
1.430676558
```

Both arguments must be greater than zero. The left argument may not be 1 unless the right argument is also 1. For example: 1⍟1 is 1.

The ⍟ symbol is formed with the ○ and ⋆ symbols.

1.1.1.16 ○ Circle

You use the dyadic ○ function to perform trigonometric functions.

The left argument of ○ specifies which trigonometric function is to be performed. Only certain combinations of arguments are valid for the circle function. For arguments *A* and *B*, Table 1–2 lists the possible values of *A* (near-integer argument), and indicates the operation associated with each value.

Table 1–2 Trigonometric Functions Performed by ○

A[1]	**Function** (*Z*←*A*○*B*)[2]	**Domain**	**Result Domain**
¯7	arc tanh*B*	1≥|*B*	
¯6	arc cosh *B*	*B*≥1	*Z*≥0
¯5	arc sinh*B*		
¯4	(¯1+*B*⋆)⋆0.5	1≤|*B*	*Z*≥0
¯3	arc tan *B*		(|*Z*)≤○0.5
¯2	arc cos *B*	1≥|*B*	(0≤*Z*)∧*Z*≤○1
¯1	arc sin *B*	1≥|*B*	(|*Z*)≤○0.5
0	(1-*B*⋆2)⋆0.5	1≥|*B*	(*Z*≥0)∧*Z*≤1
1	sin *B*		(|*Z*)≤1

[1] The value of *A* must be a near-integer from ¯7 through 7.

[2] The value of *B* is given in radians.

(continued on next page)

Table 1–2 (Cont.) Trigonometric Functions Performed by ○

A[1]	Function (*Z*←*A*○*B*)[2]	Domain	Result Domain
2	cos *B*		(|*Z*)≤1
3	tan*B*	*B*≠2|*B*÷○0.5	
4	(1+*B**2)*0.5		*Z*≥1
5	sinh *B*		
6	cosh *B*		*Z*≥1
7	tanh *B*		(|*Z*)≤1

[1] The value of *A* must be a near-integer from ¯7 through 7 .

[2] The value of *B* is given in radians.

1.1.1.17 ⌊ Minimum

The dyadic ⌊ function returns the smaller of its two arguments. For example:

```
      4⌊5
4
      4 5 3⌊1 2 6
1 2 3
```

1.1.1.18 ⌈ Maximum

The dyadic ⌈ function returns the greater of its two arguments. For example:

```
      4⌈5
5
      4 5 3⌈1 2 6
4 5 6
```

1.1.1.19 | Residue

The dyadic | function returns the residue of the right argument with respect to the left argument. The residue is obtained by adding or subtracting multiples of the left argument from the right argument. The result of a residue operation takes the sign of the left argument.

If the left and right arguments are equal, the residue is 0 . (Note that the residue function is ⎕*CT*-dependent.) If the left argument is 0 , then the residue equals the value of the right argument. If the left argument is not 0 , then the residue is in the range of the left argument through 0 ; it may equal 0 but may not equal the value of the left argument. For example:

```
      5|8
3
      ¯5|7
¯3
      7|7
0
      7|0
0
      0|7
7
      2|5.8
1.8
      1|123.4567
0.4567
      5 5|8 8
3 3
      5 5 5|2
2 2 2
      5|2 2 2
2 2 2
      A←3 0 ¯3
      B←¯6 ¯5 ¯4 ¯3 ¯2 ¯1 0 1 2 3 4 5 6
      A∘.|B
 0  1  2  0  1  2 0  1  2 0  1  2 0
¯6 ¯5 ¯4 ¯3 ¯2 ¯1 0  1  2 3  4  5 6
 0 ¯2 ¯1  0 ¯2 ¯1 0 ¯2 ¯1 0 ¯2 ¯1 0
```

1.1.1.20 ! Combinations

For arguments A and B, the dyadic ! function returns the number of combinations of B elements taken A at a time. For example:

```
      2!4
6
      10!10
1
```

For arguments A and B, the function's domain is described as follows:

∼(B<0)∧(∼ INTEGER B)∧∼ INTEGER A

INTEGER is a function that returns 1 if all the items in its argument are integers, and 0 otherwise.

APL determines the result of the dyadic ! function based on the algorithms explained in Table 1–3. The value 1 in the table for A, B, or B-A means that the argument or the difference between the arguments is a negative integer; the value 0 means that the argument or the difference between them is not a negative integer.

Table 1–3 Determining Result for Dyadic !

A	*B*	*B*-*A*	Result
0	0	0	`(!B)÷(!A)×!B-A`
0	0	1	0
0	1	0	APL signals `DOMAIN ERROR`
0	1	1	`(¯1*A)×A!A-B+1`
1	0	0	0
1	0	1	Not a possible case
1	1	0	`(¯1*B-A)×(|B+1)!(|A+1)`
1	1	1	0

Note that the dyadic ! function is related to the mathematical function BETA as follows:

```
BETA(A,B)←→ ÷B× (A-1)!A+B-1 ←→ ÷A× (B-1)!A+B-1
```

The ! symbol is formed with the ' and . symbols.

For more information on the Beta function, see Milton Abramowitz and Irene A. Stegun, eds., *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables* (National Bureau of Standards, November 1964), pp. 255-293; or John F. Hart, et al., *Computer Approximations* (Robert E. Krieger Publishing Company, 1978), pp. 130-136, 243-254.

1.1.2 Logical Functions

The monadic ~ (Not) and the dyadic ∧, ∨, ⍲, and ⍱ functions (And, Or, Nand, Nor, respectively) are commonly called logical functions. The domain and range of logical functions are restricted to the Boolean values 0 and 1.

Table 1–4 is a truth table that shows the results of logical operations for arguments *A* and *B*.

Table 1–4 Truth Table for Logical Functions

Arguments		Functions				
		And	**Or**	**Nand**	**Nor**	**Not**
A	*B*	*A*∧*B*	*A*∨*B*	*A*⍲*B*	*A*⍱*B*	~*B*
A	*B*	*A*∧*B*	*A*∨*B*	*A*⍲*B*	*A*⍱*B*	~*B*
0	0	0	0	1	1	—
0	1	0	1	1	0	—
1	0	0	1	1	0	—
1	1	1	1	0	0	—
—	0	—	—	—	—	1
—	1	—	—	—	—	0

The ⍲ symbol is formed with the ∧ and ~ symbols. The ⍱ symbol is formed with the ∨ and ~ symbols.

1.1.3 Relational Functions

The dyadic <, ≤, =, ≠, >, and ≥ functions are commonly called relational functions. The domain of relational functions is not restricted; they can take both numeric and character arguments. However, only the equal and not equal functions can have mismatched arguments, that is, one numeric and one character argument simultaneously. For example:

```
      'A'=5
0
      '5'=5
0
```

The result domain of relational functions is restricted to the Boolean values 0 and 1. A relational function returns the result 1 if true and 0 if false. For example:

```
      9>6
1
      4>6
0
      'C'>'A'
1
```

When <, ≤, ≥, or > have character arguments, the order of characters in ⎕AV is used as a collating sequence, and the evaluation is based on the respective positions of the arguments. When the relational functions have numeric arguments, the comparisons between the arguments are affected by the value of ⎕CT.

When you use relational functions with Boolean arguments, the relational functions can perform logical operations. For example, the not equal (≠) function performs an exclusive OR operation if its arguments are 0 s and 1 s:

```
      (0≠0),(0≠1),(1≠0),1≠1
0 1 1 0
```

1.2 Primitive Mixed Functions

The primitive mixed functions allow more extensive array manipulation than the scalar functions. Scalar functions take scalar arguments, return scalar results, and are extended to arrays on an item-by-item basis. Mixed functions are not as predictable. For example, depending on the values of their arguments, mixed functions may do the following:

- Take a scalar argument and return a vector result:

```
      ⍳9
1 2 3 4 5 6 7 8 9
```

- Take a vector argument and return a scalar result:

```
      ⍎'1234'
1234
```

- Take a matrix argument and return a vector result:

```
      ⎕←B←4 3⍴⍳12              ⍝CREATE AND DISPLAY B
 1  2  3
 4  5  6
 7  8  9
10 11 12
      ,B                        ⍝RAVEL B (MAKE B A VECTOR)
1 2 3 4 5 6 7 8 9 10 11 12
```

Table 1–5 summarizes the primitive mixed functions, which are described in this section.

Table 1–5 Primitive Mixed Functions

Function	Name	Meaning
A⊥*B*	Base	Bases the representation of *B* in number system *A*.
→*B*	Branch	Modifies the standard order of execution in a user-defined operation.
A,*B*	Catenate	Catenates *A* to *B* along the last axis of *A*.
A,[*K*]*B* *A*⍪[*K*]*B*	Catenate/ Laminate	Catenates/laminates *A* to *B* along the *K* th axis of *A*.
A⍪*B*	Catenate	Catenates *A* to *B* along the first axis of *A*.
A⊇*B*	Contain	Determines whether all the items in array *B* are also found in array *A*.
A?*B*	Deal	Deals *A* integers selected randomly in the range ⍳ *B*.
⊃*B*	Disclose	Reduces the depth in an array.
⊃[*K*]*B*	Disclose	Discloses *B* and arranges the substructure axes (*K*).
A↓*B*	Drop	For *A*>0, drops the first *A* items of *B*;for *A*<0, drops the last | *A* items of *B*.

(continued on next page)

Table 1–5 (Cont.) Primitive Mixed Functions

Function	Name	Meaning
A↓ [*K*]*B*	Drop	For *A*> 0, drops the first *A* items along the axes of *B* specified by *K*; for *A*< 0, drops the last | *A* items along the axes of *B* specified by *K*.
⊂*B*	Enclose	Builds enclosed arrays. Returns a scalar containing *B*.
⊂ [*K*]*B*	Enclose	Builds enclosed arrays; subarrays along axes *K* become scalars.
∊*B*	Enlist	Builds a simple vector with all of the simple scalars in its argument.
⍎*B*	Execute	Executes the character string *B*.
⍇*B* ⍇ [*K*]*B*	File Input	Reads records from an external file into an APL workspace.
A⍈*B* *A*⍈ [*K*]*B* ⍈ [*K*]*B* ⍈*B*	File Output	Writes information to an external file from an APL workspace.
⍕*B*	Format	Formats array *B*.
A⍕*B*	Format	Formats character array *B* with width and precision specified by *A*.
⍒*B*	Grade Down	Generates an index vector that can be used to sort *B* in descending order.
⍒ [*K*]*B*	Grade Down	Generates an index vector that can be used to sort *B* in descending order, row by row or column by column.
A⍒*B*	Grade Down	Generates an index vector that can be used to sort *B* in descending order using collating sequence *A*.
⍋*B*	Grade Up	Generates an index vector that can be used to sort *B* in ascending order.
⍋ [*K*]*B*	Grade Up	Generates an index vector that can be used to sort *B* in ascending order, row by row or column by column.
A⍋*B*	Grade Up	Generates an index vector that can be used to sort *B* in ascending order using collating sequence *A*.
⍳*B*	Index Generator	Generates the first *B* consecutive integers from the current index origin.
A⍳*B*	Index Of	Finds the first occurrence of *B* in vector *A*.

(continued on next page)

Table 1–5 (Cont.) Primitive Mixed Functions

Function	Name	Meaning
A∩B	Intersection	Returns a vector of the common items in the arrays A and B.
A≡B	Match	Determines whether arrays A and B are identical in rank, shape, and value.
A⌹B	Matrix Divide	Performs matrix division, solves linear equations, and finds a least-squares solution.
⌹B	Matrix Inverse	Inverts the matrix B.
A∊B	Membership	Determines if A is a member of array B.
A⊃B	Pick	Discloses an item from any depth of an array.
,B	Ravel	Returns the ravel of B (makes B a vector).
,[K]B ⍪[K]B	Ravel	Merges or adds axes to the shape of B depending on the value of K.
A⊤B	Represent	Represents B in number system A.
A⍴B	Reshape	Reshapes B to the shape specified by A.
⌽B	Reverse	Reverses along the last axis of B.
⌽[K]B ⊖[K]B	Reverse	Reverses along the K th axis of B.
⊖B	Reverse	Reverses along the first axis of B.
A⌽B	Rotate	Rotates by A along the last axis of B.
A⌽[K]B A⊖[K]B	Rotate	Rotates by A along the K th axis of B.
A⊖B	Rotate	Rotates by A along the first axis of B.
⍴B	Shape	Returns the shape of B.
A⊆B	Subset	Determines whether all the items in array A are also found in array B.
A↑B	Take	For A>0, takes the first A items of B; forA<0, takes the last ∣A items of B.
A↑[K]B	Take	For A>0, takes the first A items along the axes of B specified by K; for A<0, takes the last ∣A items along the axes of B specified by K.
⍉B	Transpose	Transposes the axes of B (for a matrix, exchanges the rows and columns).

(continued on next page)

Table 1–5 (Cont.) Primitive Mixed Functions

Function	Name	Meaning
A⍉B	Transpose	Transposes the axes of array B according to A.
A∪B	Union	Returns a vector of the items in the arrays A and B.
∪B	Unique	Removes the duplicate items of array B.
A~B	Without	Returns a vector of the items of array A that are not found in array B.

⊥ Base

Form

A⊥*B*

Left Argument Domain

Type	Numeric
Shape	Any
Depth	0 or 1 (simple)

Right Argument Domain

Type	Numeric
Shape	Any
Depth	0 or 1 (simple)

Result Domain

Type	Numeric
Rank	0⌈¯2+(ρρ*A*)+ρρ*B*
Shape	(¯1↓ρ*A*),1↓ρ*B*
Depth	0 or 1 (simple)

Implicit Arguments

None.

Description

The dyadic ⊥ function (known as base or decode) reduces a representation in a number system to a value. More specifically, it converts to decimal those vectors along the first axis of the right argument that are expressed in the positional number bases of radices given by vectors along the last axis of the left argument.

The base function is best explained as the converse of the represent function (⊤). The following example shows the two functions operating on a quantity expressed in yards, feet, and inches:

```
                                        ⍝1 YARD, 2 FEET, 3 INCHES IS 63 INCHES
      1760 3 12⊥1 2 3
63
      1760 3 12⊤63
1 2 3
```

The expressions *A*⊤*B* and *A*⊥*B* differ only in the value included in *B*; *A* expresses the number base in both cases.

The number of items in both arguments, for example *A* and *B*, must generally be the same; the first item in *A* expresses the radix in which the first item in *B* is decoded, and so on. However, if *A* is a singleton, it is extended so that its length is the same as that of the first axis of *B*. For example, the following expression has the effect of producing the base 10 value of the base 8 number 3777 (octal-to-decimal conversion):

```
      8⊥3 7 7 7
2047
```

For arguments *A* and *B*, the argument arrays for ⊥ must conform to one of the following rules:

- *A* or *B* is a scalar.
- The results of ¯1↑⍴*A* and 1↑⍴*B* are equal.
- Either ¯1↑⍴*A* or 1↑⍴*B* equals 1.

If the argument arrays conform to the last rule, the axis that equals 1 is extended to match the appropriate other axis. For example:

```
      (2 3⍴5)⊥(3 4⍴3)
93 93 93 93
93 93 93 93
      (2 3⍴5)⊥(1 4⍴3)
93 93 93 93
93 93 93 93
      (2 1⍴5)⊥(3 4⍴3)
93 93 93 93
93 93 93 93
```

The following are some other uses of the base function:

```
                         ⍝CONVERT 3 YDS. 2 FT. 4 IN. TO INCHES
      1 3 12 ⊥ 3 2 4
136
                 ⍝DETERMINE IF 2.5 IS A ZERO OF THE POLYNOMIAL
                         ⍝ ((6×X*2)-(7×X))-20
      2.5⊥6 ¯7 ¯20
0
                         ⍝YES
                         ⍝BASE 10 EQUIVALENT OF BASE 5 NUMBER
      5⊥4 3 4
119
```

You can use the base function to evaluate polynomials; the expression X⊥C evaluates a polynomial in X with coefficients given by the vector C.

For vectors A and B, the base function can be thought of as a form of the inner product operator. The expression A⊥B is equal to W+.×B, where W is the weighting vector (W←⌽×\¯1↓⌽A,1) given by the expression W[ρA] ←→ 1, and W[(-N)+ρA] is equal to A[(-N)+1ρA]×W[(-N)+1+ρA]. The value of A[1] is irrelevant. The following example shows two equivalent operations:

```
      A←1760 3 12
      B←1 2 3
      A⊥B
63
      36 12 1+.×B
63
```

Note that if the right argument is empty, the type of the left argument is not significant:

```
      (3 2 1ρ'A')⊥ ''
0 0
0 0
0 0
      (3 2 1ρ0)⊥ ⍳0
0 0
0 0
0 0
```

Possible Errors Generated

10 *LENGTH ERROR* (*LENGTHS OF INNER AXES DO NOT MATCH*)

15 *DOMAIN ERROR* (*ENCLOSED ARRAY NOT ALLOWED*)

15 *DOMAIN ERROR* (*INCORRECT TYPE*)

27 *LIMIT ERROR* (*FLOATING OVERFLOW*)

27 *LIMIT ERROR* (*VOLUME TOO LARGE*)

→ Branch

Form

→*B*

Argument Domain

Type	Near-Integer
Shape	Any
Depth	0 or 1 (simple)

Result Domain

None.

Implicit Arguments

None.

Description

The monadic → function (known as branch) modifies the standard order of execution in a user-defined operation.

Normally, APL lines in operations are executed in the order of their line numbers; execution begins at the first line following the operation header and ends with the last line in the operation. Branch changes the sequence of execution by transferring control to another line in the operation.

There are two types of branches: unconditional and conditional. Unconditional branches specify the next line to be executed. The result of an expression evaluation determines the next statement in a conditional branch.

Unconditional branches consist of a branch symbol (→), followed by a representation of the number of the operation line to which you want to transfer control. The argument can be a label, a constant, a variable, or an expression. Its value (or, if it is a vector, the value of its first item) is equivalent to an integer line number within the current definition. Execution continues at that line.

Conditional branches can be expressed in one of the following three forms:

- → *line-number* × ⍳ *logical-expression*

  Here APL evaluates the logical expression that is the right argument of ⍳ . The logical expression returns either a `1` (true) and the control passes to the specified line or a `0` (false) and the control passes to the next statement. (This form only works when `⎕IO←0`.) In the following example a simple counter controls the number of times the statements in a loop are executed. The example branches to line number 0, an out-of-range number, and forces an exit from the operation:

```
      ∇ COUNTER
[1]   ⎕←'NUMBER OF ENTRIES:' ⋄ N←⎕
[2]   C←0
[3]   LOOP: →0×⍳C=N
[4]   C←C+1
[5]   →LOOP
[6]   ∇
```

- →*logical-expression* / *line-numbers*

  This type of conditional branch specifies several line numbers and associated logical expressions as possible branch destinations. Control passes to the line number corresponding to the first logical expression that evaluates to 1 (true). For example:

```
      ∇F A
[1]   →(A>0)/3
[2]   'WILL NOT ACCEPT NEGATIVE NUMBERS' ⋄ →0
[3]   'FUNCTION CONTINUING NORMALLY'
[4]   ∇
      F 5
FUNCTION CONTINUING NORMALLY
      F ¯2
WILL NOT ACCEPT NEGATIVE NUMBERS
```

- *→line-numbers* [K]

  Here the value of K is used as an index to select the corresponding line number. For example:

```
      ∇ labs
[1]   K←2
[2]   →(LAB1,LAB2,LAB3)[K]
[3]   LAB1: 'LAB1 IS EXECUTED' ◊ →0
[4]   LAB2: 'LAB2 IS EXECUTED' ◊ →0
[5]   LAB3: 'LAB3 IS EXECUTED' ◊ →0
[6]   ∇
      LABS
LAB2 IS EXECUTED
```

Note that → is described in greater detail in Chapter 3 of the *VAX APL User's Guide* along with other information on user-defined operations.

Possible Errors Generated

```
7  SYNTAX ERROR (BRANCH NOT ALLOWED IN MIDDLE OF AN EXPRESSION)

11  VALUE ERROR (BRANCH HAS NO RESULT)

15  DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

15  DOMAIN ERROR (INCORRECT TYPE)

15  DOMAIN ERROR (NOT AN INTEGER)

27  LIMIT ERROR (INTEGER TOO LARGE)
```

, and ⍪ Catenate/Laminate

Form

`A,B     A,[K]B     A⍪B     A⍪[K]B`

⍪ is formed with , and -

Left Argument Domain

Type	Any
Shape	–
Depth	Any

Right Argument Domain

Type	Any
Shape	–
Depth	Any

Result Domain

Type	–
Rank	1⌈(⍴⍴A)⌈⍴⍴B (for catenate) or 1+(⍴⍴A)⌈⍴⍴B (for laminate)
Shape	–
Depth	(≡A)⌈≡B

Implicit Arguments

None.

Description

The dyadic APL function joins together the specified axis of two arrays. If, for A,[K]B or A⍪[K]B, K is a near-integer, the function is called catenation, and A and B are joined along the K th axis. If K is not a near-integer, the function is called lamination, and A and B are joined along a new axis lying between the axes named by ⌊K and ⌈K. The forms A,B and A⍪B represent catenation and join the arrays along their last or first axis, respectively.

If one of the arguments is a scalar, its length is extended to match the shape of the other argument. If both arguments are scalars, the result is a two-item vector. For example:

```
      5,6            ⍝CATENATE 2 SCALARS, RESULT IS A VECTOR
5 6
      B←2 3⍴⍳6
      B,[1]7            ⍝CATENATE SCALAR TO FIRST AXIS OF B
1 2 3
4 5 6
7 7 7
      B,7            ⍝CATENATE SCALAR TO LAST AXIS OF B
1 2 3 7
4 5 6 7
```

For catenation, the arguments' ranks must differ after scalar extension by at most 1. Note that a singleton argument is not extended to conform to the other argument:

```
                          ⍝CATENATE SINGLETONS OF DIFFERENT RANKS
      (1 1⍴7),1 1 1 1⍴8
  9 RANK ERROR (RANKS DIFFER BY MORE THAN ONE)
       (1 1⍴7),1 1 1 1⍴8
       ∧
                          ⍝TRY AGAIN
      ⎕ ← R ← (1 1⍴7), 1 1 1⍴8
7 8
      ⍴R
1 1 2
```

In the following example, two arrays of equal rank are catenated. The shapes of the arguments match except for the axis `[(K)]` along which the arrays are being joined:

```
      B←3 4 5⍴12
      ⍴B
3 4 5
      C←3 6 5⍴14
      ⍴C
3 6 5
      R←B,[2]C
      ⍴R
3 10 5
```

Note that `B` is equal to `R[;⍳4;]` and `C` to `R[;4+⍳6;]`.

The next example shows the catenation of two arrays whose ranks differ by 1. Again, the shapes of the arguments match except for the axis along which the arrays are being joined:

```
      B←3 4 5⍴12
      ⍴B                    ⍝B IS RANK 3
3 4 5
      C←4 5⍴33
      ⍴C                    ⍝C IS RANK 2
4 5
      R←B,[1]C              ⍝CATENATE ALONG FIRST AXIS OF B
      ⍴R
4 4 5
                            ⍝ATTEMPT TO CATENATE ALONG SECOND AXIS
      B,[2]C
 10 LENGTH ERROR (SHAPES OFF AXIS DO NOT MATCH)
       B,[2]C
       ^
```

Here, `B` is equal to `R[⍳3;;]` and `C` to `R[4;;]`.

The following are more examples of catenation:

```
      A←5 8 9
      B←6 7
      A,B                      ⍝CATENATE TWO VECTORS
5 8 9 6 7
      10,A,B,12
10 5 8 9 6 7 12
      'NAME','XY'
NAMEXY
      B←2 3⍴1 2 3 4 5 6        ⍝CREATE B
      C←2 3⍴7 8 9 10 11 12     ⍝CREATE C
      B
1 2 3
4 5 6
      C
 7  8  9
10 11 12
      B,[1]C                   ⍝CATENATE ALONG FIRST AXIS
 1  2  3
 4  5  6
 7  8  9
10 11 12
      B,[2]C                   ⍝CATENATE ALONG SECOND AXIS
1 2 3  7  8  9
4 5 6 10 11 12
```

```
      B⍪C                       ⍝USE COLUMN CATENATE
 1  2  3
 4  5  6
 7  8  9
10 11 12
      B,C                       ⍝CATENATE ALONG SECOND AXIS
1 2 3  7  8  9
4 5 6 10 11 12
      ⎕←A←2 3 3⍴'ABCDEFGHIJKLMNOPQR'
ABC
DEF
GHI

JKL
MNO
PQR
      ⎕←B←2 3 3⍴'SSSTTTUUUVVVWWWXXX'
SSS
TTT
UUU

VVV
WWW
XXX
      A,B                       ⍝CATENATE RANK 2 OBJECTS
ABCSSS
DEFTTT
GHIUUU

JKLVVV
MNOWWW
PQRXXX
```

Note that the catenation of scalars produces a vector:

```
      ⍴⍴4,5
1
```

For lamination (`A,[K]B` and `A⍪[K]B` where `K` is not a near-integer), the arguments must have the same ranks and shapes after singleton extension. The following are examples of lamination:

```
                         ⍝CREATE NEW DIMENSION BEFORE THE FIRST
                         ⍝ DIMENSION WHEN [K]<1
      ⎕←X←'ABC',[0.5]'DEF'   ⍝ADD A ROW
ABC
DEF
      ⍴X
2 3
```

```
                          ⍝CREATE NEW DIMENSION AFTER THE FIRST
                          ⍝ DIMENSION WHEN 1<[K]<2
      ⎕←X←'ABC',[1.3]'DEF' ⍝ ADD A COLUMN
AD
BE
CF
      ⍴X
3 2
                          ⍝NOW TRY EXAMPLE WITH HIGHER RANK OBJECTS
                          ⍝NOTE THAT APL RESHAPES EACH ARGUMENT
                          ⍝BEFORE JOINING
      ⎕←E←3 2⍴'ABCDEF'
AB
CD
EF
      ⎕←F←3 2⍴'UVWXYZ'
UV
WX
YZ
                          ⍝CREATE NEW DIMENSION BEFORE THE FIRST
                          ⍝ DIMENSION WHEN [K]<1
      ⎕←R←E,[.2]F         ⍝ADD A PLANE
AB
CD
EF

UV
WX
YZ
      ⍴R
2 3 2
                          ⍝CREATE NEW DIMENSION AFTER THE FIRST
                          ⍝ DIMENSION WHEN 1<[K]<2
                          ⍝ADD A ROW, PREVIOUS ROWS BECOME PLANES
      ⎕←R←E,[1.9]F
AB
UV

CD
WX

EF
YZ
      ⍴R
3 2 2
```

```
                          ⍝CREATE NEW DIMENSION AFTER THE SECOND
                          ⍝ DIMENSION WHEN 2<[K]<3
                     ⍝ADD A COLUMN, PREVIOUS COLUMNS BECOME ROWS
      ⎕←R←E,[2.3]F
AU
BV

CW
DX

EY
FZ
      ⍴R
3 2 2
                          ⍝TRY EXAMPLE USING SINGLETON EXTENSION
      ⎕←R←E,[.5]'Z'       ⍝ADD A PLANE
AB
CD
EF

ZZ
ZZ
ZZ
      ⍴R
2 3 2
                          ⍝ADD A ROW, PREVIOUS ROWS BECOME PLANES
      ⎕←R←E,[1.5]'X'
AB
XX

CD
XX

EF
XX
      ⍴R
3 2 2
      'Y',[2.5]E              ⍝ADD A COLUMN
YA
YB

YC
YD

YE
YF
```

Note that if ⎕IO ←→ 0, then ¯.5 is valid as the axis value for lamination. This is the only case in which an axis may take a negative argument (range: ¯1<K).

Further examples:

```
      ⎕←A←(0 ('AB'))
0 +--+
  |AB|
  +--+
      ⎕←B←⊂,4
+-+
|4|
+-+
      ⎕←C←''
                                     (APL outputs a blank line)
      ⎕←MAX←2 3 ⍴ A 1 0 'AB' B C
+------+ 1     0
|0 +--+|
|  |AB||
|  +--+|
+------+
+--+     +---+ +-+
|AB|     |+-+| | |
+--+     ||4|| +-+
         |+-+|
         +---+
      MAX,[1]B                 ⍝CATENATE ALONG FIRST AXIS
+------+ 1     0
|0 +--+|
|  |AB||
|  +--+|
+------+
+--+     +---+ +-+
|AB|     |+-+| | |
+--+     ||4|| +-+
         |+-+|
         +---+
+-+      +-+   +-+
|4|      |4|   |4|
+-+      +-+   +-+
      ⎕←D←0 'DP'
0 +--+
  |DP|
  +--+
      D,[2]MAX                ⍝CATENATE ALONG SECOND AXIS
0    +------+ 1     0
     |0 +--+|
     |  |AB||
     |  +--+|
     +------+
+--+ +--+     +---+ +-+
|DP| |AB|     |+-+| | |
+--+ +--+     ||4|| +-+
              |+-+|
              +---+
        ⍝SHOW CATENATION OF TWO ARRAYS WHOSE RANKS DIFFER BY 1
      ⍴MAX
2 3
      ⎕←VIC←B,D
+-+ 0 +--+
|4|   |DP|
+-+   +--+
      ⍴VIC
3
      ⎕←W←MAX,[1]VIC
+------+ 1     0
|0 +--+|
|  |AB||
|  +--+|
+------+
+--+     +---+ +-+
|AB|     |+-+| | |
+--+     ||4|| +-+
         |+-+|
         +---+
+-+      0     +--+
|4|            |DP|
+-+            +--+
      ⍴W
3 3
                          ⍝SHOW LAMINATION
      ⎕←X←4,⊂,1
4 +-+
  |1|
  +-+
      ⍴X
2
      B
+-+
|4|
+-+
      ⍴B
                                (APL outputs a blank line)
      ⎕←Y←X,[0.5]B
4   +-+
    |1|
    +-+
+-+ +-+
|4| |4|
+-+ +-+
      ρY
2 2
      ⎕←Z←X,[1.9]B
4   +-+
    |4|
    +-+
+-+ +-+
|1| |4|
+-+ +-+
      ρZ
2 2
```

Possible Errors Generated

```
9  RANK ERROR

9  RANK ERROR (RANKS DIFFER BY MORE THAN ONE)

10  LENGTH ERROR (SHAPES OFF AXIS DO NOT MATCH)

15  DOMAIN ERROR (INCORRECT TYPE)

27  LIMIT ERROR (INTEGER TOO LARGE)

28  AXIS RANK ERROR (NOT VECTOR DOMAIN)

30  AXIS DOMAIN ERROR (ARGUMENT RANK AND AXIS INCOMPATIBLE)

30  AXIS DOMAIN ERROR (AXIS LESS THAN INDEX ORIGIN)

30  AXIS DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

30  AXIS DOMAIN ERROR (INCORRECT TYPE)

29  AXIS LENGTH ERROR (NOT SINGLETON)

30  AXIS DOMAIN ERROR (SEMICOLON LIST NOT ALLOWED)
```

⊇ Contains

Form

A⊇*B*
⊇ is formed with ⊃ and _

Left Argument Domain

Type	Any
Shape	Any
Depth	Any

Type

Shape	Any
Depth	Any

Result Domain

Type	Boolean
Rank	0
Shape	⍳ 0 (scalar)
Depth	0 (simple scalar)

Implicit Arguments

⎕*CT* (determines comparison precision)

Description

The dyadic ⊇ function determines whether the left argument contains all the items found in the right argument. The result is a Boolean scalar: true, if the left argument is a superset of the right argument, and false if it is not. Duplicate items in either argument do not affect the result. For example:

```
      A←3 4⍴ 23 54 98 34 98 47 98 32 78 65 12 23
      A ⊇ B←⍳100
0
      B ⊇ A
1
```

The ⊇ function compares items in terms of the match (≡) function, which uses the value of ⎕CT. Because ≡ allows mixed-type arguments, you can compare characters with numbers. However, such a comparison is always false, so that if you use mixed-type arguments for dyadic ⊇, the result will be zero. For example:

```
      '23 24 25' ⊇ 22 23 24 25 26
0
```

Further examples:

```
      ⎕←WRL←0 'AB' (⊂,3)
0 +--+ +---+
  |AB| |+-+|
  +--+ ||3||
       |+-+|
       +---+
      ⎕←POOL←2 2 ρ 0 'AB' 'EB' (⊂,3)
0    +--+
     |AB|
     +--+
+--+ +---+
|EB| |+-+|
+--+ ||3||
     |+-+|
     +---+
      POOL ⊇ WRL
1
      ⎕←VAN←0 'QTW' ¯1 (⊂,3)
0 +---+ ¯1 +---+
  |QTW|    |+-+|
  +---+    ||3||
           |+-+|
           +---+
      ⎕←VIC←(⊂,4),0,(⊂'DP')
+-+ 0 +--+
|4|   |DP|
+-+   +--+
      VAN ⊇ VIC
0
```

Possible Errors Generated

None.

? Deal

Form

A?B

Left Argument Domain

Type	Nonnegative near-integer
Shape	Singleton
Depth	0 or 1 (simple)

Right Argument Domain

Type	Nonnegative near-integer
Shape	Singleton
Depth	0 or 1 (simple)

Result Domain

Type	Nonnegative integer
Rank	1
Shape	,*A*
Depth	1 (simple)

Implicit Arguments

⎕*RL* ⎕*IO* (*A?B* when ⎕*IO* ← 1 is identical to 1 + *A?B* when ⎕*IO* ← 0 for the same ⎕*RL*)

Description

For *A?B*, the dyadic ? function generates a vector of integers randomly selected from ⍳ *B*; no number is selected more than once. The length of the result vector is specified by *A*. For example:

```
      5?5
4 2 3 1 5
      5?1.0E7
2047059 8326627 1771140 853115 3809508
      5?1.0E7
8895125 7387197 6272379 6940437 9062050
      5?1.0E7
6693744 185074 2861354 853279 5088023
```

Unlike the roll function, dyadic ? is analogous to dealing a number of cards from a deck with no two cards alike. Roll is analogous to rolling several dice independently; roll may generate duplicates, but deal will not.

The value of the system variable ⎕RL affects the result of the deal operation, and the value of ⎕RL changes each time a deal operation completes successfully. For more details about ⎕RL, see Chapter 2.

Possible Errors Generated

```
9 RANK ERROR (NOT SINGLETON)

10 LENGTH ERROR (NOT SINGLETON)

15 DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

15 DOMAIN ERROR (INCORRECT TYPE)

15 DOMAIN ERROR (NOT AN INTEGER)

15 DOMAIN ERROR (NEGATIVE NUMBER NOT ALLOWED)

15 DOMAIN ERROR (RIGHT ARGUMENT IS LESS THAN LEFT)

27 LIMIT ERROR (INTEGER TOO LARGE)
```

≡ Depth

Form

`≡B`

Argument Domain

Type	Any
Shape	Any
Depth	Any

Result Domain

Type	Integer (non-negative)
Rank	0
Shape	⍳ 0 (scalar)
Depth	0 (simple scalar)

Implicit Arguments

None.

Description

The monadic ≡ function (known as depth) indicates the maximum level of nesting in an array. A simple array has 1 level of nesting (0 if the array is scalar). An enclosed array has a depth of at least 2.

Examples:

```
      ⎕←B←9                   ⍝CREATE A SIMPLE SCALAR
9
      ≡B
0
                              ⍝CREATE A SIMPLE ARRAY
      ⎕←C←'WHERE ARE YOU GOING?'
WHERE ARE YOU GOING?
      ≡C
1
```

```
      ⎕←D←1 (5 6 7) 11 12     ⍝CREATE AN ENCLOSED ARRAY
1 +-----+ 11 12
  |5 6 7|
  +-----+
      ≡D
2
            ⍝CREATE AN ENCLOSED ARRAY WITH MORE NESTING
      ⎕←E←1 (5 6 7 (8 9 10)) 11 12
1 +--------------+ 11 12
  |5 6 7 +------+|
  |      |8 9 10||
  |      +------+|
  +--------------+
      ≡E
3
```

Possible Errors Generated

None.

⊃ Disclose

Form

⊃*B*   ⊃[*K*]*B*

Argument Domain

Type	Any
Shape	Any
Depth	Any

Result Domain

Type	Same as constituent items in *B*
Rank	(ρρ*B*)+↑⌈/ρ¨ρ¨(,*B*),⊂↑*B*
Shape	(ρ*B*),↑⌈/(ρ¨(,*B*),⊂↑*B*)~⊂⍳0 (ρZ)[,*K*]←→↑⌈/(ρ¨(,*B*)~⊂⍳0
Depth	0⌈¯1+≡*B*

Implicit Arguments

None.

Description

The monadic ⊃ function reduces the depth of an array. It reverses the building action of the monadic enclose (⊂) function. Disclose is the left inverse of enclose (*B*←→⊃⊂*B* and *B*←→⊃[*K*]⊂[*K*]*B*).

The rank of the items in *B* must be the same (singleton items are extended). However, the lengths of the corresponding axes do not need to match. For example, three enclosed items of shape 1 3 1, 2 6 2, and 4 2 4 match in rank, but not in shape. When the shapes do not match, each item is padded along each axis, and the length of each of the result's axes is equal to the longest corresponding axis among the items of *B*. In the preceding example, the portion of the result that corresponds to the three items would have the shape 4 6 4:

```
                          ⍝ALL ITEMS SAME RANK
      F←(3 2⍴⍳6) (2 3⍴'ABCDEF') (2 4⍴10 ⌽⍳9) (5 1⍴'TUVXY')
      ⍴F
4
      ⍴¨F
+---+ +---+ +---+ +---+
|3 2| |2 3| |2 4| |5 1|
+---+ +---+ +---+ +---+
      ⎕←DD←⊃F                 ⍝NOTE FILL ITEMS
1 2 0 0
3 4 0 0
5 6 0 0
0 0 0 0
0 0 0 0

A B C
D E F




2 3 4 5
6 7 8 9
0 0 0 0
0 0 0 0
0 0 0 0

T
U
V
X
Y
      ⍴DD
4 5 4
```

Disclose only reduces one level of enclosure:

```
      A←(⊂⍳3) 'ABC' 3
      A
+-------+ +---+ 3
|+-----+| |ABC|
||1 2 3|| +---+
|+-----+|
+-------+
      ⍴A
3
      ≡A
3
```

```
      ρ¨A
++ +-+ ++
|| |3| ||
++ +-+ ++
      X←⊃A
       X
+-----+ +-----+ +-----+
|1 2 3| |0 0 0| |0 0 0|
+-----+ +-----+ +-----+
A       B       C
3       0       0
      ρX
3 3
```

Disclose with axis (form ⊃[K]B) allows you to specify the placement of the disclosed item's axes. The number of axes specified by K must be equal to the rank of the items of B (ignoring the singleton items), and the axis numbers must be less than the sum of the rank of B plus the rank of the items of B (ι(ρρB)+↑⌈\,ρ¨ρ¨B). The axis numbers must also be unique. The following example shows various combinations of axis arguments and the resulting arrays:

```
      R ← (3 3ρι9) (2 3ρ'ON' 'TI' 'MA' 'NO' 'IT' 'AM')
      R
+-----+ +--------------+
|1 2 3| |+--+ +--+ +--+|
|4 5 6| ||ON| |TI| |MA||
|7 8 9| |+--+ +--+ +--+|
+-----+ |+--+ +--+ +--+|
        ||NO| |IT| |AM||
        |+--+ +--+ +--+|
        +--------------+
      A←⊃R
      ≡A
2
```

```
      A
1    2    3


4    5    6


7    8    9



+--+ +--+ +--+
|ON| |TI| |MA|
+--+ +--+ +--+
+--+ +--+ +--+
|NO| |IT| |AM|
+--+ +--+ +--+
+--+ +--+ +--+
|  | |  | |  |
+--+ +--+ +--+
      B←⊃[1 2]R
      B
1 +--+
  |ON|
  +--+
2 +--+
  |TI|
  +--+
3 +--+
  |MA|
  +--+

4 +--+
  |NO|
  +--+
5 +--+
  |IT|
  +--+
6 +--+
  |AM|
  +--+

7 +--+
  |  |
  +--+
8 +--+
  |  |
  +--+
9 +--+
  |  |
  +--+
      ≡R
3
      ≡B
2
      ρA
2 3 3
      ρB
3 3 2
      C←⊃[1 3]R
      ≡C
2
      ρC
3 2 3
      C
1    2    3
+--+ +--+ +--+
|ON| |TI| |MA|
+--+ +--+ +--+

4    5    6
+--+ +--+ +--+
|NO| |IT| |AM|
+--+ +--+ +--+

7    8    9
+--+ +--+ +--+
|  | |  | |  |
+--+ +--+ +--+
      D←⊃[2 3]R
      ≡D
2
      ρD
2 3 3
```

```
      D
1    2    3


4    5    6


7    8    9



+--+ +--+ +--+
|ON| |TI| |MA|
+--+ +--+ +--+
+--+ +--+ +--+
|NO| |IT| |AM|
+--+ +--+ +--+
+--+ +--+ +--+
|  | |  | |  |
+--+ +--+ +--+
      E←⊃[2 1]R
      E
1 +--+
  |ON|
  +--+
4 +--+
  |NO|
  +--+
7 +--+
  |  |
  +--+

2 +--+
  |TI|
  +--+
5 +--+
  |IT|
  +--+
8 +--+
  |  |
  +--+

3 +--+
  |MA|
  +--+
6 +--+
  |AM|
  +--+
9 +--+
  |  |
  +--+
      ≡E
2
      ρE
3 3 2
      F←⊃[3 1]R
      ≡F
2
      ρF
3 2 3
      F
1    4    7
+--+ +--+ +--+
|ON| |NO| |  |
+--+ +--+ +--+

2    5    8
+--+ +--+ +--+
|TI| |IT| |  |
+--+ +--+ +--+

3    6    9
+--+ +--+ +--+
|MA| |AM| |  |
+--+ +--+ +--+
      G←⊃[3 2]R
      ≡G
2
      ρG
2 3 3
      G
1    4    7


2    5    8


3    6    9



+--+ +--+ +--+
|ON| |NO| |  |
+--+ +--+ +--+
+--+ +--+ +--+
|TI| |IT| |  |
+--+ +--+ +--+
+--+ +--+ +--+
|MA| |AM| |  |
+--+ +--+ +--+
```

If all the items of B are scalars, then the axis, if specified, must be empty:

```
      T←⊂¨(⍳3)(⍳4)(⍳5)
      T
+-------+ +---------+ +-----------+
|+-----+| |+-------+| |+---------+|
||1 2 3|| ||1 2 3 4|| ||1 2 3 4 5||
|+-----+| |+-------+| |+---------+|
+-------+ +---------+ +-----------+
      ≡T
3
      ⍴T
3
      ≡¨T
2 2 2
      ⍴¨T
++ ++ ++
|| || ||
++ ++ ++
      ⎕←S←⊃[⍳0]T
+-----+ +-------+ +---------+
|1 2 3| |1 2 3 4| |1 2 3 4 5|
+-----+ +-------+ +---------+
      ≡S
2
      ⍴S
3
```

The disclose of an array which contains only scalars and empty arrays as item will be an empty array:

```
      E←2 '' 3
      E
2 ++ 3
  ||
  ++
      ≡E
2
      ⍴E
3
      ↑E
2
```

```
      X←⊃E
      X
                              (APL outputs a blank line)
      ≡X
1
      ⍴X
3 0
      ↑X
0
```

The following expression describes the formal relationship between disclose and disclose with axis: `⊃B ←→ ⊃[(⍴⍴B)+⍳⍴⍴↑B]B`

Possible Errors Generated

```
9  RANK ERROR (ITEMS NOT SINGLETON OR ALL THE SAME RANK)

27  LIMIT ERROR (INTEGER TOO LARGE)

28  AXIS RANK ERROR (NOT VECTOR DOMAIN)

29  AXIS LENGTH ERROR (ARGUMENT RANK AND AXIS INCOMPATIBLE)

30  AXIS DOMAIN ERROR (ARGUMENT RANK AND AXIS INCOMPATIBLE)

30  AXIS DOMAIN ERROR (AXIS LESS THAN INDEX ORIGIN)

30  AXIS DOMAIN ERROR (DUPLICATE AXIS NUMBER)

30  AXIS DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

30  AXIS DOMAIN ERROR (INCORRECT TYPE)

30  AXIS DOMAIN ERROR (NOT AN INTEGER)

30  AXIS DOMAIN ERROR (SEMICOLON LIST NOT ALLOWED)
```

↓ Drop

Form

`A↓B     A↓[K]B`

Left Argument Domain

Type	Near-integer
Shape	Vector domain
Depth	`0` or `1` (simple)

Right Argument Domain

Type	Any
Shape	Any
Depth	Any

Result Domain

Type	Same as right argument
Rank	`(ρ,A)⌈ρρB`
Shape	`0⌈(ρB)-|A` (if no explicit axis)
Depth	`≡B`

Implicit Arguments

None.

Description

The dyadic ↓ function builds an array by dropping a specified number of items from an existing array. The left argument specifies how many items are to be dropped from each axis in the right argument array. Thus, for *A*↓*B*, item *A*[*K*] is used to drop values along the *K* th axis of *B*.

Unless the right argument is a scalar, the left argument must have a number of values equal to the rank of the right argument (for arguments *A* and *B*, ρ,*A* must equal ρρ*B*). For instance, if the right argument is a vector, the left argument must have just one value. If that value is positive, APL drops the specified number of items from the beginning of the vector; if the value is negative, APL drops items from the end of the vector. For example:

```
      2↓⍳5
3 4 5
      ¯2↓⍳5
1 2 3
```

If the right argument is a scalar, it is reshaped to a singleton with a rank equal to the length of the left argument.

If the rank of the right argument is greater than 1, the result array is said to be a "corner" of the argument array. The origin of the corner is determined by the signs of the items of the left argument. For example, if the right argument is a matrix, there are four possible corners, as shown in Figure 1–1.

The drop function leaves a corner that is diagonally opposite to the origin specified by the signs of the items of the left argument. In the following example, note how the order of the signs determines the "corner" selected from the matrix:

```
      ⎕←C←3 3 ⍴⍳9
1 2 3
4 5 6
7 8 9
      2 2↓C
9
      ¯2 ¯2 ↓C
1
      ¯2 2↓C
3
      2 ¯2 ↓C
7
```

Note that for arguments *A* and *B*, the dimension of the remaining corner is the complement of *A* with respect to ⍴*B*, or |(⍴*B*)-*A*.

If the value of an item in the left argument is greater than the length of the corresponding axis, then, for arguments *A* and *B*, *A*↓*B* returns an empty array with shape 0⌈(⍴*B*)-|*A*.

If the left argument is empty, the right argument must be a scalar, and the result is the right argument.

When you use ↓ with an axis argument, *K* is a vector of axis numbers whose lengths are determined by corresponding items of the left argument, *A*. Formally, ↓ with an axis argument can be described by the following:

```
Z ← 0>⍴B        ⋄        Z[K] ← A        ⋄        Z ← Z↓B
```

The value for *K* must be in the vector domain, and each item must be a near-integer in the set ⍳⍴⍴*B*. Therefore, the values of *K* are ⎕*IO* dependent. The items may be in any order, but they may not be duplicated. The length of *K* must be less than or equal to the rank of the right argument, and it must match the length of the left argument.

The value for *K* does not have to specify all the axes in *B*. APL regards the lengths of any missing axes as zero. This means that you can drop rows or columns of a matrix without specifying zero for the length of the other axis. For example:

```
      ⎕←A←8 5⍴⍳40
 1  2  3  4  5
 6  7  8  9 10
11 12 13 14 15
16 17 18 19 20
21 22 23 24 25
26 27 28 29 30
31 32 33 34 35
36 37 38 39 40
      3 ↓[1] A                 ⍝DROP 3 ROWS OF A
16 17 18 19 20
21 22 23 24 25
26 27 28 29 30
31 32 33 34 35
36 37 38 39 40
      ¯2 ↓[2] A                 ⍝DROP THE LAST 2 COLUMNS OF A
 1  2  3
 6  7  8
11 12 13
16 17 18
21 22 23
26 27 28
31 32 33
36 37 38
      3 4 ↓[2 1] A              ⍝DROP 4 ROWS, 3 COLUMNS OF A
24 25
29 30
34 35
39 40
      ⎕IO ← 0
      4 3 ↓[0 1] A              ⍝DROP 4 ROWS, 3 COLUMNS OF A
24 25
29 30
34 35
39 40
```

Further examples:

```
      ⎕←POL←2 3ρ0,(⊂'ABC'),1,0,(⊂'AB'),''
0 +---+ 1
  |ABC|
  +---+
0 +--+  0
  |AB|
  +--+
      POL
0 +---+ 1
  |ABC|
  +---+
0 +--+  0
  |AB|
  +--+
      2 ↓[1]POL
                            (APL outputs a blank line)
      ¯1 ↓[1]POL
0 +---+ 1
  |ABC|
  +---+
      2 ↓[2]POL
1
0
      ⎕←MEW←4 3 ρ'XY' 1 3 (⊂,1) ¯2 '' 'A' ' ' 0 1 ¯4 0
+--+  1  3
|XY|
+--+
+---+ ¯2 ++
|+-+|    ||
||1||    ++
|+-+|
+---+
A        0
1     ¯4 0
      1 ¯1 ↓ MEW
+---+ ¯2
|+-+|
||1||
|+-+|
+---+
A
1     ¯4
      2 1 ↓ [2 1]MEW
++
||
++
0
0
```

```
      2 0 ↓ MEW
A     0
1 ¯4 0
```

The following expression describes the formal relationship between drop and drop with axis: `A↓B ←→ A↓[ιρρB]B`

Possible Errors Generated

```
9  RANK ERROR (NOT VECTOR DOMAIN)

10  LENGTH ERROR (LEFT LENGTH NOT EQUAL TO RIGHT RANK)

15  DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

15  DOMAIN ERROR (INCORRECT TYPE)

15  DOMAIN ERROR (NOT AN INTEGER)

27  LIMIT ERROR (INTEGER TOO LARGE)

28  AXIS RANK ERROR (NOT VECTOR DOMAIN)

29  AXIS LENGTH ERROR (LEFT ARGUMENT HAS WRONG LENGTH)

30  AXIS DOMAIN ERROR (ARGUMENT RANK AND AXIS INCOMPATIBLE)

30  AXIS DOMAIN ERROR (AXIS LESS THAN INDEX ORIGIN)

30  AXIS DOMAIN ERROR (DUPLICATE AXIS NUMBER)

30  AXIS DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

30  AXIS DOMAIN ERROR (INCORRECT TYPE)

30  AXIS DOMAIN ERROR (NOT AN INTEGER)

30  AXIS DOMAIN ERROR (SEMICOLON LIST NOT ALLOWED)
```

⊂ Enclose

Form

```
⊂B      ⊂[K]B
```

Argument Domain

Type	Any
Shape	Any
Depth	Any

Result Domain

Type	Same as constituent items
Rank	(ρρB)-ρ,K
Shape	(ρB)[(ιρρB)~K]
Depth	(0≠≡B)+≡B

Implicit Arguments

None.

Description

The monadic ⊂ function builds enclosed arrays. For a nonsimple scalar argument, the result of the form ⊂B is always an enclosed scalar item. If the argument is a simple scalar, the depth remains the same: B←→⊂B when B is a simple scalar. The result of the form ⊂[K]B is an array of enclosed scalars:

```
      B ← 4
      C ← ⍳5
      D ← 2 2 ⍴ 'ABCD'
      ⎕←B←⊂B   ⍝ENCLOSE A SIMPLE SCALAR, NOTHING HAPPENS
4
      ⎕←B←⊂,B         ⍝MAKE A SINGLETON
+-+
|4|
+-+
      ⎕←C←⊂C
+---------+
|1 2 3 4 5|
+---------+
      ⎕←D←⊂D
+--+
|AB|
|CD|
+--+
      ⍴B ⋄ ⍴C ⋄ ⍴D     ⍝THE NEW B C AND D ARE SCALAR
                                   (APL outputs 3 blank lines)
```

Each time you use monadic ⊂, you increase the depth of the argument by one (unless the argument is a simple scalar). For example:

```
      ⎕←D←2 2⍴ 'ABCD'
AB
CD
      ⍴D
2 2
      ≡D                     ⍝DEPTH OF D SHOWS A SIMPLE ARRAY
1
      ⎕←D←⊂D
+--+
|AB|
|CD|
+--+
      ⍴D
                                    (APL outputs a blank line)
      ≡D                     ⍝DEPTH OF D HAS INCREASED TO 1
2
```

Using the catenate function (,) with ⊂ allows you to create arrays with multiple items. In such an expression, you must use parentheses to prevent the scope of ⊂ from extending to the rightmost end of the expression. You can also enclose arrays that are already enclosed. The only limit to the depth you create is the memory available to the workspace.

For example:

```
      B←4
      C←⍳5
      ⎕←E←B , (⊂B) , ⊂C     ⍝NOTE USE OF PARENTHESES
4 4 +---------+
    |1 2 3 4 5|
    +---------+
      ⍴E
3
      ≡E
2
      D←2 2 ⍴ 'ABCD'
      ⎕←E←B , (⊂B) , (⊂C) , D     ⍝NOTE USE OF PARENTHESES
4 4 +---------+ +--+
    |1 2 3 4 5| |AB|
    +---------+ |CD|
                +--+
      ⍴E
4
      ≡E
2
      ⎕←E←⊂E
+--------------------+
|4 4 +---------+ +--+|
|    |1 2 3 4 5| |AB||
|    +---------+ |CD||
|                +--+|
+--------------------+
      ⍴E                      ⍝SHAPE OF E SHOWS IT IS NOW A SCALAR
                                 (APL outputs a blank line)
      ≡E
3
```

The result of the form ⊂[K]B is in an array of items formed by enclosing subarrays along the axes given by K. The axis numbers in K must be a unique set of numbers in ⍳⍴⍴B:

```
      ←S←2 3⍴⍳6
      ⊂S
+-----+
|1 2 3|
|4 5 6|
+-----+
      ⊂[1]S
+---+ +---+ +---+
|1 4| |2 5| |3 6|
+---+ +---+ +---+
      ⊂[2]S
+-----+ +-----+
|1 2 3| |4 5 6|
+-----+ +-----+
      ⊂[1 2]S
+-----+
|1 2 3|
|4 5 6|
+-----+
      ⊂[2 1]S       ⍝CHANGING AXIS ORDER TRANSPOSES SHAPE
+---+
|1 4|
|2 5|
|3 6|
+---+
      SCHILLER←'AGAINST' 'STUPIDITY' 'THE GODS' 'THEMSELVES'
      ⎕←SCHILLER←3 2⍴SCHILLER,'CONTEND' 'IN VAIN'
+-------+  +---------+
|AGAINST|  |STUPIDITY|
+-------+  +---------+
+--------+ +----------+
|THE GODS| |THEMSELVES|
+--------+ +----------+
+-------+  +-------+
|CONTEND|  |IN VAIN|
+-------+  +-------+
      ⎕←PHRASES←⍪[1.5]⊂[2]SCHILLER
+---------------------+
|+-------+ +---------+|
||AGAINST| |STUPIDITY||
|+-------+ +---------+|
+---------------------+
+-----------------------+
|+--------+ +----------+|
||THE GODS| |THEMSELVES||
|+--------+ +----------+|
+-----------------------+
+-------------------+
|+-------+ +-------+|
||CONTEND| |IN VAIN||
|+-------+ +-------+|
+-------------------+
      ⍴PHRASES
3 1
```

If *K* is empty, than it has no effect if *B* is a simple array. If *B* is enclosed, then each item in *B* becomes enclosed one level deeper (⊂[⍳0]*B* ←→ ⊂¨*B*). For example:

```
      ⎕←S←2 3⍴6                ⍝CREATE S, A SIMPLE ARRAY
1 2 3
4 5 6
      ⊂[⍳0]S                   ⍝EMPTY K, NO CHANGE
1 2 3
4 5 6
      SCHILLER←'AGAINST' 'STUPIDITY' 'THE GODS' 'THEMSELVES'
      ⎕←SCHILLER←3 2⍴ SCHILLER, 'CONTEND' 'IN VAIN'
+-------+  +---------+
|AGAINST|  |STUPIDITY|
+-------+  +---------+
+--------+ +----------+
|THE GODS| |THEMSELVES|
+--------+ +----------+
+-------+  +-------+
|CONTEND|  |IN VAIN|
+-------+  +-------+
      ⊂[⍳0]SCHILLER            ⍝EMPTY K, ITEMS NESTED DEEPER
+---------+  +-----------+
|+-------+|  |+---------+|
||AGAINST||  ||STUPIDITY||
|+-------+|  |+---------+|
+---------+  +-----------+
+----------+ +------------+
|+--------+| |+----------+|
||THE GODS|| ||THEMSELVES||
|+--------+| |+----------+|
+----------+ +------------+
+---------+  +---------+
|+-------+|  |+-------+|
||CONTEND||  ||IN VAIN||
|+-------+|  |+-------+|
+---------+  +---------+
```

Further examples:

```
      ⎕←POL←2 3 ⍴ 'ABC' 0 (⊂,2) 99 'A' '0'
+---+ 0 +---+
|ABC|   |+-+|
+---+   ||2||
        |+-+|
        +---+
99    A 0
      ⊂POL
+-------------+
|+---+ 0 +---+|
||ABC|   |+-+||
|+---+   ||2|||
|        |+-+||
|        +---+|
|99    A 0    |
+-------------+
```

The first two of the following expressions describe the relationship between ⊂ and ⊂[K]. The third expression describes the relationship between ⊂[K] and the disclose (⊃[K]) function:

```
⊂B ↔ ⊂[⍳⍴⍴B]B
⊂[K]B ↔ ⊂(⍋K)⍉B (only true when  K includes all axes of  B )
B ↔ ⊃[K] ⊂[K] B
B ↔ ⊃⊂B
```

Possible Errors Generated

27 *LIMIT ERROR (INTEGER TOO LARGE)*

28 *AXIS RANK ERROR (NOT VECTOR DOMAIN)*

30 *AXIS DOMAIN ERROR (ARGUMENT RANK AND AXIS INCOMPATIBLE)*

30 *AXIS DOMAIN ERROR (AXIS LESS THAN INDEX ORIGIN)*

30 *AXIS DOMAIN ERROR (DUPLICATE AXIS NUMBER)*

30 *AXIS DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)*

30 *AXIS DOMAIN ERROR (INCORRECT TYPE)*

30 *AXIS DOMAIN ERROR (NOT AN INTEGER)*

30 *AXIS DOMAIN ERROR (SEMICOLON LIST NOT ALLOWED)*

∊ Enlist

Form

∊*B*

Argument Domain

Type	Any
Shape	Any
Depth	Any

Result Domain

Type	Same as argument
Rank	1
Shape	Vector
Depth	1 (simple vector)

Implicit Arguments

None.

Description

∊ builds a simple vector by recursively raveling each of the items in its argument. For example:

```
      ⎕←A←∊ (2 (2 2⍴5 6 7 8)) 'ABC' 'A' 2 5 6 7 8 'ABC'
2 5 6 7 8 ABCA 2 5 6 7 8 ABC
```

Possible Errors Generated

None.

⍎ Execute

Form

⍎*B*

⍎ is formed with ⊥ and ∘

Argument Domain

Type	Any
Shape	Any (Vector domain for characters)
Depth	Any (0 or 1 for characters)

Result Domain

Type	Any
Rank	Any
Shape	Any
Depth	Any

Implicit Arguments

None.

Description

The monadic ⍎ function executes the expression represented by its character-string argument as if that expression were entered in immediate mode or included in a user-defined operation. For example, the expressions ⍳ 5 and ⍎ ' ⍳ 5 ' return the same result:

```
      ⍳5
1 2 3 4 5
      ⍎'⍳5'
1 2 3 4 5
      B←⍳5
       ⍎'B'
2 3 4 5
```

For a numeric argument *B*, ⍎*B* returns *B*. For example:

```
      ⍎20
20
      B←⍳5
      ⍎B
1 2 3 4 5
```

For an enclosed or heterogeneous array *B*, ⍎*B* returns *B*. For example:

```
      ⍎ 1 2 'A' 3
1 2 A 3
      ⎕←POL←2 3⍴ ('ABC') 0 (⊂,2) 99 'A' 0
+---+ 0 +---+
|ABC|    |+-+|
+---+    ||2||
         |+-+|
         +---+
99    A 0
      ⍎ 'POL'
+---+ 0 +---+
|ABC|    |+-+|
+---+    ||2||
         |+-+|
         +---+
99    A 0
      ⍎ POL
+---+ 0 +---+
|ABC|    |+-+|
+---+    ||2||
         |+-+|
         +---+
99    A 0
```

The ⍎ function is similar to the ⎕*XQ* system function; however, there are several differences.

One difference is that the ⎕*XQ* system function always returns a value, but the ⍎ function returns a value only if the evaluation of its argument returns a value. Another difference is that the ⎕*XQ* function cannot execute a branch function (→), and the ⍎ function can.

The ⍎ and ⎕*XQ* functions also handle errors differently. Errors resulting from the evaluation of the ⎕*XQ* function's argument cannot be trapped; if an error occurs during the evaluation of its argument, ⎕*XQ* returns an empty array whose shape indicates the number of the error. With the ⍎ function, however, you can use ⎕*TRAP* to trap errors. If an error occurs in the character string being executed by ⍎, APL generates—in addition to the normal three-line error message—an execute error message for the line on which the actual execute error occurred.

For example:

```
      ∇ GRIFF A
[1]   B←⍎A
[2]   ∇
      GRIFF '3,'
   7 ⍎ SYNTAX ERROR (RIGHT ARGUMENT TO FUNCTION MISSING)
       3,
       ^
  25 EXECUTE ERROR
 GRIFF[1]      B←⍎A
       ^
      )SI
GRIFF[1] *
      B
  11 VALUE ERROR
       B
       ^
      GRIFF' '
  11 VALUE ERROR (REQUIRED VALUE NOT SUPPLIED BY EXECUTE)
 GRIFF[1]      B←⍎A
       ^
```

In the previous example, when the argument to the ⍎ function was invalid ('3,'), APL generates six lines of error messages and suspends operation execution. The blank argument is a valid one for the ⍎ function, but ⍎' ' does not produce a value, so APL signals *VALUE ERROR* when the assignment is made to *B*.

If you enter the attention signal while the ⍎ function is executing, APL stops and signals *ATTENTION SIGNALED*.

Note that quiet functions are still quiet when executed, provided that the execute is the leftmost function in the statement. When the argument is empty and numeric, the result is an empty numeric vector (⍎ ⍳0 ←→ ⍳0). When the argument is empty and character, the result is an empty character vector (' ' ←→ ⍎ ' ') if a value is required by the expression. For example:

```
      ⍎'Z←1'
      ,⍎'Z←1'
1
      ⍎ ⍳0
      ⍎ ''                            ⍝QUIET, NO OUTPUT
      A←⍎ ''
      A
                                (APL outputs a blank line.)
```

Possible Errors Generated

9 *RANK ERROR (NOT VECTOR DOMAIN)*

25 *EXECUTE ERROR*

⍇ and ⍈ File Input and Output

Form

⍇ ⟦[*mode* | *index*]⟧ *chan* ⟦*data-type*⟧
⍇ is formed with ⎕ and ←

Argument Domain

Type	Numeric
Shape	Vector domain
Depth	Any

Result Domain

Type	Any
Rank	Any
Shape	Any
Depth	Any

Form

⟦*data*⟧ ⍈ ⟦[*mode* | *index*]⟧ *chan* ⟦*data-type*⟧
⍈ is formed with ⎕ and →

Left Argument Domain

Type	Any
Shape	Any
Depth	Any

Right Argument Domain

Type	Numeric
Shape	Vector domain
Depth	0 or 1 (simple)

Result Domain

Type	Same as left argument
Rank	⍴ ⍴ *data-sent*
Shape	⍴ *data-sent*
Depth	≡ *data-sent*

Parameters

mode
Is an integer representing one of the modes listed in Table 5-3 in the *VAX APL User's Guide*. This parameter is used only when accessing files with ASCII sequential organization. When you specify *mode*, it must be surrounded by brackets.

index
Is the component number, record number, or key value in a direct-access, relative, or keyed file, respectively. *Index* must be surrounded by brackets.

chan
Is a positive integer scalar whose value is a channel number in the range 1 through 999.

data-type
Specifies the data type of the record you want to read or write. When you include a value for *data-type*, you imply that the record contains pure data; that is, the beginning of the record does not contain any header information. If you do not specify *data-type*, or if you specify a value of `0`, APL assumes that there is a header at the beginning of the record

data
Is the data that is to be written to the file.

Description

The file input (⍇) and file output (⍈) functions are for reading and writing files. ⍇ and ⍈ are described in greater detail in Chapter 4 of the *VAX APL User's Guide* along with other file I/O information.

The file output function (⍈) in its monadic form deletes a component or record from a direct-access, relative or keyed file. APL signals `DOMAIN ERROR (DELETION NOT ALLOWED)` if you use monadic ⍈ with a sequential file. When monadic ⍈ is not the leftmost function in the statement, it returns an empty numeric matrix of shape `0 75`.

The value of the ⍇ function is the record read from the specified file. The ⍈ function is quiet. It does not display a result if it is the leftmost function in a statement. When it is not the leftmost function, ⍈ returns the value of its left argument.

When a ⍇ or ⍈ function references a channel associated with a file that is not open, APL opens the file and executes the function.

Possible Errors Generated

```
15  DOMAIN ERROR (DELETION NOT ALLOWED)
```

↑ First

Form

```
↑B
```

Argument Domain

Type	Any
Shape	Any
Depth	Any

Result Domain

Type	Same as selected item
Rank	Same as selected item
Shape	Same as selected item
Depth	Same as selected item

Implicit Arguments

None.

Description

The monadic ↑ function builds an array by disclosing the first item from an existing array. If *B* is empty, then ↑ returns the prototype of *B*:

```
      B ← 4
      C ← ⍳5
      ↑C
1
      D ← 2 2 ⍴ 'ABCD'
      ↑D
A
```

```
      ⎕ ← E ← B , (⊂C) , ⊂D
4 +---------+ +--+
  |1 2 3 4 5| |AB|
  +---------+ |CD|
              +--+
      ↑E              ⍝FIRST OF E
4
      ≡↑E             ⍝DEPTH SHOWS A SIMPLE SCALAR ARRAY
0
      ↑E[2]           ⍝FIRST OF SECOND ITEM OF E
1 2 3 4 5
      ≡↑E[2]          ⍝DEPTH SHOWS A SIMPLE ARRAY
1
      ⍴↑E[2]          ⍝SHAPE SHOWS A VECTOR
5
      ↑E[3]           ⍝FIRST OF THIRD ITEM OF E
AB
CD
      ≡↑E[3]          ⍝DEPTH SHOWS A SIMPLE ARRAY
1
      ⍴↑E[3]          ⍝SHAPE SHOWS A MATRIX
2 2
      ↑ 0 3⍴99        ⍝EMPTY ARG RETURNS PROTOTYPE
0
      ↑ ' '           ⍝PROTOTYPE IS A CHARACTER BLANK

      ↑ 0⍴(1 2 3) 'ABC'
0 0 0
```

For simple arrays, the result of monadic ↑ is the same as it would be with the dyadic take function (↑) when all the items of the left argument are 1. Formally, this can be represented as follows: `↑B ←→ ((⍴⍴B)⍴1)↑B`. However, note that take does not disclose items of an array. First is also related to the pick (⊃) function as follows: `↑B ←→ (⊂(⍴⍴B)⍴1)⊃B`

Possible Errors Generated

None.

⍕ Monadic Format

Form

⍕*B*
⍕ is formed with ⊤ and ∘

Argument Domain

Type	Any
Shape	Any
Depth	Any

Result Domain

Type	Character
Rank	1⌈⍴⍴*B* for simple numeric or heterogeneous *B* ⍴⍴*B* for simple character *B* 2 for nonsimple *B*
Shape	(¯1↓⍴*B*)↔(¯1↓⍴*result*) for simple *B*
Depth	1 (simple)

Implicit Arguments

⎕*PP* (Determines decimal precision)
⎕*NG* (Determines minus sign placement)
⎕*DC* (Displays control of enclosed arrays)

Description

The monadic ⍕ function formats its argument array as a character array, making it look as it would appear when displayed by APL.

Thus, if the argument array is already of type character, the result is identical to the argument:

```
      ⎕←A←3 5⍴'STAN SAM  STEVE'
STAN
SAM
STEVE
      ⍕A
STAN
SAM
STEVE
      ⍴⍕A
3 5
```

If the argument array is of type numeric, the result appears to be identical to the argument; however, the blank characters displayed along with the items are actually part of the result array. For example:

```
      A←2 4⍴⍳8
      B←⍕A
      A
1 2 3 4
5 6 7 8
      ⍴A
2 4
      B
1 2 3 4
5 6 7 8
      ⍴B
2 7
      (' ',B)[;2×⍳4]
1234
5678
```

Note the difference between the shapes of the numeric array *A* and the character array *B*.

Since it is not feasible to indicate both shape and depth in a two-dimension display, the format of an enclosed array is always a matrix. Shape is indicated by blank lines in the same manner as for simple arrays. Display of depth is controlled by ⎕*DC*, the display control system variable.

Further examples:

```
      ⎕←POL←2 3⍴ ('ABC') 0 (⊂,2) 99 'A' 0
+---+ 0 +---+
|ABC|   |+-+|
+---+   ||2||
        |+-+|
        +---+
99    A 0
      ⍴POL
2 3
```

```
      ⎕←B←⍕POL
+---+ 0 +---+
|ABC|   |+-+|
+---+   ||2||
        |+-+|
        +---+
99    A 0
      ρB                ⍝THE SHAPES OF B AND POL ARE DIFFERENT
6 13
      ⎕←XT← 3 ρ (2 0 ρ 5) ('') (⍳0)     ⍝CREATE AN EMPTY ARRAY
++ ++ ++
|| || ||
|| ++ ++
++
      ρXT
3
      ⎕←B←⍕XT
++ ++ ++
|| || ||
|| ++ ++
++
      ρ B
4 8
```

Possible Errors Generated

None.

⍕ Dyadic Format

Form

A⍕*B*

⍕ is formed with ⊤ and ∘

Left Argument Domain

Type	Near-integer
Shape	Vector domain
Depth	0 or 1 (simple)

Right Argument Domain

Type	Numeric
Shape	Any
Depth	0 or 1 (simple)

Result Domain

Type	Character
Rank	1⌈ρρ*B*
Shape	(¯1↓ρ*B*),+/1 0⌿(2,⌈0.5×ρ,*A*)ρ*A* (provided no widths are 0)
Depth	1 (simple)

Implicit Arguments

⎕*NG* (determines minus sign placement)

Description

The dyadic ⍕ function formats its right argument according to the width and precision information supplied by its left argument.

The left argument generally contains one pair of numbers for each column (last axis) in the right-argument array. The first number specifies the width of the field; the second number controls the print precision. For example:

```
      ⎕←B←2 4⍴⍳8
1 2 3 4
5 6 7 8
      A←5 2 4 1 4 0 6 3
      R←A⍕B
      R
 1.00 2.0   3 4.000
 5.00 6.0   7 8.000
      ⍴R
2 19
```

Because the right argument has four columns, the left argument (`A`) has four pairs of numbers. The last axis of the formatted array (`R`) has a length of `19`, the sum of the widths specified in `A` (`5+4+4+6`). The second number of each pair in `A` specifies how many digits are to be displayed to the right of the decimal point.

You do not have to specify more than one pair of numbers as the left argument. If you specify only one pair, that pair is replicated a number of times equal to the length of the last axis of the right argument.

The last axis of the formatted array `Y`, below, has a length of `36` because the format function specifies that each of the three columns should have a width of `12`. The items are displayed with four digits to the right of the decimal point because the second number of the left argument pair is `4`.

Note the difference in the results when the array is formatted so that all columns have a width of `9` and a print precision of `2`, and then a width of `6` and a print precision of `0`.

If a print-precision specification in the left argument is negative, the associated item is formatted in scientific rather than decimal form, and the argument represents the number of digits in the item's mantissa.

```
      ⎕←X←2 3⍴ 31.16 0 ¯1.07 ¯15.578 8 ¯235.61
 31.16  0    ¯1.07
¯15.578 8 ¯235.61
      ⍴X
2 3
      ⎕←Y←12 4⍕X
     31.1600      0.0000     ¯1.0700
    ¯15.5780      8.0000   ¯235.6100
      ⍴Y
2 36
      A←9 2⍕X
      A
    31.16     0.00    ¯1.07
   ¯15.58     8.00  ¯235.61
      ⍴A
2 27
```

```
      ⎕←R←6 0⍕X
    31     0    ¯1
   ¯16     8  ¯236
      ⍴R
2 18
      ⎕←B←9 ¯2⍕X
  3.1E1    0.0E0   ¯1.1E0
 ¯1.6E1    8.0E0   ¯2.4E2
      ⎕←C←7 ¯1⍕X
  3E1    0E0   ¯1E0
 ¯2E1    8E0   ¯2E2
```

The width specification in the left argument may be omitted or may be `0`. If it is omitted, the entire left argument must be a singleton and is extended to `(2×¯1↑⍴B)⍴0,A`, for arrays `A` and `B`. If the width specification is `0`, then APL uses the minimum width possible, allowing for one blank between the formatted columns.

Two more examples of dyadic ⍕ follow. The first illustrates the formatting of a rank 3 array; the second shows how you can use ⍕ to format tables.

```
      ⎕←A←2 2 2⍴⍳8
1 2
3 4

5 6
7 8
      ⎕←C←5 2⍕A
 1.00 2.00
 3.00 4.00

 5.00 6.00
 7.00 8.00
      ⍴C
2 2 10
      ⎕←B←3 3⍴ 1 0 0 1 0 1 1 1 1
1 0 0
1 0 1
1 1 1
      1 0 ⍕B
100
101
111
```

Second example:

```
                          ⍝TABLE FORMATTING
      ROWS←5 7⍴'APL    FORTRANCOBOL  BASIC  PLI    '
      COLS←'  USERS  PROGS  SYST
      FORM←5 3⍴A
      ( (7↑' ') ⍪ROWS),COLS⍪7 0⍕FORM
         USERS  PROGS  SYSTS
APL          1      2      3
FORTRAN      4      5      6
COBOL        7      8      1
BASIC        2      3      4
PLI          5      6      7
```

If the right argument to the dyadic format function is empty, the shape of the result is determined by the following function:

```
      ∇Z←L EMPTY_SHAPE R ;C;W;P
[1]   L←((¯1↑⍴R),2)⍴((1=⍴,L)/0),L
[2]   W←L[;1]
[3]   P←L[;2]
[4]   C←W,(P+3),2,6,[1.5]6-P
[5]   Z←(W=0)×2+(-×P)+<¯1
[6]   C←(Z⌽C)[;1]
[7]   Z←(¯1↓⍴R),(-W[1]=0)++/C
[8]   ∇
```

For example:

```
      ⍴5 0 0 2⍕0 2⍴5
0 10
      ⍴0 2 5 0⍕0 2⍴5
0 9
```

Possible Errors Generated

9 *RANK ERROR (NOT VECTOR DOMAIN)*

10 *LENGTH ERROR*

15 *DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)*

15 *DOMAIN ERROR (INCORRECT TYPE)*

15 *DOMAIN ERROR (NOT AN INTEGER)*

15 *DOMAIN ERROR (NEGATIVE NUMBER NOT ALLOWED)*

15 *DOMAIN ERROR* (*WIDTH TOO SMALL*)

27 *LIMIT ERROR* (*INTEGER TOO LARGE*)

27 *LIMIT ERROR* (*PARAMETER OUT OF RANGE*)

⍒ Monadic Grade Down

Form

⍒*B*   ⍒[*K*]*B*
⍒ is formed with ∇ and |

Argument Domain

Type	Homogeneous
Shape	Matrix, vector, or scalar (not singletons of rank >2)
Depth	0 or 1 (simple)

Result Domain

Type	Nonnegative integer
Rank	1
Shape	(⌽⍴*B*)[*K*]
Depth	1 (simple)

Implicit Arguments

⎕*IO* (⍒*B* when ,⎕*IO* ← 1 is identical to 1 + ⍒*B* when ,⎕*IO* ← 0)

Description

The monadic ⍒ function returns a numeric vector whose items can be used to sort the items of the argument in descending order. Thus, grade down does not actually sort arrays. It creates a permutation vector of the index numbers of the argument array's items, and this vector can then be used to sort the array.

Sorting a vector requires two steps. First, the vector is the argument to the grade down function, and then the result is used to index the vector:

```
      A←2 9 7 4 3 10 4
      ⎕←B←⍒A
6 2 3 4 7 5 1
      A[B]
10 9 7 4 4 3 2
```

If two or more items of a vector or matrix have the same value, the order of the items is determined by their relative positions in the original array (this is called a stable sort). For character arguments, the collating sequence is determined by the value of ⎕*AV*. Note that for numeric arguments, the result is not ⎕*CT*-dependent.

When you use the grade down function to sort a matrix, APL treats each row or column as a string. Thus, you can use the function to sort row by row or column by column, but not to sort individual items within a row or column. When applied to a matrix, the grade down function produces a vector whose length is equal to the number of rows or columns in the matrix.

The following sorts the matrix *B* by rows and then sorts the matrix by columns

```
      ⎕←B←3 5⍴ 3 2 1 5 0 3 1 9 7 0 3 2 0 8 0
3 2 1 5 0
3 1 9 7 0
3 2 0 8 0
      ⍒B
1 3 2
      ⍒[2]B
1 3 2
      B[⍒B;]
3 2 1 5 0
3 2 0 8 0
3 1 9 7 0
      ⍒[1]B
4 1 2 3 5
      B[;⍒[1]B]
5 3 2 1 0
7 3 1 9 0
8 3 2 0 0
```

In this example, the original first row remains the first row, the third row becomes the second row, and the second row becomes the third row. Note that ⍒*B* and ⍒ [2] *B* are equivalent.

You can also sort character arrays by rows or by columns. For example:

```
      ⎕←B←3 5⍴'ALLENALAN ALLAN'
ALLEN
ALAN
ALLAN
      B[⍒B;]
ALLEN
ALLAN
ALAN
      B[;⍒[1]B]
NLLEA
 LANA
NLLAA
```

If the argument to ⍒ is a scalar, the ravel function is applied to extend it to a one-item vector, and the result of the ⍒ function is ,⎕*IO*:

```
      R←⍒5
      R
1
      ⍴⍴R
1
```

Note that ⎕*CT* is not an implicit argument to the grade down function.

Possible Errors Generated

```
9  RANK ERROR (NOT A SCALAR, VECTOR, OR MATRIX)

15  DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

27  LIMIT ERROR (INTEGER TOO LARGE)

28  AXIS RANK ERROR (NOT VECTOR DOMAIN)

29  AXIS LENGTH ERROR (NOT SINGLETON)

30  AXIS DOMAIN ERROR (AXIS LESS THAN INDEX ORIGIN)

30  AXIS DOMAIN ERROR (ARGUMENT RANK AND AXIS INCOMPATIBLE)

30  AXIS DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

30  AXIS DOMAIN ERROR (INCORRECT TYPE)

30  AXIS DOMAIN ERROR (NOT AN INTEGER)

30  AXIS DOMAIN ERROR (SEMICOLON LIST NOT ALLOWED)
```

⍒ Dyadic Grade Down

Form

A⍒*B*
⍒ is formed with ∇ and |

Left Argument Domain

Type	Character
Shape	Any
Depth	0 or 1 (simple)

Right Argument Domain

Type	Character
Shape	Any
Depth	0 or 1 (simple)

Result Domain

Type	Nonnegative integer
Rank	1
Shape	1↑ρ⍪*B*
Depth	1 (simple)

Implicit Arguments

⎕*IO* (*A*⍒*B* when ,⎕*IO* ← 1 is identical to 1+*A*⍒*B* when ,⎕*IO* ← 0)

Description

The dyadic ⍒ function returns a numeric vector whose items can be used to sort the items along the first axis of the right argument in descending order. (The sort is performed according to the collating sequence defined in *A*.) Grade down does not actually sort arrays; it creates a permutation vector of the index numbers of the argument array's items, and this vector can then be used to sort the array. If either argument is empty, the result of the grade function is ⍳1↑ρ*B*. If the length of the first axis of *B* is one, then the result is ,⎕*IO*.

If two or more items of the right argument have the same value, the order of the items is determined by their relative positions in the original array (this is known as a stable sort).

Sorting an array requires two steps. First, the array is the right argument to the grade function, and then the result is used to index the array. The left argument determines the order in which APL collates the items of the right argument; APL evaluates the collating sequence from right to left. For example:

```
      ALPHA1←'IVXLCDM'  ◊  N←'CMXIVCILI'
      X ← ⎕ ← ALPHA1⍒N
2 1 6 8 3 5 4 7 9
      N[X]
MCCLXVIII
      DATES←⎕BOX 'MCCLXVIII
VII
MLXXIII
DCCCXXIII
CLXVI
MDCLIII
CLXXI
XVIII'
      X ← ⎕ ← ALPHA1⍒DATES
6 1 3 4 7 5 8 2
      DATES[X;]
MDCLIII
MCCLXVIII
MLXXIII
DCCCXXIII
CLXXI
CLXVI
XVIII
VII
      HEX←' 0123456789ABCDEF'
      HD←⎕BOX '8E7
3DA
976
AE8
 F8
3D5
 40'
```

```
      X ← ⎕ ← HEX⍒HD
4 3 1 2 6 5 7
      HD[X;]
AE8
976
8E7
3DA
3D5
 F8
 40
```

To sort an array that contains more than one font, you can use sequences similar to the following, depending on the desired result:

```
                          ⍝Z SORTS AFTER Z̲ AND BEFORE Y̲
      ALPHA2←'AA̲BB̲CC̲DD̲EE̲FF̲GG̲HH̲II̲JJ̲KK̲LL̲MM̲NN̲OO̲PP̲
      QQ̲RR̲SS̲TT̲UU̲VV̲WW̲XX̲YY̲ZZ̲'
      WORDS←⎕BOX 'HOPE
N̲A̲S̲A̲L̲
H̲E̲E̲L̲
HELM
H̲EEL
NEST
N̲E̲A̲R̲
PALM'
      X ← ⎕ ← ALPHA2⍒WORDS
8 7 2 6 3 5 4 1
      WORDS[X;]
PALM
N̲E̲A̲R̲
N̲A̲S̲A̲L̲
NEST
H̲E̲E̲L̲
H̲EEL
HELM
HOPE
                          ⍝Z SORTS AFTER A̲ AND BEFORE Y
      ALPHA3←'ABCDEFGHIJKLMNOPQRSTUVWXYZA̲B̲C̲D̲E̲F̲G̲H̲I̲J̲K̲L̲
M̲N̲O̲P̲Q̲R̲S̲T̲U̲V̲W̲X̲Y̲Z̲'
      X ← ⎕ ← ALPHA3⍒WORDS
7 2 3 5 8 6 4 1
      WORDS[X;]
N̲E̲A̲R̲
N̲A̲S̲A̲L̲
H̲E̲E̲L̲
H̲EEL
PALM
NEST
HELM
HOPE
```

If any items appear in the right argument when they have not been specified in the left argument, APL considers them equal and places them at the end of the sort sequence. For example:

```
      ALPHA4←'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
      GAMES←⎕BOX 'FREEZE TAG
M̲O̲N̲O̲P̲O̲L̲Y̲
C̲H̲E̲S̲S̲
HIDE AND SEEK
BACKGAMMON
FRISBEE'
      X ← ⎕ ← ALPHA4⍒GAMES
2 3 4 6 5 1
      GAMES[X;]
M̲O̲N̲O̲P̲O̲L̲Y̲
C̲H̲E̲S̲S̲
HIDE AND SEEK
FRISBEE
BACKGAMMON
FREEZE TAG
```

When the left argument has a rank greater than one, each axis represents a level of comparison and the last axis receives the highest priority. For example, when the left argument has two rows, each containing an alphabet in a different font, APL gives higher priority to the order specified by the columns (last axis) than it gives to the fonts specified by the rows (first axis). For this reason, the word *HELM* precedes the word *H̲E̲E̲L̲* in the end result of the following example:

```
      ALPHA5←'ABCDEFGHIJKLMNOPQRSTUVWXYZ
A̲B̲C̲D̲E̲F̲G̲H̲I̲J̲K̲L̲M̲N̲O̲P̲Q̲R̲S̲T̲U̲V̲W̲X̲Y̲Z̲'
      WORDS←⎕BOX 'HOPE
N̲A̲S̲A̲L̲
H̲E̲E̲L̲
HELM
H̲EEL
NEST
N̲E̲A̲R̲
PALM'
      X← ⎕ ← ALPHA5⍒WORDS
7 2 3 5 8 6 1 4
      WORDS[X;]
N̲E̲A̲R̲
N̲A̲S̲A̲L̲
H̲E̲E̲L̲
H̲EEL
PALM
NEST
HOPE
HELM
```

Duplicate items, such as character blanks, in the left argument (*A*) may yield an unexpected collating sequence. APL compares the locations of a duplicate item and bases its position in the final collating sequence on this comparison. The final location of a duplicate item is the minimum value along each axis for each occurrence. For example, if a duplicate *W* appears at locations 1 1 3 and 2 1 2 in a three-dimensional array, then the position of the *W* in the final collating sequence is 1 1 2. If the position 1 1 2 is occupied by a value other than *W*, the two are treated as equivalents:

```
      ⎕←D←2 2 3⍴'ABCDEFGCIJKL'
ABC
DEF

GCI
JKL
      ⎕←B←4 3⍴'ABFAAFACFABF'
ABF
AAF
ACF
ABF
      B[D⍒B;]                    ⍝NOTE THAT C AND B ARE EQUIVALENT
ABF
ACF
ABF
AAF
```

In the following example, *D* appears at locations 1 2 and 2 1, and *B* appears at locations 1 1 and 2 2. In the final collating sequence, both are positioned at location 1 1 and are treated as equivalent values:

```
      ⎕←L←2 2⍴'BDDB'
BD
DB
      ⎕←R←5 2⍴'DBBDBDDBBD'
DB
BD
BD
DB
BD
      L⍒R                       ⍝D AND B ARE EQUIVALENT, NO CHANGE
1 2 3 4 5
```

For more information about how the dyadic grade function is implemented, see Smith, H.J., "Sorting - A New/Old Problem." *APL Quote Quad* 9 (June 1979) pp123-127.

Possible Errors Generated

```
10  LENGTH ERROR (ARGUMENT STRING IS TOO LONG)

15  DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

15  DOMAIN ERROR (INCORRECT TYPE)

30  AXIS DOMAIN ERROR (INCORRECT OPERATION)
```

⍋ Monadic Grade Up

Form

⍋*B* ⍋[*K*]*B*
⍋ is formed with ∆ and |

Argument Domain

Type	Homogeneous
Shape	Matrix, vector, or scalar (not singletons of rank > 2)
Depth	0 or 1 (simple)

Result Domain

Type	Nonnegative integer
Rank	1
Shape	(⌽⍴*B*)[*K*]
Depth	1 (simple)

Implicit Arguments

⎕*IO* (⍋*B* when ⎕*IO* ← 1 is identical to 1 + ⍋*B* when ⎕*IO* ← 0)

Description

The monadic ⍋ function returns a numeric vector whose items can be used to sort the items of the argument in ascending order. Thus, grade up does not actually sort arrays; it creates a permutation vector of the index numbers of the argument array's items, and this vector can then be used to sort the array.

Sorting a vector requires two steps. First, the vector is the argument to the grade up function, and then the result is used to index the vector:

```
      A←2 9 7 4 3 10 4
      ⎕←B←⍋A
1 5 4 7 3 2 6
      A[B]
2 3 4 4 7 9 10
```

If two or more items of a vector or matrix have the same value, the order of the items is determined by the relative positions of the items in the original array (this is called a stable sort). For character arguments, the collating sequence is determined by the value of ⎕AV. Note that for numeric arguments, the result is not ⎕CT-dependent.

When you use the grade up function to sort a matrix, APL treats each row or column as a string. Thus, you can use the function to sort row by row or column by column, but not to sort individual items within a row or column. When applied to a matrix, the result of the grade up function is a vector whose length is equal to the number of rows or columns in the matrix.

The following sorts the matrix B by rows and then by columns:

```
      B←3 5⍴3 2 1 5 0 3 1 9 7 0 3 2 0 8 0
      ⍋B
2 3 1
      ⍋[2]B
2 3 1
      B[⍋B;]
3 1 9 7 0
3 2 0 8 0
3 2 1 5 0
      ⍋[1]B
5 3 2 1 4
      B[;⍋[1]B]
0 1 2 3 5
0 9 1 3 7
0 0 2 3 8
```

In this example, the original second row becomes the first row, the third row becomes the second row, and the first row becomes the third row. Note that ⍋B and ⍋[2]B are equivalent. You can also sort character arrays by rows or by columns. For example:

```
      ⎕←B←3 5⍴'ALLENALLINALLAN'
ALLEN
ALLIN
ALLAN
      B[⍋B;]
ALLAN
ALLEN
ALLIN
      B[;⍋[1]B]
AELLN
AILLN
AALLN
```

If the argument to ⍋ is a scalar, the ravel function is applied to extend it to a one-item vector, and the result of the ⍋ function is ,⎕*IO*:

```
      R←⍋5
      R
1
      ⍴⍴R
1
```

Note that ⎕*CT* is not an implicit argument to the grade up function.

Possible Errors Generated

```
9  RANK ERROR (NOT A SCALAR, VECTOR, OR MATRIX)

15  DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

27  LIMIT ERROR (INTEGER TOO LARGE)

28  AXIS RANK ERROR (NOT VECTOR DOMAIN)

29  AXIS LENGTH ERROR (NOT SINGLETON)

30  AXIS DOMAIN ERROR (ARGUMENT RANK AND AXIS INCOMPATIBLE)

30  AXIS DOMAIN ERROR (AXIS LESS THAN INDEX ORIGIN)

30  AXIS DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

30  AXIS DOMAIN ERROR (INCORRECT TYPE)

30  AXIS DOMAIN ERROR (NOT AN INTEGER)

30  AXIS DOMAIN ERROR (SEMICOLON LIST NOT ALLOWED)
```

⍋ Dyadic Grade Up

Form

A⍋*B*
⍋ is formed with ∆ and |

Left Argument Domain

Type	Character
Shape	Any
Depth	0 or 1 (simple)

Right Argument Domain

Type	Character
Shape	Any
Depth	0 or 1 (simple)

Result Domain

Type	Nonnegative integer
Rank	1
Shape	1↑ρ⍪*B*
Depth	1 (simple)

Implicit Arguments

⎕*IO* (*A*⍋*B* when ⎕*IO* ← 1 is identical to 1 + *A*⍋*B* when ⎕*IO* ← 0)

Description

The dyadic ⍋ function returns a numeric vector whose items can be used to sort the items along the first axis of the right argument in ascending order. (The sort is performed according to the collating sequence defined in *A*.) Grade up does not actually sort arrays; it creates a permutation vector of the index numbers of the argument array's items, and this vector can then be used to sort the array. If either argument is empty, the result of the grade function is ⍳ 1↑ρ*B*. If the length of the first axis of *B* is one, then the result is ,⎕*IO*.

If two or more items of the right argument have the same value, the order of the items is determined by their relative positions in the original array (this is known as a stable sort).

Sorting an array is accomplished in two steps. First, the array is the right argument to the grade function, and then the result is used to index the array. The left argument determines the order in which APL collates the items of the right argument. For example:

```
      ALPHA1←'IVXLCDM'
      N←'CMXIVCILI'
      X ← ⎕ ← (⌽ALPHA1)⍋N
2 1 6 8 3 5 4 7 9
      N[X]
MCCLXVIII
      DATES←⎕BOX 'MCCLXVIII
VIII
MLXXIII
DCCCXXIII
CLXVI
MDCLIII
CLXXI
XVIII'
      X ← ⎕ ← ALPHA1⍋DATES
2 8 5 7 4 3 1 6
      DATES[X;]
VIII
XVIII
CLXVI
CLXXI
DCCCXXIII
MLXXIII
MCCLXVIII
MDCLIII
      HEX←' 0123456789ABCDEF'
      HD←⎕BOX '8E7
3DA
976
AE8
F8
3D5
40'
```

```
      X ← ⎕ ← HEX⍋HD
7 5 6 2 1 3 4
      HD[X;]
 40
 F8
3D5
3DA
8E7
976
AE8
```

To sort an array that contains more than one font, you can use sequences similar to the following, depending on the desired result:

```
      ALPHA2←⎕←'AA̲BB̲CC̲DD̲EE̲FF̲GG̲HH̲II̲JJ̲KK̲LL̲MM̲NN̲OO̲PP̲Q.
ZQRR̲SS̲TT̲UU̲VV̲WW̲XX̲YY̲ZZ̲'
AA̲BB̲CC̲DD̲EE̲FF̲GG̲HH̲II̲JJ̲KK̲LL̲MM̲NN̲OO̲PP̲Q.
ZQRR̲SS̲TT̲UU̲VV̲WW̲XX̲YY̲ZZ̲
      WORDS←⎕BOX 'HOPE
N̲A̲S̲A̲L̲
H̲E̲E̲L̲
HELM
H̲EEL
NEST
N̲E̲A̲R̲
PALM'
      X ← ⎕ ← ALPHA2⍋WORDS
4 1 5 3 6 2 7 8
      WORDS[X;]
HELM
HOPE
H̲EEL
H̲E̲E̲L̲
NEST
N̲A̲S̲A̲L̲
N̲E̲A̲R̲
PALM
      ALPHA3←⎕←'ABCDEFGHIJKLMNOPQRSTUVWXYZA̲B̲C̲D̲E̲F̲G̲H̲I̲J̲K̲L̲
M̲N̲O̲P̲Q̲R̲S̲T̲U̲V̲W̲X̲Y̲Z̲'
ABCDEFGHIJKLMNOPQRSTUVWXYZA̲B̲C̲D̲E̲F̲G̲H̲I̲J̲K̲L̲
M̲N̲O̲P̲Q̲R̲S̲T̲U̲V̲W̲X̲Y̲Z̲
      X ← ⎕ ← ALPHA3⍋WORDS
4 1 6 8 5 3 2 7
```

```
      WORDS[X;]
HELM
HOPE
NEST
PALM
H̲EEL
H̲E̲E̲L̲
N̲A̲S̲A̲L̲
N̲E̲A̲R̲
```

If any items appear in the right argument when they have not been specified in the left argument, APL considers them equal and places them at the end of the sort sequence. For example:

```
      ALPHA4←'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
      GAMES←⎕BOX 'FREEZE TAG
M̲O̲N̲O̲P̲O̲L̲Y̲
C̲H̲E̲S̲S̲
HIDE AND SEEK
BACKGAMMON
FRISBEE'
      X ← ⎕ ← ALPHA4⍋GAMES
5 1 6 4 2 3
      GAMES[X;]
BACKGAMMON
FREEZE TAG
FRISBEE
HIDE AND SEEK
M̲O̲N̲O̲P̲O̲L̲Y̲
C̲H̲E̲S̲S̲
```

When the left argument has a rank greater than one, each axis represents a level of comparison and the last axis receives the highest priority. For example, when the left argument has two rows, each containing an alphabet in a different font, APL gives higher priority to the order specified by the columns (last axis) than it gives to the fonts specified by the rows (first axis). For this reason, the word H̲E̲E̲L̲ precedes the word HELM in the end result of the following example:

```
      ALPHA5←2 26⍴'ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUWXYZ'
      WORDS←⎕BOX 'HOPE
NASAL
HEEL
HELM
HEEL
NEST
NEAR
PALM'
      X ← ⎕ ← ALPHA5⍋WORDS
5 3 4 1 2 7 6 8
      WORDS[X;]
HEEL
HEEL
HELM
HOPE
NASAL
NEAR
NEST
PALM
```

Duplicate items, such as character blanks, in the left argument (A) may yield an unexpected collating sequence. APL compares the locations of a duplicate item and bases its position in the final collating sequence on this comparison. The final location of a duplicate item is the minimum value along each axis for each occurrence. For example, if a duplicate W appears at locations 1 1 3 and 2 1 2 in a 3-dimensional array, then the position of the W in the final collating sequence is 1 1 2. If the position 1 1 2 is occupied by a value other than W, the two are treated as equivalents.

```
      ⎕←D←2 2 3⍴'ABCDEFGCIJKL'
ABC
DEF

GCI
JKL
      ⎕←B←4 3⍴'ABFAAFACFABF'
ABF
AAF
ACF
ABF
      B[D⍋B;]          ⍝NOTE THAT C AND B ARE EQUIVALENT
AAF
ABF
ACF
ABF
```

In the following example, *D* appears at locations 1 2 and 2 1, and *B* appears at locations 1 1 and 2 2. In the final collating sequence, both are positioned at location 1 1 and are treated as equivalent values.

```
      ⎕←L←2 2⍴'BDDB'
BD
DB
      ⎕←R←5 2⍴'DBBDBDDBBD'
DB
BD
BD
DB
BD
      L⍋R                  ⍝D AND B ARE EQUIVALENT, NO CHANGE
1 2 3 4 5
```

For more information about how the dyadic grade function is implemented, see Smith, H.J., "Sorting - A New/Old Problem," *APL Quote Quad* 9 (June 1979) pp123-127.

Possible Errors Generated

```
10 LENGTH ERROR (ARGUMENT STRING IS TOO LONG)

15 DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

15 DOMAIN ERROR (INCORRECT TYPE)

30 AXIS DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

30 AXIS DOMAIN ERROR (INCORRECT OPERATION)
```

ι Index Generator

Form

ι*B*

Argument Domain

Type	Nonnegative near-integer
Shape	Singleton
Depth	0 or 1 (simple)

Result Domain

Type	Nonnegative integer
Rank	1
Shape	,*B*
Depth	1 (simple)

Implicit Arguments

⎕*IO* (ι*B* when ⎕*IO* ← 1 is identical to 1 + ι*B* when ⎕*IO* ← 0)

Description

For an argument *B*, the monadic ι function generates a vector of *B* consecutive, ascending integers starting with the value of the index origin. For example:

```
      ⎕←A←ι4
1 2 3 4
      ρA
4
      2*ι12                           ⍝POWERS OF 2
2 4 8 16 32 64 128 256 512 1024 2048 4096
      2 5ρι10
1 2 3 4  5
7 8 9 10
      X←7 1 3 4
      ιρX
1 2 3 4
```

If the index origin is 1, the integers have values 1 through *B*; if the index origin is 0, the integers have values 0 through *B* - 1:

```
      ⎕IO
1
      ι5
1 2 3 4 5
      ⎕IO←0
      ι5
0 1 2 3 4
```

Regardless of the value of ⎕*IO*, ι0 is the numeric empty vector:

```
      ι0
                          (APL outputs a blank line)
      ρι0
0
```

Possible Errors Generated

```
9  RANK ERROR (NOT VECTOR DOMAIN)

10  LENGTH ERROR (NOT SINGLETON)

15  DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

15  DOMAIN ERROR (INCORRECT TYPE)

15  DOMAIN ERROR (NEGATIVE NUMBER NOT ALLOWED)

15  DOMAIN ERROR (NOT AN INTEGER)

27  LIMIT ERROR (INTEGER TOO LARGE)
```

ι Index Of

Form

A ι *B*

Left Argument Domain

Type	Any
Shape	Vector domain
Depth	Any

Right Argument Domain

Type	Any
Shape	Any
Depth	Any

Result Domain

Type	Nonnegative integer
Rank	ρ ρ *B*
Shape	ρ *B*
Depth	0 or 1 (simple)

Implicit Arguments

⎕*CT* (determines comparison precision)
⎕*IO* (*A* ι *B* when ⎕*IO* ← 1 is identical to 1 + *A* ι *B* when ⎕*IO* ← 0)

Description

The dyadic ι function returns the position of the first occurrence in the left argument of the corresponding items in the right argument. For example:

```
      4 9 6 8 ι 6 4
3 1
```

The result indicates that 6 is the third item in the left argument and 4 is the first item.

The result will always have the same shape as the right argument, so that an index is returned for each of the values in the right argument. If a particular value in the right argument does not appear in the left argument, APL supplies a value equal to the last index value of the left argument plus one. For example:

```
      'ABCDEFGH'ι'HEADER'
8 5 1 4 5 9
```

The value *R* does not appear in the left argument, so APL returns the value 9 (there are eight values in the left argument) for the position corresponding to *R*.

Note that the dyadic ι function is ⎕*IO*-dependent: when ⎕*IO* is 0, each item in the result is one less than when ⎕*IO* is 1:

```
      ⎕IO←0
      'ABCDEFGH'ι'HEADER'
7 4 0 3 4 8
```

If the right argument of the dyadic ι function is empty, the result is empty. If the left argument is empty, the result is all 1s (⎕*IO*←→1):

```
      (ι0)ι2 5ρι10
1 1 1 1 1
1 1 1 1 1
```

Note that comparisons of the items in the right and left arguments are defined in terms of the match (≡) function (and so are ⎕*CT*-dependent). Because match allows mixed-type arguments, you can compare characters with numbers. However, such a comparison is always false, so that if you use mixed-type arguments for dyadic ι, the items in the result will be equal to the last index value of the left argument plus one.

Further examples:

```
      ⎕←VIC←'ABC' 0
+---+ 0
|ABC|
+---+
      ⎕←VOOF← 'AB' 0 ¯3 'ABC' 99 1
+--+ 0 ¯3 +---+ 99 1
|AB|      |ABC|
+--+      +---+
      VOOF ι VIC
4 2
                          ⍝NOTE THAT DYADIC ι IS ⎕IO-DEPENDENT
      ⎕IO←0
      VOOF ι VIC
3 1
      ⎕←XIP← 0 ρ (1 2 3) 'ABC'
                                (APL outputs a blank line)
      ⎕←V←(1 2 3) 'ABC'
+-----+ +---+
|1 2 3| |ABC|
+-----+ +---+
      V ι XIP              ⍝EMPTY RIGHT ARGUMENT
                                (APL outputs a blank line)
      ρ V ι XIP
0
      XIP ι V              ⍝EMPTY LEFT ARGUMENT
0 0
```

Possible Errors Generated

9 *RANK ERROR (NOT VECTOR DOMAIN)*

∩ Intersection

Form

```
A∩B
```

Left Argument Domain

Type	Any
Shape	Any
Depth	Any

Right Argument Domain

Type	Any
Shape	Any
Depth	Any

Result Domain

Type	See explanation below
Rank	1
Shape	`ρ∪((,A)∊B)/,A`
Depth	–

Implicit Arguments

`⎕CT` (Determines comparison precision)

Description

The dyadic ∩ function returns the common items found in both arguments. The result is the intersection of the arguments with the duplicate items removed. Note that the order of the items in the result is not predictable. For example:

```
      'CBEFGH' ∩ 2 3⍴'ABCD'
CB
      (2 3⍴ 'ABCD') ∩ 'CBEFGH'
BC
      (⍳6) ∩ 5 7 3 4
3 4 5
      5 7 3 4 ∩ ⍳6
3 4 5
```

You can use the intersection function to remove duplicate items from an argument. However, the unique function is the preferred method for this task. For example:

```
      A←1 212 345 1 65 34 67 1 34   ⍝DUPLICATES ARE 1 AND 34
      A ∩ A
1 212 345 65 34 67
      ∪ A
1 212 345 65 34 67
```

The type of the result depends on the types of the arguments, as shown in the following table:

Argument	Resulting Type
Neither empty	Same as left argument
One empty	Same as nonempty argument
Both empty	Same as left argument

The ∩ function compares items in terms of the match (≡) function, which uses the value of ⎕CT. Since match allows mixed-type arguments, you can compare characters with numbers. However, such a comparison is always false, so that if you use mixed-type arguments for dyadic ∩, the result will be empty.

Note that the following definition applies: `A∩B ←→ ∪((,A)≡B)/,A`, where the order of the items may differ.

Further examples:

```
      ⎕←A←⊂,3
+-+
|3|
+-+
      ⎕←B←(1 2 5)
1 2 5
```

```
      ⎕←WRL←(⊂,3) (1 2 5) ¯1
+---+ +-----+ ¯1
|+-+| |1 2 5|
||3|| +-----+
|+-+|
+---+
      ⎕←MIC←2 2 ⍴ A B ¯1 0
+---+ +-----+
|+-+| |1 2 5|
||3|| +-----+
|+-+|
+---+
¯1    0
      MIC ∩ WRL                         ⍝ZERO NOT IN INTERSECT
+---+ +-----+ ¯1
|+-+| |1 2 5|
||3|| +-----+
|+-+|
+---+
      MIC ∪ ⊂,3         ⍝NO INTERSECTION BETWEEN TWO ARGUMENTS
+---+ +-----+ ¯1 0 +-+
|+-+| |1 2 5|      |3|
||3|| +-----+      +-+
|+-+|
+---+
      ⎕←VAN←(1 2 3) 'ABC' (⊂,1 2 3)     ⍝CREATE VAX
+-----+ +---+ +-------+
|1 2 3| |ABC| |+-----+|
+-----+ +---+ ||1 2 3||
              |+-----+|
              +-------+
      A←(1 2 3) 'A'                     ⍝CREATE NEW A
      A ∩ VAN
+-----+
|1 2 3|
+-----+
```

Possible Errors Generated

None.

≡ Match

Form

A≡*B*
≡ is formed with = and _

Left Argument Domain

Type	Any
Shape	Any
Depth	Any

Right Argument Domain

Type	Any
Shape	Any
Depth	Any

Result Domain

Type	Boolean
Rank	0
Shape	⍳ 0 (scalar)
Depth	0 (simple scalar)

Implicit Arguments

⎕*CT* (determines comparison precision)

Description

The dyadic ≡ function determines whether the two arguments are identical in rank, shape, and value. The result is a Boolean scalar: true, if the arguments are identical, and false if they are not. For example:

```
      'ABCD' ≡ 'ABCD'
1
      'ABCD' ≡ 'ACBD'
0
      'ABCD' ≡ 2 2⍴'ABCD'
0
      'A' ≡ ,'A'
0
      1 2 3 ≡ 1 2 3
1
      1 2 3 ≡ 1 2
0
      1 2 3 ≡ 1 2 2 3
0
      1 2 3 ≡ '1 2 3'
0
      '' ≡ ⍳0
0
```

The ≡ function compares the simple items in terms of the equal (=) function and identifies equal items based on the value of ⎕*CT*. For example:

```
      ⎕CT
1E¯15
      4 ≡ 4-5E¯16
1
```

Further examples:

```
      ⎕←A←4
4
      ⎕←B←⊂,4
+-+
|4|
+-+
      A ≡ B            ⍝NOTE DIFFERENCE BETWEEN ≡ AND =
0
      A = B
+-+
|1|
+-+
      ⎕←VIC←(1 2 3) (⊂,4)
+-----+ +---+
|1 2 3| |+-+|
+-----+ ||4||
        |+-+|
        +---+
      ⎕←N←⊂,4
+-+
|4|
+-+
      ⎕←RED←(1 2 3),N
1 2 3 +-+
      |4|
      +-+
      RED ≡ VIC
0
      ⎕←Q←⊂(1 2 3)
+-----+
|1 2 3|
+-----+
```

Possible Errors Generated

None.

⌹ Matrix Divide

Form

A⌹*B*
⌹ is formed with ⎕ and÷

Left Argument Domain

Type	Numeric
Shape	Matrix, vector, or scalar (not singletons of rank ≤ 2)
Depth	0 or 1 (simple)

Right Argument Domain

Type	Numeric
Shape	Matrix, vector, or scalar (not singletons of rank ≤ 2)
Depth	0 or 1 (simple)

Result Domain

Type	Numeric
Rank	0⌈¯2+(⍴⍴*A*)+⍴⍴*B*
Shape	(1↓⍴*B*),1↓⍴*A*
Depth	0 or 1 (simple)

Implicit Arguments

⎕*CT* (used in the test for singularity)

Description

For arguments *A* and *B*, the dyadic ⌹ function determines the generalized solution *R* to the linear system *A*=*B*+.⋆*R*. If *B* has more rows than columns, then dyadic ⌹ returns the least-squares solution to the linear system.

The matrix divide function treats scalars and vectors as one-column matrices (except when it is determining the shape of the result).

The following example shows the use of the matrix division function in solving the linear equations 3A+B=9 and 2A-B=1:

```
      X←9 1
      Y←2 2⍴3 1 2 ¯1
      X⌹Y
2 3
```

In the expression X⌹Y, Y is a matrix whose values are the coefficients of the equations, and X is a vector containing the constant terms 9 and 1.

The result is a vector in which the first item is the value of A in the linear equations, and the second is the value of B. The following example shows other uses of matrix divide, including a least-squares solution:

```
      ⎕←A←(,[1.5]2 5), 1
2 1
5 1
      B←10 19
      ⍴⎕←X←B⌹A
3 4
2
      A+.×X
10 19
      ⎕←A←(,[1.5]⍳5), 1
1 1
2 1
3 1
4 1
5 1
      ⎕PP
10
      B←2.001 2.998 4.002 4.997 6.01
      ⎕←X←B⌹A
1.0017 0.9965
      B-A+.×X
0.0028 ¯0.0019 0.0004 ¯0.0063 0.005
      ⎕←X←⌹A
¯0.2 ¯0.1 ¯9.356402631E¯19  0.1  0.2
 0.8  0.5  2.000000000E¯1  ¯0.1 ¯0.4
      X+.×A
1.000000000E0   ¯1.040834086E¯17
2.775557562E¯17  1.000000000E0
```

For more information about how the matrix divide function is implemented, see Jenkins, M. A., *The Solution of Linear Systems of Equations and Linear Least Squares Problems in APL.* New York: IBM Scientific Center, Technical Report No. pp320-2989, June 1970; and Businger, Peter, and Golub, Gene H. "Linear Least Squares Solutions by Householder Transformations." *Numerische Mathematik* 7 (1965) pp269-276.

Possible Errors Generated

```
9  RANK ERROR (NOT A SCALAR, VECTOR, OR MATRIX)

10  LENGTH ERROR (FEWER ROWS THAN COLUMNS)

10  LENGTH ERROR (NUMBER OF ROWS MUST MATCH)

15  DOMAIN ERROR (DIVISION BY ZERO)

15  DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

15  DOMAIN ERROR (INCORRECT TYPE)

15  DOMAIN ERROR (SINGULAR MATRIX)

27  LIMIT ERROR (FLOATING OVERFLOW)
```

⌹ Matrix Inverse

Form

⌹*B*
⌹ is formed with ⎕ and ÷

Argument Domain

Type	Numeric
Shape	Matrix, vector, or scalar (not singletons of rank ≤ 2)
Depth	0 or 1 (simple)

Result Domain

Type	Numeric
Rank	⍴⍴*B*
Shape	⌽⍴*B*
Depth	0 or 1 (simple)

Implicit Arguments

⎕*CT* (used in the test for singularity)

Description

The monadic ⌹ function inverts a matrix to facilitate matrix division and a variety of other matrix operations.

If the argument is a matrix, its rows must be linearly independent.

If the argument is a scalar or vector, the result is a scalar or vector, respectively, but the result's items are obtained by treating the argument as a one-column matrix. Formally expressed, for an argument *B* :

```
⌹B ←→((I,I)⍴1,(I←|↑⍴B)⍴0)⌹ (2↑ (⍴B),11))⍴B
```

Note that the matrix product of *B* and ⌹*B* is the identity array. Formally expressed, for an argument *B*:

```
B+.×⌹B←→I
```

For example:

```
      ⎕←A←÷(⍳3)∘.+¯1+⍳3
1            0.5          0.3333333333
0.5          0.3333333333 0.25
0.3333333333 0.25         0.2
      ⎕←X←⌹A
  9  ¯36   30
¯36  192 ¯180
 30 ¯180  180
      X+.×A
 1.000000000E0   2.220446049E¯16  1.665334537E¯16
¯4.440892099E¯15 1.000000000E0   ¯1.332267630E¯15
 4.440892099E¯15 2.220446049E¯15  1.000000000E0
```

For more information about how the matrix inverse function is implemented, see Jenkins, M. A., *The Solution of Linear Systems of Equations and Linear Least Squares Problems in APL.* New York: IBM Scientific Center, Technical Report No. pp320-2989, June 1970; and Businger, Peter and Golub, Gene H. "Linear Least Squares Solutions by Householder Transformations." *Numerische Mathematik* 7 (1965) pp269-276.

Possible Errors Generated

```
9  RANK ERROR (NOT A SCALAR, VECTOR, OR MATRIX)

10  LENGTH ERROR (THERE ARE FEWER ROWS THAN COLUMNS)

15  DOMAIN ERROR (DIVISION BY ZERO)

15  DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

15  DOMAIN ERROR (INCORRECT TYPE)

15  DOMAIN ERROR (SINGULAR MATRIX)

27  LIMIT ERROR (FLOATING OVERFLOW)
```

∊ Membership

Form

```
A∊B
```

Left Argument Domain

Type	Any
Shape	Any
Depth	Any

Right Argument Domain

Type	Any
Shape	Any
Depth	Any

Result Domain

Type	Boolean
Rank	⍴⍴A
Shape	⍴A
Depth	0 or 1 (simple)

Implicit Arguments

⎕CT (determines comparison precision)

Description

The dyadic ∊ function determines whether particular items of the left argument array occur as items of the right argument array. The result is a Boolean array whose shape is the same as that of the left argument: a 1 indicates that the corresponding item in the left array is present somewhere in the right array; a 0 indicates that the item is not present. For example:

```
      A←2 3⍴7 8 2 4 6 9
      A∊⍳6
0 0 1
1 1 0
```

The result identifies the items in *A* that are also items in ⍳6.

You can use the compression function (/) in conjunction with the membership function (∊) to identify the particular items that are members of both argument arrays:

```
      ⎕←A←'ABCDEFGH'∊'HEADED'
1 0 0 1 1 0 0 1
      A/'ABCDEFGH'
ADEH
```

Note that comparisons of the items in the right and left arguments are defined in terms of the match (≡) function (and so are ⎕*CT*-dependent). Since match allows mixed-type arguments, you can compare characters with numbers. However, such a comparison is always false, so that if you use mixed-type arguments for dyadic ∊, the result will be all 0 s.

Further examples:

```
      ⎕←ACT←(1 2 3) 'ABC' (⊂,4)
+-----+ +---+ +---+
|1 2 3| |ABC| |+-+|
+-----+ +---+ ||4||
              |+-+|
              +---+
      ⎕←BOY←2 2 ⍴ (⊂,4) 'BC' (1 2 3) 0
+---+   +--+
|+-+|   |BC|
||4||   +--+
|+-+|
+---+
+-----+ 0
|1 2 3|
+-----+
      BOY ∊ ACT
1 0
1 0
      ACT ∊ BOY
1 0 1
```

Possible Errors Generated

None.

⊃ Pick

Form

A⊃*B*

Left Argument Domain

Type	Nonnegative near-integer
Shape	Vector domain
Depth	Less than or equal to 2

Right Argument Domain

Type	Any
Shape	Any
Depth	Any

Result Domain

Type	Any
Rank	Any
Shape	Any
Depth	(≡*B*)-ρ*A* (provided *A* is along the deepest path)

Implicit Arguments

⎕*IO* (*A*⊃*B* when ⎕*IO*←1 is identical to (1+*A*)⊃*B* when ⎕*IO*←0)

Description

The dyadic ⊃ function selects and discloses an item from an existing array. The items in *A* specify the coordinates of items in *B*. For example:

```
      V←21 22 23 24 25 26
      2⊃V                      ⍝SELECT SECOND ITEM IN V
22
      V[2]                     ⍝NOTE SIMILARITY TO INDEXING
22
```

You can select an item from any depth in an enclosed array. The length of A determines the depth of the selected item: when A has one item, the selection is from the top level of B; when A has two items, the selection is from the second level; and so on. For example:

```
      ⎕←B←('1A' '1B') ('2A' '2B') ('3A' '3B')
+---------+ +---------+ +---------+
|+--+ +--+| |+--+ +--+| |+--+ +--+|
||1A| |1B|| ||2A| |2B|| ||3A| |3B||
|+--+ +--+| |+--+ +--+| |+--+ +--+|
+---------+ +---------+ +---------+
      ≡B
3
                          ⍝LEFT ARG LENGTH IS 1, PICK FROM TOP LEVEL
      ⎕←Z←2⊃B
+--+ +--+
|2A| |2B|
+--+ +--+
      ≡Z
2
      ⍴Z
2
      ⎕←X←2 2⊃B                 ⍝PICK FROM SECOND LEVEL
2B
      ≡X
1
      ⍴X
2
```

The length of each item of A is equal to the rank of the corresponding array in B. The first item in A has a length equal to the rank of B; the second item has a length equal to the rank of the array selected by the first item in A; the third item has a length equal to the rank of the array selected by the second item in A; and so on. In the following example, the rank of H is 2, and the rank of item H[1;2] is 3. To select an item from H[1;2], the first item of A must contain two elements, and the second item must contain three elements. When you pick from the top level of an array, A must have length 1, and if A is enclosed, the contents must be in the simple vector domain.

```
      □←H←2 2 ρ (10×⍳5) (2 3 4ρ⍳24) (2 2ρ100×⍳4) 1000
+--------------+ +-----------+
|10 20 30 40 50| | 1  2  3  4|
+--------------+ | 5  6  7  8|
                 | 9 10 11 12|
                 |           |
                 |13 14 15 16|
                 |17 18 19 20|
                 |21 22 23 24|
                 +-----------+
+-------+        1000
|100 200|
|300 400|
+-------+
      ≡H
2
      □←Z←((1 2) (2 2 3)) ⊃ H   ⍝PICK FROM SECOND LEVEL
19
      ≡Z
0
      (⊂1 2)⊃H
 1  2  3  4
 5  6  7  8
 9 10 11 12

13 14 15 16
17 18 19 20
21 22 23 24
      1 2⊃H
 10 LENGTH ERROR (LEFT ITEM LENGTH NOT EQUAL TO SELECTED ITEM RANK)
       1 2⊃H
       ^
```

When *B* and all the items in *B* are in the vector domain, then *A* is in the simple vector domain. When *A* is empty, then *A*⊃*B* ←→ *B*.

```
      □←F←'A' 'AN' ('ANT' ('ANTI' 'ANTIC'))
A +--+ +----------------------+
  |AN| |+---+ +--------------+|
  +--+ ||ANT| |+----+ +-----+||
       |+---+ ||ANTI| |ANTIC|||
       |      |+----+ +-----+||
       |      +--------------+|
       +----------------------+
      ≡F
4
      ρF
3
```

```
      ρ¨ F               ⍝SHAPE OF EACH OF F
++ +-+ +-+
|| |2| |2|
++ +-+ +-+
      ⎕←P←3 ⊃ F          ⍝PICK 3RD ITEM OF F
+---+ +--------------+
|ANT| |+----+ +-----+|
+---+ ||ANTI| |ANTIC||
      |+----+ +-----+|
      +--------------+
      ≡P
3
      ρP
2
      ⎕←Q←3 2 ⊃ F        ⍝PICK 1 LEVEL DEEPER
+----+ +-----+
|ANTI| |ANTIC|
+----+ +-----+
      ≡Q
2
      ρQ
2
      ⎕←R←3 2 1 ⊃ F      ⍝PICK ANOTHER LEVEL DEEPER
ANTI
      ≡R
1
      ρR
4
      ⎕←S←3 2 1 3 ⊃ F    ⍝PICK FROM 4TH LEVEL OF F
T
      ≡S
0
      ρS
                              (APL outputs a blank line)
      G←(⍳0)⊃F
      G≡F
1
```

When an item in *B* is a scalar, the corresponding item in *A* must be empty. For example:

```
      ⎕←X←2 2ρ'ABC' (⊂2 2ρ(1 2)(2 3)(3 4)(4 5)) 'XYZ' (⍳5)
+---+ +-------------+
|ABC| |+-----------+|
+---+ ||+---+ +---+||
      |||1 2| |2 3|||
      ||+---+ +---+||
      ||+---+ +---+||
      |||3 4| |4 5|||
      ||+---+ +---+||
      |+-----------+|
      +-------------+
+---+ +---------+
|XYZ| |1 2 3 4 5|
+---+ +---------+
      ≡X
4
      ρX
2 2
      ρ¨ X               ⍝SHAPE OF EACH OF X
+-+ ++
|3| ||
+-+ ++
+-+ +-+
|3| |5|
+-+ +-+
      (⊂ 1 2)⊃X          ⍝PICK X[1;2]
+-----------+
|+---+ +---+|
||1 2| |2 3||
|+---+ +---+|
|+---+ +---+|
||3 4| |4 5||
|+---+ +---+|
+-----------+
      (1 2) '' ⊃X        ⍝USE EMPTY TO PICK INTO SCALAR
+---+ +---+
|1 2| |2 3|
+---+ +---+
+---+ +---+
|3 4| |4 5|
+---+ +---+
      (1 2) '' (2 1)⊃X   ⍝PICK DEEPER
3 4
```

To select more than one item from an array, use pick with the each (¨) operator. For example:

```
      ⎕←Y←2 3ρ⊂[2]X←6 2ρ ,¨ 'ABCDEFGNIJKL'
+-------+ +-------+ +-------+
|+-+ +-+| |+-+ +-+| |+-+ +-+|
||A| |B|| ||C| |D|| ||E| |F||
|+-+ +-+| |+-+ +-+| |+-+ +-+|
+-------+ +-------+ +-------+
+-------+ +-------+ +-------+
|+-+ +-+| |+-+ +-+| |+-+ +-+|
||G| |N|| ||I| |J|| ||K| |L||
|+-+ +-+| |+-+ +-+| |+-+ +-+|
+-------+ +-------+ +-------+
      GETA←(1 1) 1
      GETL←(2 3) 2
      GETA ⊃ Y
A
      GETL ⊃ Y
L
      GETA GETL ⊃¨ ⊂Y    ⍝USE EACH TO PICK MULTIPLE ITEMS
+-+ +-+
|A| |L|
+-+ +-+
      GETA GETL ⊃¨ Y Y   ⍝THIS IS AN ALTERNATIVE FORM
+-+ +-+
|A| |L|
+-+ +-+
```

The following relationship between the take (↑) function and pick is true for any nonempty *B*: ↑*B* ←→ (⊂(ρρ*B*)ρ⎕*IO*)⊃*B*.

Possible Errors Generated

9 *RANK ERROR* (*LEFT ITEM NOT VECTOR DOMAIN*)

9 *RANK ERROR* (*NOT VECTOR DOMAIN*)

10 *LENGTH ERROR* (*LEFT ARGUMENT LENGTH GREATER THAN RIGHT ARGUMENT DEPTH*)

10 *LENGTH ERROR* (*LEFT ITEM LENGTH NOT EQUAL TO SELECTED ITEM RANK*)

14 *DEPTH ERROR* (*LEFT ARGUMENT DEPTH GREATER THAN* 2)

15 *DOMAIN ERROR* (*INCORRECT TYPE*)

15 *DOMAIN ERROR* (*INDEX LESS THAN INDEX ORIGIN*)

15 *DOMAIN ERROR* (*INDEX OUT OF RANGE*)

15 *DOMAIN ERROR (NOT AN INTEGER)*

27 *LIMIT ERROR (INTEGER TOO LARGE)*

, Ravel

Form

```
        ,B      ,[K] B
```

Argument Domain

Type	Any
Shape	Any
Depth	Any

Result Domain

Type	Same as argument
Rank	1 (for ,B)
Shape	×/ρB (for ,B)
Depth	1⌈≡B

Implicit Arguments

None.

Description

The monadic APL, function returns a vector made up of the items of the argument array, stored in row-major order (by increasing index position). For example:

```
      A←2 3ρ1 2 3 4 5 6
      ⎕←B←,A
1 2 3 4 5 6
      ρB
6
```

```
      ⎕←A←2 3 3 ρι18
 1  2  3
 4  5  6
 7  8  9

10 11 12
13 14 15
16 17 18
      ,A
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
      ρ,A
18
      ρA
2 3 3
```

If the argument array is a scalar, APL returns a vector that contains one item. Note the difference in the shape of a scalar and the shape of a scalar to which the ravel function has been applied:

```
      ρ4
                          (APL outputs a blank line)
      ρ,4
1
```

If the argument is a vector, APL returns a vector that is identical to the argument:

```
      A←ι6
      A
1 2 3 4 5 6
      ,A
1 2 3 4 5 6
```

If the argument is an empty array of any rank or shape, APL returns an empty vector that is the same type as the argument.

When you use ravel with an axis argument, *K* is in the vector domain and its items are numeric. The result depends on whether *K* is a noninteger singleton or a near-integer vector. (If *K* is a near-integer singleton, the shape of *B* is unchanged.)

When the axis argument is a noninteger singleton, APL inserts a new axis (of length one) in the indicated position. For example, if *K* is a fraction between 1 and 2, APL will insert an axis between the first and second axes of *B*. Note that *K* must be between ¯1+⎕*IO* and ⎕*IO*+ρρ*B*. The rank of the result is 1+ρρ*B*:

```
      A←2 3ρ9 8 7 6 5 4
      ,[1.5]A
9 8 7

6 5 4
      ρ,[1.5]A
2 1 3
```

If you specify a noninteger singleton axis when *B* is a scalar, the result is a one-item vector:

```
      ,[.5] 28
28
      ρ,[.5] 28
1
```

When the axis argument is a near-integer vector, APL merges the specified axes into a single axis. In this case, *K* must contain contiguous ascending axis numbers between ⎕*IO* and ρρ*B*. The rank of the result is 1+(ρρ*B*)-ρ,*K*. If *K* is empty, then the result is ((ρ*B*),1)ρ*B*. Note that ,[⍳ρρ*B*]*B* is the same as ,*B*:

```
      B←2 3 6ρ'SARAH SELLS SHELLSBETH  BUYS  BOATS '
      B
SARAH
SELLS
SHELLS

BETH
BUYS
BOATS
      ρB
2 3 6
      ,[2 3]B
SARAH SELLS SHELLS
BETH  BUYS  BOATS
      ρ,[2 3]B
2 18
```

If you want to add an axis to the end of the shape of an array, you can use ⍳0 as the axis argument. If you want to add an axis to the beginning of the shape of an array, you can use ¯.5+⎕*IO* as the axis argument:

```
      A←2 3ρ 9 7 6 5 4
      ,[⍳0]A
9
7
6
```

```
5
4
9
      ρ,[ι0]A
2 3 1
      ,[¯.5 + ⎕IO]A
9 7 6
5 4 9
      ρ,[¯.5 + ⎕IO]A
1 2 3
```

If you specify ⎕IO or ι0 as the axis argument when B is a scalar, the result is a one-item vector:

```
      ,[1] 28
28
      ρ,[1] 28
1
      ,[ι0] 6
6
      ρ,[ι0] 6
1
```

Further examples:

```
      ⎕←A←⊂.3
0.3
      ⎕←B←'ABC'
ABC
      ⎕←C←''

      ⎕←D←¯2
¯2
      ⎕←E←2 2 ρ A B C D
0.3 +---+
    |ABC|
    +---+
++  ¯2
||
++
      ρ E
2 2
      ,E
0.3 +---+ ++ ¯2
    |ABC| ||
    +---+ ++
      ρ ,E
4
      ,[1.5]E
0.3 +---+
    |ABC|
    +---+
++  ¯2
||
++
      ρ ,[1.5]E
2 1 2
      ,[0.5]E
0.3 +---+
    |ABC|
    +---+
++  ¯2
||
++
      ρ ,[0.5]E
1 2 2
```

Possible Errors Generated

27 *LIMIT ERROR* (*INTEGER TOO LARGE*)

28 *AXIS RANK ERROR* (*NOT VECTOR DOMAIN*)

29 *AXIS LENGTH ERROR* (*ARGUMENT RANK AND AXIS INCOMPATIBLE*)

29 *AXIS LENGTH ERROR* (*NOT SINGLETON*)

30 *AXIS DOMAIN ERROR* (*ARGUMENT RANK AND AXIS INCOMPATIBLE*)

30 *AXIS DOMAIN ERROR* (*AXIS LESS THAN INDEX ORIGIN*)

30 *AXIS DOMAIN ERROR* (*AXES NOT IN CONTIGUOUS ASCENDING ORDER*)

30 *AXIS DOMAIN ERROR* (*ENCLOSED ARRAY NOT ALLOWED*)

30 *AXIS DOMAIN ERROR* (*INCORRECT TYPE*)

30 *AXIS DOMAIN ERROR* (*SEMICOLON LIST NOT ALLOWED*)

⊤ Represent

Form

A⊤*B*

Left Argument Domain

Type	Numeric
Shape	Any
Depth	0 or 1 (simple)

Right Argument Domain

Type	Numeric
Shape	Any
Depth	0 or 1 (simple)

Result Domain

Type	Numeric
Rank	(⍴⍴*A*)+⍴⍴*B*
Shape	(⍴*A*),⍴*B*
Depth	0 or 1 (simple)

Implicit Arguments

None.

Description

The dyadic ⊤ function (known as represent or encode) represents an array in any number system. The left argument specifies the number system; the right argument specifies the array to be represented. For example, to represent the decimal value 7 as a four-digit binary number, specify the following:

```
      2 2 2 2⊤7
0 1 1 1
```

In the expression *A*⊤*B*, *A* can be considered as the representation rule to be applied to *B*. Each item of the vector *A* is defined in terms of the item immediately to its left. You can specify mixed bases in the left argument. For example, the represent function can express some number of inches in miles, yards, feet, and inches, or some number of milliseconds in days, hours, minutes, seconds, and milliseconds:

Thus, in representing a number as miles, yards, feet, and inches, the left argument specifies, from right to left, 12 inches in 1 foot, 3 feet in 1 yard, and 1760 yards in 1 mile. In the following example, a miles specification is not defined in terms of another quantity, so 0 is printed in the miles column.

```
       ⍝MILES, YARDS, FEET, INCHES
      0 1760 3 12⊤273125
4 546 2 5
       ⍝DAYS, HOURS, MINUTES, SECONDS, MILLISECONDS
      0 24 60 60 1000⊤713732523
8 6 15 32 523
```

The following examples of base 3 conversions demonstrate the specification of different numbers of columns in the left argument and illustrate the way in which negative numbers are represented:

```
      3 3 3 3⊤17              ⍝PRODUCES 3'S COMPLEMENT OF 17
0 1 2 2
      3 3 3 3⊤¯17             ⍝PRODUCES 3'S COMPLEMENT OF ¯17
2 1 0 1
```

Another useful application of ⊤ is to return the integer and fractional portions of a number:

```
      X←823.7513
      0 1⊤X
823 0.7513
```

The following are more examples of the use of the ⊤ function:

```
      A←⍉3 2⍴2 3
      B←5 2
      ⎕←R←A⊤B
1 0
1 0
1 0
```

```
2 2
2 2
2 2
      ρB
2
      ρA
2 3
      ρR
2 3 2
      C←2 2ρ865 429 103 692
      ⎕ ← X ← 10 10 10⊤C
8 4
1 6

6 2
0 9

5 9
3 2
      ρX
3 2 2
      3 1 2 ⍉ X
8 6 5
4 2 9

1 0 3
6 9 2
             ⍝PRODUCES 2'S COMPLEMENT OF 13
      2 2 2 2 2⊤13
0 1 1 0 1
             ⍝PRODUCES 2'S COMPLEMENT OF ¯13
      2 2 2 2 2⊤¯13
1 0 0 1 1
```

If A is a scalar, A⊤B is the same as A|B with ⎕CT←0. Note that ⎕CT is not an implicit argument to the represent function.

Possible Errors Generated

```
15 DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

15 DOMAIN ERROR (INCORRECT TYPE)

27 LIMIT ERROR (FLOATING OVERFLOW)
```

⍴ Reshape

Form

A⍴*B*

Left Argument Domain

Type	Nonnegative near-integer
Shape	Vector domain
Depth	0 or 1 (simple)

Right Argument Domain

Type	Any
Shape	Any
Depth	Any

Result Domain

Type	Same as right argument
Rank	⍴,*A*
Shape	,*A*
Depth	–

Implicit Arguments

None.

Description

The dyadic ⍴ function creates an array of items from the right argument taken in row-major order and arranged in the shape specified by the left argument. For example:

```
      2 3 ⍴ ⍳6                 ⍝2 ROWS, 3 COLUMNS
1 2 3
4 5 6
```

If the right argument does not contain enough items to fill an array that has the shape specified by the left argument, the right argument is reused starting at its beginning:

```
      3 3 ρ ι6
1 2 3
4 5 6
1 2 3
      3ρ5
5 5 5
```

If the right argument has more items than are required for an array that has the shape specified by the left argument, the extra items are ignored:

```
      2 2 ρ ι6
1 2
3 4
```

Note that the right argument may be any type and shape (it is, in effect, raveled before it is reshaped):

```
      ⎕←B←3 5ρ'STAN SAM  STEVE'
STAN
SAM
STEVE
      20ρB
STAN SAM  STEVESTAN
```

For arguments *A* and *B*, if *B* is empty, *A* must contain at least one 0 value, and the result is empty with the shape ,*A*. For example:

```
      ⎕←R←2 0 ρ ι0
                              (APL outputs a blank line)
      ρR
2 0
```

If *A* is empty, the result is a scalar whose value is the first item of *B* in row-major order; formally expressed:

```
(ι0)ρB ←→ ' 'ρB ←→ (,B)[⎕IO]
```

For example:

```
       (ι0)ρ5 7 9
5
```

Further examples:

```
      ⎕←VAN←'ABC' (1 2 3 4) 1.2 (⊂,3)
+---+ +-------+ 1.2 +---+
|ABC| |1 2 3 4|     |+-+|
+---+ +-------+     ||3||
                    |+-+|
                    +---+
      ρVAN
4
      ⎕←VAN←2 2 ρ VAN
+---+ +-------+
|ABC| |1 2 3 4|
+---+ +-------+
+---+
      |+-+|
      ||3||
      |+-+|
      +---+
      ρVAN
2 2
      ⎕←(⍳0) ρ VAN              ⍝EMPTY LEFT ARGUMENT
+---+
|ABC|
+---+
```

Possible Errors Generated

9 *RANK ERROR* (*NOT VECTOR DOMAIN*)

10 *LENGTH ERROR*

15 *DOMAIN ERROR* (*ENCLOSED ARRAY NOT ALLOWED*)

15 *DOMAIN ERROR* (*INCORRECT TYPE*)

15 *DOMAIN ERROR* (*NOT AN INTEGER*)

15 *DOMAIN ERROR* (*NEGATIVE NUMBER NOT ALLOWED*)

27 *LIMIT ERROR* (*INTEGER TOO LARGE*)

⌽ and ⊖ Reverse

Form

⌽*B*   ⌽[*K*]*B*   ⊖*B*   ⊖[*K*]*B*
⌽ is formed with ○ and |
⊖ is formed with ○ and -

Argument Domain

Type	Any
Shape	Any
Depth	Any

Result Domain

Type	Same as argument
Rank	ρρ*B*
Shape	ρ*B*
Depth	≡*B*

Implicit Arguments

None.

Description

The monadic ⌽ function returns the items of the argument array in reverse order along the relevant axis. You specify the axis to be reversed in square brackets. For example:

```
      ⎕←A←2 4ρ⍳8
1 2 3 4
5 6 7 8
      ⊖[1]A
5 6 7 8
1 2 3 4
      ⌽[2]A
4 3 2 1
8 7 6 5
```

If you do not specify an axis, ⌽ reverses the items along the last axis, and ⊖ reverses the items along the first axis. For example:

```
      ⎕←G←3 3⍴⍳9
1 2 3
4 5 6
7 8 9
      ⊖G
7 8 9
4 5 6
1 2 3
      ⌽G
3 2 1
6 5 4
9 8 7
```

The following reverses a matrix along both axes simultaneously:

```
      ⎕←X←2 3⍴⍳6
1 2 3
4 5 6
      ⌽⊖X
6 5 4
3 2 1
```

For singleton, vector, or empty arguments, both ⌽ and ⊖ return the same value. For an empty array or singleton, they return the original argument; for a vector, they return the items of the vector in reverse order. For example:

```
      ⌽5
5
      ⊖5
5
      ⌽⍳0
                              (APL outputs a blank line)
      ⌽1 1 1⍴6
6
      ⌽⍳5
5 4 3 2 1
      ⊖⍳5
5 4 3 2 1
```

Note that reverse is not the same as transpose:

```
      ⎕←X←2 3⍴1 2 3 4 5 6
1 2 3
4 5 6
      ⍉X
1 4
2 5
3 6
```

Further examples:

```
      ⎕←MIZZ←2 4 ⍴ 'ABC' 0 ¯1 2 'XYZ' 4 (⊂,3) 100
+---+ 0 ¯1    2
|ABC|
+---+
+---+ 4 +---+ 100
|XYZ|   |+-+|
+---+   ||3||
        |+-+|
        +---+
      ⊖[1]MIZZ
+---+ 4 +---+ 100
|XYZ|   |+-+|
+---+   ||3||
        |+-+|
        +---+
+---+ 0 ¯1    2
|ABC|
+---+
      ⌽[2]MIZZ
2   ¯1    0 +---+
            |ABC|
            +---+
100 +---+ 4 +---+
    |+-+|   |XYZ|
    ||3||   +---+
    |+-+|
    +---+
      ⌽⊖MIZZ
100 +---+ 4 +---+
    |+-+|   |XYZ|
    ||3||   +---+
    |+-+|
    +---+
2   ¯1    0 +---+
            |ABC|
            +---+
```

Possible Errors Generated

27	*LIMIT ERROR (INTEGER TOO LARGE)*
28	*AXIS RANK ERROR (NOT VECTOR DOMAIN)*
29	*AXIS LENGTH ERROR (NOT SINGLETON)*
30	*AXIS DOMAIN ERROR (ARGUMENT RANK AND AXIS INCOMPATIBLE)*
30	*AXIS DOMAIN ERROR (AXIS LESS THAN INDEX ORIGIN)*
30	*AXIS DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)*
30	*AXIS DOMAIN ERROR (INCORRECT TYPE)*
30	*AXIS DOMAIN ERROR (NOT AN INTEGER)*
30	*AXIS DOMAIN ERROR (SEMICOLON LIST NOT ALLOWED)*

⌽ and ⊖ Rotate

Form

`A⌽B     A⌽[K]B     A⊖B     A⊖[K]B`

⌽ is formed with ○ and |
⊖ is formed with ○ and -

Left Argument Domain

Type	Near-integer
Shape	Conforms to right argument `(ρA)≡(ρB)[ιρ ρB)~)K]`
Depth	`0` or `1` (simple)

Right Argument Domain

Type	Any
Shape	Any
Depth	Any

Result Domain

Type	Same as right argument
Rank	`ρρB`
Shape	`ρB`
Depth	`≡B`

Implicit Arguments

None.

Description

The dyadic ⌽ or ⊖ function rotates items along the relevant axis of the right argument in a way specified by the left argument. The rotation is cyclical and, for each axis, continues for the number of places specified by the corresponding item in the left argument. If the left argument is positive, the shift is to the left; if it is negative, the shift is to the right. For example:

```
      3⌽⍳5
4 5 1 2 3
      ¯3⌽⍳5
3 4 5 1 2
```

The axis to be rotated must be specified in square brackets, as in the following example:

```
      ⎕←A←3 5⍴⍳15
 1  2  3  4  5
 6  7  8  9 10
11 12 13 14 15
      2 1 4⌽[2]A
 3  4  5  1  2
 7  8  9 10  6
15 11 12 13 14
      2 1 0 2 3⌽[1]A
11  7  3 14  5
 1 12  8  4 10
 6  2 13  9 15
```

If no axis is specified, ⌽ rotates the items along the last axis, and ⊖ rotates the items along the first axis:

```
      ⎕←G←2 4⍴⍳8
1 2 3 4
5 6 7 8
      2 1⌽G
3 4 1 2
6 7 8 5
      1 2 0 1⊖G
5 2 3 8
1 6 7 4
```

Note that, in general, the shape of the left argument must be the same as the shape of the relevant axis in the right argument. If the left argument is a singleton, it is extended to conform to the relevant axis of the right argument. For example:

```
      2⌽2 5⍴⍳10
3 4  5 1 2
8 9 10 6 7
```

Further examples:

```
      ⎕←MIZZ←2 4 ρ 'ABC' 0 ¯1 1 'XYZ' 4 (⊂,3) 100
+---+ 0 ¯1    1
|ABC|
+---+
+---+ 4 +---+ 100
|XYZ|   |+-+|
+---+   ||3||
        |+-+|
        +---+
      1 3 ⌽ MIZZ
0   ¯1    1 +---+
            |ABC|
            +---+
100 +---+ 4 +---+
    |XYZ|   |+-+|
    +---+   ||3||
            |+-+|
            +---+
      ¯1 0 1 1 ⌽ [1]MIZZ
+---+ 0 +---+ 100
|XYZ|   |+-+|
+---+   ||3||
        |+-+|
        +---+
+---+ 4 ¯1    1
|ABC|
+---+
      ¯1 0 1 1 ⊖ MIZZ
+---+ 0 +---+ 100
|XYZ|   |+-+|
+---+   ||3||
        |+-+|
        +---+
+---+ 4 ¯1    1
|ABC|
+---+
```

Possible Errors Generated

9 *RANK ERROR (RANKS DIFFER BY MORE THAN ONE)*

10 *LENGTH ERROR (SHAPES OFF AXIS DO NOT MATCH)*

15 *DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)*

15 *DOMAIN ERROR (INCORRECT TYPE)*

15 *DOMAIN ERROR* (*NOT AN INTEGER*)

27 *LIMIT ERROR* (*INTEGER TOO LARGE*)

28 *AXIS RANK ERROR* (*NOT VECTOR DOMAIN*)

29 *AXIS LENGTH ERROR* (*NOT SINGLETON*)

30 *AXIS DOMAIN ERROR* (*AXIS LESS THAN INDEX ORIGIN*)

30 *AXIS DOMAIN ERROR* (*ENCLOSED ARRAY NOT ALLOWED*)

30 *AXIS DOMAIN ERROR* (*INCORRECT TYPE*)

30 *AXIS DOMAIN ERROR* (*NOT AN INTEGER*)

30 *AXIS DOMAIN ERROR* (*RIGHT ARGUMENT HAS WRONG RANK*)

30 *AXIS DOMAIN ERROR* (*SEMICOLON LIST NOT ALLOWED*)

ρ Shape

Form

ρ*B*

Argument Domain

Type	Any
Shape	Any
Depth	Any

Result Domain

Type	Nonnegative integer
Rank	1
Shape	ρρ*B*
Depth	1 (simple)

Implicit Arguments

None.

Description

The monadic ρ function returns a vector of nonnegative integers that represent the lengths of each of the axes of the argument array.

If the argument is a vector, APL returns an integer vector that represents the number of items in the vector:

```
      A←2 4 6 8
      ρA
4
      B←'ABCDEF'
      ρB
6
      ρ,9
1
```

If the argument is a matrix, APL returns the number of rows and columns:

```
     ⎕←A←2 3ρι6
1 2 3
4 5 6
     ρA
2 3
```

If the argument is a scalar, APL returns an empty numeric vector:

```
     K←3
     ρK
                    (APL outputs a blank line)
```

You can use the shape function to determine an array's rank. Because the shape function returns one item for each axis of the array, the shape of shape is an integer vector that represents the number of axes in the array:

```
     ⎕←A←5 6ρι30
 1  2  3  4  5  6
 7  8  9 10 11 12
13 14 15 16 17 18
19 20 21 22 23 24
25 26 27 28 29 30
     ρA
5 6
     ρρA
2
```

Further examples:

```
     ⎕←V←'XY' (1 2 3) ' '
+--+ +-----+
|XY| |1 2 3|
+--+ +-----+
     ρV
3
     ⎕←B←(2 0 ρ5)(' ')(ι0)
++   ++
||   ||
||   ++
++
     ρB
3
```

```
      ⎕←M←2 3 ρ 1 ('') 'ABC' 0 ¯2 4
1 ++ +---+
  || |ABC|
  ++ +---+
0 ¯2 4
      ρM
2 3
      ρ ρ M
2
```

Note that for all `B`: `ρρρB ←→ ,1`

Possible Errors Generated

None.

⊆ Subset

Form

A⊆*B*
⊆ is formed with ⊂ and _

Left Argument Domain

Type	Any
Shape	Any
Depth	Any

Right Argument Domain

Type	Any
Shape	Any
Depth	Any

Result Domain

Type	Boolean
Rank	0
Shape	⍳ 0 (scalar)
Depth	0 (simple scalar)

Implicit Arguments

⎕*CT* (determines comparison precision)

Description

The dyadic ⊆ function determines whether the right argument contains all the items in the left argument. The result is a Boolean scalar: true, if the left argument is a subset of the right argument, and false if it is not. Duplicate items in either argument do not affect the result. For example:

```
      ⎕←A←3 4 ⍴ 23 54 98 34 98 47 98 32 78 65 12 23
23 54 98 34
98 47 98 32
78 65 12 23
      A⊆ ⍳100
1
      A⊆ ⍳90
0
```

The ⊆ function compares items in terms of the match (≡) function, which uses the value of ⎕CT. Because match allows mixed-type arguments, you can compare characters with numbers. However, such a comparison is always false, so that if you use mixed-type arguments for dyadic ⊆, the result will be zero. For example:

```
      '23 24 25'⊆ 22 23 24 25 26
0
```

Further examples:

```
      ⎕←V←0 'AB' (1 2 3)
0 +--+ +-----+
  |AB| |1 2 3|
  +--+ +-----+
      ⎕←M←2 2 ⍴ (1 2 3) '0' 'AB' 'A'
+-----+ 0
|1 2 3|
+-----+
+--+    A
|AB|
+--+
      V ⊆ M        ⍝NOTE CHARACTER AND NUMERIC ZEROS
0
```

Note that the following definition applies: A⊆B ←→ ∧/,A∊B

Possible Errors Generated

None.

↑ Take

Form

```
A↑B     A↑[K]B
```

Left Argument Domain

Type	Near-integer
Shape	Vector domain
Depth	0 or 1 (simple)

Right Argument Domain

Type	Any
Shape	Any
Depth	Any

Result Domain

Type	Same as right argument
Rank	(ρ,A)⌈ρρB
Shape	|,A (if no explicit axis)
Depth	–

Implicit Arguments

Fill item

Description

The dyadic ↑ function builds an array by taking a specified number of items from an existing array. Each item in A corresponds to an axis in B. The value of each item in A specifies how many items to take from the axis. Thus, for A↑B, item A[K] is used to take values along the K th axis of B.

If an item in A is a positive integer *n*, APL takes the first *n* items from the appropriate axis of B. If an item in A is negative, APL takes the last *n* items from the appropriate axis of B.

```
      R←1 2 3 4
      2↑R                 ⍝TAKE FIRST TWO ITEMS OF R
1 2
      ¯2↑R                ⍝TAKE LAST TWO ITEMS OF R
3 4
```

Unless the right argument is a scalar, the number of items in *A* must equal the rank of *B* (⍴ ,*A* must equal ⍴ ⍴ *B*). (When the right argument is a scalar, it is extended to be a singleton of the appropriate rank.) If you use the axis form ([*K*]), the number of items in *A* must equal the length of *K*. (Examples of axis form are presented at the end of this section.) Thus, if the right argument is a matrix, the left argument must have two values:

```
      ⎕←R←3 3⍴⍳9
1 2 3
4 5 6
7 8 9
                         ⍝LEFT ARG MUST BE LENGTH 2
      2↑R
 10 LENGTH ERROR (LEFT LENGTH NOT EQUAL TO RIGHT RANK)
       2↑R
       ^
                         ⍝TAKE TWO ITEMS ALONG EACH AXIS
      2 2↑R
1 2
4 5
```

If the value of an item in *A* is greater than the length of the corresponding axis of *B*, APL pads the result array with fill items. This operation is known as overtake. For example:

```
      NUM←1 2 3
      CHA←'ABC'
                         ⍝OVERTAKE NUM, FILL ITEMS ARE ZEROS
      5↑NUM
1 2 3 0 0
                         ⍝OVERTAKE CHA, FILL ITEMS ARE BLANKS
                         ⍝CATENATE X TO SHOW END OF FILL ITEMS
      (5↑CHA),'X'
ABC  X
```

The fill items are determined by the prototype of each vector along the relevant axis. This is important for arrays of rank 2 or more because the fill item for a given position depends on the prototype of that particular column, row, or plane. The following expressions describe such an operation. Note where the fill items are blanks and where they are zeros. (Because the array *M* is simple, all the fill items are scalars. If *M* were enclosed, some of the fill items might also have been enclosed.)

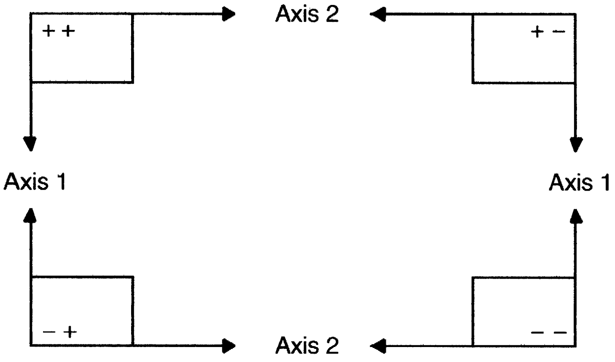
```
      ⎕←M←2 3⍴ 1 'A' 2 'B' 4 5
1 A 2
B 4 5
      ⍴M
2 3
          ⍝OVERTAKE M ALONG FIRST AXIS
          ⍝PROTOTYPE BASED ON VECTORS ALONG FIRST AXIS
      4↑[1]M
1 A 2
B 4 5
0   0
0   0
                          ⍝FIRST AXIS IS CHANGED
      ⍴4↑[1]M
4 3
                          ⍝OVERTAKE M ALONG 2ND AXIS
      5↑[2]M
1 A 2 0 0
B 4 5
      ⍴5↑[2]M            ⍝2ND AXIS IS CHANGED
2 5
      4 5↑M               ⍝OVERTAKE M ALONG BOTH AXIS
1 A 2 0 0
B 4 5
0   0 0 0
0   0 0 0
      ⍴4 5↑M              ⍝BOTH AXIS ARE CHANGED
4 5
```

Note that if *A* is positive, any needed fill items are placed at the end of the result array. If *A* is negative, any needed fill items precede the result array. For example:

```
      6↑12 24 36 48             ⍝FILL ITEMS AT END OF RESULT
12 24 36 48 0 0
      (10↑ 'TEST'),'X'      ⍝CATENATE X TO SHOW END OF FILL ITEMS
TEST      X
      ¯6↑12 24 36 48            ⍝FILL ITEMS AT BEGINNING OF RESULT
0 0 12 24 36 48
      ¯10↑ 'TEST'               ⍝FILL ITEMS AT BEGINNING OF RESULT
       TEST
```

If the rank of the right argument is greater than 1, the result array is called a corner of the argument array. The origin of the corner is determined by the signs of the items of the left argument. For example, if the right argument is a matrix, there are four possible corners as shown in Figure 1–1.

Figure 1–1 Argument Corners Selected by Take Function



NU–2233A–RA

In the following example, note how the order of the signs in the left argument determines the corner selected from the matrix right argument:

```
      ⎕←A←3 3⍴⍳9
1 2 3
4 5 6
7 8 9
      2 2↑A
1 2
4 5
      ¯2 ¯2↑A
5 6
8 9
      ¯2 2↑A
4 5
7 8
      2 ¯2↑A
2 3
5 6
```

If the left argument contains a 0, then, for arguments *A* and *B*, *A*↑*B* returns an empty array with shape , | *A*. For example:

```
      A←2 3 0↑2 3 3⍴⍳18
      ⍴A
2 3 0
```

If the left argument is empty, the right argument must be a scalar, and the result is the right argument.

If the right argument is a scalar, it is extended to a singleton with a rank equal to the length of the left argument. For example:

```
      ¯2 3 ↑ 5
0 0 0
5 0 0
```

Note that for any array *A*, `0=1↑0⍴A` is true if *A* is numeric and false if *A* is character.

When you use ↑ with an axis argument, *K* is a vector of axis numbers whose lengths are determined by corresponding items of the left argument, *A*. Formally, ↑ with an axis argument can be described by the following:

```
Z ← ⍴B ⋄ Z[K] ← A ⋄ Z ← Z↑B
```

The value for *K* must be in the vector domain, and each item must be a near-integer in the set `⍳⍴⍴B`. Therefore, the values of *K* are ⎕*IO*-dependent. The items may be in any order, but they may not be duplicated. The length of *K* must be less than or equal to the rank of the right argument, and it must match the length of *A*.

The value for *K* does not have to specify all the axes in *B*. APL determines the lengths of any missing axes by the lengths of the corresponding axes of *B*. This means that you can take rows or columns of a matrix without specifying the length of the other axis. For example:

```
      ⎕←A←8 5⍴⍳40
 1  2  3  4  5
 6  7  8  9 10
11 12 13 14 15
16 17 18 19 20
21 22 23 24 25
26 27 28 29 30
31 32 33 34 35
36 37 38 39 40
      3 ↑[1] A                ⍝TAKE 3 ROWS OF A
 1  2  3  4  5
 6  7  8  9 10
11 12 13 14 15
      ¯2 ↑[2] A               ⍝TAKE THE LAST 2 COLUMNS OF A
 4  5
 9 10
14 15
19 20
24 25
29 30
34 35
39 40
```

```
      3 4 ↑[2 1] A              ⍝TAKE 4 ROWS, 3 COLUMNS OF A
 1  2  3
 6  7  8
11 12 13
16 17 18
      ⎕IO ← 0
      4 3↑[0 1] A               ⍝TAKE 4 ROWS, 3 COLUMNS OF A
 1  2  3
 6  7  8
11 12 13
16 17 18
      ⎕←WRL←(1 2 3) 'ABC' 0
+-----+ +---+ 0
|1 2 3| |ABC|
+-----+ +---+
      5↑WRL                     ⍝OVERTAKE TO SHOW FILL ELEMENT
+-----+ +---+ 0 +-----+ +-----+
|1 2 3| |ABC|   |0 0 0| |0 0 0|
+-----+ +---+   +-----+ +-----+
```

Possible Errors Generated

9 *RANK ERROR (NOT VECTOR DOMAIN)*

10 *LENGTH ERROR (LEFT LENGTH NOT EQUAL TO RIGHT RANK)*

15 *DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)*

15 *DOMAIN ERROR (INCORRECT TYPE)*

15 *DOMAIN ERROR (NOT AN INTEGER)*

27 *LIMIT ERROR (INTEGER TOO LARGE)*

27 *LIMIT ERROR (VOLUME TOO LARGE)*

29 *AXIS LENGTH ERROR (LEFT ARGUMENT HAS WRONG LENGTH)*

30 *AXIS DOMAIN ERROR (ARGUMENT RANK AND AXIS INCOMPATIBLE)*

30 *AXIS DOMAIN ERROR (AXIS LESS THAN INDEX ORIGIN)*

30 *AXIS DOMAIN ERROR (DUPLICATE AXIS NUMBER)*

30 *AXIS DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)*

```
30 AXIS DOMAIN ERROR (INCORRECT TYPE)

30 AXIS DOMAIN ERROR (NOT AN INTEGER)

28 AXIS RANK ERROR (NOT VECTOR DOMAIN)

30 AXIS DOMAIN ERROR (SEMICOLON LIST NOT ALLOWED)
```

⍉ Monadic Transpose

Form

⍉*B*
⍉ is formed with ○ and \

Argument Domain

Type	Any
Shape	Any
Depth	Any

Result Domain

Type	Same as argument
Rank	ρρ*B*
Shape	⌽ρ*B*
Depth	≡*B*

Implicit Arguments

None.

Description

The monadic ⍉ function transposes the axes of an array; thus, ⍉*B* is *B* with the order of the axes reversed. For example, if the argument is a matrix, ⍉ exchanges rows and columns:

```
      ⎕←A←2 3ρ⍳6
1 2 3
4 5 6
      ⍉A
1 4
2 5
3 6
      ρ⍉A
3 2
```

If the rank of the argument is less than 2, the function has no effect:

```
      A←1 2 3 4 5
      ⍉A
1 2 3 4 5
```

In the next example, a rank 3 array is transposed:

```
      ⎕←B←2 3 4⍴⍳8
1 2 3 4
5 6 7 8
1 2 3 4

5 6 7 8
1 2 3 4
5 6 7 8
      ⍉B
1 5
5 1
1 5

2 6
6 2
2 6

3 7
7 3
3 7

4 8
8 4
4 8
      ⍴⍉B
4 3 2
```

Further examples:

```
      ⎕←MIZZ←2 4 ⍴ 'ABC' 0 ¯1 1 'XYZ' 4 (⊂,3) 100
+---+ 0 ¯1    1
|ABC|
+---+
+---+ 4 +---+ 100
|XYZ|   |+-+|
+---+   ||3||
        |+-+|
        +---+
      ⍉MIZZ
+---+ +---+
|ABC| |XYZ|
+---+ +---+
0     4
¯1    +---+
      |+-+|
      ||3||
      |+-+|
      +---+
1     100
      ⍴ MIZZ
2 4
      ⍴⍉MIZZ
4 2
```

Note that ⍉*B* ←→ (⌽⍴*B*)⍉*B*

Possible Errors Generated

None.

⍉ Dyadic Transpose

Form

A⍉*B*
⍉ is formed with ∘ and \

Left Argument Domain

Type	Nonnegative near-integer
Shape	Vector domain
Depth	0 or 1 (simple)

Right Argument Domain

Type	Any
Shape	Any
Depth	Any

Result Domain

Type	Same as right argument
Rank	*RANK*←⌈/*A* +~⎕*IO*
Shape	(1+ρ*A*)+⌊/((⍳*RANK*) ∘.=*A*) × (*RANK*,ρ*A*)ρρ*B*
Depth	≡*B*

Implicit Arguments

A⍉*B* when ⎕*IO* ← 1 is identical to (1 + *A*)⍉*B* when ⎕*IO* ← 0

Description

The dyadic ⍉ function permutes the axes of the right argument in a way specified by the left argument.

The shape (length) of the left argument must equal the rank of the right argument; thus, one item of the left argument is associated with each axis of the right argument. In general, the item in the left argument specifies the position to be assumed by the associated axis in the result array. For example:

```
      ⎕←A←2 3 4⍴⍳24
 1  2  3  4
 5  6  7  8
 9 10 11 12

13 14 15 16
17 18 19 20
21 22 23 24
      ⍴A
2 3 4
      ⎕←B←1 3 2⍉A
 1  5  9
 2  6 10
 3  7 11
 4  8 12

13 17 21
14 18 22
15 19 23
16 20 24
      ⍴B
2 4 3
      (⍴A)[1 3 2]
2 4 3
```

Note that the shape of the result of the ⍉ function is equal to the shape of its right argument subscripted by its left argument.

The values in the left argument must be less than or equal to the rank of the right argument; thus, if the right argument's rank is `3`, then `1`, `2`, and `3` are the only permissible values in the left argument (when `⎕IO` is `1`). However, there is one exception: if the right argument is a scalar, then either `1` (or `0` if `⎕IO` is `0`) or ⍳ 0 is permissible as the left argument; the value returned is the value of the scalar right argument.

You may repeat values in the left argument. When you do, the result is a diagonal slice of the right argument. For example:

```
      X←⎕←2 4 4⍴'*----*----*----*'
*---
-*--
--*-
---*

*---
-*--
--*-
---*
      1 2 2⍉X
****
****
      2 1 1⍉X
**
**
**
**
      Y←⎕←2 4 4⍴'****----------------'
****
----
----
----

----
****
----
----
      1 1 2⍉Y
****
****
      2 2 1⍉Y
**
**
**
**
      Z←⎕←2 4 4⍴'*---*---*---*----'
*---
*---
*---
*---

-*--
-*--
-*--
-*--
      1 2 1⍉Z
****
****
```

```
      2 1 2⍉Z
**
**
**
**
```

When you repeat values in the left argument, they must form a dense sequence; that is, in counting from 1 (or 0 if ⎕IO is 0) to the largest item you specify, no number may be left out.

Note that dyadic ⍉ is sometimes the same as monadic ⍉. Expressed formally, this means ⍉B ←→ (⌽ρB)⍉B. For example:

```
      ⎕←A←2 3ρι6
1 2 3
4 5 6
      ⍉A
1 4
2 5
3 6
      2 1⍉A
1 4
2 5
3 6
```

Table 1–6 lists transpositions for a variety of arrays: V is a vector, M is a matrix, and A is any array.

Table 1–6 Dyadic Transpose Definitions

Expression	Shape of R	Definition
R←1⍉V	ρV	R←V
R←1 2⍉M	ρM	R←M
R←2 1⍉M	(ρM)[2 1]	R[I;J]←M[J;I]
R←1 1⍉M	⌊/ρM	R[I]←M[I;I]
R←1 2 3⍉A	ρA	R←A
R←1 3 2⍉A	(ρA)[1 3 2]	R[I;J;K]←A[I;K;J]
R←2 3 1⍉A	(ρA)[3 1 2]	R[I;J;K]←A[J;K;I]
R←3 1 2⍉A	(ρA)[2 3 1]	R[I;J;K]←A[K;I;J]
R←1 1 2⍉A	(⌊/(ρA)[1 2]),(ρA)[3]	R[I;J]←A[I;I;J]

(continued on next page)

Table 1–6 (Cont.) Dyadic Transpose Definitions

Expression	Shape of *R*	Definition
R←1 2 1⍉*A*	(⌊/(⍴*A*)[1 3]),(⍴*A*)[2]	*R*[*I*;*J*]←*A*[*I*;*J*;*I*]
R←2 1 1⍉*A*	(⌊/(⍴*A*)[2 3]),(⍴*A*)[1]	*R*[*I*;*J*]←*A*[*J*;*I*;*I*]
R←1 1 1⍉*A*	⌊/⍴*A*	*R*[*I*]←*A*[*I*;*I*;*I*]

Further examples:

```
      ⎕←MIZZ←2 4⍴('ABC') 0 ¯1 1 ('XYZ') 4 (⊂,3) 100
+---+ 0 ¯1    1
|ABC|
+---+
+---+ 4 +---+ 100
|XYZ|   |+-+|
+---+   ||3||
        |+-+|
        +---+
      2 1 ⍉ MIZZ        ⍝THIS IS THE SAME AS MONADIC ⍉
+---+ +---+
|ABC| |XYZ|
+---+ +---+
0     4
¯1    +---+
      |+-+|
      ||3||
      |+-+|
      +---+
1     100
      ⎕←MIC←2 2 2 ⍴'AB' (1 20) 1 ¯2 0 '' 'A' 'XYZ'
+--+ +----+
|AB| |1 20|
+--+ +----+
1    ¯2


0    ++
     ||
     ++
A    +---+
     |XYZ|
     +---+
      1 2 2 ⍉ MIC
+--+ ¯2
|AB|
+--+
0    +---+
     |XYZ|
     +---+
      2 1 1 ⍉MIC
+--+ 0
|AB|
+--+
¯2   +---+
     |XYZ|
     +---+
      1 2 1 ⍉MIC
+--+ 1
|AB|
+--+
++   +---+
||   |XYZ|
++   +---+
      2 1 2 ⍉ MIC
+--+ ++
|AB| ||
+--+ ++
1    +---+
     |XYZ|
     +---+
```

Possible Errors Generated

9 *RANK ERROR (NOT VECTOR DOMAIN)*

10 *LENGTH ERROR (LEFT LENGTH NOT EQUAL TO RIGHT RANK)*

15 *DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)*

15 *DOMAIN ERROR (INCORRECT TYPE)*

15 *DOMAIN ERROR (LEFT ARGUMENT NOT DENSE FROM QUAD IO)*

15 *DOMAIN ERROR (NOT AN INTEGER)*

27 *LIMIT ERROR (INTEGER TOO LARGE)*

∪ Union

Form

```
AUB
```

Left Argument Domain

Type	Any
Shape	Any
Depth	Any

Right Argument Domain

Type	Any
Shape	Any
Depth	Any

Result Domain

Type	Any
Rank	1
Shape	ρ∪(,A),,B
Depth	1⌈≡A,B

Implicit Arguments

⎕*CT* (determines comparison precision)

Description

The dyadic ∪ function joins the two arguments and removes all duplicate items. The result is a vector that includes all the items from both arguments. For example:

```
      'ABCB' ∪ 2 3ρ'DDEDCC'
ABCDE
      (2 3 ρ 4 4 5 4 3 3 ) ∪ 1 2 3 2
4 5 3 1 2
```

The ∪ function compares the items in terms of the match (≡) function and eliminates duplicate items based on the value of ⎕*CT*.

Further examples:

```
      ⎕←V←(⊂,100) 'TTY' '99'
+-----+ +---+ +--+
|+---+| |TTY| |99|
||100|| +---+ +--+
|+---+|
+-----+
      ⎕←M←2 2 ρ 100 99 'TTY' 0
100   99
+---+ 0
|TTY|
+---+
      V ∪ M
+-----+ +---+ +--+ 100 99 0
|+---+| |TTY| |99|
||100|| +---+ +--+
|+---+|
+-----+
```

Note that the following definition applies: `A∪B←→∪(,A),,B`

Possible Errors Generated

None.

∪ Unique

Form

∪*B*

Argument Domain

Type	Any
Shape	Any
Depth	Any

Result Domain

Type	Same as argument
Rank	1
Shape	Equal to number of unique items
Depth	1⌈≡*B*

Implicit Arguments

⎕*CT* (determines comparison precision)

Description

The monadic ∪ function removes duplicate items from an array. The result is a vector of the unique items in the argument. For example:

```
      ⎕← A ← ? 3 4 ρ 7
6 5 4 2
7 2 6 2
1 3 7 6
      ∪ A
6 5 4 2 7 1 3
      B ← 'DR.GRANT''S CHEWING GUM'
      ∪B
DR.GANT'S CHEWIUM
```

The ∪ function compares the items in terms of the match (≡) function and eliminates duplicate items based on the value of ⎕*CT*. For example:

```
      ⎕CT
1E¯15
      ∪4 4-5E¯16
4
```

Note that the following definition applies: ∪*B*←→((*B*⍳*B*)=⍳⍴*B*)/*B*←,*B*

Possible Errors Generated

None.

~ Without

Form

```
A ~ B
```

Left Argument Domain

Type	Any
Shape	Any
Depth	Any

Right Argument Domain

Type	Any
Shape	Any
Depth	Any

Result Domain

Type	Any
Rank	1
Shape	`ρ(~(,A)∊B)/,A`
Depth	–

Implicit Arguments

⎕*CT* (determines comparison precision)

Description

The dyadic ~ function returns all the items in the left argument that are not found in the right argument. Duplicate items in the right argument do not affect the result. Duplicates in the left argument are not removed unless they are specified in the right argument. For example:

```
      RAMBLE←'RUN ON RUN ON RUN ON.'
      SQUISH←' '        ⍝SQUISH CONTAINS A BLANK SPACE
      RAMBLE ~ SQUISH   ⍝ELIMINATE THE BLANKS FROM RAMBLE
RUNONRUNONRUNON.
      ⎕←A←3 4⍴ 56 78 105 137 49 329 97 235 142 105 56 59
 56  78 105 137
 49 329  97 235
142 105  56  59
      A ~ ⍳100
105 137 329 235 142 105
```

If your data represent sets, and you want to remove duplicates from your result, you can use the unique function along with the ~ function:

```
      A←3 4⍴ 56 78 105 137 49 329 97 235 142 105 56 59
      ∪ A ~ ⍳100
105 137 329 235 142
```

If the left argument is a subset of the right argument, the result is an empty vector. For example:

```
      ⎕←B← 3 2 ⍴ 2 6 25 65 9 34 76 13 43 21
 2  6
25 65
 9 34
      ∪ B ~ ⍳30
65 34
```

The ~ function compares items in terms of the match (≡) function, which uses the value of ⎕*CT*. Because match allows mixed-type arguments, you can compare characters with numbers. However, such a comparison is always false, so that if you use mixed-type arguments for dyadic ~, the result will be equal to the left argument. For example:

```
      (A←'ABC') ~ B←'BA'
C
      B ~ A
                           (APL outputs a blank line)
```

Note that the following definition applies: `A~B↔(~(,A)∊B)/,A`

Possible Errors Generated

None.

1.3 APL Operators

APL operators take either functions or arrays as operands, and produce results called derived functions.

Operators are either monadic or dyadic, but not ambivalent. Monadic operators bind to the left; that is, they take a left operand and not a right operand. Dyadic operators take a left and a right operand. Derived functions are either monadic, dyadic, or ambivalent (their classification depends on the arguments to the derived function and not on the valence of the operator).

You can specify an axis when you use some of the operators. Because axis binds to the left, it must appear to the right of the operator.

There are four APL primitive operators: slash (`/` and `⌿`), backslash (`\` and `⍀`), each (`¨`), and dot (.). The following table describes the valence of the operators, the derived functions, and the valence of the derived functions. Note that `A`, `B`, *f*, and *g* are all operands where `A` and `B` are arrays, and *f* and *g* are functions.

Operator	**Valence**	**Derived Function**	**Valence**
Slash	Monadic	Compress (`A/` and `A⌿`)	Monadic
		Replicate (`A/` and `A⌿`)	Monadic
		Reduce (*f* / and *f*⌿)	Monadic
Backslash	Monadic	Expand (`A\` and `A⍀`)	Monadic
		Scan *f* \ and *f*⍀	Monadic
Each	Monadic	Itemwise application (*f*¨)	Ambivalent
Dot	Dyadic	Inner product (*f*.*g*)	Dyadic
		Outer product (∘.*f*)	Dyadic

Operators may accept functions or arrays for their operands. You can specify any valid function, including primitive functions, system functions, user-defined functions, and derived functions. (A derived function is a function resulting from the use of an operator.)

Because derived functions may be operands for operators, it is possible to build sequences of operators to form function expressions.

For example, you can use the inner product derived function (`+.×`) as the left operand to the slash operator (`/`). The result is the inner product reduce derived function, which allows you to perform matrix multiplication along a vector of matrices. Note that the left and right sides of the following expression are equivalent. However, the left side is more concise. The arrays `M1`, `M2`, and `M3` represent matrix arrays:

```
↑.×/M1 M2 M3 ←→ M1 +.× M2 +.× M3
```

The following example uses the outer product derived function (∘ . ,) as the left operand to the slash operator. The result is the catenate outer product reduce derived function, which in this case extends the monadic iota function (⍳) to vector arguments to produce the odometer function:

```
      V←1 2 3
      ,⊃ ∘., / ⍳¨ V
+-----+ +-----+ +-----+ +-----+ +-----+ +-----+
|1 1 1| |1 1 2| |1 1 3| |1 2 1| |1 2 2| |1 2 3|
+-----+ +-----+ +-----+ +-----+ +-----+ +-----+
```

The following expression adds parentheses to show the binding action of the operator sequence: `,⊃ ((∘.,) /) (⍳ ¨) V`

Table 1–7 summarizes the operators and derived functions in greater detail, including the forms with axis. The subsequent subsections describe all the forms.

Table 1–7 APL Operators

Operator	Name	Meaning
A/B	Slash	A compression/replication along the last axis of B
A/[K]B	Slash	A compression/replication along the K th axis of B
A⌿[K]B		
A⌿B	Slash	A compression/replication along the first axis of B first axis of A
f/A	Slash	The *f* reduction along the last axis of A
f/[K]A	Slash	The *f* reduction along the K th axis of A
f⌿[K]A		
f⌿A	Slash	The *f* reduction along the first axis of A
f¨B	Each	The application of monadic *f* on each item of B
A*f*¨B	Each	The application of dyadic *f* on corresponding pairs of each item of A and B
A\B	Backslash	A expansion along the last axis of B
A\[K]B	Backslash	A expansion along the K th axis of B
A⍀[K]B		

(continued on next page)

Table 1–7 (Cont.) APL Operators

Operator	Name	Meaning
A⍀B	Backslash	A expansion along the first axis of B
f\A	Backslash	The f scan along the last axis of A
f\[K]A	Backslash	The f scan along the K th axis of A
f⍀[K]A		
f⍀A	Backslash	The f scan along the first axis of A
A∘.fB	Dot	Outer product
Af.gB	Dot	Inner product

1.3.1 / and ⌿ Slash

The monadic slash (/ and ⌿) operator takes a left operand and produces a monadic derived function. When the operand is an array, the derived function is either compression or replication. When the operand is a function, the derived function is reduction.

1.3.2 \ and ⍀ Backslash

The monadic backslash (\ and ⍀) operator takes a left operand and produces a monadic derived function. When the operand is an array, the resulting function is expansion. When the operand is a function, the result is scan.

1.3.3 . The Dot Operator

The dyadic dot (.) operator takes a left and right operand and produces a dyadic-derived function. When the left operand is a jot (∘), the derived function is an outer product. When the left operand is a function, the derived function is an inner product. The right operand is always a dyadic function.

/ and ⌿ Compression and Replication

Form

`A/B     A/[K]B     A⌿B     A⌿[K]B`

⌿ is formed with / and -

Left Operand Domain

Type	Near-integer
Shape	Vector domain
Depth	0 or 1 (simple)

Right Argument Domain

Type	Any
Shape	Any
Depth	Any

Result Domain (of derived function)

Type	Same as right argument
Rank	1⌈ρρB
Shape	((K-1)↑ρB),(+/|A),K↓ρB (for ⎕IO←1)
Depth	–

Implicit Arguments

None.

Description

Compression and replication are monadic functions derived from the slash (/) operator. They build arrays by specifying the items to be deleted, preserved, or duplicated from an existing array, and by indicating where fill items are to be added in the new array. When items only are preserved or deleted, this is known as compression (the left operand is Boolean). When items are d(uplicated, deleted, or filled, this is known as replication (the left operand is integer). You can also use the ⎕REP system function to perform the compress and replicate operations (see Chapter 2 for more information).

For compression, each Boolean item in A corresponds to the position of an item in B. When A is 1, the item in B is preserved in the result array. When A is 0, the item in B is deleted from the result array.

```
      1 1 0 1 0/5 7 9 11 13   ⍝THIS IS COMPRESSION
5 7 11
```

For replication, each positive scalar and each zero in A correspond to the position of an item in B. Negative integers, which specify fill items, are not associated with explicit positions in B. When A is Boolean, the effects are the same as for compression (items are either preserved or deleted in the result array). When 1>A, the item in B is repeated A times in the result array. When A is negative, APL builds |A occurrences of the fill item into the new array:

```
      1 3 1 0 ¯4 2 ¯2/5 7 9 11 13         ⍝THIS IS REPLICATION
5 7 7 7 9 0 0 0 0 13 13 0 0
```

If A contains only 1 s, the result is B itself; if A contains only 0 s, the result is an empty array. For example:

```
      1 1 1 1 1/⍳5
1 2 3 4 5
      0 0 0 0 0/⍳5
                         (APL outputs a blank line)
```

In general, the length of the relevant axis of B must equal the number of nonnegative items in A ((⍴B)[K]←→+/A≥0). That is, you must specify an operation (either copy, drop, or replicate) for each item in the right argument. However, APL does perform singleton extension in certain conditions. If A is a positive singleton, it is extended to the length of B. (Negative values are not extended. When A is a negative singleton, B must be empty along the axis being replicated.) If B is a singleton, it is extended to the length of A.

```
      G←5 7 9 11 13
      K←1 1 0 1 0
      2/G               ⍝SINGLETON EXTENSION ON LEFT ARGUMENT
5 5 7 7 9 9 11 11 13 13
      K/5               ⍝SINGLETON EXTENSION ON RIGHT ARGUMENT
5 5 5
      ⎕←M←3 0 1⍴9
                           (APL outputs a blank line)
      ⍴M
3 0 1
      2 3/M             ⍝EXTENSION ON LAST (DEFAULT) AXIS
                           (APL outputs a blank line)
      ⍴ (2 3/M)         ⍝THIRD AXIS EXTENDED 2+3 TIMES
3 0 5
```

```
      ⍝NEXT EXPRESSIONS USE NEGATIVE SINGLETON IN LEFT ARGUMENT
      ⍝THE RIGHT ARGUMENTS MUST BE EMPTY ON APPLICABLE AXIS
      ¯2/⍳0
0 0
      ¯2/3 0⍴9           ⍝LAST (DEFAULT) AXIS IS EMPTY
0 0
0 0
0 0
      ¯2/3 3 0⍴9         ⍝AGAIN, LAST AXIS IS EMPTY
0 0
0 0
0 0

0 0
0 0
0 0

0 0
0 0
0 0
      G
5 7 9 11 13
      ¯2/G               ⍝RIGHT ARGUMENT NOT EMPTY
 10 LENGTH ERROR
       ¯2/G                    ⍝RIGHT ARGUMENT NOT EMPTY
       ∧
      ¯2/0 3⍴9           ⍝WRONG AXIS IS EMPTY
 10 LENGTH ERROR
       ¯2/0 3⍴9                ⍝WRONG AXIS IS EMPTY
       ∧
```

If B is a vector, all four forms of the compression function have the same effect. If the rank of B is greater than 1, the form used determines which axis of the array is affected.

For the forms A/[K]B and A⌿[K]B, the affected axis is axis K:

```
      ⎕←B←3 4⍴⍳12
1  2  3  4
5  6  7  8
9 10 11 12
      1 0 1/[1]B
1  2  3  4
9 10 11 12
```

```
      1 0 1⌿[1]B
1  2  3  4
9 10 11 12
      1 0 1 0/[2]B
1  3
5  7
9 11

      1 0 1 0⌿[2]B
1  3
5  7
9 11
```

The forms *A*/*B* and *A*⌿*B* affect the last and first axis of *B*, respectively:

```
      X←2 3⍴⍳6
      X
1 2 3
4 5 6
      0 1 1/X
2 3
5 6
      1 0⌿X
1 2 3
```

If *A* is empty, then *B* (after extension, if necessary) must have length 0 along the relevant axis.

If the left argument contains all negative numbers (indicating fill characters), then the applicable axis in the right argument must be empty, and the result will be the prototype of *B* repeated +/|*A* times along the axis. If the applicable axis is not empty, APL signals *LENGTH ERROR*. For example:

```
      ⎕←B←3 0⍴5
                          (APL outputs a blank line)
      ¯2 ¯3/B         ⍝CORRECT AXIS IS EMPTY
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
      ⎕←C←0 3⍴5
                          (APL outputs a blank line)
      ¯2 ¯3/C        ⍝INCORRECT AXIS IS EMPTY
10 LENGTH ERROR
       ¯2 ¯3/C        ⍝INCORRECT AXIS IS EMPTY
       ∧
```

APL inserts fill items that are determined by the prototype of each vector along the relevant axis. This is important for arrays of rank 2 or more because the fill item for a given position depends on the prototype of that particular column, row, or plane. The following expressions describe such an operation. Note where the fill items are blanks and where they are zeros. (Because the array *M* is simple, all the fill items are scalars. If *M* were enclosed, some of the fill items might also have been enclosed.)

```
                          ⍝CREATE M, A HETEROGENEOUS ARRAY OF RANK 3
      ⎕←M←2 2 3ρ 1 'A' 2 3 4 5 'A' 3 4 5 'B' 6
1 A 2
3 4 5

A 3 4
5 B 6
      ρM
2 2 3
      COL←1 ¯1 1 1
                ⍝REPLICATE M ALONG LAST AXIS (DEFAULT)
                ⍝PROTOTYPE BASED ON VECTORS ALONG LAST AXIS
      COL/M
1 0 A 2
3 0 4 5

A   3 4
5 0 B 6
      ρCOL/M              ⍝LAST AXIS IS CHANGED
2 2 4
      COL←1 ¯1 1
      COL/[2]M            ⍝REPLICATE M ALONG 2ND AXIS
1 A 2
0   0
3 4 5

A 3 4
  0 0
5 B 6
      ρCOL/[2]M           ⍝2ND AXIS IS CHANGED
2 3 3
      COL/[1]M            ⍝EXPAND M ALONG 1ST AXIS
1 A 2
3 4 5

0   0
0 0 0

A 3 4
5 B 6
      ρCOL/[1]M           ⍝1ST AXIS IS CHANGED
3 2 3
```

Further examples:

```
       ⎕←WRL←(1 2 3) 'ABC' 0
+-----+ +---+ 0
|1 2 3| |ABC|
+-----+ +---+
       1 1 0/WRL               ⍝COMPRESSION
+-----+ +---+
|1 2 3| |ABC|
+-----+ +---+
       3 ¯2 2 0 ¯1/WRL         ⍝REPLICATION
+-----+ +-----+ +-----+ +-----+ +-----+ +---+ +---+ +-----+
|1 2 3| |1 2 3| |1 2 3| |0 0 0| |0 0 0| |ABC| |ABC| |0 0 0|
+-----+ +-----+ +-----+ +-----+ +-----+ +---+ +---+ +-----+
```

Possible Errors Generated

7 *SYNTAX ERROR (NO DYADIC FORM OF DERIVED FUNCTION)*

9 *RANK ERROR (NOT VECTOR DOMAIN)*

10 *LENGTH ERROR*

15 *DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)*

15 *DOMAIN ERROR (INCORRECT TYPE)*

15 *DOMAIN ERROR (NOT AN INTEGER)*

27 *LIMIT ERROR (INTEGER TOO LARGE)*

28 *AXIS RANK ERROR (NOT VECTOR DOMAIN)*

29 *AXIS LENGTH ERROR (NOT SINGLETON)*

30 *AXIS DOMAIN ERROR (AXIS LESS THAN INDEX ORIGIN)*

30 *AXIS DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)*

30 *AXIS DOMAIN ERROR (INCORRECT TYPE)*

30 *AXIS DOMAIN ERROR (NOT AN INTEGER)*

30 *AXIS DOMAIN ERROR (RIGHT ARGUMENT HAS WRONG RANK)*

30 *AXIS DOMAIN ERROR (SEMICOLON LIST NOT ALLOWED)*

/ and ⌿ Reduction

Form

```
f/B     f/[K]B     f⌿B     f⌿[K]B
```

⌿ is formed with / and -

Left Operand Domain

Type	Dyadic value-returning function

Right Argument Domain

Type	Any
Shape	Any
Depth	Any

Result Domain (of derived function)

Type	Depends on *f*
Rank	`0⌈¯1+⍴⍴B`
Shape	`(⍴B)[(⍳⍴⍴B)~K]`
Depth	Depends on *f*

Implicit Arguments

None.

Description

Reduction is a monadic function derived from the slash (/) operator. To derive the reduction function, use any dyadic function as the operand (*f* in the form) to slash. *f* can be a primitive dyadic function, a dyadic system function, a dyadic user-defined function, or a dyadic-derived function. *f* cannot be a user-defined operator. The result operates as if *f* were applied between successive items along a specified axis of an array (B). For example:

```
      ⎕←X←ι6
1 2 3 4 5 6
      1+2+3+4+5+6
21
      +/X
21
      1×2×3×4×5×6
720
      ×/X
720
      </'A'
A
      </'AB'
1
      </'ABC'
 15 DOMAIN ERROR (INCORRECT TYPE)
       </'ABC'
       ∧
```

The reduction of a scalar always returns the scalar itself. Thus, the last expression in the preceding example results in a *DOMAIN ERROR* because 'B'<'C' evaluates to 1, and 'A'<1 is invalid because the data types do not match.

Remember that APL evaluates expressions from right to left. For example:

```
      </1 2 3
0
```

Here, APL evaluated 2<3 and the result was 1. APL then evaluated 1<1 and returned 0.

The result of the derived function has a rank that is one less than the rank of the original array (unless the original array is a scalar). Thus, the reduction of a matrix yields a vector, the reduction of a vector yields a scalar, and so forth.

For the forms *f*/ [K] B and *f*⌿ [K] B, the applicable axis is axis K:

```
      ⎕←A←2 4ρι6
1 2 3 4
5 6 1 2
      +/[2]A
10 14
      +/[1]A
6 8 4 6
```

Further examples:

```
      ⎕←Y←(1 2 3) (¯3 2 ¯2) (¯3 2 ¯2) (1 0 ¯1)
+-----+ +-------+ +-------+ +------+
|1 2 3| |¯3 2 ¯2| |¯3 2 ¯2| |1 0 ¯1|
+-----+ +-------+ +-------+ +------+
      +/Y
+-------+
|¯4 6 ¯2|
+-------+
      ,/⍳3                      ⍝SIMPLE ARG YIELDS NESTED RESULT
+-----+
|1 2 3|
+-----+
```

If the length of the K th axis is 1, the result of the derived function is the original array with the K th axis removed:

```
      ⎕←A←5 1⍴⍳5
1
2
3
4
5
      +/[1]A
15
      +/[2]A
1 2 3 4 5
```

The forms *f*/ B and *f*⌿ B affect the last and first axes of B, respectively:

```
      ⎕←A←2 4⍴⍳6
1 2 3 4
5 6 1 2
      +/A
10 14
      +⌿A
6 8 4 6
```

If the length of the applicable axis is 0, and all other axes have nonzero lengths, each result item is the identity function applied to the prototype of the argument, if one exists. The identity function for all scalar dyadic functions is *p*+*f*/ ⍳ 0 where *p* is the prototype of the right argument (*p*← ↑ 0⍴ B) and *f* is the scalar dyadic function. The identity elements for the identity function of the scalar dyadic functions are listed in the following table:

Identity Items for the Scalar Dyadic Functions

Dyadic Function	Symbol	Identity Item (*f*/⍳0)
Plus	+	0
Minus	-	0
Times	×	1
Divide	÷	1
Power	*	1
Residue	|	0
Maximum	⌈	Most negative representable number (¯1.7E38 approx)
Minimum	⌊	Largest representable positive number (1.7E38 approx)
Logarithm	⍟	None
Combination	!	1
Circle	○	None
And	∧	1
Or	∨	0
Nand	⍲	None
Nor	⍱	None
Less	<	0
Not Greater	≤	1
Equal to	=	1
Not Less	≥	1
Greater	>	0
Not Equal	≠	0

The identity functions for the nonscalar dyadic functions are listed in the following table. Note that `P` is the prototype of the argument `B` (defined formally as `P←↑0⍴B`). Any functions not listed (including system functions, user-defined operations, and derived functions from arbitrary operator sequences) do not have identity functions.

Identity Functions for the Nonscalar Dyadic Functions

Dyadic Function	Symbol	Identity Function
Reshape	⍴	⍴P
Catenate	,	((¯1↓⍴P),0)⍴⊂((¯1↓⍴P),0)⍴P
Rotate	⌽	(¯1↓⍴P)⍴0
Rotate	⊖	(1↓⍴P)⍴0
Transpose	⍉	⍳⍴⍴P
Pick	⊃	⍳0
Drop	↓	(⍴⍴P)⍴0
Take	↑	⍴P
Without	~	⍳0
Matrix Divide	⌹	(⍳↑⍴P)∘.=⍳↑⍴P

Possible Errors Generated

7 *SYNTAX ERROR (NO DYADIC FORM OF DERIVED FUNCTION)*

7 *SYNTAX ERROR (NO DYADIC FORM OF FUNCTION)*

11 *VALUE ERROR*

15 *DOMAIN ERROR*

15 *DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)*

15 *DOMAIN ERROR (FUNCTION HAS NO IDENTITY ITEM)*

15 *DOMAIN ERROR (INCORRECT TYPE)*

15 *DOMAIN ERROR (NOT A DYADIC FUNCTION)*

27 *LIMIT ERROR (INTEGER TOO LARGE)*

28 *AXIS RANK ERROR (NOT VECTOR DOMAIN)*

29 *AXIS LENGTH ERROR (NOT SINGLETON)*

30 *AXIS DOMAIN ERROR (ARGUMENT RANK AND AXIS INCOMPATIBLE)*

30 *AXIS DOMAIN ERROR (AXIS LESS THAN INDEX ORIGIN)*

```
30 AXIS DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

30 AXIS DOMAIN ERROR (INCORRECT TYPE)

30 AXIS DOMAIN ERROR (NOT AN INTEGER)

30 AXIS DOMAIN ERROR (SEMICOLON LIST NOT ALLOWED)
```

¨ Each

Form

f¨B   A*f*¨B

Left Operand Domain

Type	Any function

Left Argument Domain

Type	Depends on the function *f*
Shape	Depends on the function *f*
Depth	Depends on the function *f*

Right Argument Domain

Type	Depends on the function *f*
Shape	Depends on the function *f*
Depth	Depends on the function *f*

Result Domain (of derived function)

Type	Depends on the function *f*
Rank	ρρB (after singleton extension)
Shape	ρB (after singleton extension)
Depth	Depends on the function *f*

Implicit Arguments

None.

Description

The monadic ¨ operator (known as each) takes a function (*f*) as the left operand. The result is either a monadic or dyadic derived function (depending on the valence of *f*). *f* can be a primitive function, a system function, a user-defined operation, or a derived function from an arbitrary operator sequence. The function *f* does not have to be a value-returning function.

When you use ¨, the action of a monadic *f* is applied to successive items of an array (B in the form), and the action of a dyadic *f* is applied between corresponding pairs of items (A and B in the form). The action of *f* is only applied to the top level of nesting in an enclosed array (¨ is not pervasive).

```
      B ← 4
      C ← ⍳5
      D ← 2 2 ⍴ 'ABCD'
      ⎕ ← E ← B , (⊂C), ⊂D  ⍝NOTE USE OF PARENTHESES
4 +---------+ +--+
  |1 2 3 4 5| |AB|
  +---------+ |CD|
              +--+
      ⍴E                ⍝SHAPE OF E SHOWS A 3-ITEM VECTOR
3
      ⍴¨E    ⍝SHAPE OF EACH OF E SHOWS SHAPE OF ALL ITEMS
++ +-+ +---+
|| |5| |2 2|
++ +-+ +---+
      ≡E              ⍝DEPTH OF E SHOWS ONE NESTING LEVEL
2
      ≡¨E    ⍝DEPTH OF EACH OF E SHOWS DEPTH OF ALL ITEMS
0 1 1
      ⎕ ← E ← ⊂E
+------------------+
|4 +---------+ +--+|
|  |1 2 3 4 5| |AB||
|  +---------+ |CD||
|              +--+|
+------------------+
      ⍴¨⍴¨E              ⍝RANK OF EACH ITEM OF E
+-+
|1|
+-+
      ⎕ ← F ← ⊂E,E,E
+--------------------------------------------------------------+
|+------------------+ +------------------+ +------------------+|
||4 +---------+ +--+| |4 +---------+ +--+| |4 +---------+ +--+||
||  |1 2 3 4 5| |AB|| |  |1 2 3 4 5| |AB|| |  |1 2 3 4 5| |AB|||
||  +---------+ |CD|| |  +---------+ |CD|| |  +---------+ |CD|||
||              +--+| |              +--+| |              +--+||
|+------------------+ +------------------+ +------------------+|
+--------------------------------------------------------------+
      ⍴F                 ⍝SHAPE OF F SHOWS IT IS NOW A SCALAR
                             (APL outputs a blank line)
      ≡F     ⍝DEPTH OF F SHOWS ENCLOSED ARRAY, 3 NESTING LEVELS
4
```

```
                ⍝SHAPE OF EACH OF F SHOWS 1 VECTOR OF SHAPE 3)
      ρ¨F
+-+
|3|
+-+
                ⍝SHAPE OF EACH/EACH OF F SHOWS 3 VECTORS OF SHAPE 3
      ρ¨¨F
+-----------+
|+-+ +-+ +-+|
||3| |3| |3||
|+-+ +-+ +-+|
+-----------+
                              ⍝SHAPE OF EACH/EACH/EACH OF F
      ρ¨¨¨F
+--------------------------------------------+
|+------------+ +------------+ +------------+|
||++ +-+ +---+| |++ +-+ +---+| |++ +-+ +---+||
|||| |5| |2 2|| ||| |5| |2 2|| ||| |5| |2 2|||
||++ +-+ +---+| |++ +-+ +---+| |++ +-+ +---+||
|+------------+ +------------+ +------------+|
+--------------------------------------------+
      ∇Z←SH X
[1]   ⍝THIS USER-DEFINED FUNCTION RETURNS A RESULT
[2]   ⍝SH DETERMINES IF ARRAY X IS SIMPLE HOMOGENEOUS
[3]    Z←(2>≡X)∧ (↑Z)∧.=Z←↑¨0ρ¨,¨X
[4]    ∇
      SH 'A' 2 3
0
      SH ⍳5
1
      SH 2 2ρ'ABCD'
1 1
      SH (1 2) 3
0
      SH 'ABC' 5
0
```

The following example shows the use of system functions with the each operator. The example creates a vector of function definitions and then displays the canonical representation of each of the list of functions:

```
      X ← 2 2ρ'F 2 '
      X
F
2
      Y ← 2 3ρ'FOO1+2'
      Y
FOO
1+2
      )FNS
      ⎕FX¨ X Y
+-+ +---+
|F| |FOO|
+-+ +---+
      )FNS
F FOO
      ⎕CR ¨ 'F' 'FOO'
+-+ +---+
|F| |FOO|
|2| |1+2|
+-+ +---+
      ⎕BOX¨ ⎕VR¨ ⊂[2]⎕NL 3
+-------+ +----------+
|     ∇F| |      ∇FOO|
|[1]   2| |[1]    1+2|
|     ∇ | |      ∇   |
+-------+ +----------+
```

The next example shows the use of each to derive a dyadic function:

```
      X←'WENDY' 'STAN' 'PETER'
      X
+-----+ +----+ +-----+
|WENDY| |STAN| |PETER|
+-----+ +----+ +-----+
      (⊂⎕ALPHA) ⍋¨ X
+---------+ +-------+ +---------+
|4 2 3 1 5| |3 4 1 2| |2 4 1 5 3|
+---------+ +-------+ +---------+
      Y ← (⍳4) (⍳6) (⍳3)
      Y
+-------+ +-----------+ +-----+
|1 2 3 4| |1 2 3 4 5 6| |1 2 3|
+-------+ +-----------+ +-----+
      2 1 3 ⌽¨ Y
+-------+ +-----------+ +-----+
|3 4 1 2| |2 3 4 5 6 1| |1 2 3|
+-------+ +-----------+ +-----+
```

The next example shows each as part of a derived function:

```
      +/¨ (2 2⍴⍳4)(9 8 7)
+---+ 24
|3 7|
+---+
```

Possible Errors Generated

9 *RANK ERROR*

10 *LENGTH ERROR*

40 *OPERATOR DOMAIN ERROR (OPERAND TO EACH NOT A FUNCTION)*

\ and ⍀ Expansion

Form

A\B   *A*\\[*K*]*B*   *A*⍀*B*   *A*⍀[*K*]*B*
⍀ is formed with \ and -

Left Operand Domain

Type	Near-Boolean
Shape	Vector domain
Depth	0 or 1 (simple)

Right Argument Domain

Type	Any
Shape	Any
Depth	Any

Result Domain (of derived function)

Type	Same as argument
Rank	1⌈⍴⍴*B*
Shape	(⍴*B*)[(⍳⍴⍴*B*)~*K*] and ⍴,*A* on axis *K*
Depth	1⌈≡*B*

Implicit Arguments

None.

Description

Expansion is a monadic function derived from the backslash (\ operator. It builds an array by combining the items of an existing array with fill items. You can also use the ⎕*EXP* system function to perform the expand operation (see Chapter 2 for more information).

Each item in the operand (*A* in the form) is a Boolean scalar that corresponds to the position of an item in the right argument (*B*). When *A* is 1, APL inserts the corresponding item along the relevant axis of *B* into the result array. When *A* is 0, APL inserts a fill item into that position in the result array. There must be a 1 for each item along the relevant axis in the right argument, so that all the items in *B* appear in the result array. Any number of fill items may be included:

```
      ⎕←LIS←12 13 15
12 13 15
      V←1 0 1 0 1
                           ⍝ZEROS IN V DECIDE LOCATION OF FILL ITEMS
      V\LIS
12 0 13 0 15
```

A singleton right argument is extended along the axis to a length that matches the number of 1 s in the left argument:

```
      1 0 1\5
5 0 5
```

If the left argument is a singleton, APL signals an error:

```
      1\5 6 7 8
  10 LENGTH ERROR
       1\5 6 7 8
       ^
      0\5 6 7 8
  10 LENGTH ERROR
       0\5 6 7 8
       ^
```

If the right argument is a vector, all four forms of the expansion function have the same effect. If the rank of the right argument is greater than 1, the form used determines which axes of the array are affected.

For the forms *A*\[*K*]*B* and *A*⍀[*K*]*B*, the affected axis is axis *K*:

```
      ⎕←A←2 3⍴⍳6
1 2 3
4 5 6
      1 0 1\[1]A                ⍝EXPAND ALONG 1ST AXIS
1 2 3
0 0 0
4 5 6
```

```
      1 0 1⍀[1]A              ⍝EXPAND ALONG 1ST AXIS
1 2 3
0 0 0
4 5 6
      1 0 1 1\[2]A            ⍝EXPAND ALONG 2ND AXIS
1 0 2 3
4 0 5 6
      1 0 1 1⍀[2]A            ⍝EXPAND ALONG 2ND AXIS
1 0 2 3
4 0 5 6
```

The forms `A\B` and `A⍀B` affect the last and first axis of `B`, respectively:

```
      X←3 9⍴'*THISISANEXPANSIONEXAMPLE**'
      X
*THISISAN
EXPANSION
EXAMPLE**
      ⍴X
3 9
      V←1 1 1 1 1 0 1 1 0 1 1
      V\X                     ⍝EXPAND X ALONG LAST AXIS
*THIS IS AN
EXPAN SI ON
EXAMP LE **
      1 0 1 1⍀X               ⍝EXPAND X ALONG FIRST AXIS
*THISISAN

EXPANSION
EXAMPLE**
```

When you expand an array, APL uses fill items that are determined by the prototype of each vector along the relevant axis. This is important for arrays of rank 2 or more because the fill item for a given position depends on the prototype of that particular column, row, or plane. The following expressions describe such an operation. Note where the fill items are blanks and where they are zeros. (Because the array `M` is simple, all the fill items are scalars. If `M` were enclosed, some of the fill items might also have been enclosed.)

```
      ⎕←M←2 2 3⍴ 1 'A' 2 3 4 5 'A' 3 4 5 'B' 6
1 A 2
3 4 5

A 3 4
5 B 6
      ⍴M
2 2 3
      BOO←1 1 0 1
```

```
                         ⍝EXPAND M ALONG LAST AXIS (DEFAULT)
                  PROTOTYPE BASED ON VECTORS ALONG LAST AXIS
      BOO\M
1 A 0 2
3 4 0 5

A 3   4
5 B 0 6
      ρBOO\M             ⍝LAST AXIS IS CHANGED
2 2 4
      BOO←1 0 1
      BOO\[2]M           ⍝EXPAND M ALONG 2ND AXIS
1 A 2
0   0
3 4 5

A 3 4
  0 0
5 B 6
      ρBOO\[2]M          ⍝2ND AXIS IS CHANGED
2 3 3
      BOO\[1]M           ⍝EXPAND M ALONG 1ST AXIS
1 A 2
3 4 5

0   0
0 0 0

A 3 4
5 B 6
      ρBOO\[1]M          ⍝1ST AXIS IS CHANGED
3 2 3
```

Note that the right argument may be empty:

```
      0 0 0\⍳0
0 0 0
      ⎕←A←0 0 0\''
                               (APL outputs a blank line)
      ρA
3
```

If the left argument is empty, the right argument (after extension, if necessary) must have length 0 along the relevant axis.

For a simple, homogeneous array `A`, the result of the expression `0=0\0ρA` is 1 if `A` is numeric, and 0 if `A` is character. For any array `X`, the result of the expression `(2>≡X)∧(↑Z)∧.=Z←↑¨0ρ¨,¨X` is 1 if `X` is simple or homogeneous, and 0 if `X` is either nonsimple or heterogeneous.

Further examples:

```
      ⎕←WRL←(1 2 3) ('ABC') 0
+-----+ +---+ 0
|1 2 3| |ABC|
+-----+ +---+
      1 0 1 0 1\WRL
+-----+ +-----+ +---+ +-----+ 0
|1 2 3| |0 0 0| |ABC| |0 0 0|
+-----+ +-----+ +---+ +-----+
```

Possible Errors Generated

7 *SYNTAX ERROR* (*NO DYADIC FORM OF DERIVED FUNCTION*)

9 *RANK ERROR* (*NOT VECTOR DOMAIN*)

10 *LENGTH ERROR*

15 *DOMAIN ERROR*

15 *DOMAIN ERROR* (*ENCLOSED ARRAY NOT ALLOWED*)

15 *DOMAIN ERROR* (*INCORRECT TYPE*)

27 *LIMIT ERROR* (*INTEGER TOO LARGE*)

28 *AXIS RANK ERROR* (*NOT VECTOR DOMAIN*)

29 *AXIS LENGTH ERROR* (*NOT SINGLETON*)

30 *AXIS DOMAIN ERROR* (*AXIS LESS THAN INDEX ORIGIN*)

30 *AXIS DOMAIN ERROR* (*ENCLOSED ARRAY NOT ALLOWED*)

30 *AXIS DOMAIN ERROR* (*INCORRECT TYPE*)

30 *AXIS DOMAIN ERROR* (*NOT AN INTEGER*)

30 *AXIS DOMAIN ERROR* (*RIGHT ARGUMENT HAS WRONG RANK*)

30 *AXIS DOMAIN ERROR* (*SEMICOLON LIST NOT ALLOWED*)

\ and ⍀ Scan

Form

f< *B*   *f*[*K*] *B*   *f*⍀ *B*   *f*⍀ [*K*] *B*
⍀ is formed with \ and -

Left Operand Domain

Type	Dyadic value-returning function

Argument Domain

Array	Any
Type	Any
Shape	Any
Depth	Any

Result Domain (of derived function)

Type	Depends on *f*
Rank	⍴⍴*B*
Shape	⍴*B*
Depth	Depends on *f*

Implicit Arguments

None.

Description

Scan is a monadic function derived from the backslash (\ operator. To derive the scan function, use any dyadic function as the operand (*f* in the form) to backslash. *f* can be a primitive dyadic function, a dyadic system function, a dyadic user-defined function, or a dyadic-derived function from an arbitrary operator sequence. The result operates as if *f* were applied between successive items along a specified axis of an array (*B*). Thus, a scan of an array works the same as a reduction, except that the scan returns the results as the function is applied to each successive group of items.

The result has the same shape as B. The first item of the result is always identical to the first item of B, and the last item is equal to the *f* reduction of B. For example:

```
      +\3 4 5
3 7 12
```

As the function is applied to each successive group of items, APL evaluates the resultant expressions from right to left:

```
      -\1 2 3 4
1 ¯1 2 ¯2
```

Here, APL returned the following:

- The first item in the argument array
- The result of the expression 1-2
- The result of the expression 1-2-3
- The result of the expression 1-2-3-4

Note that APL treated each expression in the example independently; for example, the result of the expression 1-2 did not affect the evaluation of the expression 1-2-3.

If B is an empty array, the result is an empty array.

For the forms *f*\ [K] B and *f*⍀ [K] B, the applicable axis for the scan is axis K:

```
      ⎕←A←2 4⍴⍳6
1 2 3 4
5 6 1 2
      +\[2]A
1  3  6 10
5 11 12 14
      +\[1]A
1 2 3 4
6 8 4 6
```

The forms *f*\ B and *f*⍀ B affect the last axis and first axis of B, respectively:

```
      ⎕←A←2 4⍴⍳6
1 2 3 4
5 6 1 2
      +\A
1  3  6 10
5 11 12 14
      +⍀A
1 2 3 4
6 8 4 6
```

Note that the scan operator is never applied if B has the length 1. Thus, +\'A'←→'A'. Also note that for =\'AB' ←→ 'A',0 is heterogeneous, because the first item of the result would be a character ('A'), and the second item would be a number 0, the result of 'A'='B'.

If the dyadic function specified with scan is one of the associative primitive functions (+, ×, ⌊, ⌈, < , and ∨ for all arguments; = and ≠ for Boolean arguments), APL uses an optimization that changes the way scan is computed. The definition of R←f\B (for vectors R[K]=f/K↑B0) is changed as follows:

```
 R[1] = B[1]
 R[K] = R[K-1] f  B[K] for  K∊1↓⍳⍴B
```

This optimized scan requires fewer operations than the traditional scan. Note that the result of an associative operation may differ slightly from the nonassociative approach, and you should use it carefully if your results require a high degree of precision. For example:

```
      A←1E6 ¯1E6 1E¯16
      A
1000000 ¯1000000 1E¯16
      +\A
1000000 0 1E¯16
      +/A
0
```

Further examples:

```
      ⎕←W←(2 2 3) (2 1 0)
+-----+ +-----+
|2 2 3| |2 1 0|
+-----+ +-----+
      ⍴\W
+-----+ +-----+
|2 2 3| |2 1 0|
+-----+ |2 1 0|
        |     |
        |2 1 0|
        |2 1 0|
        +-----+
      ρ\3 2 1            ⍝SIMPLE ARG YIELDS NESTED RESULT
3 +-----+ +-----+
  |2 2 2| |1 1 1|
  +-----+ +-----+
```

Possible Errors Generated

7 *SYNTAX ERROR (NO DYADIC FORM OF DERIVED FUNCTION)*

7 *SYNTAX ERROR (NO DYADIC FORM OF FUNCTION)*

11 *VALUE ERROR*

15 *DOMAIN ERROR*

15 *DOMAIN ERROR (INCORRECT TYPE)*

27 *LIMIT ERROR (INTEGER TOO LARGE)*

28 *AXIS RANK ERROR (NOT VECTOR DOMAIN)*

29 *AXIS LENGTH ERROR (NOT SINGLETON)*

30 *AXIS DOMAIN ERROR (ARGUMENT RANK AND AXIS INCOMPATIBLE)*

30 *AXIS DOMAIN ERROR (AXIS LESS THAN INDEX ORIGIN)*

30 *AXIS DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)*

30 *AXIS DOMAIN ERROR (INCORRECT TYPE)*

30 *AXIS DOMAIN ERROR (NOT AN INTEGER)*

30 *AXIS DOMAIN ERROR (SEMICOLON LIST NOT ALLOWED)*

40 *OPERATOR DOMAIN ERROR (NOT A DYADIC FUNCTION)*

∘ . f Outer Product

Form

A∘.*fB*

Left Operand Domain

Type	Always jot (∘)

Right Operand Domain

Type	Dyadic function

Left Argument Domain

Type	Any
Shape	Any
Depth	Any

Right Argument Domain

Type	Any
Shape	Any
Depth	Any

Result Domain (of derived function)

Type	Depends on *f*
Rank	(ρρ*A*)+ρρ*B*
Shape	(ρ*A*),ρ*B*
Depth	Depends on *f*

Implicit Arguments

None.

Description

Outer product is a derived function that specifies an operation to be performed between every item of one array and every item of another array. In the expression *R←A∘ .fB*, *R* is an array that results from the application of the function *f* to every pair of items of *A* and *B*. *f* can be a primitive dyadic function, a dyadic system function, a dyadic user-defined function, or a dyadic-derived function from an arbitrary operator sequence. The function *f* does not have to be a value-returning function.

In the following example, note how the outer product operator affects the operation of the primitive scalar function multiply:

```
      1 2 3×2 3 4     ⍝SCALAR PRODUCT APPLIES × TO EACH PAIR
2 6 12
                  ⍝OUTER PRODUCT APPLIES × BETWEEN ALL ITEMS
      1 2 3∘.×2 3 4
2 3  4
4 6  8
6 9 12
```

In the next example, the outer product operator affects the operation of the equal function so that each data item in the left argument is compared to each item in the right argument. Then, the reduction function (derived from the slash operator) is used to determine how many times each item in the left argument appears in the right argument. Note that the left argument determines the number of rows in the result, and the right argument determines the number of columns:

```
      G←1 2 3 2 2 1
      (⍳3)∘.=G            ⍝FIND THE LOCATIONS OF 1S,2S,AND 3S IN G
1 0 0 0 0 1
0 1 0 1 1 0
0 0 1 0 0 0
      +/(⍳3)∘.=G          ⍝USE REDUCE TO TOTAL THE ROWS
2 3 1
       ⍝THERE ARE TWO 1S THREE 2S AND ONE 3 IN G
```

Further examples:

```
      ⎕←X←(1 2 3) 'ABC' ¯2
+-----+ +---+ ¯2
|1 2 3| |ABC|
+-----+ +---+
      (⍳2) ∘.= X
+-----+ +-----+ 0
|1 0 0| |0 0 0|
+-----+ +-----+
+-----+ +-----+ 0
|0 1 0| |0 0 0|
+-----+ +-----+
      ⎕←W←(2 2 3) (2 1 0)
+-----+ +-----+
|2 2 3| |2 1 0|
+-----+ +-----+
      W ∘.⍴ W
+-----+ +-----+
|2 2 3| |2 1 0|
|2 2 3| |2 1 0|
|     | |     |
|2 2 3| |2 1 0|
|2 2 3| |2 1 0|
+-----+ +-----+
++      ++
||      ||
||      ||
||      ||
++      ++
      (⍳3) ∘.⍴ ¯1 2 0 ⍝SIMPLE ARGS YIELD NESTED RESULT
+--+       +-+     +-+
|¯1|       |2|     |0|
+--+       +-+     +-+
+-----+    +---+   +---+
|¯1 ¯1|    |2 2|   |0 0|
+-----+    +---+   +---+
+--------+ +-----+ +-----+
|¯1 ¯1 ¯1| |2 2 2| |0 0 0|
+--------+ +-----+ +-----+
```

Possible Errors Generated

7 *SYNTAX ERROR (NO MONADIC FORM OF DERIVED FUNCTION)*

15 *DOMAIN ERROR*

15 *DOMAIN ERROR (INCORRECT TYPE)*

40 *OPERATOR DOMAIN ERROR (NOT A DYADIC FUNCTION)*

f. g Inner Product

Form

Af.gB

Left Operand Domain

Type	Dyadic value-returning function

Right Operand Domain

Type	Dyadic value-returning function

Left Argument Domain

Type	Any
Shape	Any, inner axes of *A* and *B* must conform
Depth	Any

Right Argument Domain

Type	Any
Shape	Any, inner axes of *A* and *B* must conform
Depth	Any

Result Domain (of derived function)

Type	Depends on *f* and *g*
Rank	`0⌈¯2+(⍴⍴A)+⍴⍴B`
Shape	`(¯1↓⍴A),1↓⍴B`
Depth	Depends on *f* and *g*

Implicit Arguments

None.

Description

The derived function inner product produces the common algebraic matrix product of two arrays. The name inner product comes from the application of the function (*g*) along the inner axes of the two arguments. *f* and *g* can be a primitive dyadic function, a dyadic system function, a dyadic user-defined function, or a dyadic-derived function from an arbitrary operator sequence. (The inner axes are the last axis of the left argument and the first axis of the right argument.) For example:

```
      ⎕←A←2 3⍴⍳6
1 2 3
4 5 6
      ⎕←B←⍳3
1 2 3
      A+.×B
14 32
       ⍝INNER AXES DO NOT MATCH IN NEXT EXPRESSION
      B+.×A
  10 LENGTH ERROR (LENGTHS OF INNER AXES DO NOT MATCH)
       B+.×A
       ^
```

In the preceding example, each item along the first axis of the right argument (in this case, B has only one axis) is multiplied by the corresponding item along the last axis of the right argument, and the products of each row are summed. The lengths of these inner axes must conform (in this case they are both 3). The shape of the result is the shape of A (2 3) catenated to the shape of B (3) without their inner axes.

When each of the arguments has only one axis, the result is a scalar:

```
      (⍳3)+.×⍳3
14
```

When you want to perform the inner product with one object on itself, use transpose:

```
      ⎕←A←2 3⍴⍳6
1 2 3
4 5 6
      A+.×⍉A
14 32
32 77
```

You can also specify an inner product in which an operation other than multiplication is performed. Commonly, you might also use ∧.= (And Dot Equals), ∨.≠ (Or Dot Not equals), or ×.⋆ (Times Dot Star). Using this method, you can locate values containing specific characters or search for a row of one array in which all the items are equal to those in a column of another array. For example:

```
      ⎕←B←2 3⍳6
3
                          ⍝NEXT EXPRESSION COUNTS WHERE 2 AND 6
      2 6+.≤B             ⍝ ARE ≤ THE TWO ROWS OF EACH COLUMN IN B
1
      ⎕←X←4 3⍴'ONETWOSIXTEEN'
ONE
TWO
SIX
TEE
      ⍴X
4 3
      ⍴Y←'SIX'            ⍝FIND WHERE Y OCCURS IN SIX
3
```

To be used in an inner product operation, the two arguments, denoted A and B, must conform to at least one of the following rules:

- A or B is a singleton
- The inner axes (the results of ¯1↑⍴A and 1↑⍴B) are equal
- Either the last axis of A (¯1↑⍴A) or the first axis of B (1↑⍴B) equals 1

If the first or third rule is true, then the corresponding argument is extended (through the process of singleton extension) so that the arguments have equal lengths along the matching axes.

If (0=¯1↑⍴A)∧0=1↑⍴B, but no other axes of A and B are equal to 0, then the inner product operator returns an array of identity items for the function *f*, as in reduction.

Further examples:

```
      ⎕←G←1 3 ⍴ (1 2 3) (¯2 0 1) ¯2
+-----+ +------+ ¯2
|1 2 3| |¯2 0 1|
+-----+ +------+
      G + .× ⍉ G
+------+
|9 8 14|
+------+
                              ⍝SIMPLE ARGS YIELD NESTED RESULT
      (⍳3) ,.⍴ ⍳3
+-----------+
|1 2 2 3 3 3|
+-----------+
```

Possible Errors Generated

7 *SYNTAX ERROR (NO MONADIC FORM OF DERIVED FUNCTION)*

11 *VALUE ERROR*

10 *LENGTH ERROR (LENGTHS OF INNER AXES DO NOT MATCH)*

15 *DOMAIN ERROR*

15 *DOMAIN ERROR (FUNCTION HAS NO IDENTITY ELEMENT)*

15 *DOMAIN ERROR (INCORRECT TYPE)*

40 *OPERATOR DOMAIN ERROR (NOT A DYADIC FUNCTION)*

[ ] Axis

Form

f[*K*] *B*   *Af*[*K*] *B*

Left Argument Domain

Type	Monadic or dyadic function

Right Argument Domain

Type	Near-integer (floating for laminate and ravel, Any for user-defined operations)
Shape	Singleton (Vector for drop, enclose, disclose, ravel, take and all dyadic scalar functions, Any for user-defined operations)
Depth	0 or 1 (simple), Any for user-defined operations

Result Domain

Type	Same as left argument

Implicit Arguments

⎕*IO* (*f*[*K*] when ⎕*IO*←1 is identical to *f*[*K*+1] when ⎕*IO*←0)

Description

Axis makes the function to its left apply to the axis specified by the value surrounded by brackets. The following functions and operators may be affected by axis:

- Catenate (, and ⍪)
- Derived compress/replicate (/ and ⌿)
- Derived reduction (/ and ⌿)
- Derived expand (\ and ⍀)
- Derived scan (\ and ⍀)
- Disclose (⊃)
- Drop (↓)

- Enclose (⊂)
- Laminate (, and ⍪)
- Monadic grade up (⍋)
- Monadic grade down (⍒)
- Ravel (, and ⍪)
- Rotate (⌽ and ⊖)
- Reverse (⌽ and ⊖)
- Take (↑)
- *⎕EXP*
- *⎕REP*
- All dyadic scalar functions (see Table 1–1 in Section 1.1.1)
- User-defined operations

The use of axis with these functions (and operators) is described in the individual explanations of the functions. For examples and further descriptions of axis with scalar functions, see Section 1.1.

When you use axis with the ⍪, ⌿, ⍀, and ⊖ functions, the functions are equivalent to , , / , \ , and ⌽ used with axis. The following list shows the definitions and equivalences of these symbols. Note the following:

> *f* represents either , , / , \ or ⌽
> *g* represents ⍪ , ⌿ , ⍀ , or ⊖
> *A* and *B* represent any arrays
> *S* is any scalar
> *K* is any axis
> ⎕*IO* is used to select the first axis of an array.

- *gB* ←→ *f*[⎕*IO*] *B*
- *f*[*K*] *B* ←→ *g*[*K*] *B* for all*K*
- *fS* ←→ *f*[⎕*IO*] *S* ←→ *g S* ←→ *g*[⎕*IO*] *S*
- *Ag B* ←→ *A f* [⎕*IO*] *B*
- *Af*[*K*] *B* ←→ *A g*[*K*] *B* for all *K*
- *AfS* ←→ *A f*[⎕*IO*] *S* ←→ *A g S* ←→ *A g*[⎕*IO*] *S*

Axis is ⎕*IO*-dependent; thus, all the functions named are ⎕*IO*-dependent when they are affected by axis, except user-defined operations.

Possible Errors Generated

```
28  AXIS RANK ERROR (NOT VECTOR DOMAIN)

29  AXIS LENGTH ERROR (NOT SINGLETON)

30  AXIS DOMAIN ERROR (ARGUMENT RANK AND AXIS INCOMPATIBLE)

30  AXIS DOMAIN ERROR (AXES NOT IN CONTIGUOUS ASCENDING ORDER)

30  AXIS DOMAIN ERROR (AXIS LESS THAN INDEX ORIGIN)

30  AXIS DOMAIN ERROR (DUPLICATE AXIS NUMBER)

30  AXIS DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

30  AXIS DOMAIN ERROR (INCORRECT OPERATION)

30  AXIS DOMAIN ERROR (INCORRECT TYPE)

30  AXIS DOMAIN ERROR (LEFT ARGUMENT HAS WRONG LENGTH)

30  AXIS DOMAIN ERROR (NOT AN INTEGER)

30  AXIS DOMAIN ERROR (RIGHT ARGUMENT HAS WRONG RANK)

30  AXIS DOMAIN ERROR (SEMICOLON LIST NOT ALLOWED)
```

← Specification Function

Form

```
A←B        A[K]←B
```

Left Argument Domain

Type	Variable name or undefined name
Shape	Any

Right Argument Domain

Type	Any
Shape	Conforms to index argument K, if any
Depth	Any

Result Domain

Type	Same as right argument
Rank	⍴⍴B
Shape	⍴B
Depth	≡B

Implicit Arguments

None.

Description

The specification function (←) stores values in identifiers. The left argument (A) must be a variable name or undefined. When the function is executed, the value of the right argument (B) becomes associated with the name A.

In addition to the uses described below, specification can also be used for strand and selective assignment statements.

Specification functions can be included in the construction of other statements. For example, the following assigns the value 7 to C, 11 to B, and 14 to A:

```
A←3+B←4+C←7
```

For the form A[K]←B, axes of length 1 are dropped from B to allow B to conform to A[K]. (For more details about the A[K]←B form of specification, see the *VAX APL User's Guide.*) For example:

```
      ⎕←A←2 3ρι6
1 2 3
4 5 6
      B←3 1 1ρ7 8 9
      A[2;]←B
      A
1 2 3
7 8 9
```

The specification function is a quiet function; it does not return a value if it is the leftmost function in a statement.

```
      A←2
      A
2
      (A←2)
2
      ⎕←A←2
2
```

Note that the value returned by the specification function (when you require that it returns a value) is the value of the right argument, even if the left argument is indexed. For example:

```
      A←5 4 3 2 1
      ⎕←A[1]←4
4
      A
4 4 3 2 1
```

Possible Errors Generated

Specification not subscripted (form A←B)

4 *NOT A VALID SYSTEM IDENTIFIER*

7 *SYNTAX ERROR (MISSING LEFT ARGUMENT TO ASSIGNMENT)*

11 *VALUE ERROR (NO VALUE TO ASSIGN)*

15 *DOMAIN ERROR (ILLEGAL LEFT ARGUMENT TO ASSIGNMENT)*

15 *DOMAIN ERROR (NOT A SYSTEM VARIABLE)*

Subscripted specification (form *A*[*K*]←*B*)

11	*VALUE ERROR (NO VALUE TO ASSIGN)*
11	*VALUE ERROR (SUBSCRIPTED NAME IS UNDEFINED)*
15	*DOMAIN ERROR (INVALID OBJECT IN INDEXED ASSIGNMENT)*
15	*DOMAIN ERROR (NOT A SYSTEM VARIABLE)*
27	*LIMIT ERROR (INTEGER TOO LARGE)*
36	*INDEX RANK ERROR*
36	*INDEX RANK ERROR (CANNOT INDEX A SCALAR)*
37	*INDEX LENGTH ERROR*
37	*INDEX LENGTH ERROR (INDEX LESS THAN INDEX ORIGIN)*
37	*INDEX LENGTH ERROR (INDEX OUT OF RANGE)*
38	*INDEX DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)*
38	*INDEX DOMAIN ERROR (INCORRECT TYPE)*
38	*INDEX DOMAIN ERROR (NOT AN INTEGER)*

Strand Assignment with the Specification Function

Form

```
(A1...An)←B
```

Left Argument Domain

Type	List of variable or undefined names
Shape	Any

Right Argument Domain

Type	Any
Shape	Vector domain
Depth	Any

Result Domain

Type	Same as right argument
Rank	⍴⍴B
Shape	⍴B
Depth	≡B

Implicit Arguments

None.

Description

Strand assignment (also known as vector assignment) allows you to assign a list of values to a list of objects. APL applies the assignment along successive pairs of items in the left (A) and right (B) arguments in a manner similar to scalar extension. The objects in A may be undefined names, variable names, or system variable names. The result of the strand assignment function is the right argument.

The length of B must conform to the number of objects in A, or B must be a singleton, in which case APL performs singleton extension. For example:

```
      BURR ← 32 ⋄ TEMP ← 0 ⋄ COLD ← ¯12
      BURR ⋄ TEMP ⋄ COLD
32
0
¯12
                          ⍝PARENTHESES REQUIRED
      (BURR TEMP COLD) ← 20 ¯4 ¯15
      BURR ⋄ TEMP ⋄ COLD
20
¯4
¯15
                          ⍝SINGLETON EXTENSION
      (BURR TEMP COLD) ← ¯3
      BURR ⋄ TEMP ⋄ COLD
¯3
¯3
¯3
```

You can use strand assignment to allow multiple arguments in user-defined operations. For example, *FRET* is a monadic user-defined function containing three local variables (*X*, *Y*, and *Z*). The header definition of *FRET* is as follows:

```
∇FRET B;X;Y;Z ∇
```

When *FRET* is called, the argument (*B*) contains three items. Inside *FRET*, there is an expression that performs a strand assignment in which each item in *B* is assigned to a local variable. For example:

```
BIP←23 41 'RUE'   ⍝BIP CONTAINS 3 ITEMS
FRET BIP          ⍝THE CALL TO FRET IS STILL MONADIC
   .
   .
   .
(X Y Z)←BIP       ⍝THIS IS EXPRESSION INSIDE OF FRET
```

Note that the length (3) of the left argument to the specification function conforms to the number of items in *BIP*. If *BIP* were a singleton, APL would perform singleton extension.

Strand assignment is an atomic operation; if any of the assignments fail, no change occurs to any of the names in the left argument list. However, If you have set the display option on the ⎕*WATCH* system function (see Chapter 2 if you have set the signal option, the signal is held until APL completes the entire strand assignment and only the last watched name is signaled.

Possible Errors Generated

4 *NOT A VALID SYSTEM IDENTIFIER*

7 *SYNTAX ERROR (MISSING LEFT ARGUMENT TO ASSIGNMENT)*

9 *RANK ERROR (NOT VECTOR DOMAIN)*

10 *LENGTH ERROR*

15 *DOMAIN ERROR (INVALID OBJECT IN STRAND ASSIGNMENT)*

11 *VALUE ERROR (NO VALUE TO ASSIGN)*

15 *DOMAIN ERROR (NOT A SYSTEM VARIABLE)*

Selective Assignment with the Specification Function

Form

(*fA*)←*B*   (*CfA*)←*B*

Left Argument Domain

Type	*A* is a variable name *f* is a function (see list below) *C* is any valid left argument to *f*
Shape	Any

Right Argument Domain

Type	Any
Shape	Conforms to left argument
Depth	Any

Result Domain

Type	Same as right argument
Rawrite nk	ρρ*B*
Shape	ρ*B*
Depth	≡*B*

Implicit Arguments

None.

Description

Selective assignment allows you to assign values to specified items of an array. The left argument (*fA*) contains an expression that selects items from an array. The length of the right argument (*B*) either equals the number of items selected or is 1 (*B* is a singleton), in which case APL performs singleton extension. For example:

```
      ⎕←GUT←⍳5
1 2 3 4 5
      ⎕← (3↑GUT)←48 49 50 ⍝ASSIGN TO FIRST 3 ITEMS OF GUT
48 49 50 45
      ⎕← (3↑GUT)←48       ⍝SINGLETON EXTENSION
48 48 48 45
                          ⍝SHAPES DO NOT MATCH
      (3↑GUT)←48 49
 10 LENGTH ERROR
       (3↑GUT)←48 49
       ^
```

The following table describes the primitive functions you can use in the left argument expression to select items from an array. The symbol *I* refers to an expression that is a valid argument to the function in the form.

Assignment Form	Function Name
(,A)←B	Ravel
(,[K]A) ← B	Ravel with axis
(⌽A) ← B	Reverse
(⊖A) ←B	
(⌽[K]A) ← B	Reverse with axis
(⊖[K]A) ← B	
(⍉A) ← B	Transpose
(I↓A) ← B	Drop
(I↓[K]A) ← B	Drop with axis
(I↑A) ← B	Take
(I↑[K]A) ← B	Take with axis
(I⌽A) ← B	Rotate
(I⊖A) ← B	
(I⌽[K]A) ← B	Rotate with axis
(I⊖[K]A) ← B	
(I⍉A) ← B	Transpose
(I⍴A) ← B	Reshape
(I\A) ← B	Expand
(I⍀A) ← B	
(I\[K]A) ← B	Expand with axis

Assignment Form	Function Name
(*I*⍀[*K*]*A*) ← *B*	
(*I*/*A*) ← *B*	Replicate
(*I*⌿*A*) ← *B*	
(*I*/[*K*]*A*) ← *B*	Replicate with axis
(*I*⌿[*K*]*A*) ← *B*	

You can use more than one of the eligible functions in the left argument expression. For example:

```
      ⎕←BOP←3 3⍴⍳9
1 2 3
4 5 6
7 8 9
      ⎕←(2↑1 1⍉BOP)←0 0  ⍝CHANGE FIRST 2 ITEMS ON THE DIAGONAL
0 2 3
4 0 6
7 8 9
```

You can use other primitive functions in the portion of the left argument expression that evaluates the argument of one of the eligible functions. For example:

```
      ⎕←BOP←3 3⍴⍳9
1 2 3
4 5 6
7 8 9
      BE←1
      EP←2
      ⎕←((BE+EP)↑1 1⍉BOP)←0 0 0
0 2 3
4 0 6
7 8 0
```

Possible Errors Generated

4 *NOT A VALID SYSTEM IDENTIFIER*

11 *VALUE ERROR (NO VALUE TO ASSIGN)*

15 *DOMAIN ERROR (CANNOT MODIFY SELECTIVE ASSIGNMENT TARGET)*

```
15 DOMAIN ERROR (INVALID FUNCTION IN SELECTIVE ASSIGNMENT)

15 DOMAIN ERROR (INVALID OBJECT IN SELECTIVE ASSIGNMENT)

15 DOMAIN ERROR (NOT A SYSTEM VARIABLE)

36 INDEX RANK ERROR

37 INDEX LENGTH ERROR
```

2

VAX APL System Variables and Functions

Conceptually, there are two parts to the VAX APL interpreter: the APL language and the APL environment. The APL language comprises the lexical and symbolic elements of APL, the parts of APL that are included when it is used as a mathematical notation in a classroom. The APL environment is the setting in which the APL language elements are applied.

The APL interpreter recognizes a set of system variables, functions, and commands that allow you to control your APL sessions, as well as to facilitate and preserve the work you do in those sessions. For example, the interpreter allows you to:

- Determine or set the values of the index origin, print precision, comparison tolerance, and other elements that affect the operation of functions.
- Get information about a workspace, such as its name and size, the names of its user-defined operations and variables, the state of its active operations, and so on.
- Manipulate workspaces; that is, load, save, or delete them, copy objects from them, or change their size.
- Get the system time and date, or get accounting information for a session.

2.1 System Variables

VAX APL system variables, like ordinary variables, can be used in any language expression or function. Unlike ordinary variables, system variables have special meaning to the system, and they allow you to do the following:

- Set the index origin and comparison tolerance.
- Change the output precision and line width.
- Specify an operation to be performed when a workspace is activated.
- Automatically save an active workspace after function editing and data input.

2.1.1 System Variable Names

The names of APL system variables begin with a quad character (⎕). The names are considered to be distinguished names, meaning that they are reserved for a specific purpose. You cannot use them as names for user-defined operations or variables, and you cannot copy, erase, or collect them in a group.

2.1.2 System Variable Characteristics

System variables are similar to ordinary variables in the following ways:

- They retain their values until new ones are assigned.
- Their current values are saved with a workspace (except for ⎕*GAG*, ⎕*TT*, ⎕*TLE* and ⎕*VPC*).
- They can be localized in user-defined operation definitions.

Each of the system variables in APL can be assigned a value and can be localized in user-defined operations.

Table 2–1 lists the system variables, the range of values you can specify for them, and their default values.

Table 2–1 System Variable Value Ranges

Variable	Value Range	Default
⎕*AUS*	0, 1, 2	0
⎕*CT*	0 to 2.328E¯10	1E¯15
⎕*DC*	Nested vector	(¯1 1 0 2) ' '
⎕*DML*	512 to 2048	2048
⎕*ERROR*	Error message	' '
⎕*GAG*	0, 1, 2, 3	Terminal dependent
⎕*IO*	0, 1	1
⎕*L*	Any	⍳0
⎕*LX*	Expression	' '
⎕*NG*	0, 1, 2	1
⎕*PP*	1 to 16	10
⎕*PW*	35 to 2044	Terminal width
⎕*R*	Any	⍳0

(continued on next page)

Table 2–1 (Cont.) System Variable Value Ranges

Variable	Value Range	Default
⎕*RL*	¯2147483648 to 2147483647	695197565
⎕*SF*	Prompt	⎕:<CR><LF> 6 spaces
⎕*SINK*	Any	Always ⍳ 0
⎕*TERSE*	0, 1	0
⎕*TIMELIMIT*	¯1 to 255	0
⎕*TIMEOUT*	0, 1	0
⎕*TLE*	0, 1	Terminal dependent
⎕*TRAP*	Expression	' '
⎕*TT*	1 to 19	Terminal dependent
⎕*VPC*	Non-negative integer	30

Note that ⎕*ERROR*, ⎕*LX*, ⎕*SF*, and ⎕*TRAP* must have character values; ⎕*DC* has a two-item heterogeneous value; all the other system variables must have numeric values. The exceptions are ⎕*L*, ⎕*R*, and ⎕*SINK*, which may take any type of value.

2.2 System Functions

APL system functions supplement the primitive functions by providing additional processing capabilities. For instance, they allow you to do the following:

- Express the canonical representation of a user-defined operation and store the operation definition as data.
- Expunge a named object.
- Construct a name list of labels, variables, or functions, and return the classification of a named object.
- Delay execution of an operation for a specified period of time.

You access a system function by stating its name and arguments (if any), just as you would access a primitive or user-defined operation.

For the system functions that take character arguments, white space (spaces and tabs) is allowed before and after the name (workspace name, function name, and so on) in the argument. For example, all of the following will load the workspace *MYWS*:

```
⎕QLD ' MYWS '
⎕QLD '       MYWS '
⎕QLD 'MYWS'
⎕QLD '     MYWS          '
```

Anything other than white space is not allowed before or after the name:

```
      ⎕QLD ' MYWS          A'
 22 INCORRECT PARAMETER (EXTRANEOUS CHARACTERS AFTER COMMAND)
      ⎕QLD ' MYWS          A'
        ^
```

2.2.1 System Function Names

System functions, like system variables, are identified by unique names that begin with a quad character (⎕); you cannot use these names for user-defined operations or variables, and you cannot copy, erase, or collect them in a group. APL assumes that any system functions in an expression are ambivalent, even though most system functions have a specific valence. This means that if an expression contains a left argument for a monadic system function, APL signals an error. For example:

```
                          ⍝⎕ARBOUT IS MONADIC
      2 ⎕ARBOUT 3
 7 SYNTAX ERROR (NO DYADIC FORM OF FUNCTION)
      2 ⎕ARBOUT 3
         ^
```

2.2.2 Types of System Functions

System functions can be categorized as follows:

- Niladic system functions—those that do not take arguments.
- Monadic system functions—those that take one argument.
- Dyadic system functions—those that take two arguments.
- Ambivalent system functions—those that take either one or two arguments.

The niladic system functions do not take arguments (you may not assign a value to them), and they cannot be localized in user-defined operations. The niladic system functions and their values (where applicable) in a clear workspace may be summarized as follows:

Niladic System Functions

Function	Description (value in clear workspace)
⎕*AI*	Account information as 4-integer vector
⎕*ALPHA*	'∆*ABCDEFGHIJKLMNOPQRSTUVWXYZ*'
⎕*ALPHAL*	'*abcdefghijklmnop qrstuvwxyz*'
⎕*ALPHAU*	'⍙*A̲B̲C̲D̲E̲F̲G̲H̲I̲J̲K̲L̲M̲N̲ O̲P̲Q̲R̲S̲T̲U̲V̲W̲X̲Y̲Z̲*'
⎕*ASCII*	⎕*AV* subset; approximates ASCII characters
⎕*AV*	Atomic vector
⎕*CHANS*	Assigned file channels (empty numeric vector)
⎕*CTRL*	The first 32 ASCII characters and Delete
⎕*LC*	Line numbers in state indicator (⍳ 0)
⎕*NUM*	'0123456789'
⎕*RESET*	Clears the state indicator (no value)
⎕*TS*	Time stamp as 7-integer vector
⎕*UL*	Process identification number (PID)
⎕*VERSION*	Interpreter and workspace versions
⎕*WA*	Workspace available in bytes

The monadic system functions take one argument, which is placed immediately to the right of the function. The following table of the monadic system functions describes the type, shape, and, where applicable, the units associated with each function's argument. Note that there are two entries for ⎕*ASS*, which has both action and query uses.

Monadic System Functions

Function	Shape	Type	Units
⎕*ARBOUT*	Vector domain	Integer	Character codes
⎕*ASS*	Vector domain	Character	File information
⎕*ASS*	Vector domain	Near-int	Channel numbers
⎕*BREAK*	Any	Any	N/A
⎕*CHS*	Vector domain	Near-int	Channel numbers
⎕*CLS*	Vector domain	Near-int	Channel numbers
⎕*CR*	Vector domain	Character	Operation name
⎕*DAS*	Vector domain	Near-int	Channel numbers

Monadic System Functions

Function	Shape	Type	Units
⎕*DL*	Singleton	Floating	Seconds
⎕*DVC*	Vector domain	Near-int	Channel numbers
⎕*EFC*	Vector domain	Near-int	Channel numbers
⎕*EFR*	Vector domain	Near-int	Channel numbers
⎕*EFS*	Vector domain	Near-int	Channel numbers
⎕*EX*	Matrix domain	Character	Name list
⎕*FI*	Vector domain	Character	Numeric string
⎕*FLS*	Vector domain	Near-int	Channel numbers
⎕*FX*	Matrix domain	Character	Operation definition
⎕*MBX*	Vector domain	Near-int	Channel numbers
⎕*NC*	Matrix domain	Character	Name list
⎕*OM*	Vector domain	Near-Bool	N/A
⎕*QCO*	Vector domain	Character	Workspace name, object names
⎕*QLD*	Vector domain	Character	Workspace name
⎕*QPC*	Vector domain	Character	Workspace name, object names
⎕*RELEASE*	Vector domain	Near-int	Channel numbers
⎕*VI*	Vector domain	Character	Numeric string
⎕*VR*	Vector domain	Any	Value or object name
⎕*XQ*	Vector domain	Any	N/A

The dyadic system functions take both a left and a right argument. The dyadic system functions and the type, shape, and units, if any, associated with their arguments are as follows.

Dyadic System Functions

Function		Shape	Type	Units
⎕*CIQ*	Left:	Vector domain	Near-int	Packed data
	Right:	Vector domain	Near-int	Control information
⎕*COQ*	Left:	Array	Any	Data to be packed
	Right:	Vector domain	Near-int	Control information
⎕*EXP*	Left:	Vector domain	Near-Bool	Expand information

Dyadic System Functions

Function		Shape	Type	Units
	Right:	Any	Any	Array to be expanded
⎕FMT	Left:	Vector domain	Character	Format string
	Right:	Any	Any	Data to be formatted
⎕REP	Left:	Vector domain	Near-int	Replicate information
	Right:	Any	Any	Array to be replicated
⎕SS	Left:	Vector domain	Character	Pattern string
	Right:	Vector domain	Character	String

The ambivalent system functions may be monadic or dyadic; thus, they take either a right argument only, or they take both a right and a left argument. The following table of the ambivalent system functions describes the type, shape, and, where applicable, the units associated with each function's arguments:

Ambivalent System Functions

Function		Shape	Type	Units
⎕BOX	Left:	Vector domain	Character	Delimiter
	Right:	Matrix domain	Character	Delimited lines
⎕MAP	Left:	Vector domain	Character	Function header
	Right:	Vector domain	Character	Shared image def/function name
⎕MONITOR	Left:	Vector domain	Numeric	Line numbers
	Right:	Matrix domain	Character	Operation names
⎕NL	Left:	Vector domain	Character	Letter list
	Right:	Vector domain	Near-int	Name classes
⎕PACK	Left:	Vector domain	Numeric	Data packets
	Right:	Matrix domain	Character	Variable names
⎕REWIND	Left:	Singleton	Near-int	Key of reference
	Right:	Vector domain	Near-int	Channel numbers
⎕SIGNAL	Left:	Vector domain	Character	Error message
	Right:	Singleton	Near-int	Error number
⎕STOP	Left:	Vector domain	Near-int	Line numbers

		Ambivalent System Functions		
Function		**Shape**	**Type**	**Units**
	Right:	Matrix domain	Character	Operation names
⎕*TRACE*	Left:	Vector domain	Near-int	Line numbers
	Right:	Matrix domain	Character	Operation names
⎕*WAIT*	Left:	Singleton	Near-int	Time limit
	Right:	Vector domain	Near-int	Channel numbers
⎕*WATCH*	Left:	Singleton	Near-int	Watch mode
	Right:	Matrix domain	Character	Variable names

Another type of system function is the quiet function, a category that is independent of the valence of the function. Quiet functions do not generally cause APL to display a value when they are evaluated as the leftmost function in a statement. The following table shows the quiet functions:

	Quiet System Functions	
Monadic	**Dyadic**	**Ambivalent**
⎕*ARBOUT*	←	⍈
→ (always)	⎕*WAIT*	⎕*REWIND*
⎕*CLS*	⎕*XQ* (sometimes)	
⎕*DAS*	⍎ (sometimes)	
⎕*QCO*		
⎕*QLD*		
⎕*QPC*		
⎕*RELEASE*		

However, a quiet function displays a value if you enclose the function and its arguments in parentheses (note that the branch function (→) is always quiet). The ⎕*XQ* and ⍎ functions are quiet when the argument is quiet; otherwise ⎕*XQ* and ⍎ cause APL to display a value. For example:

```
      A←5      ⍝SPECIFICATION FUNCTION IS QUIET.  NO DISPLAY
      (A←5)    ⍝ADD PARENTHESIS IF YOU WANT A DISPLAY
5
      (B←'THIS WILL PRINT BECAUSE OF THE PARENTHESES')
THIS WILL PRINT BECAUSE OF THE PARENTHESES
      ⎕XQ 'A+10'    ⍝⎕XQ ARGUMENT IS NOT QUIET
15
      ⎕XQ 'C←A+10'  ⍝⎕XQ ARGUMENT IS QUIET.  NO DISPLAY
```

2.3 System Variables and Functions Reference

The following sections describe the APL system variables and functions in alphabetical order. Table 2–2 lists the system variables and functions and gives a brief description of their uses. APL displays an alphabetical list of these variables and functions when you enter the following expression: `X←⎕NL ¯2 ¯3 ¯5 ⋄ X[⍋X;]`

Table 2–2 System Variables and Functions

Name	Meaning
⎕AI	Maintains account information on the current APL session. Includes user identification, CPU time, and connect time.
⎕ALPHA	Vector of 27 characters: ∆ and A through Z.
⎕ALPHAL	Vector of 26 lowercase characters: a through z.
⎕ALPHAU	Vector of 27 underscored characters.
⎕ARBOUT	Writes arbitrary output to the terminal.
⎕ASCII	Subset of ⎕AV approximates the ASCII character set.
⎕ASS	Associates a file or mailbox with a channel.
⎕AUS	Specifies periodic workspace backup.
⎕AV	Vector of all APL characters.
⎕BOX	Returns a matrix from a character vector and vice versa. (The rows of the matrix are delimited by a specified string.)
⎕BREAK	Suspends operation execution and returns control to immediate mode.
⎕CHANS	Identifies channel numbers associated with files.
⎕CHS	Returns file organization and open status on one or more channels.
⎕CIQ	Unpacks data packed by ⎕COQ.

(continued on next page)

Table 2–2 (Cont.) System Variables and Functions

Name	Meaning
⎕CLS	Closes the files on one or more channels.
⎕COQ	Packs data of different types for storage as one record.
⎕CR	Returns a canonical representation of a user-defined operation whose name is the character string specified.
⎕CT	Determines the degree of tolerance applied in numeric comparisons.
⎕CTRL	Vector of ASCII control characters.
⎕DAS	Disassociates files from one or more channels.
⎕DC	Controls the display of enclosed arrays.
⎕DL	Delays execution by the number of seconds specified.
⎕DML	Controls default maximum record length used to save the workspace or to create a file.
⎕DVC	Returns the device characteristics longword for one or more channels.
⎕EFC	Clears event flags associated with one or more channels.
⎕EFR	Returns the setting for event flags on one or more channels.
⎕EFS	Sets event flags associated with one or more channels.
⎕ERROR	Character vector that describes last error to occur.
⎕EX	Expunges existing use of a name in the workspace.
⎕EXP	Expands an array by adding fill items in the same manner as the expansion derived function.
⎕FI	Converts character argument to numeric, placing 0s in each position not corresponding to a valid number.
⎕FLS	Returns information about files on one or more channels.
⎕FMT	Converts argument to character matrix in designated format.
⎕FX	Establishes an operation from its canonical representation.
⎕GAG	Indicates whether to accept broadcast messages.
⎕IO	Sets index origin for arrays; must be 0 or 1.
⎕L	Contains the name of a changed variable that is being watched by ⎕WATCH.
⎕LC	Vector of line numbers in state indicator; most recently suspended operation appears first.

(continued on next page)

Table 2–2 (Cont.) System Variables and Functions

Name	Meaning
⎕*LX*	Contains an expression to be executed automatically when workspace is loaded.
⎕*MAP*	Associates an external routine with a user-defined function.
⎕*MBX*	Returns information about mailboxes on one or more channels.
⎕*MONITOR*	Gathers information about operation execution counts and CPU times.
⎕*NC*	Returns the classification of one or more names.
⎕*NG*	Controls recognition and printing of negative sign.
⎕*NL*	Constructs a list of named objects residing in the active workspace.
⎕*NUM*	Vector of 10 numeric characters: 0 through 9.
⎕*OM*	Returns the index of every occurrence of a 1 in a Boolean vector.
⎕*PACK*	Packs and unpacks data for storage as one record.
⎕*PP*	Controls precision of noninteger numeric output.
⎕*PW*	Sets maximum number of characters in output line.
⎕*QCO*	Quietly copies a workspace.
⎕*QLD*	Quietly loads a workspace.
⎕*QPC*	Quietly copies a workspace with certain protection.
⎕*R*	Contains the previous value of a changed variable that is being watched by ⎕*WATCH*.
⎕*RELEASE*	Releases all locked records in files on one or more channels.
⎕*REP*	Compresses or replicates an array in the same manner as the compression and replication derived functions.
⎕*RESET*	Clears the state indicator.
⎕*REWIND*	Repositions the next record pointer to the first record of a file on one or more channels.
⎕*RL*	Forms link in chain of random numbers used in roll and deal functions.
⎕*SF*	Prompt for evaluated input.
⎕*SIGNAL*	Passes an error up the stack one level to the caller of the operation in error.
⎕*SINK*	Discards unwanted output; always ⍳0.

(continued on next page)

Table 2–2 (Cont.) System Variables and Functions

Name	Meaning
⎕SS	Searches the right argument for every occurrence of a character string specified in the left argument.
⎕STOP	Sets or clears stop bits associated with operation lines.
⎕TERSE	Suppresses display of secondary error messages.
⎕TIMELIMIT	Limits time to respond to quote quad and quad del input requests.
⎕TIMEOUT	Equals 1 if time runs out during quote quad or quad del input request; otherwise, equals 0.
⎕TLE	Equals 1 when the terminal line editing attribute is on and 0 when line editing is off.
⎕TRACE	Sets or clears trace bits associated with operation lines.
⎕TRAP	Contains an expression to be executed when an error occurs.
⎕TS	Current date and time in base 10 format.
⎕TT	Determines the type of terminal being used for the current APL session.
⎕UL	Process identification number.
⎕VERSION	Interpreter and workspace versions.
⎕VI	Returns logical vector giving position of valid numbers in ⎕FI of argument.
⎕VPC	Controls the use of vector processing hardware.
⎕VR	Returns a visual representation of a value or user-defined operation whose name is the argument specified.
⎕WA	Maximum amount in bytes by which the active workspace can be increased.
⎕WAIT	Determines how long a read function waits for control of a shared record.
⎕WATCH	Watches changes or references to the values of variables.
⎕XQ	Executes character strings with error handling.

⎕*AI* Accounting Information

Type

Niladic System Function

Form

uic / cpu-time / connect-time ← ⎕*AI*

Result Domain

Type	Integer
Rank	1 (vector)
Shape	4
Depth	1 (simple)

Description

⎕*AI* returns a vector of the user identification number (*uic*), computer time (*cpu-time*) used during the current APL session, and time elapsed (*connect-time*) since the beginning of the current APL session.

For the user identification code GROUP, MEMBER, the *uic* is *MEMBER*+(*GROUP*×2*16). All times are expressed in milliseconds. The fourth element is always 0. For example:

```
      ⎕AI
589825 390 1190 0
```

VMS expresses the GROUP and MEMBER numbers in ⎕*AI*[1] in octal. The following APL expression returns those numbers:

```
      10⊥8 8 8⊤(0,2*16)⊤⎕AI[1]
11 1
```

Possible Errors Generated

None.

⎕*ALPHA* Alphabetic Characters

Type

Niladic System Function

Form

```
'∆ABCDEFGHIJKLMNOPQRSTUVWXYZ' ← ⎕ALPHA
```

Result Domain

Type	Character
Rank	1 (vector)
Shape	27
Depth	1 (simple)

Description

⎕*ALPHA* is a subset of ⎕*AV*; it returns a vector of the 27 alphabetic characters that may be used in identifiers. They are ∆ and *A* through *Z*, or, expressed in terms of ⎕*AV*:

```
⎕AV[(⎕IO+72),97+⍳26]
```

For example:

```
      ⎕ALPHA
∆ABCDEFGHIJKLMNOPQRSTUVWXYZ
      ⎕IO←0 ⋄ ⎕PW←52
      ⎕AV ⍳ ⎕ALPHA
72 97 98 99 100 101 102 103 104 105 106 107 108 109
      110 111 112 113 114 115 116 117 118 119 120
      121 122
```

Possible Errors Generated

None.

⎕*ALPHAL* Lowercase Alphabetics

Type

Niladic System Function

Form

```
'abcdefghijklmnopqrstuvwxyz' ← ⎕ ALPHAL
```

Result Domain

Type	Character
Rank	1 (vector)
Shape	26
Depth	1 (simple)

Description

⎕*ALPHAL* is a subset of ⎕*AV*; it returns a vector of the 26 lowercase alphabetic characters. They are *a* through *z*, or, expressed in terms of ⎕*AV*:

```
⎕AV[129+⍳26]
```

For example:

```
      ⎕ALPHAL
abcdefghijklmnopqrstuvwxyz
      ⎕IO←0 ⋄ ⎕PW←52
      ⎕AV ⍳ ⎕ALPHAL
129 130 131 132 133 134 135 136 137 138 139 140 141
      142 143 144 145 146 147 148 149 150 151 152
      153 154
```

Possible Errors Generated

None.

⎕*ALPHAU* Underscored Alphabetics

Type

Niladic System Function

Form

```
'⍙A̲B̲C̲D̲E̲F̲G̲H̲I̲J̲K̲L̲M̲N̲O̲P̲Q̲R̲S̲T̲ U̲V̲W̲X̲Y̲Z̲' ← ⎕ALPHAU
```

Result Domain

Type	Character
Rank	1 (vector)
Shape	27
Depth	1 (simple)

Description

⎕*ALPHAU* is a subset of ⎕*AV*; it returns a vector of the 27 underscored alphabetic characters that may be used in identifiers. They are ⍙ and A̲ through Z̲, or, expressed in terms of ⎕*AV*:

```
⎕AV[160+⍳27]
```

For example:

```
      ⎕ALPHAU
⍙A̲B̲C̲D̲E̲F̲G̲H̲I̲J̲K̲L̲M̲N̲O̲P̲Q̲R̲S̲T̲U̲V̲W̲X̲Y̲Z̲
      ⎕IO←0 ⋄ ⎕PW←52
      ⎕AV ⍳ ⎕ALPHAU
160 161 162 163 164 165 166 167 168 169 170 171 172
      173 174 175 176 177 178 179 180 181 182 183
      184 185 186
```

Possible Errors Generated

None.

⎕*ARBOUT* Arbitrary Output

Type

Monadic System Function (quiet)

Form

```
ι0 ← ⎕ARBOUT B
```

Argument Domain

Type	Near-integer
Shape	Vector domain
Depth	1 (simple)
Value	0 through 255

Result Domain

Type	Numeric
Rank	1 (vector)
Shape	0 (empty)
Depth	1 (simple)

Description

⎕*ARBOUT* allows you to send untranslated output to the terminal (actually, to the default output device). ⎕*ARBOUT* outputs the argument's items as if they were character codes.

One use of ⎕*ARBOUT* is to write a file of ASCII characters, where each of the integers corresponds to a character in the ASCII character set. You cannot use the file system function ⍈ (see the *VAX APL User's Guide*) with ⎕*ARBOUT* to write the file because ⎕*ARBOUT* sends output only to your default output device, usually your terminal. You can use the)*OUTPUT* system command (see the *VAX APL User's Guide*), however, to divert output from your terminal to a file. For example:

```
      )OUTPUT ASCFILE
      ⎕ARBOUT 35 37 38 42 64 94 95
      )OUTPUT
                           ⍝CHANGE TO ASCII CHARACTER SET
      )PUSH
$TYPE ASCFILE.AAS
      ⎕ARBOUT 35 37 38 42 64
#%&*@      )OUTPUT
$
```

APL does not append a <CR><LF> to ⎕*ARBOUT* output.

If you use ⎕*ARBOUT* immediately following ⍞ or ⍌ output, ⎕*ARBOUT* resets the bare output buffer. For details, see the *VAX APL User's Guide*.

⎕*ARBOUT* is a quiet function; that is, it does not return a result if it is the leftmost function in a statement. If it is not the leftmost function, ⎕*ARBOUT* returns ⍳0 as its result.

Possible Errors Generated

```
9  RANK ERROR (NOT VECTOR DOMAIN)

15  DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

15  DOMAIN ERROR (INCORRECT TYPE)

15  DOMAIN ERROR (NOT AN INTEGER)

27  LIMIT ERROR (INTEGER TOO LARGE)

15  DOMAIN ERROR
```

⎕*ASCII* APL Approximation to the ASCII Character Set

Type

Niladic system function

Form

ASCII-characters ← ⎕*ASCII*

Result Domain

Type	Character
Rank	1 (vector)
Shape	128
Depth	1 (simple)

Description

⎕*ASCII* is a subset of ⎕*AV*; it returns a vector of 128 characters that approximates the 7-bit ASCII character set. ⎕*ASCII* contains the ASCII control characters (⎕*CTRL*) and the lowercase letters (⎕*ALPHAL*). For example:

```
      (32↑⎕ASCII) ≡ 32↑⎕CTRL
1
      (¯1↑⎕ASCII) ≡ ¯1↑⎕CTRL
1
      ⎕ASCII[33] = ' '        ⍝33RD ITEM IS AN EMPTY SPACE
1
                     ⍝DISPLAY ALL BUT THE CTRL CHARACTERS
      2 47 ⍴ 94↑33↓⎕ASCII
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_'abcdefghijklmnopqrstuvwxyz{|}~
```

Possible Errors Generated

None.

⎕*ASS* Associating Files with Channels

Type

Monadic System Function (action form)

Form

variable ← ⎕*ASS* ' ⟦*channel*⟧ *filespec* ⟦*/fileorganization*⟧ '

Argument Domain

Type	Character
Shape	Vector domain
Depth	1 (simple)

Result Domain

Rank	0 (scalar)
Shape	⍳ 0 (scalar)
Depth	0 (simple scalar)

Type

Monadic System Function (query form)

Form

current-assignments ← ⎕*ASS channel*

Argument Domain

Type	Near-Integer
Shape	Vector domain
Depth	0 (simple scalar)
Value	¯999 through 999 (but not 0)

Result Domain

Type	Character
Rank	1 or 2
Shape	Vector or matrix
Depth	1 (simple)

Parameters

variable
Is an optional variable used when writing to or reading from this file and channel combination.

channel
Is an optional integer scalar whose absolute value represents a channel number in the range 1 through 999. If you do not specify a channel number, APL assigns one for you. APL picks the first available channel number, beginning at 12 and counting down to 1; then APL begins at 13 and counts up to 999.

filespec
Is the VMS file specification associated with the specified channel. If you do not include the file extension, APL uses the default file extension for the file organization qualifier specified. (See Table 2–3.)

/fileorganization
Identifies the file organization of the file specified by *filespec*. The possible values of /*fileorganization* are listed in Table 2–3. The default value is */DA*.

Table 2–3 File Organization Qualifiers

/fileorganization Qualifier	Default File Extension	Type of File
/AS	*.AAS*	ASCII sequential; can open for either read or write, or both (when you specify */UPDATE*).
/AS⋆	*.AAS*	ASCII sequential; file is positioned at end-of-file to allow appending.

(continued on next page)

Table 2–3 (Cont.) File Organization Qualifiers

/fileorganization Qualifier	Default File Extension	Type of File
/IS	*.AIS*	Internal sequential; can open for either read or write, or both (when you specify */UPDATE*).
/IS⋆	*.AIS*	Internal sequential; file is positioned at end-of-file to allow appending.
/DA	*.AIX*	Direct-access; can do read and write (this is the default).
/RF	*.ARF*	Relative; can do read and write.
/KY	*.AKY*	Keyed; can do read and write.

current-assignments
A vector containing the current value of assignments.

Qualifiers

/BLOCKSIZE ⟦:***blocksize***⟧
For input on nondisk devices, it specifies the minimum size memory buffer for APL to make available. The default is 2044 bytes or the current */MAXLEN* value, whichever is smaller.

In all other cases, it is ignored. In addition, it is always ignored for ASCII sequential files (the blocksize is always 2044 bytes.)

/BUFFERCOUNT ⟦:***n***⟧
Specifies how many I/O buffers you want allocated to read and write to a file. The acceptable values for *n* is 0 through 127. The default is 0, which means that the number of allocated buffers will be the same as the current system default value.

/CCONTROL ⟦:***keyword***⟧
Specifies the carriage control attribute for a new, sequential file. (The qualifier is ignored for nonsequential file organizations.) When you do not specify */CCONTROL*, or when you do not specify a *keyword*, the carriage controls are set according to the file type.

Valid keywords include the following:

Keyword	Carriage control Attribute	Default
FORTRAN	The first character of each record will contain the appropriate carriage control information	
LIST	Implied carriage control (single spacing between records)	Default for */AS* files.
NONE	No carriage control information (any carriage control information will be placed in individual records)	Default for */IS* files.

/DEFAULTFILE **:*defaultspec***
Specifies a default to be applied to any missing components of the *filespec*. The *defaultspec* must be specified. APL first looks at the file specification named in the argument. If any components are missing, APL looks for a default in the */DEFAULTFILE* qualifier. If you omit the *defaultspec*, APL specifies the appropriate APL file type.

/DISPOSE **⟦:*keyword*⟧**
Specifies whether the file is temporary or permanent. */DISPOSE:KEEP*, the default, means the file is permanent. */DISPOSE:DELETE* means the file will be deleted when it is closed.

Other *keywords* send the file to a queue when the file is closed in accordance with the following:

Keyword	Definition
PRINT	Sends the file to SYS$PRINT. The file is not deleted.
PRINTDELETE	Sends the file to SYS$PRINT. The file is deleted when job is finished.
SUBMIT	Sends the file to SYS$BATCH. The file is not deleted.
SUBMITDELETE	Sends the file to SYS$BATCH The file is deleted when job is finished.

Note that you must have VMS delete privileges to use any of the delete *keywords*. If you do not have delete privileges, APL signals *FILE PROTECTION VIOLATION* when the file closes. As a result, APL closes the file, but does not delete it. If you receive the file protection violation error when you press Ctrl/Z, you can exit from APL by pressing Ctrl/Z a second time.

/EFN **:*n***
Associates an event flag with a channel number. For more information on event flags see ⎕*EFR*,⎕*EFS*, and ⎕*EFC*.

/MAXLEN **[[:*length*]]**
Specifies the maximum record length (in bytes) for a new file. It is ignored for existing files. The default length is the value of the ⎕*DML* system variable. The maximum record length value is also used as the maximum segment size for segmented records on output.

The maximum values are as follows:

- 32232 (for prolog 1 or 2)
- 32224 for */DA* and */KY* files (prolog 3)
- 32767 for */IS* files
- 2048 for */AS* files
- 32253 for */RF* files

When you write to an */AS* file in quad output mode, the maximum record length is determined by the current setting of ⎕*PW*. In all other output modes for all file types, the maximum record length is determined by */MAXLEN*.

/MBX
Indicates that an assigned file name actually refers to a mailbox.

/NFS
A non-file-structured qualifier that tells APL to read from the device without trying to interpret the data. In other words, to return the data on the device as a string of bits. This qualifier is useful when reading foreign devices.

/NOSHARE

/NOWRITERS
Allows you to write to a shareable file, but prevents other users from doing so.

/OPEN **[[:*keyword*]]**
Specifies that you want APL to open or create a file when the channel is assigned. Using the */OPEN* qualifier allows you to detect errors related to the openning or creating of a file at the time of assignment instead of at the time of the first I/O operation. Values for *keyword* include *NEW*, used to create a new file, and *OLD*, used to open an existing file.

/PROTECTION ⟦:***protection***⟧
Specifies the protection to be associated with a new file. It is ignored for existing files.

/READONLY ⟦:*NOLOCKS*⟧
Allows you to read the file but not write to it. The *NOLOCKS* argument specifies that records should be read even if they have been locked by another user. Using */READONLY:NOLOCKS* avoids waiting for a locked record to become unlocked. However, note that when ⎕*WAIT* is set to any value but the default (wait indefinitely), it overrides the *NOLOCKS* argument.

/RECORDTYPE ⟦:***keyword***⟧
Specifies the record format used by VAX RMS for each record of the file. The default is variable length records. APL ignores this qualifier if the file already exists or if the file type is */DA*, */RF*, or */KY*. You can use the following keywords as values to */RECORDTYPE*.

Keyword	**Record Format**
VARIABLE	Variable length
FIXED	Fixed length
STREAM	Stream format
STREAMCR	Stream format delimited with <CR>s
STREAMLF	Stream format delimited with <LF>s

Note that when you use fixed-length records, the record size is defined with the */MAXLEN* qualifier. The default value is ⎕*DML*.

Because APL adds a prefix containing system information to each record of a */IS* file, you may want to write data out to these files in pure data mode when using fixed-length records. Otherwise you need to calculate the size of the prefixed information before writing the data.

/SHARE
Specifies that several users may access the file simultaneously. All users sharing the file must use the */SHARE* qualifier when associating a given file with a channel. Sequential file users are exempted from this rule.

/SIGNAL
Specifies that APL signal the end-of-file indicator when you perform a read operation on a nonexistent record. For */AS* and */IS* files the indicator is *EOF ENCOUNTERED*. For */DA*, */RF*, and */KY* files the indicator is *EOF ENCOUNTERED* for a sequential read and *RECORD NOT FOUND* for a random read. If you do not

specify */SIGNAL*, APL returns an empty numeric matrix with the shape of 0 75 as the end-of-file indicator.

/UPDATE
Specifies that you want both read and write access to a sequential file and that APL should change the rules slightly for sharing the file. */UPDATE* is relevant for */AS* and */IS* files only and is ignored for all other file types.

When you use */UPDATE* you should consider how you want APL to deal with locked records. See */READONLY:NOLOCKS* and ⎕*RELEASE* for more information.

/WRITEONLY
Allows you to write to a file, but not read it. A new file is created when APL writes to the file. If the assignment specifies */OPEN:OLD*, a new file is not created. However, APL can write to an existing file only if the file is empty, or if */IS⋆* was specified for appending. Subsequent assignments can gain read access to the file.

Description

The action form of ⎕*ASS* associates files with channels. ⎕*ASS* does not create or open a file (unless you use the */OPEN* qualifier) or perform any input or output. It establishes a connection between a file specification (and related file information) and a specified channel.

When you perform I/O functions, you must refer to channel numbers rather than to file specifications. The APL functions that perform file I/O (⍇ and ⍈) require channel numbers—not file specifications—as part of their arguments. So, to read or write a file, you must first associate it with a channel.

The query form of ⎕*ASS* returns the current value of assignments made previously with the action form.

The result of the query form is a character vector or matrix that identifies the parameters you associated with the channels specified.

Note that the action and query forms of ⎕*ASS* are described in in the *VAX APL User's Guide*, along with other file I/O information.

Possible Errors Generated

Action Form

15 *DOMAIN ERROR (ERROR PARSING ARGUMENT TO CCONTROL)*

15 *DOMAIN ERROR (REDUNDANT KEYWORD OR QUALIFIER)*

15 *DOMAIN ERROR (CONFLICTING QUALIFIERS SPECIFIED)*

33 *IO ERROR (INVALID RECORD SIZE)*

33 *IO ERROR (FILE CURRENTLY LOCKED BY ANOTHER USER)*

68 *END OF FILE ENCOUNTERED*

69 *RECORD NOT FOUND*

74 *BLOCK TOO BIG*

Query Form

9 *RANK ERROR (NOT VECTOR DOMAIN)*

15 *DOMAIN ERROR (ENCLOSED HETEROGENEOUS ARRAY NOT ALLOWED)*

15 *DOMAIN ERROR (INVALID CHANNEL NUMBER)*

15 *DOMAIN ERROR (NOT AN INTEGER)*

27 *LIMIT ERROR (INTEGER TOO LARGE)*

⎕*AUS* Automatic Save of the Workspace

Type

System Variable

Form

⎕*AUS* ← *near-integer-singleton*
integer-scalar ← ⎕*AUS*

Value Domain

Type	Near-Integer
Shape	Singleton
Depth	0 or 1 (simple)
Value	0, 1, or 2
Default	0

Result Domain

Type	Integer
Rank	0 #(scalar)
Shape	⍳ 0 (scalar)
Depth	0 #(simple scalar)

Description

⎕*AUS* controls a feature that allows you to save the active workspace automatically at periodic intervals.

Workspace backup is often critical when you are performing extensive operation editing and debugging, or when you are using quad input to type a large table of values. You could back up your work by periodically issuing a)*SAVE* command. However, if you set ⎕*AUS* to 1 or 2, APL automatically saves the workspace every time you exit from function-definition mode, or every time quad input is requested from the terminal. Then, if the system crashes, you probably will have to reenter only a small amount of input. In addition, when ⎕*AUS* is set to 2, the)*OFF* command acts like the)*CONTINUE* command (see Chapter 3).

You can set ⎕*AUS* to 0, 1, or 2; the default is 0. When ⎕*AUS* equals 0, the automatic save feature is not activated. When ⎕*AUS* equals 1 or 2, the feature is activated and the workspace is saved in your default directory as follows:

Value of ⎕*AUS*	File Name of Saved Workspace
1	*APLxxxxxxxx.TMP*, where xxxxxxxx is the value of ⎕*UL*, represented in hexadecimal. (Note that ⎕*UL* is an integer that represents your process identification number.)
2	*CONTINUE.APL*

The name of the file saved when ⎕*AUS* is 1 can be represented as the following APL expression:

```
'APL',('0123456789ABCDEF'[⎕IO+(8⍴16)⊤⎕UL]),'.TMP'
```

The value of ⎕*AUS* is saved when you save the active workspace and can be localized in user-defined operations.

When ⎕*AUS* is 2, APL keeps all versions of *CONTINUE.APL*, even after the APL session ends.

When ⎕*AUS* is 1, APL deletes all old versions of *APLxxxxxxxx.TMP* each time a new version is created during the same APL session. When you successfully execute a *)SAVE*, *)OFF*, or *)CONTINUE* system command, APL deletes all *APLxxxxxxxx.TMP* files created during the session, including the one most recently created. Note, however, that APL does not delete *APLxxxxxxxx.TMP* files created in past APL sessions, unless such an *APLxxxxxxxx.TMP* file has exactly the same name as a newly created *APLxxxxxxxx.TMP* file. For example:

```
      ⎕AUS←1
      ⎕UL
88
      '0123456789ABCDEF'[⎕IO+(8⍴16)⊤⎕UL]

      ∇G
[1]   '1'
[2]   ∇
                       ⍝APL WRITES .TMP FILE
      )LIB *.TMP

Directory APLGRP:[USER]

APL00000058.TMP;1
```

```
Total of 1 file.
   .
   .
   .
         ⍝START NEW APL SESSION WITH NEW ⎕UL
      ⎕AUS←1
      ⎕UL
92
      '0123456789ABCDEF'[⎕IO+(8⍴16)⊤⎕UL]
0000005D
      ∇G
[1]   '1'
[2]   ∇
                          ⍝APL WRITES .TMP FILE
      )LIB *.TMP

Directory APLGRP:[USER]

APL00000058.TMP;1
APL0000005D.TMP;1

Total of 2 files.
      )SAVE ABC
SAVED THURSDAY 15-NOV-1990 14:33:21.32 6 BLKS
      )LIB *.TMP

Directory APLGRP:[USER]

APL00000058.TMP;1

Total of 1 file.
       ⍝APL DELETED NEW .TMP FILE BUT NOT .TMP
       ⍝ FILE CREATED IN EARLIER SESSION
```

Note

⎕*UL* has a value that changes each time you log in to VMS, and the names of the *APLxxxxxxxx.TMP* files are probably different if they are created during different VMS sessions. The names are the same if the files are created during the same VMS session, even if they are not created in the same APL session.

To recover the ⎕*AUS* file after a system crash, execute APL and issue a *)LIB* command to verify that the temporary file exists. Then use *)LOAD* to load the temporary file. APL prints the *)LOAD* message. The name of the active workspace is now the name that the workspace had before the backup was performed, not the name of the temporary file:

```
      )LIB *.TMP

Directory APLGRP:[USER]

APL0000005C.TMP;1

Total of 1 file.
      )LOAD APL0000005C.TMP
SAVED THURSDAY  15-NOV-1990 12:30:05.04 53 BLKS* WAS ABC
      )WSID
ABC
```

Possible Errors Generated

9 *RANK ERROR* (*NOT VECTOR DOMAIN*)

10 *LENGTH ERROR* (*NOT SINGLETON*)

15 *DOMAIN ERROR* (*INCORRECT TYPE*)

15 *DOMAIN ERROR* (*ENCLOSED ARRAY NOT ALLOWED*)

15 *DOMAIN ERROR* (*NOT AN INTEGER*)

15 *DOMAIN ERROR* (*PARAMETER OUT OF RANGE*)

27 *LIMIT ERROR* (*INTEGER TOO LARGE*)

⎕*AV* Atomic Vector

Type

Niladic System Function

Form

all-known-chars ← ⎕*AV*

Result Domain

Type	Character
Rank	1 (vector)
Shape	256
Depth	1 (simple)

Description

⎕*AV* contains a vector of the 256 characters known to APL. Table 2–4 shows the characters and their positions in the vector. The positions are based on an index origin of 0. The index of a character is the sum of its row and column numbers.

Symbols with indexes 213 through 255 have no meaning to the APL user. They are used only by the APL implementation and may not be used for input. On output, they print as squish quads (▯). If you include any of these characters in a character array, the results are unpredictable.

The following is useful to display the APL characters in ⎕*AV* order.

```
      6 32⍴32↓⎕AV
 ¨)<≤=>]∨∧≠÷,+./0123456789([;×:\
¯α⊥∩⌊∊_∇∆⍳∘'⎕|⊤○⋆?⍴⌈~↓∪⍵⊃↑⊂←⊢→≥-
⋄ABCDEFGHIJKLMNOPQRSTUVWXYZ{⊣}$
'abcdefghijklmnopqrstuvwxyz@"#%&
⍙A̲B̲C̲D̲E̲F̲G̲H̲I̲J̲K̲L̲M̲N̲O̲P̲Q̲R̲S̲T̲U̲V̲W̲X̲Y̲Z̲!⍝⌶⍎⍕
⌹⍇⍈⍞⍔⍋⍒⍫⍱⍲⍟⌽⍉⊖⍪⌿⍀⊆⊇≡^▯▯▯▯▯▯▯▯▯▯▯
```

Table 2–4 Elements of ⎕*AV*(⎕*IO*←0)

dec	0	32	64	96	128	160	192	224
0	NUL	SP	¯	⋄	‘	⍙	⍠	▯
1	SOH	¨	⍺	*A*	*a*	*A̲*	⍇	▯
2	STX	)	⊥	*B*	*b*	*B̲*	⍈	▯
3	ETX	<	∩	*C*	*c*	*C̲*	⍞	▯
4	EOT	≤	⌊	*D*	*d*	*D̲*	⍔	▯
5	ENQ	=	∊	*E*	*e*	*E̲*	⍋	▯
6	ACK	>	_	*F*	*f*	*F̲*	⍒	▯
7	BEL	]	∇	*G*	*g*	*G̲*	⍫	▯
8	BS	∨	∆	*H*	*h*	*H̲*	⍱	▯
9	HT	∧	⍳	*I*	*i*	*I̲*	⍲	▯
10	LF	≠	∘	*J*	*j*	*J̲*	⍟	▯
11	VT	÷	'	*K*	*k*	*K̲*	⌽	▯
12	FF	,	⎕	*L*	*l*	*L̲*	⍉	▯
13	CR	+	|	*M*	*m*	*M̲*	⊖	▯
14	SO	.	⊤	*N*	*n*	*N̲*	⍪	▯
15	SI	/	○	*O*	*o*	*O̲*	⌿	▯
16	DLE	0	⋆	*P*	*p*	*P̲*	⍀	▯
17	DC1	1	?	*Q*	*q*	*Q̲*	⊆	▯
18	DC2	2	⍴	*R*	*r*	*R̲*	⊇	▯
19	DC3	3	⌈	*S*	*s*	*S̲*	≡	▯
20	DC4	4	~	*T*	*t*	*T̲*	^	▯
21	NAK	5	↓	*U*	*u*	*U̲*	▯	▯
22	SYN	6	∪	*V*	*v*	*V̲*	▯	▯
23	ETB	7	⍵	*W*	*w*	*W̲*	▯	▯
24	CAN	8	⊃	*X*	*x*	*X̲*	▯	▯
25	EM	9	↑	*Y*	*y*	*Y̲*	▯	▯

(continued on next page)

Table 2–4 (Cont.) Elements of ⎕*AV*(⎕*IO*←0)

dec	0	32	64	96	128	160	192	224
26	SUB	(	⊂	*Z*	z	*Z̲*	▯	▯
27	ESC	[	←	{	@	!	▯	▯
28	FS	;	⊢	⊣	"	⍝	▯	▯
29	GS	×	→	}	#	⌶	▯	▯
30	RS	:	≥	$	%	⍎	▯	▯
31	US	\	–	DEL	&	⍕	▯	▯

The index of a character in ⎕*AV* is the sum of its row and column numbers.

Possible Errors Generated

None.

⎕*BOX* Forming Character Matrices and Vectors

Type

Ambivalent System Function

Form

boxed-text ← ⟦*delimiter*⟧ ⎕*BOX* *text*

Left Argument Domain

Type	Character
Shape	Vector domain
Depth	0 or 1 (simple)

Right Argument Domain

Shape	Matrix domain
Depth	0 or 1 (simple)

Result Domain

Rank	1 or 2
Shape	Matrix or Vector
Depth	1 (simple)

Description

⎕*BOX* produces a character matrix from a character vector or vice versa.

When the right argument is in the vector domain, ⎕*BOX* forms a matrix. When the right argument is a matrix, ⎕*BOX* forms a vector. If the right argument is an empty vector, the result is an empty character matrix with the shape 0 0. If the right argument is an empty matrix, the result is a vector containing a number of delimiters equal to the number of rows in the right argument matrix.

When producing a matrix, APL uses a delimiter to determine where to form rows. The left argument optionally specifies a delimiting string. The default delimiter is <CR> <LF> . The number of columns is equal to the longest string contained between any pair of delimiters. Shorter strings are padded with trailing blanks.

When producing a vector with the monadic form, APL removes any trailing blanks and inserts the <CR> <LF> delimiter at the end of each row.

When producing a vector with the dyadic form, APL does not remove trailing blanks from the rows of the matrix argument. It does insert the specified delimiter at the end of each row.

⎕BOX is particularly useful for forming a matrix from a vector consisting of lines of data delimited by <CR> <LF> pairs. ⎕BOX removes the <CR> <LF> and makes each line of data that was between <CR> <LF> s (or between a <CR> <LF> and the end or beginning of the vector) a row in the result matrix. Thus, the result matrix and right argument vector appear the same when displayed by APL.

Examples:

```
      B ← 'FIRST LINE
SECOND LINE .........IS LONGER

LINE FOUR'
      ⍴B
70
      (34 ≠ ⎕CTRL ⍳ B)/⍳⍴B    ⍝GENERATE INDEX OF <CR><LF>S
11 12 48 49 55 56
      ⍴⎕←⎕BOX B
FIRST LINE
SECOND LINE .........IS LONGER

LINE FOUR
4 35
                          ⍝NO <CR><LF> ADDED TO LAST ROW
      ⍴⎕←⎕BOX (⎕BOX B)
FIRST LINE
SECOND LINE .........IS LONGER

LINE FOUR
65
      ⍴⎕←⎕BOX 'A'
A
1 1
```

```
                           ⍝TRAILING <CR><LF> IGNORED
      ⍴⎕←⎕BOX 'A', ⎕CTRL [14 11]
A
1 1
      ⍴⎕←Y←',' ⎕BOX 'ABC,DE,FGHI'
ABC
DE
FGHI
3 4
      ⍴⎕←'!' ⎕BOX Y
ABC!DE!FGHI
11
      ⍴⎕←'AB' ⎕BOX 'XXXABYYYABZZZ'
XXX
YYY
ZZZ
3 3
      ⍴⎕←'A' ⎕BOX 1 1 1 ⍴ 'B'
B
1 1
      ⍴⎕←'A' ⎕BOX 1 1 1 ⍴ 'A'
                                (APL outputs a blank line.)
1 0
```

Possible Errors Generated

9 *RANK ERROR (NOT MATRIX DOMAIN)*

9 *RANK ERROR (NOT VECTOR DOMAIN)*

15 *DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)*

15 *DOMAIN ERROR (INCORRECT TYPE)*

⎕*BREAK* Suspending Execution

Type

Monadic System Function

Form

⎕*BREAK apl-expression*

Argument Domain

Type	Any
Shape	Any
Depth	0 or 1 (simple)

Result Domain

Type	None
Shape	None
Depth	None

Description

⎕*BREAK*suspends execution of the operation in which it is contained and returns you to immediate mode. It takes any APL object as an argument and prints the value of that argument before breaking to the terminal. However, the function itself has no explicit result.

To return to execution after a break, you can either branch to a specific line number (→3), or you can use the system function ⎕*LC*. Use of ⎕*LC* would return you to the breakpoint, that is, to the line where the ⎕*BREAK* executes. To resume at the line after the breakpoint, specify ⎕*LC*+1. For example:

```
      ∇FUNC
[1]   'FIRST LINE'
[2]   ⎕BREAK 'BREAK AT LINE 2'
[3]   'RESUME AT LINE 3'
[4]   ∇
      FUNC
FIRST LINE
BREAK AT LINE 2
      →⎕LC+1
RESUME AT LINE 3
```

It is illegal to use ⎕BREAK in immediate mode or as part of the argument to the execute (⍎)function.

Possible Errors Generated

```
15 DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)
```

⎕*CHANS* Returning Channel Numbers

Type

Niladic System Function (query)

Form

current-channels ← ⎕*CHANS*

Result Domain

Type	Integer
Rank	1
Shape	Vector
Depth	1 (simple)

Description

⎕*CHANS* displays all of the channel numbers currently associated with file specifications. The result is a vector. In the following example, channels 1 and 5 are each associated with a file:

```
      ⎕ASS '1 PLAN/AS' ⋄ ⎕ASS '5 ANALYSIS/AS'
1
5
      ⎕CHANS
1 5
      )CLEAR
CLEAR WS
      ⎕CHANS
                               (APL outputs a blank line)
```

If no channels are assigned, ⎕*CHANS* returns an empty numeric vector.

⎕*CHANS* is also described in the *VAX APL User's Guide* along with other file I/O information.

Possible Errors Generated

None.

⎕*CHS* Returning File Organization and Open Status

Type

Monadic System Function

Form

file-org/status ← ⎕*CHS chans*

Argument Domain

Type	Near-integer
Shape	Vector domain
Range	¯999 to 999 (but not 0)
Depth	0 or 1 (simple)

Result Domain

Type	Integer
Rank	1 or 2
Shape	Vector or matrix
Depth	1 (simple)

Description

⎕*CHS* returns the file organization and status of the files associated with the specified channels. The absolute value of *chans* represents the channels associated with the files you want information on. The following table gives the possible codes resulting from ⎕*CHS*.

First Element		Second Element	
Code	File Organization	Code	Open Status
0	Not applicable	0	Channel free
1	*/AS*	1	Assigned but not open
2	*/IS*	2	Open for output
3	Not applicable	3	Open for input
4	*/DA*	4	Open for input and output
5	Not applicable		
6	Not applicable		
7	*/RF*		
8	*/KY*		

If the argument is a singleton, ⎕*CHS* returns a two-item vector: the first item identifies the file's organization, and the second item identifies the file's open status. For example:

```
      ⎕CHS 1
1 3
```

This means that the file associated with channel 1 is an ASCII sequential file that is assigned and open for input.

If the argument is a vector of *n* items, the result is an array of shape *n* by 2 . For example, the following expression returns a 3-by-2 array:

```
      ⎕←FILS←⎕CHS⍳3
1 3
7 4
2 2
      ⍴FILS
3 2
```

⎕*CHS* returns a result of 0 2⍴0 if its argument is empty.

⎕*CHS* is also described in the *VAX APL User's Guide* along with other file I/O information.

Possible Errors Generated

9 *RANK ERROR (NOT VECTOR DOMAIN)*

15 *DOMAIN ERROR (ENCLOSED HETEROGENEOUS ARRAY NOT ALLOWED)*

15 *DOMAIN ERROR* (*INVALID CHANNEL NUMBER*)

15 *DOMAIN ERROR* (*NOT AN INTEGER*)

27 *LIMIT ERROR* (*INTEGER TOO LARGE*)

⎕CIQ and ⎕COQ Packing and Unpacking Data

Type

Dyadic System Function

Form

unpacked-data ← *packed-data* ⎕CIQ *header* ⟦*type*⟧

Left Argument Domain

Type	Near-integer
Shape	Vector domain
Depth	0 or 1 (simple)

Right Argument Domain

Type	Near-integer
Shape	Vector domain
Depth	0 or 1 (simple)

Result Domain

Type	Any
Rank	Any
Shape	Any
Depth	Any

Type

Dyadic System Function

Form

packed-data ← *data* ⎕COQ *header* ⟦*type*⟧

Left Argument Domain

Type	Any
Shape	Any
Depth	Any

Right Argument Domain

Type	Near-integer
Shape	Vector domain
Depth	0 or 1 (simple)

Result Domain

Type	Integer
Rank	1
Shape	Vector
Depth	1 (simple)

Parameters

unpacked-data
The variable associated with the unpacked data; it must be in the format of the result of ⎕*COQ*, with or without a header. It may be empty only if header is 0.

packed-data
Specifies the variable associated with the packed data.

data
Any array you want to pack into an integer vector.

header
0, 2, or 4. For ⎕*CIQ*, if a header exists, it is 2; if no header exists, it is 0. If you specify 0 and a header does exist, the header is treated as part of the data to be unpacked. With ⎕*COQ*, use 0 if you do not want a header; 2 if you do want a header; and 4 if you want only a header.

type
If specified, it indicates whether the data is to be converted to another data type before being packed. The possible values are listed in Table 2–5. The

possible effects of such a conversion are summarized in the *VAX APL User's Guide*. Omitting type has the same effect as using type 0.

Description

⎕*CIQ* and ⎕*COQ* allow you to accumulate data of different types into variables. You can then catenate the variables for storage as one logical record.

⎕*COQ* takes an argument of any data type and packs it into an integer vector by treating it as if it were a stream of bits. For example, you can put ASCII characters in one variable and internal APL characters in another variable, then catenate them, and write them to a file. Later, you can read the file and use ⎕*CIQ* to change the variables from integer vectors back into ASCII and APL characters.

The value of ⎕*COQ* is the packed data or, if the header parameter equals 4, the header information associated with the left argument.

The value of ⎕*CIQ* is the unpacked data. ⎕*CIQ* converts the packed data to the internal data type specified by the corresponding external data type identified by the type parameter (the possible effects of such a conversion are summarized in the *VAX APL User's Guide*.)

Table 2–5 Type Parameter Values

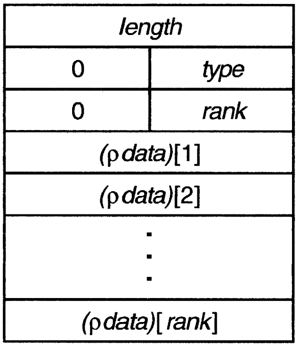
Type	External Data Type
0	No conversion; use type of "*data*"
1	Convert to 32-bit integer
2	Convert to 1-bit Boolean
3	Convert to F_floating single-precision floating-point
4	Convert to D_floating double-precision floating-point
5	Convert to 8-bit ^{2}AV characters
6	Convert to 8-bit ASCII characters
7	Convert to 8-bit unsigned numeric bytes
8	Convert to G_floating double-precision floating-point
9	Convert to H_floating floating-point
10	Convert to 16-bit integer
11	Convert to 8-bit Digital Multinational Characters

(continued on next page)

Table 2–5 (Cont.) Type Parameter Values

Type	External Data Type
12	Convert to 8-bit ⎕*AV* characters in TTY mnemonics
13	Convert to 8-bit ⎕*AV* characters in KEY-paired APL
14	Convert to 8-bit ⎕*AV* characters in BIT-paired APL
15	Convert to 8-bit ⎕*AV* characters in APL COMPOSITE

The header generated by ⎕*COQ* has the following format:



NU–2234A–RA

Each large box represents a longword as described in the following list:

- *length* is the length of the integer vector result of ⎕*COQ* 2
- *type* is one of the external types in Table 2–5
- *rank* is the rank of the data that was packed by ⎕*COQ*

The next *n* (*n* = *rank*) boxes contain the shape of the data that was packed.

For example:

```
      A←⍳5
      P←A ⎕COQ 2
      P
9 1 1 5 1 2 3 4 5
      B←3 4ρ1 0
      B
1 0 1 0
1 0 1 0
1 0 1 0
      Q←B ⎕COQ 2
      Q
6 2 2 3 4 1365
      BOOL←(32ρ2)⊤1365
      ¯12 ↑ BOOL
0 1 0 1 0 1 0 1 0 1 0 1
      P ⎕CIQ 2
1 2 3 4 5
      ⎕CIQ 2
1 0 1 0
1 0 1 0
1 0 1 0
```

The first example uses ⎕COQ to pack the vector ⍳ 5. The value of P shows that when ⍳ 5 was packed, the packed data had length 9 (including the header), type 1 (integer), rank 1, and shape 5.

In the second example, the Boolean matrix B is packed into Q. The value of Q indicates that the packed data with its header has length 6, type 2 (Boolean), rank 2, and shape 3 4. Note that when Q[6] is converted to Boolean, the value is the same as that of ⌽ , B (the Boolean data values are stored right to left).

Finally, ⎕CIQ was used to unpack the data, to retrieve the data and translate it back to the original data.

In the following example, the functions PACK and UNPACK use ⎕COQ and ⎕CIQ to pack and unpack values of different types:

```
      ∇P←PACK LIST;I
[1]    ⍝PACK USES ⎕COQ TO PACK A SET OF VALUES
[2]    ⍝INTO A SINGLE VARIABLE.  THE VALUES CAN BE
[3]    ⍝DIFFERENT TYPES OR SHAPES.  THUS, YOU CAN
[4]    ⍝USE PACK WHEN CATENATE WON'T WORK.
[5]    ⍝LIST IS A CHARACTER MATRIX, EACH ROW OF WHICH
[6]    ⍝CONTAINS THE NAME OF A VARIABLE WHOSE VALUE
[7]    ⍝IS TO BE PACKED.
[8]    ⍝P IS THE RESULTANT PACKED VALUE; IT IS AN
[9]    ⍝INTEGER ARRAY.
[10]   ⍝
[11]   P←⍳0
[12]   I←1
[13]   TEST:→(I>1↑⍴LIST)/0
[14]   P←P, (⍎LIST[I;])⎕COQ 2
[15]   I←I+1
[16]   →TEST
[17]   ∇
      ∇P UNPACK LIST;DATA;I;J;LEN;ENTRY
[1]    ⍝UNPACK USES ⎕CIQ TO UNPACK A VARIABLE CREATED
[2]    ⍝BY PACK INTO A SET OF VARIABLE NAMES.
[3]    ⍝P IS THE PACKED VALUE, CREATED BY PACK.
[4]    ⍝LIST IS A CHARACTER MATRIX, EACH ROW OF WHICH
[5]    ⍝CONTAINS THE NAME OF A VARIABLE TO RECEIVE ONE
[6]    ⍝OF THE PACKED VALUES.  UNPACKED VALUES FROM P
[7]    ⍝ARE STORED INTO SUCCESSIVE VARIABLES IN LIST.
[8]    DATA←P
[9]    I←⍴DATA
[10]   J←1
[11]   TEST:→(I≤0)/0
[12]   →(J>1↑⍴LIST)/0
[13]   LEN←DATA[1]
[14]   ENTRY←DATA [⍳LEN]
[15]   ⍎ LIST[J;], '←ENTRY ⎕CIQ 2'
[16]   DATA←LEN ↓ DATA
[17]   J←J+1
[18]   I←I-LEN
[19]   →TEST
[20]   ∇
                          ⍝DEFINE SOME NUMERIC VARIABLES:
      A←1
      AA← 1 1
      AAA← 1 1 1
                          ⍝DEFINE SOME CHARACTER VARIABLES:
      B←'B'
      BB←'BB'
```

```
                          ⍝MAKE A LIST OF INPUT VARIABLE NAMES:
      L1←5 3⍴ 'A  AA AAAB  BB '
      L1
A
AA
AAA
B
BB
                          ⍝PACK THESE VARIABLES INTO ONE ITEM
                          ⍝CATENATE WON'T WORK
      P←PACK L1
      P
4 2 0 1 5 2 1 2 3 4 5 2 1 3 7 4 5 0 98 5 5 1 2 25186
                          ⍝MAKE A LIST OF OUTPUT VARIABLE NAMES:
      L2←5 3⍴'X  XX XXXY  YY '
      L2
X
XX
XXX
Y
Y
               ⍝UNPACK THE PREVIOUS DATA INTO NEW VARIABLES:
      P UNPACK L2
               ⍝THE RESTORED DATA IS THE SAME AS THE
               ⍝ DATA THAT WAS PACKED.
      X=A
1
      XX=AA
1 1
      XXX=AAA
1 1 1
      Y=B
1
      YY=BB
1 1
```

Possible Errors Generated

For ⎕CIQ:

9 *RANK ERROR (NOT VECTOR DOMAIN)*

10 *LENGTH ERROR (ARGUMENT MUST BE* 1 *OR* 2 *ELEMENTS)*

10 *LENGTH ERROR (DATA TYPE MISSING)*

10 *LENGTH ERROR (DATA TYPE EXCEEDS DATA LENGTH)*

15 *DOMAIN ERROR*

15 *DOMAIN ERROR (DATA TYPE MUST BE UNSPECIFIED OR ZERO)*

15 *DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)*

15 *DOMAIN ERROR (INCORRECT TYPE)*

15 *DOMAIN ERROR (INVALID CIQ HEADER)*

15 *DOMAIN ERROR (INVALID EXTERNAL DATA TYPE)*

15 *DOMAIN ERROR (INVALID HEADER TYPE)*

15 *DOMAIN ERROR (NOT AN INTEGER)*

27 *LIMIT ERROR (INTEGER TOO LARGE)*

For ⎕*COQ*:

9 *RANK ERROR (NOT VECTOR DOMAIN)*

10 *LENGTH ERROR (ARGUMENT MUST BE* 1 *OR* 2 *ELEMENTS)*

15 *DOMAIN ERROR*

15 *DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)*

15 *DOMAIN ERROR (INCORRECT TYPE)*

15 *DOMAIN ERROR (INVALID HEADER TYPE)*

15 *DOMAIN ERROR (INVALID EXTERNAL DATA TYPE)*

15 *DOMAIN ERROR (NOT AN INTEGER)*

27 *LIMIT ERROR (INTEGER TOO LARGE)*

⎕*CLS* Closing Files

Type

Monadic System Function (quiet)

Form

```
ι0 ← ⎕CLS chans
```

Argument Domain

Type	Near-integer
Shape	Vector domain
Depth	0 or 1 (simple)
Value	¯999 through 999 (but not 0)

Result Domain

Type	Numeric
Rank	1 (vector)
Shape	0 (empty)
Depth	1 (simple)

Description

⎕*CLS* closes one or more files without deassigning their corresponding channels. The absolute values of *chans* represent the channels associated with the files you want to close.

⎕*CLS* is useful when you want to return to the beginning of a sequential file. (You can also use the ⎕*REWIND* system function, which does not close files.)

After you close a channel, a read function opens the file and reads the first record, and a write function creates a new version of the file (except for */DA*, */RF*, and */KY* files, where a new file is created only if no version currently exists). With ⎕*CLS*, there is no need to reassign the file to the channel.

The following line closes the file associated with channel 2:

```
⎕CLS 2
```

Any unassigned channels in the argument are ignored.

⎕*CLS* is a quiet function; it does not return a result if it is the leftmost function in a statement. When ⎕*CLS* is not the leftmost function, it returns an empty numeric vector. If its argument is empty, ⎕*CLS* has no effect.

Note that when you use ⎕*CLS*, you activate whichever parameter has been set for the */DISPOSE* qualifier on ⎕*ASS*. For example, if you specify */DISPOSE:DELETE*, APL deletes the file when you specify the ⎕*CLS* function.

APL automatically closes and deassigns all open files when you press Ctrl/Z or execute a)*LOAD*,)*CLEAR*,)*OFF*, or)*CONTINUE* system command (the)*MON* and)*PUSH* system commands do not have this effect).

⎕*CLS* is described in the *VAX APL User's Guide* along with other file I/O information.

Possible Errors Generated

```
9  RANK ERROR (NOT VECTOR DOMAIN)

15  DOMAIN ERROR (ENCLOSED HETEROGENEOUS ARRAY NOT ALLOWED)

15  DOMAIN ERROR (INVALID CHANNEL NUMBER)

15  DOMAIN ERROR (NOT AN INTEGER)

27  LIMIT ERROR (INTEGER TOO LARGE)
```

⎕*CR* Obtaining a Canonical Representation

Type

Monadic System Function

Form

canonical-rep ← ⎕*CR* *operation-name*

Argument Domain

Type	Character
Shape	Vector domain
Depth	0 or 1 (simple)

Result Domain

Type	Character
Rank	2
Shape	Matrix
Depth	1 (simple)

Implicit Arguments

⎕*PP* (controls precision of numeric constants)

Description

⎕*CR* provides a canonical representation of a user-defined operation, which enables you to treat the operation as data. A canonical representation is a character matrix with rows that are the original lines of the operation definition, but are reformatted so that they are the same length.

The canonical representation consists of exactly what you typed when you defined the operation, minus the beginning and ending ∇ s, plus blanks added to the end of lines to make their lengths the same as that of the longest line of the operation. Line numbers and brackets are removed from the definition. White space at the beginning (but not at the end) of a line is preserved. Lines that contain labels are not shifted.

The argument of the ⎕*CR* system function is a character array representing the name of the operation. The shape of the argument must be in the vector domain.

If the argument is empty or does not represent the name of an existing unlocked operation, the resulting character matrix is an empty matrix, with the shape `0 0`. (APL considers primitive system functions and external functions as locked.)

The display of numeric constants in an operation definition is ⎕*PP*-dependent.

⎕*CR* does not work on operands to user-defined operators that contain derived functions. Use ⎕*VR* instead.

For example:

```
      ∇MEANX←NSUBJ MEAN X
[1]   ⍝SUM VECTOR X
[2]   SUMX←+/X
[3]   MEANX←SUMX÷NSUBJ
[4]   ∇
      ⎕ ← SHOWCRFX ← ⎕CR 'MEAN'
MEANX←NSUBJ MEAN X
⍝SUM VECTOR X
SUMX←+/X
MEANX←SUMX÷NSUBJ
      ⍴SHOWCRFX
4 21
      X← 8 6 3 9 5 4 2 1 7 4
      10 MEAN X
4.9
```

The ⎕*FX* system function is the inverse of ⎕*CR*.

Possible Errors Generated

```
9  RANK ERROR (NOT VECTOR DOMAIN)

15  DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

15  DOMAIN ERROR (INCORRECT TYPE)
```

⎕*CT* Comparison Tolerance

Type

System Variable

Form

⎕*CT* ← *tolerance-value*
floating-scalar ← ⎕*CT*

Value Domain

Type	Non-negative numeric
Shape	Singleton
Depth	0 or 1 (simple)
Value	0 to 2.328*E*¯10
Default	1*E*¯15

Result Domain

Type	Numeric
Rank	0
Shape	⍳0 (scalar)
Depth	0 (simple)

Description

⎕*CT* specifies the degree of tolerance applied when two numbers are compared for equality. If the difference between two numbers is less than or equal to the value of ⎕*CT* times the larger number, the numbers are considered equal.

The value of ⎕*CT* affects the following primitive functions:

Function	Function Name	Function	Function Name
∊	Set membership	|	Residue
⍳	Index of	⌈	Ceiling
>	Greater than	⌊	Floor

Function	Function Name	Function	Function Name
≥	Greater than or equal to	∪	Set union and unique
=	Equal to	∩	Set intersection
≠	Not equal to	~	Without
≤	Less than or equal to	⊆	Subset
<	Less than	⊇	Contains
≡	Match	⌹	Matrix inverse and divide

The value of ⎕*CT* is saved when you save the active workspace and can be localized in user-defined operations.

For example:

```
      ⎕CT
1E¯15
      1 = 1.00000000009
0
      ⎕CT←1E¯10
      1 = 1.00000000009
1
```

The following function is the APL metafunction. It describes an exact definition of how ⎕*CT* is applied.

```
      ∇Z←A DFEQ B ;⎕CT;T ⍝A=B WITHIN BXCT
[1]   ⎕CT←0
[2]   T←0≤(×A)B
[3]   A←A×T
[4]   B←B×T
[5]   Z←(|A-B)≤⎕CT×(|A)⌈|B
[6]   Z←Z×T ∇
```

Possible Errors Generated

9 *RANK ERROR (NOT VECTOR DOMAIN)*

10 *LENGTH ERROR (NOT SINGLETON)*

15 *DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)*

15 *DOMAIN ERROR (INCORRECT TYPE)*

15 *DOMAIN ERROR (PARAMETER OUT OF RANGE)*

⎕*CTRL* Control Characters

Type

Niladic System Function

Form

control-chars ← ⎕*CTRL*

Result Domain

Type	Character
Rank	1 (vector)
Shape	33
Depth	1 (simple)

Description

⎕*CTRL* is a subset of ⎕*AV*. It returns a vector of the 32 ASCII control characters and Delete, or, expressed in terms of ⎕*AV*:

```
⎕AV[⍳32,⎕IO+127]
```

The control characters are listed in the table below. Note that for any formatting control character, the internal code that appears in ⎕*CTRL* is the same as the internal code used by APL for that character. For example:

```
      ⎕IO←0    ⋄  ⎕AV ⍳ 'ABCDEF'
97 98 99 13 10 32 32 32 32 32 100 101 102
```

Index	Name	Description	Octal Value	Hex Value
0	NUL	Null	000	00
1	SOH	Start Of Heading	001	01
2	STX	Start of TeXt	002	02
3	ETX	End of TeXt	003	03
4	EOT	End Of Transmission	004	04
5	ENQ	ENQiry	005	05

Index	Name	Description	Octal Value	Hex Value
6	ACK	ACKnowledge	006	06
7	BEL	BELl	007	07
8	BS	BackSpace	010	08
9	HT	Horizontal Tabulation	011	09
10	LF	<LF>	012	0A
11	VT	Vertical Tabulation	013	0B
12	FF	Form Feed	014	0C
13	CR	<CR>	015	0D
14	SO	Shift Out	016	0E
15	SI	Shift In	017	0F
16	DLE	Data Line Escape	020	10
17	DC1	Device Control 1	021	11
18	DC2	Device Control 2	022	12
19	DC3	Device Control 3	023	13
20	DC4	Device Control 4	024	14
21	NAK	Negative AcKnowledge	024	15
22	SYN	SYNchronous Idle	026	16
23	ETB	End-of-Transmission Block	027	17
24	CAN	CANcel	030	18
25	EM	End of Medium	031	19
26	SUB	SUBstitute	032	1A
27	ESC	ESCape	033	1B
28	FS	File Separator	034	1C
29	GS	Group Separator	035	1D
30	RS	Record Separator	036	1E
31	US	Unit Separator	037	1F
32	DEL	DELete	177	7F

Possible Errors Generated

None.

⎕*DAS* Deassigning Files

Type

Monadic System Function (quiet)

Form

```
ι0 ← ⎕DAS chans
```

Argument Domain

Type	Near-integer
Shape	Vector domain
Depth	0 or 1 (simple)
Value	¯999 through 999 (but not 0)

Result Domain

Type	Numeric
Rank	1 (vector)
Shape	0 (empty)
Depth	1 (simple)

Description

⎕*DAS* dissociates or deassigns file specifications from channel numbers. The absolute value of *chans* represents the channels associated with the files you want to deassign. If any files associated with the specified channel numbers have not been closed (by ⎕*CLS*), ⎕*DAS* closes them, and then deassigns them.

In general, ⎕*DAS* reverses the operations performed by the ^{2}*ASS* system function. The following line deassigns the files associated with channels 1, 3, and 5:

```
⎕DAS 1 3 5
```

Any unassigned channels in the argument are ignored.

⎕*DAS* is described in the *VAX APL User's Guide* along with other file I/O information.

Possible Errors Generated

```
9  RANK ERROR (NOT VECTOR DOMAIN)

15  DOMAIN ERROR (ENCLOSED HETEROGENEOUS ARRAY NOT ALLOWED)

15  DOMAIN ERROR (INVALID CHANNEL NUMBER)

15  DOMAIN ERROR (NOT AN INTEGER)

27  LIMIT ERROR (INTEGER TOO LARGE)
```

⎕*DC* Display Control

Type

System Variable

Form

⎕*DC* ← *display-area box-characters*
current-setting ← ⎕*DC*

Value Domain

Type	Enclosed, heterogeneous (see below)
Shape	2 (vector)
Depth	2 or 3 (enclosed)
Default	(¯1 1 0 2) ''

Result Domain

Type	Enclosed, heterogeneous
Rank	1
Shape	2 (vector)
Depth	2 or 3 (enclosed)

Description

⎕*DC* specifies how APL displays enclosed arrays. You can set ⎕*DC* to draw boxes around enclosed items of an array, and the resulting display makes the nested structure of the array clearer. You can also increase the blank space that APL uses to surround an enclosed item.

The value you assign to ⎕*DC* is a two-item enclosed vector.

The first item of the ⎕*DC* value is a simple numeric vector of length 4. Data elements 1 and 2 of this item specify where an item is displayed when its display area is larger than the structure of the item itself. The first data element controls the vertical placement; the item can be at the top, center, or bottom of the display area. The second data element controls the horizontal placement; the item can be at the left, center, or right of the display area. The following table describes the meaning of the values you can specify for these two data elements:

Positioning Items in Display Areas

First Element	Location	Second Element	Location
¯1	Top	¯1	Left
0	Center	0	Center
1	Bottom	1	Right

Data elements 3 and 4 of the first item of the ⎕*DC* value allow you to change the size of the display areas. The third data element controls the vertical space between rows of items; the integer you specify indicates how many blank rows you want to add. The fourth element controls the horizontal space between columns; the integer you specify indicates how many blank columns you want to add. (Note that the rows and columns containing the characters that form the boxes are included in the number you specify. When you display boxes, the minimum value you can specify for the third and fourth elements is 2.) The default for the third and fourth elements are 0 (no extra rows between rows of items) and 2 (two extra columns between columns of items), respectively.

The second item is a character vector that is either empty (''), if you do not want boxes around enclosed items, or has length 8. The vector specifies the characters for APL to use when it draws boxes around enclosed items. The first four items specify the symbols for the corners of boxes (upper left, upper right, lower left, lower right), the next two items specify the left and right sides, and the last two specify the top and bottom.

The following table describes the order and meaning of the eight items in the second item of the ⎕*DC* value:

Position in Second Item	Portion of Box Described	Shape
1	Upper left corner	Singleton
2	Upper right corner	Singleton
3	Lower left corner	Singleton
4	Lower right corner	Singleton
5	Left side	Vector

Position in Second Item	Portion of Box Described	Shape
6	Right side	Vector
7	Top	Vector
8	Bottom	Vector

The default value of ⎕*DC* is `(¯1 1 0 2) ''`. The four data elements of the non-empty item have the following meanings: `¯1` positions data at the top of each display area, `1` justifies data to the right, `0` places no extra blank lines between rows of data, and `2` places an extra blank between columns of enclosed items. For example:

```
      ⎕←CSNY←3 3ρ'LONG' 9823 834 'TIME' 98 23 'COMIN' ¯2 'YO'
+----+  9823 834
|LONG|
+----+
+----+  98   23
|TIME|
+----+
+-----+ ¯2   +--+
|COMIN|      |YO|
+-----+      +--+
      ⎕←MUSC←(1 2) (3 4 5) 6 (7 8 9 10)
+---+ +-----+ 6 +--------+
|1 2| |3 4 5|   |7 8 9 10|
+---+ +-----+   +--------+
      ⍝WITHOUT EXTRA BLANKS, MUSC WOULD APPEAR AS ⍳10
      ⍳10
1 2 3 4 5 6 7 8 9 10
```

The displays that follow show CSNY and MUSC with the addition of the delta (Δ) symbol to clarify the location of the blanks.

```
ΔLONGΔΔ9823ΔΔ834
ΔTIMEΔΔΔ98ΔΔΔ23
COMINΔΔΔΔ¯2ΔΔΔYO

1Δ2ΔΔ3Δ4Δ5ΔΔ6ΔΔ7Δ8Δ9Δ10
```

The following examples describe the use of the second item of the ⎕*DC* value, which specifies the boxes for APL to draw. If you specify `'++++||=='` as the second item, APL draws boxes that look like the following:

```
       +=====+
       |     |
       |     |
       +=====+
```

The items you specify for the corners of boxes must be singleton items. However, the four sides may be character strings (vectors). (Note that the depth increases from two to three when you specify a character string for one or more of the four sides.) For example, if you specified `'+' '+' '+' '+' '\/' '\/' 'PETER' 'PETER'` as the second item of the ⎕*DC* value, APL would draw boxes that look like the following:

```
+PETER+
\     \
/     /
+PETER+
```

If a dimension of the box requires fewer characters than the string you specify, APL uses only the number required. If the box requires more characters, APL reuses the string. For example, the boxes might look like the following:

```
+PET+
\   \
+PET+

+PETERPET+
\        \
/        /
\        \
/        /
+PETERPET+
```

When APL displays an array, it places each item of the array into a display area. If all items have the same shape, the display areas are all the same size. If the items vary in shape (as they often do), the display areas also vary in size. For any given row, the vertical dimension of the display area is determined by the maximum number of rows in any item in that row. For any given column, the horizontal dimension of the display area is determined by the maximum number of columns in any item in that column. For example:

```
      ⎕DC
+---------+ +--------+
|¯1 ¯1 2 3| |++++||--|
+---------+ +--------+
      BUMP←2 2⍴(⍳10) 5 (3 4⍴⍳12) 'ABC'
      BUMP
+----------------------+ 5
|1 2 3 4 5 6 7 8 9 10|
+----------------------+
+----------+           +---+
|1  2  3  4|           |ABC|
|5  6  7  8|           +---+
|9 10 11 12|
+----------+
```

Note the dimensions of the display areas in the preceding example. BUMP[1;1](⊂⍳10) determines the dimension for the first column because it is wider than BUMP[2;1](⊂3 4⍴⍳12). BUMP[2;2] determines the dimension for the second column because it is wider than BUMP[1;2] (5). BUMP[1;1] determines the dimension for the first row because it has more rows than BUMP[1;2] (the rows of the box are part of the display size of BUMP[1;1]). Finally, BUMP[2;1] determines the dimension for the second row because it has more rows than BUMP[2;2](⊂'ABC').

Note that this manual displays enclosed items as if the ⎕DC default were the following:

```
      ⎕DC
+---------+ +--------+
|¯1 ¯1 2 3| |++++||--|
+---------+ +--------+
```

The ⎕DC setting shown in the preceding example places items in the top left corner of each display area, adds two extra rows between rows of items, adds three extra columns between columns of items, and draws boxes using plus signs, vertical bars, and hyphens.

Examples:

```
      ⎕DC←(¯1 ¯1 2 3) '++++||--'
      ⎕DC
+---------+ +--------+
|¯1 ¯1 2 3| |++++||--|
+---------+ +--------+
      BUMP←2 2⍴(⍳10) 5 (3 4⍴⍳12) 'ABC'
                          ⍝CHANGE POSITION IN DISPLAY AREA
      ⎕DC←(0 0 2 3) '++++||--'
      ⎕DC
+-------+ +--------+
|0 0 2 3| |++++||--|
+-------+ +--------+
      BUMP
+--------------------+
|1 2 3 4 5 6 7 8 9 10|    5
+--------------------+
     +----------+
     |1  2  3  4|      +---+
     |5  6  7  8|      |ABC|
     |9 10 11 12|      +---+
     +----------+
                          ⍝CHANGE SIZE OF DISPLAY AREAS
      ⎕DC←(0 0 4 5) '++++||--'
      ⎕DC
+-------+   +--------+
|0 0 4 5|   |++++||--|
+-------+   +--------+
      ≡⎕DC
2
      BUMP
+--------------------+
|1 2 3 4 5 6 7 8 9 10|      5
+--------------------+


     +----------+
     |1  2  3  4|        +---+
     |5  6  7  8|        |ABC|
     |9 10 11 12|        +---+
     +----------+
      ⍝CHANGE BOX, REDUCE DISPLAY AREA  USE E FOR ELEMENT 7
      ⍝ OF SECOND ITEM, F FOR ELEMENT 8 OF SECOND ITEM
      E←'--ρ--------------'
      F←'--∇--------------'
      ⎕DC←(0 0 2 3) ('+' '+' '+' '+' '|' '|' E F)
      ⎕DC                 ⍝DISPLAY ⎕DC VALUE
          +--ρ----------------ρ----------------ρ--------------+
+--ρ----+ |            +--ρ--------------+ +--ρ--------------+|
|0 0 2 3| |+ + + + | | |--ρ--------------| |--∇--------------||
+--∇----+ |            +--∇--------------+ +--∇--------------+|
          +--∇----------------∇----------------∇--------------+
      ≡⎕DC
3
```

```
      BUMP
+--⍴----------------⍴+
|1 2 3 4 5 6 7 8 9 10|    5
+--∇----------------∇+
     +--⍴-------+
     |1  2  3  4|      +--⍴+
     |5  6  7  8|      |ABC|
     |9 10 11 12|      +--∇+
     +--∇-------+
                          ⍝STOP DRAWING BOXES
      ⎕DC←(0 0 2 3) ''
      BUMP
1 2 3 4 5 6 7 8 9 10    5


     1  2  3  4
     5  6  7  8        ABC
     9 10 11 12
      ⎕DC←(¯1 ¯1 2 3) '++++||--'
      B←4 ⋄ C← ⍳5 ⋄ D←2 2⍴ 'ABCD'
      ⎕←A←⊂A←B, (⊂C), ⊂D
+------------------+
|4 +---------+ +--+|
|  |1 2 3 4 5| |AB||
|  +---------+ |CD||
|              +--+|
+------------------+
         ⍝THE FOLLOWING WORKS WELL TO DISPLAY NESTS OF VECTORS
         ⍝ IN A FORM SIMILAR TO STRAND NOTATION.
      ⎕DC←(0 ¯1 0 1)('+' '+' '+' '+' '(' ')' '' '')
      A
(4 (1 2 3 4 5) (AB))
(              (CD))
         ⍝THE FOLLOWING PUTS PARENTHESES AROUND ARRAYS
      ⎕DC←(0 0 2 3) ('/' '\' '\' '/' '|' '|' ' ' ' ')
      A
/                  \
|  /         \ /  \|
|4 |1 2 3 4 5| |AB||
|  \        /  |CD||
|              \  /|
\                  /
```

Possible Errors Generated

9 *RANK ERROR (MUST BE VECTOR)*

9 *RANK ERROR (NOT SINGLETON)*

9 *RANK ERROR (NOT VECTOR DOMAIN)*

10 *LENGTH ERROR (DISPLAY CONTROL ITEM WRONG LENGTH)*

10 *LENGTH ERROR (DISPLAY CONTROL VECTOR MUST BE TWO ITEMS)*

10 *LENGTH ERROR (NOT SINGLETON)*

15 *DOMAIN ERROR*

15 *DOMAIN ERROR (ENCLOSED HETEROGENEOUS ARRAY NOT ALLOWED)*

15 *DOMAIN ERROR (ENCLOSED VALUE REQUIRED)*

15 *DOMAIN ERROR (INCORRECT TYPE)*

15 *DOMAIN ERROR (NEGATIVE INTEGER NOT ALLOWED)*

15 *DOMAIN ERROR (PARAMETER OUT OF RANGE)*

⎕*DL* Delaying Execution

Type

Monadic System Function

Form

actual-delay ← ⎕DL *seconds*

Argument Domain

Type	Numeric
Shape	Singleton
Depth	0 or 1 (simple)
Value	*seconds* < ¯1+2*18

Result Domain

Type	Non-negative numeric
Rank	0
Shape	⍳0 (scalar)
Depth	0 (simple)

Description

⎕*DL* delays execution for the number of seconds specified in the argument. If the argument is less than 0.001, there is no delay. If the argument is negative, there is no delay.

Although ⎕*DL* specifies the desired duration of the delay, the actual delay may be longer because of other demands on the system. The result returned is the actual delay in seconds. For example:

```
      ⎕ ← ⎕DL 8.5              ⍝SET DELAY AND DISPLAY
9
      ⎕ ← ⎕DL ¯74.36           ⍝NEGATIVE ARG = NO DELAY
0
```

Here, the user instructed APL to wait 8.5 seconds before prompting for input; the actual delay was 9 seconds.

The ⎕*DL* function uses a negligible amount of computer time. Thus, you can issue it freely when tests are required at periodic intervals to determine whether an event has occurred as expected.

The delay resulting from the execution of ⎕*DL* may be canceled by the weak attention signal. When the weak attention signal is thus used, APL stops ⎕*DL* and returns the actual delay but does not signal attention.

Possible Errors Generated

9 *RANK ERROR* (*NOT VECTOR DOMAIN*)

10 *LENGTH ERROR* (*NOT SINGLETON*)

15 *DOMAIN ERROR* (*ENCLOSED ARRAY NOT ALLOWED*)

15 *DOMAIN ERROR* (*INCORRECT TYPE*)

27 *LIMIT ERROR* (*DELAY VALUE TOO LARGE*)

⎕*DML* Maximum Record Length

Type

System Variable

Form

⎕DML ← default-length
integer-scalar ← ⎕DML

Value Domain

Type	Near-integer
Shape	Singleton
Depth	0 or 1 (simple)
Value	512 through 2048 (bytes)
Default	2044

Result Domain

Type	Integer
Rank	0
Shape	⍳0 (scalar)
Depth	0 (simple scalar)

Description

⎕*DML* specifies the default maximum record length to be used when you save a workspace or create an external file. The value of ⎕*DML* is saved with the workspace and can be localized within user-defined operations.

If you do not want to use the default maximum record length, you can use the */MAXLEN* qualifier when you save a workspace or create an external file. If you omit the */MAXLEN* qualifier, APL uses the value of ⎕*DML* as the maximum record length.

Possible Errors Generated

9 *RANK ERROR (NOT VECTOR DOMAIN)*

10 *LENGTH ERROR (NOT SINGLETON)*

15 *DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)*

15 *DOMAIN ERROR (INCORRECT TYPE)*

15 *DOMAIN ERROR (NOT AN INTEGER)*

15 *DOMAIN ERROR (PARAMETER OUT OF RANGE)*

27 *LIMIT ERROR (INTEGER TOO LARGE)*

⎕*DVC* Returning Device Characteristics

Type

Monadic System Function (query)

Form

characteristics ← ⎕*DVC* *chans*

Argument Domain

Type	Near-integer
Shape	Vector domain
Depth	0 or 1 (simple)
Value	¯999 through 999 (but not 0)

Result Domain

Type	Integer
Rank	1 or 2
Shape	Vector or matrix (*n* by 2)
Depth	1 (simple)

Description

⎕*DVC* displays device characteristics. The absolute value of *chans* represents the channels associated with the files you want information on.

For each channel specified in the argument, ⎕*DVC* returns one row containing two values. The first value is the VMS device-characteristics longword, and the second value is always 0. For unassigned channels, ⎕*DVC* returns 0 0.

⎕*DVC* returns a two-element vector if a single channel is specified. If more than one channel is specified, the result is a matrix of shape *n* by 2, where *n* is the length of the argument.

If its argument is empty, ⎕*DVC* returns a result of 0 2⍴0.

Note that to return a value for ⎕*DVC*, APL must open files that have been associated with channels but have not yet been opened. Thus, unopened files associated with channels identified by positive integers in the ⎕*DVC* argument are opened for input; unopened files associated with channels identified in the argument by negative integers are opened for output. Note that when you open a sequential file for output, APL makes a new copy of the file with a version number that is one higher than that of the previous copy.

It is usually helpful to convert the device-characteristics longword to binary format before examining it. For example:

```
      ⎕ASS '15 DESIGN/DA'
15
      'XXXYYY'⍈ 15
      A←⎕DVC 15
      A
474824712 0
      (32⍴1)⊤A[1]
0 0 0 1 1 1 0 0 0 1 0 0 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0
      (-⎕IO)+⎕OM (32⍴2)⊤A[1]
3 4 5 9 12 13 15 17 28
```

You can compare the binary value of the longword with the device characteristics in Table 2–6. The first element in the table is associated with the rightmost bit in the longword, the second element is associated with the next rightmost bit, and so forth. Thus, in the previous example, the three rightmost 0s indicate that the device is not record-orientated, is not a carriage-control device, and is not a terminal; the 1 in the fourth position from the right indicates that the device is directory-structured.

Table 2–6 Device Characteristics Longword

Bit	Type or Condition of Device
0	Record-oriented
1	Carriage-control
2	Terminal
3	Directory-structured
4	Single directory-structured
5	Sequential, block-oriented
6	Being spooled
7	Operator console
8	RA50,RA81,RA82,RH60
9-12	(Bits reserved)

(continued on next page)

Table 2–6 (Cont.) Device Characteristics Longword

Bit	Type or Condition of Device
13	Network
14	File-oriented
15	(Bit reserved)
16	Shareable
17	Generic
18	Available for use
19	Mounted
20	Mailbox
21	Marked for dismount
22	Error logging enabled
23	Allocated
24	Non-file-structured
25	Software write-locked
26	Capable of providing input
27	Capable of providing output
28	Allows random access
29	Real-time
30	Read-checking enabled
31	Write-checking enabled

Possible Errors Generated

```
9 RANK ERROR (NOT VECTOR DOMAIN)

15 DOMAIN ERROR (ENCLUDES ARRAY NOT ALLOWED)

15 DOMAIN ERROR (NOT AN INTEGER)

15 DOMAIN ERROR (INVALID CHANNEL NUMBER)

27 LIMIT ERROR (INTEGER TOO LARGE)
```

⎕*EFR* ⎕*EFS* ⎕*EFC* Event Flag System Functions

Type

Monadic System Functions

Form

event-flag-values ← ⎕*EFR* *chans* (read)
previous-values ← ⎕*EFS* *chans* (set)
previous-values ← ⎕*EFC* *chans* (clear)

Argument Domain

Type	Near-integer
Shape	Vector domain
Depth	0 or 1 (simple)
Value	¯999 to 999 (but not 0)

Result Domain

Type	Numeric
Rank	1 or 2
Shape	Vector or matrix (*n* by 2)
Depth	1 (simple)

Description

There are three event-flag-system functions: ⎕*EFR* to read event flag values, ⎕*EFS* to set event flags (make them equal 1), and ⎕*EFC* to clear event flags (make them equal 0).

The absolute values of *chans* represent the channels associated with the event flags you want to manipulate.

The ⎕*EFR* function returns the values of the event flags associated with the channel numbers in its argument. For channels not associated with an event flag, ⎕*EFR* returns ¯1.

The result is a matrix (or vector, if the argument is a singleton) of shape *n* 1, where *n* is the shape of the argument. In the example, the result indicates that the event flags are associated with channels 1, 2, and 5. The event flag associated with channel 2 is set, and then cleared, and no event flag is associated with channel 4:

```
      ⎕ASS '1 MYFILE/RF/SHARE/EFN:77'
1
      ⎕ASS '2 PUBLIC/DA/SHARE/EFN:68'
2
      ⎕ASS '4 MINE/IS'
4
      ⎕ASS '5 MAILBOX/AS/SHARE/MBX/EFN:65'
5
      ⎕EFS 2
0
      ⎕EFR⍳5
 0
 1
¯1
¯1
 0
      ⎕EFC 2
1
      ⎕EFR⍳5
 0
 0
¯1
¯1
 0
```

The ⎕*EFS* and ⎕*EFC* functions set and clear, respectively, the event flags associated with the channel numbers in their arguments. They return a matrix of shape *n* by 1, where *n* is the shape of the argument, and the values are the previous values of the event flags. For channel numbers not associated with event flags, ⎕*EFS* and ⎕*EFC* return ¯1.

If the argument to ⎕*EFR*, ⎕*EFS*, or ⎕*EFC* is empty, APL returns `0 1⍴0` as the result.

⎕*EFR*, ⎕*EFS*, and ⎕*EFC* are described in the *VAX APL User's Guide* along with other file I/O information.

Possible Errors Generated

9 *RANK ERROR* (*NOT VECTOR DOMAIN*)

15 *DOMAIN ERROR* (*ENCLOSED HETEROGENEOUS ARRAY NOT ALLOWED*)

15 *DOMAIN ERROR* (*NOT AN INTEGER*)

15 *DOMAIN ERROR* (*INVALID CHANNEL NUMBER*)

27 *LIMIT ERROR* (*INTEGER TOO LARGE*)

⎕*ERROR* Error Message

Type

System Variable

Form

← ⎕*ERROR*
⎕*ERROR* ← *error-text*

Value Domain

Type	Character
Shape	Vector domain
Depth	0 or 1 (simple)
Default	''

Result Domain

Type	Character
Rank	1
Shape	Vector
Depth	1 (simple)

Description

⎕*ERROR* contains either the text of the last error message that occurred or the text that you assign to it. (⎕*ERROR* is set implicitly by the system when an error occurs, but can also be set by the user.) ⎕*ERROR* contains one error at a time; when a new error occurs, the new message overwrites the old one. You can, however, localize ⎕*ERROR* within user-defined operations to save error information within an operation's own environment.

The text of ⎕*ERROR* is a character vector of variable-length lines and is delimited by a <CR><LF>. The lines of text in ⎕*ERROR* are the same as the lines of the error message that APL displays on the terminal (except when ⎕*TERSE* is 1), including secondary error messages and execute error messages. The error number is always contained in the first four characters of ⎕*ERROR*, so you can always extract the error number with the expression ⍎4↑⎕*ERROR*. ⎕*ERROR* always contains the entire error message text, even if some of the text was not displayed on the terminal because it was truncated to ⎕*PW* characters. For a description of the text of error messages, see Appendix A.

Note that like all the system variables, ⎕*ERROR* can be set by the user; that is, you can use the specification function (¼) to assign a value to it.

It is possible that when a *WORKSPACE FULL* error occurs, there will not be enough memory available to build ⎕*ERROR*. In that case, ⎕*ERROR* will equal '' (an empty character array). It is also possible that there will not be enough room to display ⎕*ERROR*. In that case, APL signals *WORKSPACE FULL* with the line in error being ⎕*ERROR*.

For example:

```
      ∇ABC;⎕TRAP;⎕ERROR   ⍝LOCALIZE ⎕TRAP AND ⎕ERROR
[1]   ⎕TRAP←'→ LAB'
[2]   5+
[3]            ⍝NEXT LINE PRINTS MESSAGE AND INTERRUPTS EXECUTION
[4]   LAB:⎕BREAK 'CHECK ERROR MESSAGE'
[5]   'RESUME AT LINE 5'
[6]   ∇
               ⍝NOW GENERATE AN IMMEDIATE MODE ERROR
      C+A      ⍝ADD TWO UNDEFINED VARIABLES
  11 VALUE ERROR
      C+A          ⍝ADD TWO UNDEFINED VARIABLES
       ∧
               ⍝CHECK GLOBAL VALUE OF ⎕ERROR
      ⎕ERROR
  11 VALUE ERROR
      C+A          ⍝ADD TWO UNDEFINED VARIABLES
       ∧
               ⍝NOW EXECUTE ABC TO GENERATE LOCAL ERROR
      ABC
CHECK ERROR MESSAGE
               ⍝ABC HAS SUSPENDED
               ⍝NOW CHECK CONTENTS OF ⎕ERROR
      ⎕ERROR
   7 SYNTAX ERROR (RIGHT ARGUMENT TO FUNCTION MISSING)
ABC[2] 5+
       ∧
      ⍝NOW BRANCH TO CURRENT LINE + 1 TO CONTINUE EXECUTION
      →⎕LC+1
RESUME AT LINE 5
               ⍝FUNCTION HAS FINISHED EXECUTION
               ⍝AND LOCAL ⎕ERROR IS GONE
               ⍝CHECK GLOBAL VALUE OF ⎕ERROR
      ⎕ERROR
  11 VALUE ERROR
      C+A                     ⍝ADD TWO UNDEFINED VARIABLES
       ∧
```

Note that if an error occurs during an ⍎ execute, ⎕*ERROR* contains six lines of text. For example:

```
      ⍎ '⎕BREAK 1'
  79 ⍎ SYSTEM FUNCTION ILLEGAL IN EXECUTE
      ⎕BREAK 1
          ∧
  25 EXECUTE ERROR
      ⍎ '⎕BREAK 1'
      ∧
      ⎕ERROR
  79 ⍎ SYSTEM FUNCTION ILLEGAL IN EXECUTE
      ⎕BREAK 1
          ∧
  25 EXECUTE ERROR
      ⍎ '⎕BREAK 1'
      ∧
```

Note that ⎕*ERROR* is set by ⎕*FX*, ⎕*ASS*, and ⎕*XQ*, even though no message is displayed when the error occurs.

The value of ⎕*ERROR* is saved when you save the active workspace and can be localized within user-defined operations. (See the *VAX APL User's Guide*.) The default value is ' '.

Possible Errors Generated

```
9  RANK ERROR (NOT VECTOR DOMAIN)

15  DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

15  DOMAIN ERROR (INCORRECT TYPE)
```

⎕*EX* Erasing a Named Object

Type

Monadic System Function

Form

erased/not-erased ← ⎕*EX* *name-list*

Argument Domain

Type	Character
Shape	Matrix domain
Depth	1 (simple)

Result Domain

Type	Boolean
Rank	1
Shape	ρ *name-list*
Depth	1 (simple)

Description

⎕*EX* erases the local APL objects named by the rows of its argument. You cannot erase a named object that refers to a label, a group (a group name is always global), or to a suspended or pendent operation.

The result of the ⎕*EX* system function is a Boolean vector that indicates which objects were erased: a 1 signifies that the object now has no value; a 0 signifies that the object cannot be erased. Note that ⎕*EX* returns 0 if you specify an ill-formed identifier.

For example:

```
      )FNS
ABCD GROW TEST
      )SI
TEST[2] *
      A←3 4⍴'ABCDTESTGROW'
      A
ABCD
TEST
GROW
      ⎕EX A
1 0 1
      )FNS
TEST
```

If the argument to ⎕*EX* is empty, the result also is empty.

Note that the memory allocated from VMS remains allocated even if you expunge an object from the workspace. If you want to release this memory, follow the sequence of steps discussed in the section on space considerations in Chapter 3 of the *VAX APL User's Guide.*

Possible Errors Generated

```
9  RANK ERROR (NOT MATRIX DOMAIN)

15  DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

15  DOMAIN ERROR (INCORRECT TYPE)
```

⎕*EXP* Expansion

Type

Dyadic System Function

Form

```
A ⎕EXP B     A ⎕EXP[K] B
```

Left Argument Domain

Array	Simple, homogeneous
Type	Near-Boolean
Shape	Vector domain
Depth	0 or 1 (simple)

Right Argument Domain

Type	Any
Shape	Any
Depth	Any

Result Domain

Type	Same as argument
Rank	1⌈⍴⍴*B*
Shape	(*K*−1)↑⍴*B*),(⍴,*A*),*K*↓⍴*B* (for ⎕*IO*=1)
Depth	1⌈≡*B*

Implicit Arguments

None.

Description

⎕*EXP* builds an array by combining the items of an existing array with fill items.

⎕*EXP* works the same as the expansion derived function. The examples shown for the Expansion operator in Chapter 1a also apply to ⎕*EXP* (you can substitute ⎕*EXP* for the backslash (\) operator). The difference between ⎕*EXP* and the backslash operator is that you can use ⎕*EXP* as an operand to an operator. Operators cannot be used as operands to operators. ⎕*EXP* applies along the last axis of *B* unless modified by an explicit axis (*K*) in brackets. The shape of the result is the same as the original array *B* except along the applicable axis (⍴ *B*[*K*]) where the shape becomes the length of *A* (⍴ ,*A*).

The following examples show ⎕*EXP* with the each (¨) operator. Although the variables *A* and *C* are nested in the examples, they conform to the left argument domain requirement that specifies a simple array. This is because the each operator reduces the nesting by one level.

```
      ⎕←A←(1 0 1 1) (1 1 0 1)
+-------+ +-------+
|1 0 1 1| |1 1 0 1|
+-------+ +-------+
      ⎕←B←(2 3⍴⍳6) (4 3⍴'ABCDEFGHIJKL')
+-----+ +---+
|1 2 3| |ABC|
|4 5 6| |DEF|
+-----+ |GHI|
        |JKL|
        +---+
                          ⍝ATTEMPT TO USE OPERATOR AS ARG TO ¨
      A\¨B                ⍝APL EVALUATES AS (A\)¨B
 15 DOMAIN ERROR (ENCLOSED/HETEROGENEOUS ARRAY NOT ALLOWED)
      A\¨B                ⍝APL EVALUATES AS (A\)¨B
      ^
      A ⎕EXP¨ B
+-------+ +----+
|1 0 2 3| |AB C|
|4 0 5 6| |DE F|
+-------+ |GH I|
          |JK L|
          +----+
      A ⎕EXP[2]¨B        ⍝EXPAND ITEMS OF B USING THE ITEMS OF A
+-------+ +----+
|1 0 2 3| |AB C|
|4 0 5 6| |DE F|
+-------+ |GH I|
          |JK L|
          +----+
      ⎕←C←(1 0 1) (1 1 0 1 1)
+-----+ +---------+
|1 0 1| |1 1 0 1 1|
+-----+ +---------+
      C ⎕EXP[1]¨B        ⍝USE ⎕EXP WITH AXIS ARGUMENT
+-----+ +---+
|1 2 3| |ABC|
|0 0 0| |DEF|
|4 5 6| |   |
+-----+ |GHI|
        |JKL|
        +---+
```

Possible Errors Generated

9 *RANK ERROR* (*NOT VECTOR DOMAIN*)

10 *LENGTH ERROR*

15 *DOMAIN ERROR*

15 *DOMAIN ERROR* (*ENCLOSED ARRAY NOT ALLOWED*)

15 *DOMAIN ERROR* (*INCORRECT TYPE*)

27 *LIMIT ERROR* (*INTEGER TOO LARGE*)

28 *AXIS RANK ERROR* (*NOT VECTOR DOMAIN*)

29 *AXIS LENGTH ERROR* (*NOT SINGLETON*)

30 *AXIS DOMAIN ERROR* (*AXIS LESS THAN INDEX ORIGIN*)

30 *AXIS DOMAIN ERROR* (*ENCLOSED ARRAY NOT ALLOWED*)

30 *AXIS DOMAIN ERROR* (*INCORRECT TYPE*)

30 *AXIS DOMAIN ERROR* (*NOT AN INTEGER*)

30 *AXIS DOMAIN ERROR* (*RIGHT ARGUMENT HAS WRONG RANK*)

30 *AXIS DOMAIN ERROR* (*SEMICOLON LIST NOT ALLOWED*)

⎕*FI* Converting Characters to Numerics

Type

Monadic System Function

Form

numeric-values ← ⎕*FI* *numeric-character-string*

Argument Domain

Type	Character
Shape	Vector domain
Depth	0 or 1 (simple)

Result Domain

Type	Numeric
Rank	1
Shape	Vector
Depth	1 (simple)

Implicit Arguments

⎕*NG* (determines minus sign placement)

Description

⎕*FI* converts a numeric character argument to a vector of numeric values, placing a 0 in each position that does not correspond to a valid number. The shape of the argument must be in the vector domain. If the argument is empty, ⎕*FI* returns an empty numeric vector.

⎕*FI* separates the argument into fields that are delimited by one or more spaces, tabs, or a carriage return (optionally followed by a line feed); converts each field that contains a valid number; and inserts a 0 to replace each field containing an invalid number. For example:

```
      A←⎕FI '12 55ϵ ¯4 C 2951 8 +5'
      A
12 0 ¯4 0 2951 8 0
      ρA
7
```

Note that a plus sign preceding a number is not part of the number but is rather an operation to be performed on the number. However, in APL, the negative sign in the expression ¯5 is a valid part of the number.

⎕*FI* is often used in conjunction with ⎕*VI* and the compression derived function (see the Section 1.3.2 section) to select the valid numbers from a character string: ⎕*VI* produces the left argument of the compression function, and ⎕*FI* produces the right argument. For example:

```
      ∇ Z←AVERAGE
[1]    ⎕←'ENTER A LIST OF NUMBERS'  ⋄ Z←,⍞
[2]    Z←(⎕VI Z)/⎕FI Z
[3]    Z←(+/Z)÷ρZ
[4]    ∇
      AVERAGE
ENTER A LIST OF NUMBERS
1 3.5 A 0 +2 ¯.5 6. .
2
```

In the previous example, ⎕*VI* of *Z* equals `1 1 0 1 0 1 1 0` and ⎕*FI* of *Z* equals `1 3.5 0 0 0 ¯ .5 6 0`

Recognition of negative numbers in the ⎕*FI* argument depends upon the value of the system variable ⎕*NG*. If ⎕*NG* equals 1 (the default), negative numbers in the ⎕*FI* argument must begin with the high minus sign (¯) to be recognized. If ⎕*NG* equals 0, numbers preceded by a minus sign (-) are recognized as negative numbers. If ⎕*NG* equals 2, negative numbers are preceded by an APL "+" symbol. (APL "+" prints as an ASCII "-" so that ⎕*NG*←2 can be used to handle negative numbers in strings that are read or written in ASCII.) For example:

```
      ⎕NG←1                   ⍝¯ MEANS NEGATIVE
      X←'66 G ¯7 +9 ¯4'
      ⎕FI X
66 0 ¯7 0 ¯4
      ⎕NG←0                   ⍝- MEANS NEGATIVE
      ⎕FI X
66 0 0 0 0
      ⎕NG←2                   ⍝+ MEANS NEGATIVE
      ⎕FI X
66 0 0 +9 0
```

Note that when ⎕*NG* is 0, it may be useful for you to use APL to interpret data created by other languages, specifically those that do not use the high minus sign.

Possible Errors Generated

```
9  RANK ERROR (NOT MATRIX DOMAIN)

15  DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

15  DOMAIN ERROR (INCORRECT TYPE)
```

⎕*FLS* Returning File Information

Type

Monadic System Function (query)

Form

file-info ← ⎕*FLS* *chans*

Argument Domain

Type	Near-integer
Shape	Vector domain
Depth	0 or 1 (simple)
Value	¯999 to 999 (but not 0)

Result Domain

Type	Integer
Rank	1 or 2
Shape	Vector or matrix (*n* by 5)
Depth	1 (simple)

Description

⎕*FLS* returns information about files. The absolute values of *chans* represent the channels associated with the files you want to specify. The result contains one row of five values for each channel specified in the argument. The meanings of the values differ according to each file's organization.

The values returned by ⎕*FLS* have the following meanings (from left to right):

First value	Share bit: 1 means that you specified /*SHARE* in the argument for the associated ⎕*ASS* function; 0 means that you did not.

Second value — For sequential files, the second value is the number of records read or written since the file was opened. For direct-access and relative files, it is the value of the last record or component number used for a successful read or write operation. For keyed files, it is the value of the last key of reference used for a successful read, write, or rewind.

Third value — The maximum record length of the file (0 means there is no user limit on record size).

Fourth value — The */BLOCKSIZE* setting for the file.

Fifth value — The type of the most recent I/O operation

Value Returned	I/O Operation
0	None
1	Sequential read
2	Random read
3	Sequential write
4	Random write
5	Sequential delete
6	Random delete

⎕*FLS* is described in the *VAX APL User's Guide* along with other file I/O information.

Possible Errors Generated

9 *RANK ERROR (NOT VECTOR DOMAIN)*

15 *DOMAIN ERROR (ENCLOSED HETEROGENEOUS ARRAY NOT ALLOWED)*

15 *DOMAIN ERROR (NOT AN INTEGER)*

15 *DOMAIN ERROR (INVALID CHANNEL NUMBER)*

27 *LIMIT ERROR (INTEGER TOO LARGE)*

⎕*FMT* The Report Formatter

Type

Dyadic System Function

Form

report ← *format-phrases* ⎕*FMT* {*array* | (*array* ; *array*;...)}

Left Argument Domain

Type	Character
Shape	Vector domain
Depth	1 (simple)

Right Argument Domain

Type	Any
Shape	Any
Depth	≤ 2 (vector of arrays or a simple array)

Result Domain

Type	Character
Rank	2
Shape	Matrix
Depth	1 (simple)

Implicit Arguments

⎕*NG* (determines minus sign placement)

Description

⎕*FMT* combines the data from all the arrays in the right argument and arranges it as a single character matrix whose columns are then formatted according to corresponding format phrases specified in the left argument. The arrays in the right argument can be both character and numeric data.

⎕*FMT* can edit the data as it is moved to an output field. For example, ⎕*FMT* fills or erases zeros in numeric fields; round numeric data; and inserts commas, dollar signs, and other text where appropriate.

The right argument is a list of arrays of any type or rank. The list must be surrounded by parentheses (unless there is only one array in the list), and the arrays must be separated by semicolons. Alternatively, the right argument may be a single nested vector of simple arrays, which are treated in the same manner as a semicolon list.

The left argument is a character vector comprised of one or more format phrases of the form described in Chapter 4 of the *VAX APL User's Guide*. The phrases must be separated by commas.

The following table summarizes the syntax of the format phrase in the left argument. Note that *rep* (repetitions) refers to the number of consecutive target columns to be affected by the format phrase; *quals* refers to one of the qualifiers or decorators described in the following table; *width* refers to the width in the result array of a value from a column of data in the right argument; *dig* (digits) refers to the number of decimal places included in the result array; and *col* (column) refers to either the leftmost column that a value is to occupy in the result array (for type T), or the number of columns to be shifted before the next value is output to the result array.

Phrase	Type of Data
⟦*rep*⟧ ⟦*quals*⟧ A *width*	Character
⟦*rep*⟧ ⟦*quals*⟧ E *width.dig*	Floating-point with exponent
⟦*rep*⟧ ⟦*quals*⟧ F *width.dig*	Fixed-point
⟦*rep*⟧ ⟦*quals*⟧ G ⍝*pattern*⍝	Picture
⟦*rep*⟧ ⟦*quals*⟧ I *width*	Integer
⟦*rep*⟧ ⟦*quals*⟧ Y *width*	Byte
⟦*rep*⟧ T ⟦*col*⟧	Absolute tab
⟦*rep*⟧ X ⟦*col*⟧	Relative tab
⟦*rep*⟧⍝ *text*⍝	Literal

The following table summarizes the qualifiers and decorators used in the format phrase:

Qualifiers	**Meaning**
B	For types I, E, F, G, and Y, if the value of the item in the target column is zero, make the field in the target column blank in the result array.
C	For types I and F, insert commas between each group of three digits in the integer part of the formatted value.
L	For types I, F, E, A, and Y, left-justify the fields in the target column.
K*n*	For types I, F, G, and E, before formatting the fields in the target column, multiply the fields by the scale factor `10*`*n*.
SA *symbol pairs*A	For types I, E, F, G, and Y, replace, in the formatted output, all occurrences of the first character in each symbol pair with the corresponding second character of the symbol pair.
W*n*	For type E, use *n* exponent digits in the formatted output.
Z	For types I, F, and Y, fill leading blanks in the formatted output with zeros.
Decorator	**Meaning**
MA *text*A	For types I, F, and G, replace the sign of negative-formatted values with *text* placed to the left of the value.
NA *text*A	For types I, F, and G, place *text* to the right of negative-formatted values.
OA *text*A	For types I, F, G, and Y, replace formatted zero values with *text*.
PA *text*A	For types I, F, and G, place *text* to the left of positive-formatted values.

Decorator	Meaning
Q⍝ *text*⍝	For types I, F, and G, place *text* to the right of positive-formatted values.
R⍝ *text*⍝	For types I, F, E, A, G, and Y, fill unused columns in the formatted output with *text*.

Note that the delimiting pair ⍝ ⍝ may also be any of the following pairs:

⍞ ⍞   ¨ ¨   ⎕ ⎕   < >   ⊂ ⊃

⎕*FMT* is also described in Chapter 4 of the *VAX APL User's Guide*.

Possible Errors Generated

9 *RANK ERROR (NOT VECTOR DOMAIN)*

10 *LENGTH ERROR*

14 *DEPTH ERROR*

15 *DOMAIN ERROR (DUPLICATE FMT QUALIFIER)*

15 *DOMAIN ERROR (DUPLICATE FMT STANDARD SUBSTITUTION CHARACTER)*

15 *DOMAIN ERROR (EMPTY FMT STRING PARAMETER NOT ALLOWED)*

15 *DOMAIN ERROR (ENCLOSED ARRAY IS NOT ALLOWED)*

27 *LIMIT ERROR (FLOATING OVERFLOW)*

15 *DOMAIN ERROR (FMT DECORATION OR LITERAL STRING TOO LONG)*

15 *DOMAIN ERROR (FMT RIGHT ARGUMENT DOES NOT MATCH FORMAT PHRASE)*

15 *DOMAIN ERROR (ILL FORMED FMT PARAMETER)*

15 *DOMAIN ERROR (ILLEGAL CHARACTER IN FMT LEFT ARGUMENT)*

15 *DOMAIN ERROR (ILLEGAL FMT FORMAT PHRASE)*

15 *DOMAIN ERROR (ILLEGAL FMT G FORMAT PHRASE PATTERN CHARCTER)*

15 *DOMAIN ERROR (ILLEGAL FMT LITERAL STRING DELIMITER)*

```
15  DOMAIN ERROR (ILLEGAL FMT S QUALIFIER SYMBOL)

15  DOMAIN ERROR (ILLEGAL USE OF FMT QUALIFIER)

15  DOMAIN ERROR (INCORRECT TYPE)

15  DOMAIN ERROR (MISSING FMT FORMAT PHRASE SEPARATOR)

15  DOMAIN ERROR (MISSING FMT FORMAT PHRASE/QUALIFIER)

15  DOMAIN ERROR (MISSING LITERAL STRING IN FMT LEFT ARGUMENT)

15  DOMAIN ERROR (NO DIGIT SELECTOR IN FMT G FORMAT PHRASE PATTERN)

15  DOMAIN ERROR (NO FMT EDITING FORMAT PHRASE)

15  DOMAIN ERROR (PARAMETER OUT OF RANGE)

15  DOMAIN ERROR (RIGHT ARG TOO DEEPLY NESTED)

15  DOMAIN ERROR (UNBALANCED TEXT DELIMITER IN FMT LEFT ARGUMENT)

15  DOMAIN ERROR (UNBALANCED PARENS IN FMT LEFT ARGUMENT)

15  DOMAIN ERROR (UNPAIRED SYMBOL IN FMT S QUALIFIER)
```

⎕*FX* Establishing an Operation

Type

Monadic System Function

Form

operation-name← ⎕*FX* *operation-definition*

Argument Domain

Type	Character
Shape	Matrix domain
Depth	1 (simple)

Result Domain

Type	Character (Numeric if error is detected)
Rank	0 or 1
Shape	Vector (Scalar if error is detected)
Depth	0 or 1 (simple)

Implicit Arguments

⎕*IO* (controls origin of line number in error)

Description

⎕*FX* reverses the operation of the ⎕*CR* system function; that is, it creates in internal form the operation defined by its argument.

The argument is assumed to be a character matrix that contains the canonical representation of an operation. The shape of the argument must be in the matrix domain. Blank lines in the argument are removed in the operation established by ⎕*FX*.

⎕*FX* fails if the operation's name is the same as that of an existing label, variable, or group, or if it is the same as that of an existing operation that is pendent or suspended. If an operation already exists in your workspace with the same name, ⎕*FX* replaces it and removes any trace, stop, or monitor bits that were set on it.

The ⎕*FX* function executes properly if the matrix it references is identical to a canonical representation. If ⎕*FX* fails, APL returns a scalar index (which is ⎕*IO*-sensitive) representing the row in the matrix where the error occurred, and no change is made to any operation or array in your workspace. You can check the value of ⎕*ERROR* for a description of what was wrong with the line in error.

If ⎕*FX* is successful, its result is a character vector containing the name of the operation defined. If the argument is empty, the result is empty.

The following example begins where the ⎕*CR* example from the ⎕*CR* section ended. Here, the plus sign in *SHOWCRFX* is changed to a multiplication sign; then, the ⎕*FX* system function is applied to *SHOWCRFX* to replace the function *MEAN* with its new version:

```
      SHOWCRFX[3;6]←'×'
      ⎕FX SHOWCRFX
MEAN
      ∇ MEAN[⎕]∇
      ∇MEANX←NSUBJ MEAN X
[1]   ⍝SUM VECTOR X
[2]   SUMX←×/X
[3]   MEANX←SUMX÷NSUBJ
      ∇
      X
8 6 3 9 5 4 2 1 7 4
      10 MEAN X
145152
      X←2 4⍴'F  X1234'
      X[1;2 3]←⎕CTRL[14 11]   ⍝EMBED CRLF IN OPERATION HEADER
      ⎕FX X
1
      ⎕ERROR
  5 DEFN ERROR (EXTRANEOUS CHARACTERS AFTER COMMAND)
F
X

^
```

Possible Errors Generated

```
9  RANK ERROR (NOT MATRIX DOMAIN)

15  DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

15  DOMAIN ERROR (INCORRECT TYPE)
```

⎕*GAG* Preventing Interruptions

Type

System Variable (session)

Form

⎕*GAG* ← *near-integer-singleton*
integer-scalar ← ⎕*GAG*

Value Domain

Type	Integer
Shape	Singleton
Depth	0 or 1 (simple)
Value	0, 1, 2, or 3
Default	Determined when APL is invoked

Result Domain

Type	Integer
Rank	0
Shape	⍳0 (scalar)
Depth	0 (simple scalar)

Description

⎕*GAG* allows you to specify how APL handles messages that arrive at your terminal from other users. You can set ⎕*GAG* to the following values:

Value	Meaning
0	Display messages
1	Refuse messages
2	Trap, translate, and display messages
3	Signal *BROADCAST RECEIVED*

Setting ⎕*GAG* to 0 is equivalent to executing the DCL command set terminal /broadcast, and setting ⎕*GAG* to 1 has the same effect as the DCL command set terminal/nobroadcast. When ⎕*GAG* is 1, messages from nonprivileged users are suppressed (note that senders are not told that their messages were not received). For more details, see the *VMS DCL Dictionary*. When you return to DCL from APL, the original system value for ⎕*GAG* is restored, unless the exit from APL was a panic exit; in that case, the setting established in the APL session remains in effect.

Setting ⎕*GAG* to 2 is equivalent to executing the DCL command SET TERMINAL/BROADCAST, with the addition of instructing APL to display the message in the character set that is currently set for the terminal. If you use an APL terminal, the default setting is 2 when APL is invoked.

Setting ⎕*GAG* to 3 allows you to trap messages with ⎕*TRAP* and to view them at a later time. As messages arrive at the terminal, APL signals *BROADCAST RECEIVED* followed by a secondary message of the broadcast text.

The default setting of ⎕*GAG* is the current monitor setting. Note that ⎕*GAG* is a session variable; that is, its value is not saved with the workspace, and ⎕*GAG* is not reset by the execution of a)*CLEAR* command (see Chapter 3). ⎕*GAG* can, however, be localized in user-defined operations.

Possible Errors Generated

9 *RANK ERROR* (*NOT VECTOR DOMAIN*)

10 *LENGTH ERROR* (*NOT SINGLETON*)

15 *DOMAIN ERROR* (*ENCLOSED ARRAY NOT ALLOWED*)

15 *DOMAIN ERROR* (*INCORRECT TYPE*)

15 *DOMAIN ERROR* (*NOT AN INTEGER*)

27 *LIMIT ERROR* (*INTEGER TOO LARGE*)

⎕*IO* Index Origin

Type

System Variable

Form

⎕*IO* ← *near-integer-singleton*
integer-scalar-current-value ← ⎕*IO*

Value Domain

Type	Integer
Shape	Singleton
Depth	0 or 1 (simple)
Value	0 or 1
Default	1

Result Domain

Type	Integer
Rank	0
Shape	ι 0 (scalar)
Depth	0 (simple scalar)

Description

⎕*IO* specifies the setting of the index origin. This setting determines whether the values of an array are indexed beginning with position 0 or 1. The default position is 1.

⎕*IO* also affects the operation of axis ([*K*]), exceopt when axis is used with user-defined operations. In addition, ⎕*IO* affects the operation of the following primitive and system functions:

ι*A*   *A*ι*B*   ?*A*   *A*?*B*   ⍋*A*   ⍒*A*   *A*⍉*B*   ⎕*OM B*   ⎕*FX B*

The value of ⎕*IO* is saved when you save the active workspace and can be localized in user-defined operations.

Examples:

```
      ⎕IO←1
      ι 3
1 2 3
      A←2 4ρι6
      +/[2]A
10 14
      +/[1]A
8 4 6
      +/[0]A
  30 AXIS DOMAIN ERROR (AXIS LESS THAN INDEX ORIGIN)
      +/[0]A
       ^
      ⎕IO←0
      ι3
0 1 2
      +/[2]A
  30 AXIS DOMAIN ERROR (ARGUMENT RANK AND AXIS INCOMPATIBLE)
      +/[2]A
       ^
      +/[1]A
10 14
      +/[0]A
6 8 4 6
      ⎕IO←7
  15 DOMAIN ERROR (SYSTEM VARIABLE VALUE MAY ONLY BE 0 OR 1)
      ⎕IO←7
        ^
```

Possible Errors Generated

9 *RANK ERROR (NOT VECTOR DOMAIN)*

10 *LENGTH ERROR (NOT SINGLETON)*

15 *DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)*

15 *DOMAIN ERROR (INCORRECT TYPE)*

15 *DOMAIN ERROR (NOT AN INTEGER)*

15 *DOMAIN ERROR (SYSTEM VARIABLE VALUE MAY ONLY BE* 0 *OR* 1)

27 *LIMIT ERROR (INTEGER TOO LARGE)*

⎕*L* Monitoring Variable Changes

Type

System Variable

Form

⎕*L* ← *any-value*
variable-name← ⎕*L*

Value Domain

Type	Any
Shape	Any
Depth	Any

Result Domain

Type	Character (any when set by user)
Rank	1 (any when set by user)
Shape	Vector (any when set by user)
Depth	1 (simple) (any when set by user)
Default	⍳0

Description

⎕*L* and ⎕*R* are system variables that are implicitly used by ⎕*WATCH*. (⎕*L* is set implicitly by the system when a variable changes, but can also be set by the user.) ⎕*WATCH* is a system function that is used to monitor any changes in one or more variables. When a change occurs in a monitored variable, APL assigns information to ⎕*L* and ⎕*R*: ⎕*L* contains a character vector showing the name of the variable that has changed; ⎕*R* contains the previous value of the changed variable. APL assigns this information regardless of whether monitoring is set for signal or display mode.

The default value for both ⎕*L* and ⎕*R* is ⍳0. Immediately after a ⎕*WATCH* event occurs, ⎕*L* and ⎕*R* contain the new information that results from the event. However, this information may change as an operation continues execution (this is especially true if an error occurs during an assignment or reference of a variable that is associated with a watchpoint).

Both ⎕*L* and ⎕*R* can be localized, explicitly assigned values of any type, and saved in the workspace.

Note that you cannot include ⎕*L* or ⎕*R* in the right argument to dyadic ⎕*WATCH*.

Possible Errors Generated

None.

⎕*LC* Line Counter

Type

Niladic System Function

Form

current-line-number ← ⎕*LC*

Result Domain

Type	Integer
Rank	1
Shape	Vector
Depth	1 (simple)
Default	Empty

Description

⎕*LC* (line counter) allows you to obtain a partial report on operations that are currently being executed. The function returns a vector of all the line numbers contained in the state indicator; the numbers are arranged as they would appear in the)*SI* system command display (see Chapter 3.) If the state indicator is empty, ⎕*LC* returns an empty numeric vector.

The ⎕*LC* system function is particularly useful in restarting suspended operations. For more information, see the *VAX APL User's Guide.* For example:

```
      ∇NEW
[1]   →1  ∇
      NEW  ⍝CALL FUNCTION, THEN SEND ATTENTION SIGNAL
  18 ATTENTION SIGNALED
NEW[1] →1
       ^
      ⎕LC
1
```

Possible Errors Generated

None.

⎕*LX* Latent Expression

Type

System Variable

Form

⎕*LX* ← *character-vector*
current-value ← ⎕*LX*

Value Domain

Type	Character
Shape	Vector domain
Depth	0 or 1 (simple)
Default	''

Result Domain

Type	Character
Rank	1
Shape	Vector
Depth	1 (simple)

Description

⎕*LX* specifies an APL expression that is executed automatically when the workspace is loaded.

The value you assign to ⎕*LX* must be a character vector. The default value is ''. APL processes the expression as if you had specified ⍎ ⎕*LX*. Any error messages you receive are produced by the execute function.

The ⎕*LX* system variable is useful in restarting a suspended operation. For example:

```
⎕LX←'→⎕LC'
```

⎕*LX* is also useful for invoking a particular user-defined operation (see the *VAX APL User's Guide*) when you load the workspace. For example:

```
⎕LX←' STARTUP'
```

The ⎕*LX* system variable is often used to display a message when the workspace in which it is defined is loaded. For example:

```
      ⎕LX←'''NOTE NEW LINE PRINTER IN OPERATION'''
      )SAVE MYWS
FRIDAY 16-NOV-1990 10:09:27:02 7 BLKS
      )CLEAR
CLEAR WS
      )LOAD MYWS
SAVED FRIDAY 10-NOV-1990 10:09:27.02 7 BLKS
NOTE NEW LINE PRINTER IN OPERATION
```

When you want to load a workspace without invoking ⎕*LX*, you can use the)*XLOAD* command (see Chapter 3) if you are the owner of the workspace.

APL executes ⎕*LX* only in immediate mode and only when the state indicator stack is either empty or has a suspended operation on top. If the top of the stack contains a ⎕ input function, the latent expression is executed only after the pendent ⎕ input is removed from the stack. The latent expression is not executed if the top of the stack contains an execute function, or if the loaded workspace is in function-definition mode. For example:

```
      )LOAD MYWS
SAVED THURSDAY  8-NOV-1990 19:42:58.52 15 BLKS
      ∇ F
[1]   A←1
[2]   X←⎕XQ')SAVE MYWS'
[3]   'X IS ';X
[4]   'END OF F' ∇
      F
X IS THURSDAY  8-NOV-1990 17:01:59.54 16 BLKS
END OF F
      )CLEAR
CLEAR WS
      )LOAD MYWS
X IS SAVED THURSDAY  8-NOV-1990 17:01:59.54 16 BLKS
END OF F
```

In this example, the note about the new line printer is not displayed when the workspace is loaded because the workspace was saved during the execution of an ⎕*XQ* system function; thus, the ⎕*XQ* function is at the top of the stack when the workspace is reloaded, and APL completes the ⎕*XQ* function rather than executing the latent expression.

If you were to save the workspace after execution of the function *F* completed, the latent expression would be executed the next time the workspace was loaded:

```
      )SAVE MYWS
FRIDAY 16-NOV-1990 10:43:59.54 8 BLKS
      )CLEAR
CLEAR WS
      )LOAD MYWS
SAVED FRIDAY 16-NOV-1990 10:43:59.54 8 BLKS
NOTE NEW LINE PRINTER IN OPERATION
```

Note that when the function *F* was executed, the value of *X* displayed by operation line [3] was equivalent to the message displayed by the)*SAVE* system command:

```
X IS MONDAY 27-SEP-1982 18:07:42.02 8 BLKS
```

However, when the execution of function *F* was resumed because the saved workspace was loaded by a)*LOAD* command, the value of *X* displayed by operation line [3] was equivalent to the message displayed by the)*LOAD* command:

```
X IS SAVED MONDAY 27-SEP-1982 18:07:42.02 8 BLKS
```

Thus, as shown by this example, it is possible to determine whether the workspace has just been saved or has just been loaded.

The value of ⎕*LX* is saved when you save the active workspace and can be localized in user-defined operations.

Possible Errors Generated

```
9  RANK ERROR (NOT VECTOR DOMAIN)

15  DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

15  DOMAIN ERROR (INCORRECT TYPE)
```

⎕*MAP* Defining External Routines to APL

Type

System Function

Form

external-routine-definition ← ⎕MAP *function-name*
function-name ← *function-header* ⎕MAP *image-definition*

Monadic Argument Domain

Type	Character
Shape	Vector domain
Depth	0 or 1 (simple)

Dyadic Left Argument Domain

Type	Character
Shape	Vector domain
Depth	0 or 1 (simple)

Dyadic Right Argument Domain

Type	Character
Shape	Vector domain
Depth	0 or 1 (simple)

Result Domain

Type	Character
Rank	1
Shape	Vector
Depth	1 (simple)

Parameters

external-routine-definition
The operation header returned by the monadic form of ⎕*MAP*. This is the same header that dyadic ⎕*MAP* uses when you successfully define the external routine to APL.

function-name
Specifies the name of the external routine. For dyadic ⎕*MAP*, if both *function-header* and *image-definition* are empty, and are in the vector domain, then the result is an empty vector. For monadic ⎕*MAP*, if *function-name* is empty, the result is an empty character vector. If the value of *function-name* does not name an external routine, APL signals *DOMAIN ERROR (NOT AN EXTERNAL FUNCTION)*.

function-header
Describes the external routine. *function-header* has the following form:

⟦*result/att*← ⟧ *entry-point* ⟦*arg1/att* ⟦*arg2/att*⟧ . . . ⟧

result/att specifies that the external function returns a result. Note that the result must be a scalar. (If you want to return data that has a rank greater than 0, you can modify a formal parameter with the external routine.) The */att* qualifier specifies the type of the result. It has the form */TYPE*:*vms-data-type*, and must be one of the external data types in the Table 2–7 (excluding */TYPE*:*Z*).

Do not specify the */MECHANISM* attribute for the result of an external routine; APL determines the mechanism by the value specified for */TYPE*. Table 2–7 describes these default mechanisms.

If the result type occupies 8 bytes or less, APL assumes the mechanism is *IMMEDIATE*. If the result type occupies more than 8 bytes, APL assumes the mechanism is *DESCRIPTOR* for strings and *REFERENCE* for all others. *entry-point* is the name that you want APL to associate with the shared image entry point specified in *image-definition*. After you define *entry-point*, you can call the external routine as if it were a user-defined operation. Note that *entry-point* has a name class value of 3.

Dyadic ⎕*MAP* signals *DOMAIN ERROR (NAME IN USE)* if *entry-point* is the same name as an existing label, variable, or group, or if it is the same name as an existing operation that is pendent or suspended. If an operation already exists in your workspace with the same name, and it is not pendent or suspended, ⎕*MAP* replaces it.

argn specifies the names of the function's formal parameters. These names are similar to the dummy arguments of a user-defined operation; they are placeholders only, and you specify the actual values for these parameters when you invoke the function.

The maximum number of formal parameters you can specify is 255 (including *result*). The names must be valid APL identifiers; they do not have to be unique. An external function can only be monadic or niladic; all of the formal parameters belong to the function's right argument. */att* determines the attributes for each of the formal parameters and for the external routine's result. For parameters, the attributes specify the kind of access that the external routine has to the parameter (either read, write, or both), the data type of the parameter, and the passing mechanism used to send the parameter between APL and the external routine. Valid qualifiers for */att* include */ACCESS*, */TYPE* and */MECHANISM*.

image-definition

The name of a shared image (in the form of a VMS file name or logical name) and its entry point. If you use a logical name you cannot change the name once the shared image is mapped by APL. *image-definition* has the following form:

{*vms-filename* | *vms-logical-name*} ⟦{*/ENTRY* | */VALUE*} ⟦:*symbol*⟧⟧

vms-filename or vms-logical-name specifies the name of the VMS shared image. If you do not use a logical name, you can only specify a file name, not a complete file specification. The default directory for *vms-filename* is SYS$SHARE:, and the default file type is .EXE. If you use *vms-logical-name*, you should not redefine the logical name to point to a different file once the shared image is mapped.

Note that you can use the equal sign delimiter (=) in place of the colon (:). Spaces are allowed before and after the */ENTRY* or */VALUE* qualifier, the delimiter, and the value for *symbol*.

Qualifiers

/ENTRY⟦:***symbol***⟧

Used with dyadic ⎕*MAP*, specifies the name of the entry point in the shared image. An entry point is the starting address of executable code. If you do not specify */ENTRY*, or if you specify */ENTRY* with no value for *symbol*, APL assumes that the name of the entry point is the same as the value for *function-name*.

/VALUE[[**:*symbol***]]

Used with dyadic ⎕*MAP*, specifies the name of a global constant in the shared image. A global constant is a 32-bit signed longword value. When you specify */VALUE*, then *function-header* must specify a niladic function that returns a value with a return type of *L* (for example, *'Z/TYP:L← F'*). If */VALUE* is specified when there is no value for *symbol*, APL assumes that the name of the global constant is the same as the value for *function-name*.

/ACCESS

Specifies whether the parameter is read only or modifiable. The value *IN* indicates that the external routine reads the parameter and does not modify its value. The value *INOUT* indicates that the routine reads the parameter and may modify it. The value *OUT* indicates that the routine writes a value to the parameter.

When you specify *INOUT* or *OUT*, the actual parameter that you specify when you invoke the function must be a character string that names the variable that the routine will read (in the case of *INOUT*) and write. If the variable does not have a value when you call the external routine, APL assumes the variable is a scalar and will accept a scalar only when the value is returned.

If you do not specify a value for */ACCESS*, APL uses the *IN* value as the default. You cannot specify the */ACCESS* attribute for the result of an external routine; by definition the access is always *OUT*.

Note that you can abbreviate the values for */ACCESS* to their shortest unique prefixes.

/TYPE

Specifies the attribute for both the formal parameters and the result (if any). It is one of the external types shown in Table 6-1 in the *VAX APL User's Guide*. On a formal parameter, */TYPE* specifies the VAX data type that the external routine is expecting. On the result, */TYPE* specifies the VAX data type that will be returned. Data internal to APL has one of the following types:

- Character data in the APL character set (8-bits per value)
- Boolean data, a subset of numeric data (1-bit per value)
- Integer data, a subset of numeric data (32-bits, signed, per value)
- Floating-point, a subset of numeric data (64-bits, D_floating, per value)

Because VMS supports many more data types than APL, conversions will take place as data leaves and returns to APL from the external routine. Tables 6-2 and 6-3 in the *VAX APL User's Guide* summarize these possible conversions.

The default data type is `/TYPE:Z` (unspecified), which indicates that data is passed out of the workspace without conversion. Data that is passed out of APL as `/TYPE:Z` cannot return to APL; for this reason, a formal parameter with the attribute `/TYPE:Z` must also have the attribute `/ACCESS:IN`.

Note that you cannot abbreviate any of the values to the `/TYPE` qualifier.

Table 2–7 Characteristics of External Data Types

External Type	Type Name	DEFAULT *result* /*MECHANISM*	Length in Bytes
Z	Unspecified	N/S	
BU	Byte Logical	IMM	1
WU	Word Logical	IMM	2
LU	Longword Logical	IMM	4
QU	Quadword Logical	N/S	
OU	Octaword Logical	N/S	
B	Byte Integer	IMM	1
W	Word Integer	IMM	2
L	Longword Integer	IMM	4
Q	Quadword Integer	N/S	
O	Octaword Integer	N/S	
F	F_floating	IMM	4
D	D_floating	IMM	8
G	G_floating	IMM	8
H	H_floating	REF	16
FC	F complex	IMM	8
DC	D complex	REF	16
GC	G complex	REF	16
HC	H complex	REF	32

Key to Default result /*MECHANISM*

N/S—not supported
IMM—by value
REF—by reference
DES—by description

(continued on next page)

Table 2–7 (Cont.) Characteristics of External Data Types

External Type	Type Name	DEFAULT *result* */MECHANISM*	Length in Bytes
CIT	COBOL Temp	N/S	
T	8-bit Text	DES	1
VT	Varying Text	REF	1
NU	Numeric String	DES	1
NL	Left Sign String	DES	1
NLO	Left Overpunch String	DES	1
NR	Right Sign String	DES	1
NRO	Right Overpunch String	DES	1
NZ	Zoned Sign String	DES	1
P	Packed Decimal	N/S	
V	Bit	IMM	1
VU	Bit Unaligned	N/S	
ZI	Instructions	N/S	
ZEM	Entry Mask	N/S	
DSC	Descriptor	N/S	
BPV	Bound Procedure	N/S	
BLV	Bound Label	N/S	
ADT	Date/Time	N/S	
other	DEC or user reserved	N/S	

Key to Default result */MECHANISM*

N/S—not supported
IMM—by value
REF—by reference
DES—by description

/MECHANISM
Specifies one of the three techniques for passing formal parameters from APL to the external routine. These techniques are *IMMEDIATE*, *REFERENCE*, and *DESCRIPTOR*. *IMMEDIATE* specifies that the value of the parameter is the value you want to pass. *REFERENCE* specifies that the value of the parameter is the

address of the value you want to pass. *DESCRIPTOR* specifies that the value is the address of a descriptor that contains the address and length of the data as well as other attributes (if the descriptor requires them). Note that the descriptor length field contains the length of the object.

If you do not specify the */MECHANISM* attribute when you invoke dyadic ⎕*MAP*, APL uses a default when you call the external routine. If the parameter is */TYPE:Z*, APL assumes */MECHANISM:REFERENCE*. Otherwise, the default is based on the rank of the actual argument being passed: */MECHANISM:REFERENCE* is chosen for scalars and vectors, and */MECHANISM:DESCRIPTOR* is used for arrays of rank 2 or higher.

When you specify */MECHANISM:IMMEDIATE*, the formal parameter must be a scalar; if the internal length of the actual value that you specify when you invoke the external function is greater than 4 bytes, APL signals *LENGTH ERROR*.

When you specify */MECHANISM:DESCRIPTOR*, APL uses string descriptors (CLASS_S) for the vector domain, and array descriptors (CLASS_A with a multipliers block) for arrays of rank 2 or greater. (The type of the value being passed does not affect the choice of descriptor.) For more information on descriptors, see the *Introduction to VMS System Routines*.

Note that you can abbreviate the values for */MECHANISM* to their shortest unique prefixes.

Description

Dyadic ⎕*MAP* defines an external routine to APL. Once a routine is defined in a workspace, the workspace can be saved, loaded, or copied, and the definition for the routine remains intact.

The monadic ⎕*MAP* system function returns an operation header that provides information on the current definition associated with an external routine. APL returns an operation header (*external-routine-definition*). This is the same header that dyadic ⎕*MAP* uses when you successfully define the external routine to APL. The header's form is as follows:

⟦*result/att* ←⟧*function-name/info* ⟦*arg1/att arg2/att ...*⟧

function-name shows the name that APL currently associates with the external routine.

/info shows the name and entry point of the shared image that contains the external routine. The shared image name is preceded by */IMAGE:*, and the entry point name is preceded by */ENTRY:*. If the external symbol defines a

global constant instead of an entry point, then the symbol name is preceded by */VALUE:*.

argn/att ... shows the names and attributes of the external routine's formal parameters. The attributes describe the settings for */ACCESS*, */TYPE*, and */MECHANISM* that APL associates with each formal parameter. If you did not specify a value for any of the attributes when you defined the external routine, monadic ⎕*MAP* reports the following default selections: */ACCESS:IN*, */TYPE:Z*, and */MECHANISM:UNSPECIFIED* (APL does not choose a default mechanism until you call the external routine).

result/att shows the name and attributes of the result that is returned by the external routine if there is one. the attributes describe the settings for the following: */TYPE*, which you defined with dyadic ⎕*MAP*; and */MECHANISM*, which APL determines based on the value for */TYPE*. Monadic ⎕*MAP* does not report the */ACCESS* attribute, which is assumed to be */ACCESS:OUT* by definition.

If the routine does not return a result, then monadic ⎕*MAP* does not report a value for *result/att*.

The result of monadic ⎕*MAP* contains one blank before each formal parameter, and one blank following the ← symbol (unless the function has no result).

Both monadic and dyadic ⎕*MAP* are described in Chapter 6 of the *VAX APL User's Guide*.

Possible Errors Generated

Dyadic Form

9 *RANK ERROR (NOT VECTOR DOMAIN)*

10 *LENGTH ERROR (ILLEGAL EMPTY ARGUMENT)*

15 *DOMAIN ERROR (ENCLOSED HETEROGENEOUS ARRAY NOT ALLOWED)*

15 *DOMAIN ERROR (ERROR ACTIVATING IMAGE)*

15 *DOMAIN ERROR (EXTRANEOUS CHARACTERS AFTER COMMAND)*

15 *DOMAIN ERROR (FUNCTION NAME MISSING)*

15 *DOMAIN ERROR (ILL FORMED NAME)*

15 *DOMAIN ERROR (INCORRECT TYPE)*

15 *DOMAIN ERROR* (*INCORRECT PARAMETER*)

15 *DOMAIN ERROR* (*INVALID FILE SPECIFICATION*)

15 *DOMAIN ERROR* (*KEY NOT FOUND IN TREE*)

15 *DOMAIN ERROR* (*NAME IN USE*)

15 *DOMAIN ERROR* (*OPERATION SUSPENDED OR PENDENT*)

15 *DOMAIN ERROR* (*WILDCARDS NOT ALLOWED IN FILE SPEC*)

27 *LIMIT ERROR* (*ARGUMENT TOO LONG*)

Monadic Form

9 *RANK ERROR* (*NOT VECTOR DOMAIN*)

15 *DOMAIN ERROR* (*ENCLOSED ARRAY NOT ALLOWED*)

15 *DOMAIN ERROR* (*EXTRANEOUS CHARACTERS AFTER COMMAND*)

15 *DOMAIN ERROR* (*INCORRECT TYPE*)

15 *DOMAIN ERROR* (*NOT AN EXTERNAL FUNCTION*)

⎕*MBX* Mailbox System Function

Type

System Function

Form

mailbox-info ← ⎕*MBX* *chans*

Argument Domain

Type	Near-integer
Shape	Vector domain
Depth	0 or 1 (simple)
Value	¯999 through 999 (but not 0)

Result Domain

Type	Integer
Rank	1 or 2
Shape	Vector or matrix (*n* by 3)
Depth	1 (simple)

Description

⎕*MBX* returns information on the status of mailboxes. For each channel specified, ⎕*MBX* returns a row of three elements denoting (from left to right):

- The physical device number assigned to the mailbox (or 0 if the mailbox is remote, and ¯1 if the channel is not associated with a mailbox).
- The Process IDentification number (PID, returned by ⎕*UL*) of the last user to receive a message you sent to the mailbox (or ¯1 if no messages have been sent).
- The PID of the last user from which you received a message in the mailbox (or ¯1 if no messages have been received).

The result is a matrix (or a vector if the argument is a singleton) with the shape *n* by 3, where *n* is the length of the argument.

To return a value for ⎕*MBX*, APL must open the mailbox if it is not already open. (For a list of commands that open files, see the *VAX APL User's Guide*.) For channel numbers represented in the argument by positive integers, APL opens the mailbox for input; for channel numbers represented by negative integers, APL opens the mailbox for output. Note that whether a mailbox is opened for input or output is not significant, because APL treats mailboxes like terminals: it allows both input and output at the same time, even in sequential modes.

⎕*MBX* is described in the *VAX APL User's Guide* along with other file I/O information.

Possible Errors Generated

```
9  RANK ERROR (NOT VECTOR DOMAIN)

15  DOMAIN ERROR (ENCLOSED HETEROGENEOUS ARRAY NOT ALLOWED)

15  DOMAIN ERROR (NOT AN INTEGER)

15  DOMAIN ERROR (INVALID CHANNEL NUMBER)

27  LIMIT ERROR (INTEGER TOO LARGE)
```

⎕*MONITOR* Gathering Data on Operations

Type

System Function

Form

success/failure ← *line-numbers* ⎕*MONITOR* *operation-names*
monitor-database ← ⎕*MONITOR* *operation-name*

Left Argument Domain

Type	Near-integer
Shape	Vector domain
Depth	0 or 1 (simple)

Right Argument Domain

Type	Character
Rank	1 or 2
Shape	Matrix domain
Depth	0 or 1 (simple)

Result Domain

Type	Integer (dyadic) or Boolean (monadic)
Shape	Vector or matrix (*n* by 3)
Depth	1 (simple)

Description

⎕*MONITOR* is a debugging tool that allows you to gather statistics on an operation. These include the following:

- The execution count, or the number of times an operation or operation line is invoked while ⎕*MONITOR* is enabled. The possible range is 0 to ¯1 + 2*31.

- The accumulated CPU time charged to an operation or operation line while ⎕*MONITOR* is enabled. The possible range is `0` to `¯1 + 2*31` milliseconds (about 24.5 days).

If either of these statistics overflows its range, its value is reset to 0, and the data collection continues.

Once ⎕*MONITOR* is enabled, APL collects data from the moment the operation (or operation line) receives control to the moment it relinquishes control. APL increments the execution count each time the control is relinquished and registers the accumulated CPU time from the beginning moment to the ending moment. If the operation (or operation line) calls another operation, the result includes the time required to execute this second operation.

You can view a monitored operation with the)*EDIT* command, but if you modify the operation with)*EDIT*, ⎕*FX*, or ⎕*MAP*, you will disable ⎕*MONITOR* and lose any collected data. If you view the operation with the ∆ editor, you can change the contents of individual lines without affecting the status of ⎕*MONITOR*. Note that you cannot add or delete lines or modify the header of a monitored operation with the ∆ editor.

The dyadic form of ⎕*MONITOR* enables and disables monitoring of an operation. The form used is as follows:

success/failure ←line-numbers ⎕*MONITOR operation-names*

The right argument identifies the user-defined operations you want to monitor. It belongs to the character matrix domain, and each row specifies one operation name. The operations must be user-defined, and they must be unlocked. You can also monitor line 0 of external functions (this has the same meaning as monitoring line 0 of a user-defined operation). When the same operation name applies to more than one operation in the workspace, APL monitors the most local one.

The left argument identifies the lines you want to monitor. It belongs to the near-integer vector domain. The line numbers can be in any order. APL ignores negative line numbers, repeated line numbers, and line numbers that do not appear in the operation. If the left argument contains a 0, APL monitors the entire operation.

The result of dyadic ⎕*MONITOR* is a Boolean vector. Each position in the vector corresponds to a row of the right argument. A 1 means that the attempt to enable ⎕*MONITOR* was successful, and a 0 means the attempt was unsuccessful. If the right argument is empty, the result is also empty. For example:

```
      PHASEONE ← ⎕BOX 'FOO
DOUBLE
MOVE
SPREAD
FINAL'
      (⍳50) ⎕MONITOR PHASEONE     ⍝PARENTHESES REQUIRED
0 0 0 0 0
```

To disable ⎕*MONITOR*, use the dyadic form with an empty left argument, as follows:

```
      (⍳0) ⎕MONITOR 'DESIGN'
1
```

⎕*MONITOR* on an operation that is already being monitored, APL reinitializes any previously collected data. If you want to use this data before losing it, you must retrieve it with the monadic form of ⎕*MONITOR* before you reset it with the dyadic form.

monitor-database ← ⎕*MONITOR* *operation-name*

The monadic form of ⎕*MONITOR* queries for any collected data. The argument must be in the character matrix domain, and must have at most one row; you must query for ⎕*MONITOR* information one operation at a time. Note that the operation must be unlocked and user-defined.

Monadic ⎕*MONITOR* returns an *n* by 3 numeric matrix, where *n* is the number of monitored lines. Each row of the matrix contains the current data for each line since ⎕*MONITOR* was enabled. The result is formatted as follows:

line-number execution-count cpu-time-in-milliseconds

For example:

```
      ⎕MONITOR 'FOO'
1  1  20
2  5  104
3  5  96
4  1  20
```

The result is an empty 0 by 3 matrix in the following five instances:

- The right argument is empty
- The right argument does not specify an operation name
- The operation does not exist
- The operation is locked
- The operation is not being monitored

Possible Errors Generated

The Dyadic Form

9 *RANK ERROR* (*NOT MATRIX DOMAIN*)

9 *RANK ERROR* (*NOT VECTOR DOMAIN*)

15 *DOMAIN ERROR* (*ENCLOSED ARRAY NOT ALLOWED*)

15 *DOMAIN ERROR* (*INCORRECT TYPE*)

15 *DOMAIN ERROR* (*NOT AN INTEGER*)

27 *LIMIT ERROR* (*INTEGER TOO LARGE*)

Monadic Form

9 *RANK ERROR* (*NOT MATRIX DOMAIN*)

10 *LENGTH ERROR*

15 *DOMAIN ERROR* (*ENCLOSED ARRAY NOT ALLOWED*)

15 *DOMAIN ERROR* (*INCORRECT TYPE*)

⎕*NC* Returning a Name Classification

Type

Monadic System Function

Form

name-class-list ← ⎕*NC* *name-list*

Argument Domain

Type	Character
Shape	Matrix domain
Depth	0 or 1 (simple)

Result Domain

Type	Integer
Rank	1
Shape	1 ↑ ⍴ *name-list*
Depth	1 (simple)

Description

⎕*NC* responds with the name class of each APL object that you specify in the argument. APL objects include user-defined objects, system variables, and system functions. Each row of the argument must contain the name of one object.

The result has a length equal to the number of rows in the argument. If the argument to ⎕*NC* is empty, the result is ⍳ 0 .

The possible name classes and values returned by ⎕*NC* are as follows:

⎕*NC* Name Classes and Values

Value	Name Class
¯20	Derived function
¯5	Niladic system function
¯4	Group
¯3	Monadic, Dyadic, or Ambivalent system function
¯2	System variable
¯1	Ill-formed identifier
0	Name not in use
1	Label
2	Variable
3	User-defined function
4	User-defined operator

Examples:

```
      )FNS
AVER MEAN
      )VARS
A B C TOT
      ⎕NC 'AVER'
3
      ⎕NC 'C'
2
      ⎕NC '⎕NC'
¯3
      ⎕NC 5 4⍴'A   TOT ⎕IO MEANB   '
2 2 ¯2 0 ¯1
```

Note that ⎕*NC* returns the current local rather than global name classification of the object. For example:

```
                          ⍝F IS A FUNCTION AND A VARIABLE
      ∇F;F
[1]    F←1  ◇  ⎕BREAK  'STOP F'  ∇
      ⎕NC 'F'            ⍝THIS QUERY RETURNS THAT F IS A FUNCTION
3
      F                   ⍝EXECUTE F, WHICH GETS SUSPENDED
STOP F
      ⎕NC 'F'             ⍝NOW THE MOST LOCAL F IS A VARIABLE
2
```

Possible Errors Generated

```
9  RANK ERROR (NOT MATRIX DOMAIN)

15  DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

15  DOMAIN ERROR (INCORRECT TYPE)
```

⎕*NG* Print High Minus

Type

System Variable

Form

⎕*NG* ← *near-integer-singleton*
integer-scalar ← ⎕*NG*

Value Domain

Type	Near-Integer
Shape	Singleton
Depth	0 or 1 (simple)
Value	0, 1, or 2
Default	1 (high minus sign)

Result Domain

Type	Integer
Rank	0
Shape	⍳0 (scalar)
Depth	0 (simple scalar)

Description

⎕*NG* controls the output representation of the APL negative sign, the high minus (¯). ⎕*NG* affects the primitive functions monadic and dyadic format and the system functions ⎕*FI*, ⎕*VI*, and ⎕*FMT*. The following table describes the display of the minus sign for each of the possible values for ⎕*NG*.

Value	Meaning in APL Output
0	The minus sign (-) is used as the negative sign.
1	The high minus sign ¯ (.NG in TTY mode) is used as the negative sign.
2	The ASCII minus sign (-) is used as the negative sign.

When ⎕*NG* = 2, negative numbers are preceded by an APL "+" symbol when formatted by ⍕ and ⎕*FMT*. Because APL "+" prints as an ASCII "-", you can use ⎕*NG* = 2 to handle negative numbers in strings that will be read or written in ASCII. Note that ⎕*FI* and ⎕*VI* recognize negative numbers that are preceded by an APL "+" symbol as negative numbers.

The value of ⎕*NG* is saved when you save the active workspace, and ⎕*NG* can be localized in user-defined operations.

Possible Errors Generated

9 *RANK ERROR* (*NOT VECTOR DOMAIN*)

10 *LENGTH ERROR* (*NOT SINGLETON*)

15 *DOMAIN ERROR* (*ENCLOSED ARRAY NOT ALLOWED*)

15 *DOMAIN ERROR* (*INCORRECT TYPE*)

15 *DOMAIN ERROR* (*NOT AN INTEGER*)

15 *DOMAIN ERROR* (*PARAMETER OUT OF RANGE*)

27 *LIMIT ERROR* (*INTEGER TOO LARGE*)

⎕*NL* Constructing a List of Names

Type

Ambivalent System Function

Form

name-list ← ⎕*NL name-classes*
name-list ← *letter-list* ⎕*NL name-classes*

Left Argument Domain

Type	Character
Shape	Vector domain
Depth	0 or 1 (simple)

Right Argument Domain

Type	Near-integer
Shape	Vector domain
Depth	0 or 1 (simple)

Result Domain

Type	Character
Rank	2
Shape	Matrix
Depth	1 (simple)

Description

⎕*NL* lists the names of all existing APL objects belonging to the name classes specified in the argument. APL objects include user-defined objects, system variables, and system functions.

The right argument specifies one or more name classes. The possible values for the right argument and the classes those values represent are as follows:

Value	Names Returned
¯5	Niladic system functions
¯4	Groups
¯3	Monadic, dyadic, and ambivalent system functions
¯2	System variables
1	Labels
2	User-defined variables
3	User-defined functions
4	User-defined operators

Note that ⎕*NL* returns the current local name of an object rather than the global name. For example:

```
      ∇F;F                   ⍝F IS A FUNCTION (3) AND A VARIABLE (2)
[1]   F←1  ⋄ ⎕BREAK 'STOP F'  ∇
      ⎕NL 3                  ⍝THIS QUERY RETURNS THAT F IS A FUNCTION
F
      F                      ⍝EXECUTE F, WHICH GETS SUSPENDED
STOP F
      ⎕NL 2                  ⍝NOW THE MOST LOCAL F IS A VARIABLE
F
      ⎕NL 3                  ⍝THE FUNCTION F IS NO LONGER LISTED
                                 (APL outputs a blank line.)
```

The result of ⎕*NL* is a character matrix. If the right argument is empty, or if there are no objects belonging to the specified name class, the result is `0 0 ⍴ ''`. Otherwise, each row contains the name of one object. All rows have the same number of columns; ⎕*NL* pads the ends of the shorter names with blanks.

APL returns the objects of each name class in ⎕*AV* order. When the right argument specifies more than one name class, APL catenates the alphabetical lists without merging them together. The order of the lists is as follows:

- Niladic system functions
- Monadic, dyadic, and ambivalent system functions
- System variables
- User-defined names

The dyadic form of ⎕*NL* allows you to restrict the name list to names beginning with the characters in the left argument. For example, the following constructs a name list of function names whose initial letters are *A* through *F*:

```
NLIST← 'ABCDEF' ⎕NL 3
```

The left argument of ⎕*NL*, if used, must be a character array whose shape is in the vector domain. The order of the characters does not affect the ⎕*AV* order of the result. APL ignores the left argument if it is empty. Note that the first character of a system function or system variable is the ⎕ symbol; if you use the dyadic form of ⎕*NL* and specify either ¯5, ¯3, or ¯2 in the right argument, APL ignores the ⎕ as it searches for the names beginning with the letters contained in the left argument.

The following example shows the construction of a matrix containing the names of variables in the active workspace that begin with the letter V:

```
      NLIST←'V' ⎕NL 2
      NLIST
VAR1
VAR2
VAR203
VBMAX
```

The ⎕*NL* system function is useful for a variety of purposes. For example:

- ⎕*NL* can interact with ⎕*CR* to create functions that automatically display the definitions of some or all the functions in the workspace.
- With ⎕*EX*, ⎕*NL* can dynamically erase all the named objects in a certain category. You can also use ⎕*NL* to design a function that will clear a workspace of all but a preselected collection of named objects.
- In its dyadic form, ⎕*NL* can guide you in choosing names while you develop or interact with a workspace.

Possible Errors Generated

```
9  RANK ERROR (NOT VECTOR DOMAIN)

15  DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

15  DOMAIN ERROR (INCORRECT TYPE)

15  DOMAIN ERROR (NOT A LETTER)

15  DOMAIN ERROR (NOT AN INTEGER)
```

```
15 DOMAIN ERROR (PARAMETER OUT OF RANGE)

27 LIMIT ERROR (INTEGER TOO LARGE)
```

⎕*NUM* Digits

Type

Niladic System Function

Form

```
'0123456789' ← ⎕NUM
```

Result Domain

Type	Character
Rank	1
Shape	10
Depth	1 (simple)

Description

⎕*NUM* is a subset of ⎕*AV*. ⎕*NUM* returns a vector of the 10 digits `0123456789`, or, expressed in terms of ⎕*AV*:

```
⎕AV[48+⍳10]
```

For example:

```
      ⎕NUM
0123456789
      ⎕IO←0
      ⎕AV⍳⎕NUM
48 49 50 51 52 53 54 55 56 57
```

Possible Errors Generated

None.

⎕*OM* Indexing a Boolean Vector

Type

Monadic System Function

Form

indexes ← ⎕*OM near-Boolean*

Right Argument Domain

Type	Near-Boolean
Shape	Vector domain
Depth	0 or 1 (simple)

Result Domain

Type	Integer
Rank	1
Shape	+/*near-Boolean*
Depth	1 (simple)

Implicit Arguments

⎕*IO* (⎕*OM B* when ⎕*IO* ← 1 is identical to 1 + ⎕*OM B* when ⎕*IO* ← 0

Description

⎕*OM* produces indexes showing the locations of the 1s in a Boolean vector. If the argument is empty, the result is ⍳ 0. For example:

```
      MERZ ← 0 0 0 1 0 1 1 1 0 0 1 0
      ⎕OM MERZ
4 6 7 8 11
      A ← 'THE QUICK BROWN FOX'
      ⎕OM A = ' '
4 10 16
```

Note that the following definition applies: ⎕*OM A* ←→ *A*/⍳⍴,*A*

Possible Errors Generated

```
9  RANK ERROR (NOT VECTOR DOMAIN)

15  DOMAIN ERROR

15  DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

15  DOMAIN ERROR (INCORRECT TYPE)
```

⎕*PACK* Packing and Unpacking Data

Type

success / fail ← *data-packets* ⎕*PACK* *variable-names*

Monadic Argument Domain

Type	Character
Shape	Matrix domain
Depth	0 or 1 (simple)

Monadic Result Domain

Type	Integer
Rank	1
Shape	Vector
Depth	1 (simple)

Dyadic Left Argument Domain

Type	Near-integer
Shape	Vector
Depth	1 (simple)

Dyadic Right Argument Domain

Type	Character
Shape	Matrix domain
Depth	0 or 1 (simple)

Dyadic Result Domain

Type	Boolean
Rank	1
Shape	Vector
Depth	1 (simple)

Description

⎕*PACK* allows you to pack and unpack data of different types into a single variable known as a packet. The monadic form packs data, and the dyadic form unpacks it.

⎕*PACK* differs from ⎕*COQ* and ⎕*CIQ* because it allows you to pack and unpack variables of different data types with only one invocation of the ⎕*PACK* function. (Otherwise, to pack variables you invoke ⎕*COQ* once for each data type, and then catenate the results into a single variable; to unpack a variable you undo the catenation and then invoke ⎕*CIQ* once for each data type.) Unlike ⎕*COQ*, ⎕*PACK* does not convert data into different data types before packing it.

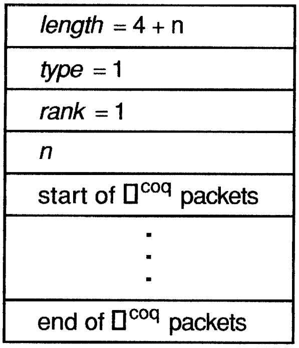
Use monadic ⎕*PACK* to pack data. When you specify a single variable, ⎕*PACK* creates a ⎕*COQ* packet with a header; it does not perform any data type conversion before creating the packet. When you specify more than one variable, ⎕*PACK* creates individual ⎕*COQ* packets for each variable and combines them in a single logical record.

The argument to the monadic form contains the names of the user-defined variables you want to pack. If the argument to monadic ⎕*PACK* is empty, then the result is ⍳0.

Use dyadic ⎕*PACK* to unpack data. The left argument must be a vector; it identifies a packet that was created by monadic ⎕*PACK*.

The right argument contains the names you want to assign to the individual packets as they are unpacked from the left argument. It must have one row for each individual packet in the left argument. Each name can have a name class of 0 or 2 (undefined name or user-defined variable). When the name class is 0, the variable becomes defined. When the name class is 2, APL redefines the variable. If the right argument contains a blank row, APL does not unpack the individual packet associated with that row.

The result indicates whether the names contained in the right argument have been successfully assigned the ⎕*CIQ* value of the individual packets. A 0 indicates an unsuccessful assignment (caused by a blank row in the right argument), and a 1 indicates a successful assignment. Each position in the result corresponds to a row in the right argument. If the left or right argument is empty, then the result is ⍳0. The header generated by ⎕*PACK* has the following format:



length = 4 + n
type = 1
rank = 1
n
start of ⎕coq packets
⋮
end of ⎕coq packets

NU–2235A–RA

Each large box represents a longword. *length* is the length of the result (an integer vector) of monadic ⎕*PACK*. *type* always has a value of 1, indicating 32-bit integers. *rank* is always 1. *n* indicates the length of the data section of the packet. The data section (elements 5 through 5 + *n*) contains the integer representations of the individual packets.

The following example shows how the individual packets created by ⎕*PACK* relate to the packets created by ⎕*CIQ*. Note the use of ⎕*BOX* in the right argument to dyadic ⎕*PACK*; it is used to facilitate the entry of the names of the individual packets as a character matrix:

```
      A←⍳5
      B←2 4 ⍴ 'ABCD'
      ⎕ ← AA ← A ⎕COQ 2
9 1 1 5 1 2 3 4 5
      ⎕ ← BB ← B ⎕COQ 2
5 2 2 4 1684234849 1684234849
      D←⎕PACK ⎕BOX 'A
B'
      D
20 1 1 16 9 1 1 5 1 2 3 4 5 7 5 2 2 4 1684234849 1684234849
      ⎕ ← N ← (⍴AA) + ⍴BB
16
      D ≡ ((N+4), 1 1, N),AA,BB
1
      D ⎕PACK ⎕BOX 'AAA
```

```
BBB'
1 1
      AAA
1 2 3 4 5
      AAA ≡ A
1
      BBB
ABCD
ABCD
      BBB ≡ B
1
```

Possible Errors Generated

9 RANK ERROR (MUST BE VECTOR)

9 RANK ERROR (NOT MATRIX DOMAIN)

10 LENGTH ERROR (ITEM COUNT MISMATCH)

15 DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

15 DOMAIN ERROR (ILLEGAL NAME CLASS)

15 DOMAIN ERROR (INCORRECT TYPE)

15 DOMAIN ERROR (INVALID CIQ HEADER)

15 DOMAIN ERROR (INVALID LENGTH IN PACK HEADER)

15 DOMAIN ERROR (INVALID RANK IN PACK HEADER)

15 DOMAIN ERROR (INVALID RHO VECTOR IN PACK HEADER)

15 DOMAIN ERROR (INVALID TYPE IN PACK HEADER)

15 DOMAIN ERROR (NOT AN INTEGER)

27 LIMIT ERROR (INTEGER TOO BIG)

⎕*PP* Print Precision

Type

System Variable

Form

⎕*PP* ← *digits-of-precision*
integer-scalar ← ⎕*PP*

Value Domain

Type	Near-integer
Shape	Singleton
Depth	0 or 1 (simple)
Value	1 to 16
Default	10

Result Domain

Type	Integer
Rank	0
Shape	⍳0
Depth	0 (simple scalar)

Description

⎕*PP* determines how many significant digits are displayed in APL floating-point output. It does not affect the display of integers or the precision of internal calculations. It does affect the conversion of numbers to characters by the dyadic format function or the display of floating-point constants in ⎕*CR* and ⎕*VR*.

Legal values for ⎕*PP* are the integers 1 through 16. The default is 10. APL rounds off numbers that contain more digits than the setting. For example:

```
      ⎕PP
10
      123456789.123456789
123456789.1
      ⎕PP←5
      123456789.123456789
1.2346E8
      ⎕PP←15
      123456789.123456789
123456789.123457
      ⎕PP←10
                         ⍝LEADING ZEROS ARE NOT SIGNIFICANT
                         ⍝NOTE THAT ROUNDING MAY MAKE RESULT
                         ⍝HAVE FEWER THAN ⎕PP DIGITS
      ⎕←A←2 1ρ1 4 ÷ 101
0.009900990099
0.0396039604
      ⎕PP←11
      A
0.009900990099
0.039603960396
      ⎕PP←12
      A
0.00990099009901
0.039603960396
```

The value of ⎕*PP* is saved when you save the active workspace and can be localized in user-defined operations.

Possible Errors Generated

9 *RANK ERROR* (*NOT VECTOR DOMAIN*)

10 *LENGTH ERROR* (*NOT SINGLETON*)

15 *DOMAIN ERROR* (*ENCLOSED ARRAY NOT ALLOWED*)

15 *DOMAIN ERROR* (*INCORRECT TYPE*)

15 *DOMAIN ERROR* (*NOT AN INTEGER*)

15 *DOMAIN ERROR* (*PARAMETER OUT OF RANGE*)

27 *LIMIT ERROR* (*INTEGER TOO LARGE*)

⎕*PW* Print Width

Type

System Variable

Form

⎕*PW* ← *print-positions*
integer-scalar ← ⎕*PW*

Value Domain

Type	Near-integer
Shape	Singleton
Depth	0 or 1 (simple)
Value	35 to 2048
Default	Determined when APL is invoked

Result Domain

Type	Integer
Rank	0
Shape	⍳0
Depth	0 (simple scalar)

Description

⎕*PW* specifies the maximum number of characters that can appear on a terminal output line before a <CR><LF> is performed. ⎕*PW* has no effect on the length of input lines. The default uses the current VMS setting for set terminal/width=*n*.

If an output line requires more than ⎕*PW* character positions, printing continues on succeeding indented lines. For example:

```
      ⎕PW←35
      A←'THIS IS A TEST OF THE PRINT WIDTH VARIABLE'
      A
THIS IS A TEST OF THE PRINT WIDTH V
      ARIABLE
```

However, if a line in an error message is longer than ⎕*PW* characters, it is truncated; it is not continued on the next line. If truncating the line prevents APL from displaying the particular point in the line at which the error was discovered, APL cuts off enough characters from the beginning of the line to allow the part in error to be displayed.

The value of ⎕*PW* is saved when you save the active workspace and can be localized in user-defined operations. When you exit from APL, the original terminal-width value is restored.

Actually, APL never changes your terminal width; in effect, it overrides the width by preventing lines from wrapping and then by formatting any output based on the value of ⎕*PW*.

Note that if you interrupt your APL session (for example, by executing a)*PUSH* command) and then execute an operating system command to change the value of the terminal width, the value of ⎕*PW* is not changed when you return to APL. For example:

```
      )CLEAR
CLEAR WS
      ⎕PW                      ⍝⎕PW INITIAL VALUE = SYSTEM TERM WIDTH
80
      ⎕PW←65                   ⍝CHANGE ⎕PW TO 65
                               ⍝INTERRUPT APL SESSION
      )PUSH
$SET TERMINAL/WIDTH = 72
$LOGOUT
 Process USER logged out at 16-NOV-1990 13:24:43.11
      ⎕PW                      ⍝⎕PW NOT CHANGED TO 72
65
```

If you exit APL via a panic exit, your system terminal width takes effect, but your terminal retains the APL setting that prevents lines from wrapping, regardless of the way wrapping was handled before you began your APL session. If you want lines to wrap, execute the DCL command set terminal /wrap.

Possible Errors Generated

```
9  RANK ERROR (NOT VECTOR DOMAIN)

10  LENGTH ERROR (NOT SINGLETON)

15  DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)
```

```
15  DOMAIN ERROR (INCORRECT TYPE)

15  DOMAIN ERROR (NOT AN INTEGER)

15  DOMAIN ERROR (PARAMETER OUT OF RANGE)

27  LIMIT ERROR (INTEGER TOO LARGE)
```

⎕*QCO* Copying Objects from a Workspace

Type

Monadic System Function (quiet)

Form

message ← ⎕QCO *wsname* ⟦*object-names*⟧

Argument Domain

Type	Character
Shape	Vector domain
Depth	0 or 1 (simple)

Result Domain

Type	Character
Rank	1
Shape	Vector
Depth	1 (simple)

Qualifiers

/PASSWORD **⟦:*pw*⟧**
If a workspace is saved with a password, you must specify the password to copy objects from the workspace.

/CHECK
The optional */CHECK* qualifier causes APL to examine the workspace for possible corruption (damage to the internal structure of the workspace). If damage is detected, a message is displayed and APL tries to recover as much information as possible from the workspace and continues the copy. The recovered workspace may be missing APL variables, user-defined operations, and other APL objects that were damaged. The user must determine what named objects have been removed from the workspace.

Description

⎕QCO (quiet copy) retrieves global objects from a workspace and places them into your active workspace.

The argument has four parts: the name of the workspace from which you want to copy the objects, an optional password, an optional qualifier (/CHECK), and an optional list of objects to be copied.

Use the list of objects to identify the specific objects you want to copy. If you omit the list, all global user-defined operations, global variables, and groups are copied. When you specify the objects, you can use the ⋆ and ÷ wildcards. Note that ⎕QCO does not transfer local values for variables, functions, and operators, nor does it copy the state indicator or system variables like the print width, index origin, or significant digits settings.

The ⎕QCO system function performs the same operation as the)COPY system command (see Chapter 3). ⎕QCO returns as its result a character vector containing the usual)COPY command message. However, because ⎕QCO is a quiet function, if it is the leftmost function in the statement, its result is not displayed on the terminal unless there is an error (warnings are not displayed). For example:

```
      )COPY AB CALC
SAVED TUESDAY  6-NOV-1990 17:49:10.14 16 BLKS
      )CLEAR
CLEAR WS
      ⎕QCO 'AB CALC'
      )CLEAR
CLEAR WS
      MSG←⎕QCO 'AB CALC TOT'
      MSG
SAVED TUESSDAY  6-NOV-1990 17:49:10.14 13 BLKS
NOT FOUND: TOT
```

If your active workspace contains objects with the same names as those in the copied workspace, ⎕QCO replaces the global (but not the local) values in your active workspace with the copied ones. For example, if B is a variable in the active workspace with a global value of 10 and a local value of 5, and the workspace being copied has a variable B with a global value of 20, the active workspace after ⎕QCO executes will have a variable B with a global value of 20 and a local value of 5. A pendent or suspended operation is not replaced, and an operation being created in the workspace being copied is not copied.

When you copy a group, all members of the group are copied along with their values. However, if a member of a group is itself a group, APL copies only the group name and not its values. For example, suppose the group *GROUP*1 has as members the variables *A* and *B*, and the group *GROUP*2. Also suppose that *GROUP*2 has as members the variables *C* and *D*. Then, if you copy *GROUP*1, you copy the values of *A* and *B*, but only the name of *GROUP*2, not the values of *C* and *D*.

If the object list contains objects that are not in the specified workspace, APL returns the warning message *NOT FOUND* followed by the names (separated by tabs) that were not found. The objects that were found are still copied, however.

Examples:

```
      )CLEAR
CLEAR WS
      )COPY T A B
SAVED TUESDAY  6-NOV-1990 17:51:20.88 13 BLKS
      A
1
      B
2
      C
 11 VALUE ERROR
      C
       ^
      )CLEAR
CLEAR WS
      MSG←⎕QCO 'T'
      MSG
SAVED TUESDAY  6-NOV-1990 17:51:20.88 13 BLKS
      A
1
      B
2
      C
3
      )CLEAR
CLEAR WS
      MSG←⎕QCO 'T A B'
      A
1
      B
2
      C
 11 VALUE ERROR
      C
       ^
      )CLEAR
CLEAR WS
      )COPY T A D
SAVED TUESDAY  6-NOV-1990 17:51:20.88 13 BLKS
NOT FOUND: D
      A
1
      B
 11 VALUE ERROR
      B
       ^
      )CLEAR
CLEAR WS
      MSG←⎕QCO 'T A D'
      MSG
SAVED TUESDAY  6-NOV-1990 17:51:20.88 13 BLKS
NOT FOUND: D
      ⍴MSG
59
```

Possible Errors Generated

9 *RANK ERROR (NOT VECTOR DOMAIN)*

10 *LENGTH ERROR (ILLEGAL EMPTY ARGUMENT)*

15 *DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)*

15 *DOMAIN ERROR (FILE SPECIFICATION IS MISSING)*

15 *DOMAIN ERROR (INCORRECT TYPE)*

⎕*QLD* Loading a Workspace

Type

Monadic System Function (quiet)

Form

⎕*QLD wsname*

Argument Domain

Type	Character
Shape	Vector domain
Depth	0 or 1 (simple)

Result Domain

None.

Qualifiers

/PASSWORD **[:*pw*]**
Specifies the password used when the workspace was saved. If a workspace is saved with a password, you must specify the password to copy objects from the workspace.

/CHECK
The optional */CHECK* qualifier causes APL to examine the workspace for possible corruption (damage to the internal structure of the workspace). If damage is detected, a message is displayed and APL tries to recover as much information as possible from the workspace and continues the copy. The recovered workspace may be missing APL variables, user-defined operations, and other APL objects that were damaged. The user must determine what named objects have been removed from the workspace. You must use the)*SAVE* command if you want to maintain an undamaged version of the recovered workspace.

Description

⎕*QLD* (quiet load) makes the specified workspace the active workspace by replacing the currently active workspace and destroying its contents.

The argument has three parts: the name of the workspace to be loaded, an optional password, and an optional qualifier (*/CHECK*). For example, the following loads a workspace named *ABC*, which was saved with the password *JOHN*:

```
      ⎕QLD 'ABC/PASSWORD:JOHN'
```

Note that the ⎕*QLD* system function performs the same operation as the *)LOAD* system command (see Chapter 3), but ⎕*QLD* does not print messages on the terminal unless there is an error.

⎕*QLD* does not return a result in the usual sense or display a message when it is successful, because the context in which ⎕*QLD* was executed is replaced by the loaded workspace.

If the ⎕*LX* system variable has a value in a workspace, it executes when ⎕*QLD* is used to load the workspace, except if the top of the state indicator stack contains an execute function (see the Execute function described in Chapter 1 for details), or if the workspace was saved in function-definition mode (if it was, you remain in function-definition mode after the workspace is loaded). If the workspace was saved inside ⎕ input, the ⎕*LX* expression is executed only after the pendent ⎕ input is removed from the state indicator stack. For example:

```
      A
1
      )CLEAR
CLEAR WS
      ⎕QLD 'T'
      )WSID
T
      A
1
```

Possible Errors Generated

9 *RANK ERROR (NOT VECTOR DOMAIN)*

10 *LENGTH ERROR (ILLEGAL EMPTY ARGUMENT)*

15 *DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)*

```
15  DOMAIN ERROR (FILE SPECIFICATION IS MISSING)

15  DOMAIN ERROR (INCORRECT TYPE)
```

⎕*QPC* Copying Objects with Protection

Type

Monadic System Function (quiet)

Form

message ← ⎕QPC *wsname* ⟦*object-names*⟧

Argument Domain

Type	Character
Shape	Vector domain
Depth	0 or 1 (simple)

Result Domain

Type	Character
Rank	1
Shape	Vector
Depth	1 (simple)

Qualifiers

/PASSWORD **⟦:*pw*⟧**
Specifies the password used when the workspace was saved. If a workspace is saved with a password, you must specify the password to copy objects from the workspace.

/CHECK
The optional */CHECK* qualifier causes APL to examine the workspace for possible corruption (damage to the internal structure of the workspace). If damage is detected, a message is displayed and APL tries to recover as much information as possible from the workspace and continues the copy. The recovered workspace may be missing APL variables, user-defined operations, and other APL objects that were damaged. The user must determine what named objects have been removed from the workspace. You must use the *)SAVE* command if you want to maintain an undamaged version of the recovered workspace.

Description

⎕*QPC* (quiet copy with protection) is the same as the ⎕*QCO* system function except that ⎕*QPC* does not replace objects in the active workspace with objects of the same name in the copy workspace. Instead, APL returns the warning message *NOT COPIED* followed by the names of objects (separated by tabs) that were not copied.

As with ⎕*QCO*, the argument for ⎕*QPC* represents the name of the workspace from which you want to copy the objects, followed by three optional items: a password, a qualifier (*/CHECK*), and a list of objects. When you specify the objects, you can use the ⋆ and ÷ wildcards.

When copying groups, ⎕*QPC* does not copy any members of the group that have the same name as a name already in the active workspace. If the group name itself is the same as a group name in the active workspace, APL does not copy the group name or any members of the group.

If the list to be copied contains an object that is not in the specified workspace, APL returns the warning message *NOT FOUND*, followed by the names of the objects (separated by tabs) that were not found. The objects that were found are still copied, however.

The ⎕*QPC* system function performs the same operation as the *)PCOPY* system command (see Chapter 3). ⎕*QPC* returns as its result a character vector that contains the usual *)PCOPY* command message. However, because ⎕*QCO* is a quiet function, if it is the leftmost function in the statement, the result is not displayed on the terminal unless there is an error (warning messages are not displayed).

Examples:

```
      )CLEAR
CLEAR WS
      A←20
      )PCOPY T
SAVED TUESDAY  6-NOV-1990 17:51:20.88 13 BLKS
NOT COPIED: A
      A
20
      B
2
      C
3
```

```
      )CLEAR
CLEAR WS
      A←20
      MSG←⎕QPC 'T'
      MSG
SAVED TUESDAY  6-NOV-1990 17:51:20.88 13 BLKS
NOT COPIED: A
      A
20
      B
2
      C
3
      MSG←⎕QPC 'T A D'
      MSG
SAVED TUESDAY  6-NOV-1990 17:51:20.88 13 BLKS
NOT FOUND: D
NOT COPIED: A
      ρMSG
74
```

Possible Errors Generated

9 *RANK ERROR (NOT VECTOR DOMAIN)*

10 *LENGTH ERROR (ILLEGAL EMPTY ARGUMENT)*

15 *DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)*

15 *DOMAIN ERROR (FILE SPECIFICATION IS MISSING)*

15 *DOMAIN ERROR (INCORRECT TYPE)*

⎕*R* Monitoring Variable Changes

Type

System Variable

Form

⎕*R* ← *any*
old-value ← ⎕*R*

Value Domain

Type	Any
Shape	Any
Depth	Any
Default	''

Result Domain

Type	Any
Rank	Any
Shape	Any
Depth	Any

Description

⎕*R* and ⎕*L* are system variables that are implicitly used by ⎕*WATCH*. ⎕*WATCH* is a system function that is used to monitor any changes in one or more variables. When a change occurs in a monitored variable, APL assigns information to ⎕*R* and ⎕*L*: ⎕*R* contains the previous value of the changed variable; ⎕*L* contains a character vector showing the name of the variable that has changed. APL assigns this information regardless of whether monitoring is set for signal or display mode.

Immediately after a ⎕*WATCH* event occurs, ⎕*R* and ⎕*L* contain the new information resulting from the event. However, this information may change as an operation continues execution (this is especially true if an error occurs during an assignment or reference of a variable that is associated with a watchpoint).

Both ⎕*R* and ⎕*L* can be localized, explicitly assigned values of any type, and saved in the workspace.

Note that you cannot include ⎕*R* or ⎕*L* in the right argument to dyadic ⎕*WATCH*.

Possible Errors Generated

None.

⎕*RELEASE* Unlocking Shared Records

Type

Monadic System Function (quiet)

Form

```
ι0 ← ⎕RELEASE chans
```

Argument Domain

Type	Near-integer
Shape	Vector domain
Depth	0 or 1 (simple)
Value	¯999 through 999 (but not 0)

Result Domain

Type	Numeric
Rank	1
Shape	0 (empty)
Depth	1 (simple)

Description

⎕*RELEASE* unlocks any locked records in files associated with the channel numbers specified in the argument. The absolute values of *chans* represent the channels associated with the files you want to unlock.

⎕*RELEASE* is quiet; it does not return a result if it is the leftmost function in a statement. When it is not the leftmost function, ⎕*RELEASE* returns an empty numeric vector. If its argument is empty, ⎕*RELEASE* has no effect and its result is an empty vector. Note that APL performs a ⎕*RELEASE* on all open files whenever a)*MON* command is executed.

If you read a record that you do not intend to rewrite, it is a good idea to unlock it as soon as possible, because other users who try to retrieve it are put in a wait state until the record becomes available.

⎕*RELEASE* is described the *VAX APL User's Guide* along with other file I/O information.

Possible Errors Generated

9 *RANK ERROR (NOT VECTOR DOMAIN)*

15 *DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)*

15 *DOMAIN ERROR (NOT AN INTEGER)*

15 *DOMAIN ERROR (INVALID CHANNEL NUMBER)*

27 *LIMIT ERROR (INTEGER TOO LARGE)*

⎕*REP* Replication

Type

Dyadic System Function

Form

A ⎕*REP B A* ⎕*REP*[*K*] *B*

Left Argument Domain

Type	Near-integer
Shape	domain
Depth	or 1 (simple)

Right Argument Domain

Type	Any
Shape	Any
Depth	Any

Result Domain

Type	as right argument
Rank	1⌈⍴⍴*B*
Shape	((*K*-1)↑⍴*B*),(+/∣*A*),*K*↓⍴*B* (for ⎕*IO* 1)
Depth	1⌈≡*B*

Implicit Arguments

None.

Description

⎕*REP* builds arrays by specifying the items to be deleted, preserved, or duplicated from an existing array, and by indicating where fill items are to be added in the new array. When items are preserved or deleted, this is known as compression (the left argument is Boolean). When items are duplicated, deleted, or filled, this is known as replication (the left argument is integer).

⎕*REP* works the same as the compress and replicate derived functions. The difference between ⎕*REP* and the slash operator is that you can use ⎕*REP* as an operand to an operator. Operators cannot be used as operands to operators. ⎕*REP* applies along the last axis of *B* unless modified by an explicit axis (*K*) in brackets. The shape of the result is the same as the original array *B* except along the applicable axis (⍴*B*)[*K*]. The shape of that axis becomes the sum of the absolute value of the items in *A*(+/|*A*).

The following examples show ⎕*REP* with the each (¨) operator. Although the variables *A* and *C* are nested in the examples, they conform to the left argument domain requirement that specifies a simple array. This is because the each operator reduces the nesting by one level:

```
      ⎕←A←(1 0 1) (1 ¯1 0 2)
+-----+ +--------+
|1 0 1| |1 ¯1 0 2|
+-----+ +--------+
      ⎕←B←(2 3⍴⍳6) (4 3⍴'ABCDEFGHIJKL')
+-----+ +---+
|1 2 3| |ABC|
|4 5 6| |DEF|
+-----+ |GHI|
        |JKL|
        +---+
                           ⍝ATTEMPT TO USE OPERATOR AS ARG TO
      A/¨B                 ⍝APL EVALUATES AS (A/)¨B
 15 DOMAIN ERROR (ENCLOSED/HETEROGENEOUS ARRAY NOT ALLOWED)
      A/¨B                     ⍝APL EVALUATES AS (A/)¨B
       ∧
      A ⎕REP¨B    ⍝REPLICATE ITEMS OF B USING THE ITEMS OF A
+---+ +----+
|1 3| |A CC|
|4 6| |D FF|
+---+ |G II|
      |J LL|
      +----+
      A ⎕REP[2]¨ B         ⍝SECOND AXIS↔ DEFAULT IN THIS CASE
+---+ +----+
|1 3| |A CC|
|4 6| |D FF|
+---+ |G II|
      |J LL|
      +----+
      ⎕←C←(2 ¯1 0) (1 1 0 1)
+------+ +-------+
|2 ¯1 0| |1 1 0 1|
+------+ +-------+
      C ⎕REP[1]¨ B        ⍝USE ⎕REP WITH AXIS ARGUMENT
+-----+ +---+
|1 2 3| |ABC|
|1 2 3| |DEF|
|0 0 0| |JKL|
+-----+ +---+
```

Possible Errors Generated

9 *RANK ERROR (NOT VECTOR DOMAIN)*

10 *LENGTH ERROR*

15 *DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)*

15 *DOMAIN ERROR (INCORRECT TYPE)*

15 *DOMAIN ERROR (NOT AN INTEGER)*

27 *LIMIT ERROR (INTEGER TOO LARGE)*

28 *AXIS RANK ERROR (NOT VECTOR DOMAIN)*

29 *AXIS LENGTH ERROR (NOT SINGLETON)*

30 *AXIS DOMAIN ERROR (SEMICOLON LIST NOT ALLOWED)*

30 *AXIS DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)*

30 *AXIS DOMAIN ERROR (INCORRECT TYPE)*

30 *AXIS DOMAIN ERROR (NOT AN INTEGER)*

30 *AXIS DOMAIN ERROR (AXIS LESS THAN INDEX ORIGIN)*

30 *AXIS DOMAIN ERROR (RIGHT ARGUMENT HAS WRONG RANK)*

⎕*RESET* Resetting the State Indicator

Type

Niladic System Function

Form

⎕*RESET*

Result Domain

None.

Description

⎕*RESET* clears the state indicator. When the state indicator is clear, no user-defined operations are suspended, no quad input requests or execute functions are pending, and the)*SI* system command (see Chapter 3) does not return a value.

⎕*RESET* does not return a value.

Possible Errors Generated

None.

⎕*REWIND* Returning Next-Record Pointer to Start of File

Type

Ambivalent System Function (quiet)

Form

```
⍳0 ← ⎕REWIND chans
⍳0 ← key-of-reference ⎕REWIND chans
```

Monadic Argument Domain

Type	Near-integer
Shape	Vector domain
Depth	0 or 1 (simple)
Value	¯999 through 999 (but not 0)

Dyadic Left Argument Domain

Type	Near-integer
Shape	Singleton
Depth	0 or 1 (simple)
Value	0 through 255 inclusive

Dyadic Right Argument Domain

Type	Near-integer
Shape	Singleton
Depth	0 or 1 (simple)
Value	¯999 through 999 (but not 0)

Result Domain

Type	Numeric
Rank	1
Shape	0 (empty)
Depth	1 (simple)

Description

⎕*REWIND* allows you to reposition the next record pointer to the first record of a file without closing the file. The absolute values of *chans* represent the channels associated with the files you want to rewind.

With the monadic form, you can specify a vector of channel numbers in the right argument. This will rewind each of the files associated with the specified channel numbers. If any of the files have a keyed organization, APL performs the rewind on the primary key of reference.

Use the dyadic form for keyed files when you want APL to perform the rewind on a key of reference other than the primary key. The right argument specifies the channel number associated with the keyed file. The left argument specifies the key of reference. Zero (0) indicates the primary key, one (1) indicates the secondary key, and so on. You can specify only one file at a time when you invoke dyadic ⎕*REWIND*.

⎕*REWIND* is described in the *VAX APL User's Guide* along with other file I/O information.

Possible Errors Generated

Monadic Form

9 *RANK ERROR* (*NOT VECTOR DOMAIN*)

15 *DOMAIN ERROR* (*INCORRECT TYPE*)

15 *DOMAIN ERROR* (*ENCLOSED HETEROGENEOUS ARRAY NOT ALLOWED*)

15 *DOMAIN ERROR* (*INTEGER TOO LARGE*)

15 *DOMAIN ERROR* (*NOT AN INTEGER*)

15 *DOMAIN ERROR* (*INVALID CHANNEL*)

15 *DOMAIN ERROR* (*CHANNEL NOT ASSIGNED*)

15 *DOMAIN ERROR (FILE IS ASSIGNED WRITE ONLY)*

33 *IO ERROR (INVALID KEY OF REFERENCE FOR $GET/$FIND)*

Dyadic Form

9 *RANK ERROR (NOT A SINGLETON)*

15 *DOMAIN ERROR (CHANNEL NOT ASSIGNED)*

15 *DOMAIN ERROR (CHANNEL NOT ASSIGNED TO A KEYED FILE)*

15 *DOMAIN ERROR (ENCLOSED HETEROGENEOUS ARRAY NOT ALLOWED)*

15 *DOMAIN ERROR (INCORRECT TYPE)*

15 *DOMAIN ERROR (INVALID CHANNEL)*

15 *DOMAIN ERROR (NOT AN INTEGER)*

15 *DOMAIN ERROR (PARAMETER OUT OF RANGE)*

27 *LIMIT ERROR (INTEGER TOO LARGE)*

33 *IO ERROR (INVALID KEY OF REFERENCE FOR $GET/$FIND)*

⎕*RL* Link

Type

System Variable

Form

⎕*RL* ← *random-seed*
integer-scalar ← ⎕*RL*

Value Domain

Type	Near-integer
Shape	Singleton
Depth	`0` or `1` (simple)
Value	`¯2*30` through `¯1+2*30`
Default	`695197565`

Result Domain

Type	Integer
Rank	`0`
Shape	`⍳0` (scalar)
Depth	`0` (simple scalar)

Description

⎕*RL* sets the seed of the pseudo-random-number generator used with the roll and deal functions (see Chapter 1). ⎕*RL* can be set by the user, and is also set implicitly by the system when roll and deal are executed.

Every time you execute a roll or deal function, the value of the random link changes. The value of ⎕*RL* is saved with a workspace and can be localized in user-defined operations. For example:

```
      ⎕RL
695197565
      5?5
4 2 3 1 5
      ⎕RL
¯47060346
      5?5
4 1 2 3 5
      ⎕RL
1636171147
```

Possible Errors Generated

```
9  RANK ERROR (NOT VECTOR DOMAIN)

10  LENGTH ERROR (NOT SINGLETON)

15  DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

15  DOMAIN ERROR (INCORRECT TYPE)

15  DOMAIN ERROR (NOT AN INTEGER)

27  LIMIT ERROR (INTEGER TOO LARGE)
```

⎕*SF* Quad Input Prompt

Type

System Variable

Form

⎕*SF* ← *prompt*
char-vector ← ⎕*SF*

Value Domain

Type	Character
Shape	Vector domain
Depth	0 or 1 (simple)
Value	*prompt* length≤ 255 keystrokes
Default	'⎕: <CR><LF> 6-spaces'

Result Domain

Type	Character
Rank	1
Shape	Vector
Depth	1 (simple)

Description

⎕*SF* specifies the text to be used as the prompt for quad input (see the *VAX APL User's Guide*). You can use any printing characters in the prompt.

The prompt is printed each time a request is made for quad input (⎕). For example:

```
      A←3+⎕+5
⎕:
      5
      A
13
      B←⎕
⎕:
      'INPUT'
      B
INPUT
      ⎕SF←'WHAT IS YOUR NAME? '
      C←⎕
WHAT IS YOUR NAME? 'CARLA'
      C
CARLA
```

Note that you must enclose character-type quad input in single quotation marks.

The maximum length for ⎕*SF* is 255 keystrokes (a keystroke occurs any time you press a key on the keyboard, including the Space bar and the Backspace key). The value of ⎕*SF* is saved when you save the active workspace and can be localized in user-defined operations.

Possible Errors Generated

```
9  RANK ERROR (NOT VECTOR DOMAIN)

15  DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

15  DOMAIN ERROR (INCORRECT TYPE)

27  LIMIT ERROR (ARGUMENT STRING IS TOO LONG)
```

⎕*SIGNAL* Signaling Errors

Type

Ambivalent System Function

Form

error-number
message-text ⎕SIGNAL *error-number*

Monadic Argument Domain

Type	Near-integer
Shape	Singleton
Depth	0 or 1 (simple)
Value	Any APL error number (except 75, 115 to 499 or greater than 999)

Dyadic Left Argument Domain

Type	Character
Shape	Vector domain
Depth	0 or 1 (simple)

Dyadic Right Argument Domain

Type	Near-integer
Shape	Singleton
Depth	0 or 1 (simple)
Value	Any APL error number (except 75, 115 to 499 or greater than 999)

Result Domain

None.

Description

⎕*SIGNAL* allows you to signal an error to the caller of the operation in error; thus, the way a user-defined operation that executes ⎕*SIGNAL* fails is similar to the way a primitive function fails.

In both the monadic and dyadic forms, the right argument is the error number. You can use an existing APL error number (except 75) as listed in Appendix A, or you can define your own number (within the range 500 through 999).

The left argument (dyadic form), if used, is the text of the error message for the error you are signaling. For example:

```
'WILL NOT ACCEPT NEGATIVE NUMBERS' ⎕SIGNAL 501
```

This statement, if executed within a user-defined operation (⎕*SIGNAL* generally appears within a user-defined operation, but this is not a requirement), signals the following error:

```
501  WILL NOT ACCEPT NEGATIVE NUMBERS
```

The message is followed by the rest of the APL standard three-line error message; that is, the text of the line in error and a caret pointing to the part of the line in error. The three-line error message generated by ⎕*SIGNAL* becomes the value of ⎕*ERROR*.

If the error number you supply to ⎕*SIGNAL* is the number of an APL error, the message displayed (and stored in ⎕*ERROR*) is the error message that coincides with that number (see Appendix A for a description of APL error messages), and the left argument to ⎕*SIGNAL* becomes the secondary error message (displayed in parentheses following the primary error message). If you do not use an existing APL error number, and you leave the left argument blank, APL signals the following error:

```
ERROR SIGNALED
```

In the following example, notice that the error is signaled at the level of the caller, function *H*, not at function *F*:

```
                           ⍝FUNCTION F HAS ⎕SIGNAL
      ∇F A
[1]    →(A>0)/3
[2]    'WILL NOT ACCEPT NEGATIVE NUMBERS' ⎕SIGNAL 15
[3]    'FUNCTION CONTINUING NORMALLY'
[4]    ∇
                           ⍝FUNCTION H CALLS F
      ∇H A
[1]    F A
[2]    ∇
      H 5
FUNCTION CONTINUING NORMALLY
      H ¯7
 15 DOMAIN ERROR (WILL NOT ACCEPT NEGATIVE NUMBERS)
  H[1] F A
        ^
```

You can use error number 80 to signal a status condition to the DCL interpreter. The right argument to ⎕*SIGNAL* must be 80, and the left argument is a character string representing a hexadecimal number that is the status code you want to return to VMS. The status code returned is stored as the value of the global symbol $STATUS. For example:

```
      '123ABC1' ⎕SIGNAL 80

                             (APL returns control to DCL)
$show symbol $status
$STATUS == "%X0123ABC1"
```

The low-order three bits of $STATUS represent the severity level of the error signaled and are contained by the global symbol $SEVERITY. For example:

```
$show symbol $severity
$SEVERITY == "1"
```

DCL command procedures interpret the $SEVERITY value 1 to mean success, and the value 2 to mean error. (For details on command procedures, see the *Guide to Using VMS Command Procedures*.) In the following command procedure, the first line means branch to the label ERROR any time $SEVERITY becomes equal to 2:

```
on error then goto error
$ apl
   .
   .
   .
     (APL statements)
   .
   .
   .
$ write sys$output"No Error From APL"
$ exit
$ error:
$ write sys$output"APL Returned Error"
```

If `'2'` ⎕*SIGNAL* `80` is executed during the APL session, the command procedure branches to ERROR and displays the message "APL Returned Error". If `'1'` ⎕*SIGNAL* `80` is executed, the command procedure displays "No Error From APL" and then exits.

For more information on the use of ⎕*SIGNAL*, see Chapter 3 in the *VAX APL User's Guide*.

Possible Errors Generated

15 *DOMAIN ERROR* (*CANNOT SIGNAL EOF*)

15 *DOMAIN ERROR* (*NOT VECTOR DOMAIN*)

10 *LENGTH ERROR* (*NOT SINGLETON*)

15 *DOMAIN ERROR* (*ENCLOSED ARRAY NOT ALLOWED*)

15 *DOMAIN ERROR* (*INCORRECT TYPE*)

15 *DOMAIN ERROR* (*NOT AN INTEGER*)

27 *LIMIT ERROR* (*INTEGER TOO LARGE*)

15 *DOMAIN ERROR* (*PARAMETER OUT OF RANGE*)

⎕*SINK* Discard Output

Type

System Variable

Form

```
⎕SINK ← any-value
⍳ 0 ← ⎕SINK
```

Value Domain

Type	Any
Shape	Any
Depth	Any

Result Domain

Type	Numeric
Rank	1
Shape	0 (empty)
Depth	1 (simple)

Description

⎕*SINK* immediately discards any value that you assign to it. The value of ⎕*SINK* is always `⍳ 0`.

⎕*SINK* is useful inside a user-defined operation; it allows you to discard output that you do not want stored or displayed.

⎕*SINK* can be localized and is saved with the workspace; however, neither operation has any effect.

Possible Errors Generated

None.

⎕*SS* String Search

Type

Dyadic System Function

Form

Boolean ← *pattern-string* ⎕SS *target-string*

Left Argument Domain

Type	Character
Shape	Vector domain
Depth	0 or 1 (simple)

Right Argument Domain

Type	Character
Shape	Vector domain
Depth	0 or 1 (simple)

Result Domain

Type	Boolean
Rank	1
Shape	⍴ , *target-string*
Depth	1 (simple)

Description

⎕*SS* searches the right argument for every appearance of the character string specified in the left argument. This allows you to determine where a substring begins in the searched string.

The result is a Boolean vector equal to the length of the ravel of the right argument. The function places a 1 in any position corresponding to the start of the specified string. For example:

```
      'ISSI' ⎕SS 'MISSISSIPPI'
0 1 0 0 1 0 0 0 0 0 0
      ⎕←MONTHS←4 11⍴'JAN FEB MARAPR MAY JUNJUL AUG SEPOCT NOV DEC'
JAN FEB MAR
APR MAY JUN
JUL AUG SEP
OCT NOV DEC
      4 11 ⍴('AUG' ⎕SS ,MONTHS)
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
```

Possible Errors Generated

9 *RANK ERROR (NOT VECTOR DOMAIN)*

15 *DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)*

15 *DOMAIN ERROR (INCORRECT TYPE)*

⎕*STOP* Suspending Operation Execution

Type

Ambivalent System Function (monadic form is query)

Form

line-numbers ← ⎕STOP *function-names*
success/fail ← *line-numbers* ⎕STOP *function-name*

Monadic Argument Domain

Type	Character
Shape	Vector domain or one-row matrix
Depth	0 or 1 (simple)

Dyadic Left Argument Domain

Type	Near-integer
Shape	Vector domain
Depth	0 or 1 (simple)

Dyadic Right Argument Domain

Type	Character
Shape	Matrix domain
Depth	0 or 1 (simple)

Result Domain

Type	Boolean (dyadic) or integer (monadic)
Rank	1
Shape	Vector
Depth	1 (simple)

Description

⎕*STOP* allows you to suspend the execution of user-defined operations at specified lines. ⎕*STOP* is a useful debugging tool; you can use it to execute a portion of a user-defined operation, then to stop execution temporarily and examine the operation's environment, including the values of its local variables. You can also stop execution at line 1 of an external routine.

The external image containing the external routine must be linked with the /SHARE and /DEBUG qualifieres; APL causes VMS DEBUG to set a breakpoint at line 1 (the routine entry point).

For dyadic ⎕*STOP*, the right argument identifies the operations you want to suspend. Each row should be the name of a valid, unlocked, user-defined operation or an external routine.

The left argument specifies where you want to suspend the operations by naming the lines on which stop bits are to be set; the line numbers do not have to be in order. Negative line numbers and line numbers that do not appear in the operation are ignored.

Note that line [0] can be stopped; APL suspends execution immediately before returning to the caller, thus enabling you to examine the operation's environment after it has finished executing.

The following example sets a stop bit at lines [5], [25], and [55] of the user-defined operations *CALC* and *AVER*:

```
      5 25 55 ⎕STOP 2 4⍴'CALCAVER'
1 1
```

Thus, if you run *CALC* or *AVER*, execution is suspended before line [5], and APL displays the operation name and the line number. Execution can be resumed if you type a branch to line [5], but it is suspended again at line [25], and so on.

The result of dyadic ⎕*STOP* is a Boolean vector that indicates whether stop bits were set for the objects named. A 1 in the position corresponding to the name in the right argument indicates that the stop bits were successfully set; a 0 indicates that stop bits were not set.

To clear all the stop bits associated with an object or objects, use ⎕STOP with an empty left argument, as follows:

```
      (⍳0) ⎕STOP 2 4⍴'CALCAVER'
1 1
```

If you modify an operation with)EDIT, ⎕FX, or ⎕MAP, you clear any stop bits set with ⎕STOP. (However, you can view an operation with)EDIT and not clear the stop bits as long as you do not perform any modifications.) If you edit an operation with the ∆ editor, stop bits remain on existing lines (provided they are not modified) even if the lines are renumbered when the operation is closed.

When operation execution is suspended because a stop bit was set for the line, APL signals STOPSET. Thus, you can trap stop bits with ⎕TRAP.

In its monadic form, ⎕STOP returns the line numbers (in ascending order) on which stop bits have been set for a specified operation. The right argument must contain only one row, which identifies the name of the operation. In the case of a stop bit that is set on an external routine, ⎕STOP returns a one-item vector with a value of 1 (the only allowable line). For example:

```
      ⎕STOP 'CALC'
5 25 55
```

If the argument is empty or contains a value other than the well-formed name of an unlocked operation, APL returns an empty Boolean vector.

For more details about ⎕STOP, see Chapter 3 in the *VAX APL User's Guide.*

Possible Errors Generated

Dyadic Form

9 RANK ERROR (NOT MATRIX DOMAIN)

9 RANK ERROR (NOT VECTOR DOMAIN)

15 DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

15 DOMAIN ERROR (INCORRECT TYPE)

15 DOMAIN ERROR (NOT AN INTEGER)

27 LIMIT ERROR (INTEGER TOO LARGE)

Monadic Form

```
9  RANK ERROR (NOT MATRIX DOMAIN)

10  LENGTH ERROR

15  DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

15  DOMAIN ERROR (INCORRECT TYPE)
```

⎕*TERSE* Terse Error Messages

Type

System Variable

Form

⎕*TERSE* ← *terse-verbose*
integer-scalar ← ⎕*TERSE*

Value Domain

Type	Near-integer
Shape	Singleton
Depth	0 or 1 (simple)
Value	0 or 1
Default	0

Result Domain

Type	Integer
Rank	0
Shape	⍳ 0 (scalar)
Depth	0 (simple scalar)

Description

Each APL error message (see Appendix A) consists of a primary error message (for example, *VALUE ERROR* or *DOMAIN ERROR*) and perhaps a secondary error message. The secondary message provides more information about why the error occurred.

⎕*TERSE* determines whether or not secondary error messages are output. When ⎕*TERSE* is 0, secondary error messages are printed; when it is 1, they are not printed. The default is 0.

Note that APL always puts secondary error messages into ⎕*ERROR*, regardless of the value of ⎕*TERSE*.

The value of ⎕*TERSE* is saved when you save the active workspace and can be localized in user-defined operations.

Possible Errors Generated

9 *RANK ERROR (NOT VECTOR DOMAIN)*

10 *LENGTH ERROR (NOT SINGLETON)*

15 *DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)*

15 *DOMAIN ERROR (INCORRECT TYPE)*

15 *DOMAIN ERROR (NOT AN INTEGER)*

15 *DOMAIN ERROR (SYSTEM VARIABLE VALUE MAY ONLY BE* 0 *OR* 1)

27 *LIMIT ERROR (INTEGER TOO LARGE)*

⎕*TIMELIMIT* User Response Time Limit

Type

System Variable

Form

⎕*TIMELIMIT* ← *seconds*
integer-scalar ← ⎕*TIMELIMIT*

Value Domain

Type	Near-integer
Shape	Singleton
Depth	0 or 1 (simple)
Value	¯1 to 255
Default	0 (unlimited response time)

Result Domain

Type	Integer
Rank	0
Shape	⍳0 (scalar)
Depth	0 (simple scalar)

Description

⎕*TIMELIMIT* limits the amount of time allowed for responses to quote quad (⍞) or quad del (⍔) input requests. Note that you cannot set ⎕*TIMELIMIT* on a VT220, VT240, VT320, VT330, VT340, or DECterm terminal designator.

If, in responding to input requests, you exceed ⎕*TIMELIMIT* seconds between characters, APL accepts only the characters you typed before you ran out of time, and appends a <CR><LF> to them. You can use the ⎕*TIMEOUT* system variable to check whether the time limit expired.

In the following example, the user sets the time limit to 5 seconds, and then supplies a value for *A* before the time limit expires. However, the user does not finish entering a value for *B* before time expires, so APL accepts what was typed before time ran out. The six spaces after the last character (an angle bracket) are APL's input prompt, and the comment (⍝ 6 *SPACES IS PROMPT*) is terminal input.

```
      ⎕TIMELIMIT←5
      A←⍞
YOU HAVE FIVE SECONDS
      A
YOU HAVE FIVE SECONDS
      B←⍞
NOW STOP TYPING BUT NO <RETURN>
                          ⍝TIMELIMIT WAS EXCEEDED ON PREVIOUS LINE
      B
NOW STOP TYPING BUT NO <RETURN>
```

A negative argument (¯1) makes APL check for type-ahead input, that is, data that was placed in the input buffer before the quote quad or quad del input request was made. You can use this feature to help you determine whether anything was typed after the time limit expired. For example:

```
      ∇TIME
[1]   ⎕TIMELIMIT←5
[2]   ⎕←'RESPOND WITHIN 5 SECONDS'
[3]   A←⍞
[4]   ⎕←'TIMED OUT'  ⋄ ⎕SINK←⎕DL 5    ⍝⎕DL DELAYS EXECUTION
[5]   ⎕TIMELIMIT←¯1
[6]   B←⍞
[7]   A,' BEFORE TIMED OUT AND ',B,' AFTER'
[8]   ∇
      TIME
RESPOND WITHIN 5 SECONDS
I STOPPED TYPING
      TIMED OUT
THIS IS TYPEAHEAD
I STOPPED TYPING BEFORE TIMED OUT AND THIS IS TYPEAHEAD AFTER
```

Because ⎕*TIMELIMIT* was set to ¯1 on line [5], the quote quad request on line [6] captured the input that was typed after time expired on the response to the first quote quad input request (line [3]), but before the second input request (line [6]) was made.

You may also want to set ⎕*TIMELIMIT* to ¯1 if you have written a function that checks periodically for a response to a poll or prompt. For example:

```
      ∇POLL;START
[1]   ⎕TIMELIMIT←¯1
[2]   ⍞←'WHEN READY TYPE THE NUMBER 1 '
[3]   ⎕SINK←⎕DL 5           ⍝⎕DL DELAYS EXECUTION 5 SECONDS
[4]   START←⍞
[5]   →(START ≡ '1')/7
[6]   →3
[7]   ⍝EXECUTION BEGINS HERE
[8]   ∇
      POLL
WHEN READY TYPE THE NUMBER 1 1
```

The function *POLL* displays a message telling the user to respond with the number 1 when ready. Until the user enters 1, *POLL* loops between operation lines [3] and [6]; thus, *POLL* delays for 5 seconds, then checks whether the user typed 1 during the delay and, if not, branches back for another 5-second delay. When the user enters 1, control passes to line [7] and operation execution continues.

The value of ⎕*TIMELIMIT* is saved when you save the active workspace and can be localized in user-defined operations.

Possible Errors Generated

9 *RANK ERROR (NOT VECTOR DOMAIN)*

10 *LENGTH ERROR (NOT SINGLETON)*

15 *DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)*

15 *DOMAIN ERROR (INCORRECT TYPE)*

15 *DOMAIN ERROR (NOT AN INTEGER)*

15 *DOMAIN ERROR (PARAMETER OUT OF RANGE)*

15 *DOMAIN ERROR (TIMEOUT READ UNSUPPORTED FOR CURRENT VALUE OF QUAD TT)*

27 *LIMIT ERROR (INTEGER TOO LARGE)*

⎕*TIMEOUT* Time Limit Report

Type

System Variable

Form

⎕*TIMEOUT* ← *0-or-1*
integer-scalar ← ⎕*TIMEOUT*

Value Domain

Type	Near-integer
Shape	Singleton
Depth	0 or 1 (simple)
Value	0 or 1
Default	0

Result Domain

Type	Integer
Rank	0
Shape	⍳ 0 (scalar)
Depth	0 (simple scalar)

Description

⎕*TIMEOUT* queries the system to see whether response time expired for a previously executed quote quad (⍞) or quad del (⍔) input request. (⎕*TIMEOUT* is set implicitly by the system when a timeout occurs, but can also be set by the user.) Its value is a Boolean scalar\ 1 means that time ran out, 0 means that it) did not. The amount of time the user has to respond to quote quad or del quad input requests is determined by the ⎕*TIMELIMIT* system variable. For example:

```
      ⎕TIMELIMIT←5
      A←⍞
YOU HAVE FIVE SECONDS
      ⎕TIMEOUT                    ⍝DID NOT RUN OUT OF TIME
0
      A←⍞
NOW STOP TYPING BUT NO <RETURN>
      ⎕TIMEOUT                        ⍝TIMELIMIT EXCEEDED
1
```

You may set ⎕*TIMEOUT* to 0 or 1. APL changes the value of ⎕*TIMEOUT* only when you type one of the following:

- Quote quad input from the terminal (⍞)
- Quad del input from the terminal (⍔)

The value of ⎕*TIMEOUT* is saved when you save the active workspace and can be localized in user-defined operations.

Possible Errors Generated

9 *RANK ERROR* (*NOT VECTOR DOMAIN*)

10 *LENGTH ERROR* (*NOT SINGLETON*)

15 *DOMAIN ERROR* (*ENCLOSED ARRAY NOT ALLOWED*)

15 *DOMAIN ERROR* (*INCORRECT TYPE*)

15 *DOMAIN ERROR* (*NOT AN INTEGER*)

15 *DOMAIN ERROR* (*SYSTEM VARIABLE VALUE MAY ONLY BE* 0 *OR* 1)

27 *LIMIT ERROR* (*INTEGER TOO LARGE*)

⎕*TLE* Terminal Line Editing Characteristics

Type

System Variable (session)

Form

⎕*TLE* ← 0-*or*-1
current-value ← ⎕*TLE*

Value Domain

Type	Near-Integer
Shape	Singleton
Depth	0 or 1 (simple)
Value	0 or 1
Default	Determined when APL is invoked

Result Domain

Type	Integer
Rank	0
Shape	⍳0 (scalar)
Depth	0 (simple scalar)

Description

⎕*TLE* controls the terminal line editing attribute. You can assign a 0 or a 1 to ⎕*TLE*. By default, ⎕*TLE* inherits the line editing status that is in effect when APL is started. Note that ⎕*TLE* is a session variable; that is, its value is not saved with the workspace and ⎕*TLE* is not reset by the execution of a)*CLEAR* command (see Chapter 3).

⎕*TLE*	**Equivalent DCL Command**
0	$SET TERMINAL/NOLINE_EDITING
1	$SET TERMINAL/LINE_EDITING

APL determines the default value for ⎕*TLE* depending on your terminal designator. For LA, VT102, GIGI, KEY, BIT, HDS201, and HDS221 (terminals that form overstruck characters with the Backspace key), the default is 0. For VT220, VT240, VT320, VT330, VT340, DECterm and VS (terminals that form overstruck characters with the Compose key or Ctrl/D), the default is 1. In all other cases (TTY for example), the default is the same as the current VMS setting when APL is invoked.

Possible Errors Generated

9 *RANK ERROR* (*NOT VECTOR DOMAIN*)

10 *LENGTH ERROR* (*NOT SINGLETON*)

15 *DOMAIN ERROR* (*ENCLOSED ARRAY NOT ALLOWED*)

15 *DOMAIN ERROR* (*INCORRECT TYPE*)

15 *DOMAIN ERROR* (*NOT AN INTEGER*)

15 *DOMAIN ERROR* (*SYSTEM VARIABLE VALUE MAY ONLY BE* 0 *OR* 1)

27 *LIMIT ERROR* (*INTEGER TOO LARGE*)

⎕*TRACE* Monitoring Operation Execution

Type

Ambivalent System Function

Form

line-numbers ← ⎕*TRACE function-name*
success/fail ← *line-numbers* ⎕*TRACE function-name*

Monadic Argument Domain

Type	Character
Shape	Vector domain or 1-row matrix
Depth	0 or 1 (simple)

Dyadic Left Argument Domain

Type	Near-integer
Shape	Vector domain
Depth	0 or 1 (simple)

Dyadic Right Argument Domain

Type	Character
Shape	Matrix domain
Depth	0 or 1 (simple)

Result Domain

Type	Integer (dyadic) or Boolean (monadic)
Rank	1
Shape	Vector
Depth	1 (simple)

Description

⎕*TRACE* is a debugging tool that allows you to obtain intermediate results of operation execution.

In the dyadic form, the right argument identifies the operations that you want to trace. Each row should be the name of a valid, unlocked, user-defined operation. You can also specify the name of an external routine.

The left argument specifies the line numbers you want to trace. The line numbers do not have to be in order. Negative line numbers and line numbers that do not appear in the operation are ignored. When you trace line 0 of an operation, APL displays the result returned by the operation before the operation exits. For external routines, you can specify only line 0.

When you execute a line of an operation that has the trace bit set, APL displays the following information:

- The name of the operation
- The line number being traced (always 0 for external routines)
- The final value returned by the statement, provided that the value is not an enclosed array, in which case APL displays a message indicating an enclosed value

When the statement traced is not the first statement on the line, APL also displays the statement number.

The result of dyadic ⎕*TRACE* is a Boolean vector that indicates whether trace bits were set for the operations named. A 1 in the position corresponding to the name in the right argument indicates that the trace bits were successfully set; a 0 indicates that trace bits were not set.

To clear all the trace bits associated with an object or objects, use ⎕*TRACE* with an empty left argument, as follows:

```
      (⍳0) ⎕TRACE 2 4⍴'CALCAVER'
1 1
```

If you modify an operation with)*EDIT*, ⎕*FX*, or ⎕*MAP*, you clear any trace bits set with ⎕*TRACE*. (However, you can view an operation with)*EDIT* and not clear the trace bits as long as you do not perform any modifications.) If you edit an operation with the ∆ editor, trace bits remain on existing lines (provided they are not modified), even if the lines are renumbered when the operation is closed.

In the monadic form, ⎕*TRACE* returns the line numbers (in ascending order) on which trace bits are set for a specified operation. The right argument must contain only one row, which identifies the name of the operation. In the case of a trace bit that is set on an external routine, ⎕*TRACE* returns a one-item vector with a value of 0 (the only allowable line). The result indicates the lines that exist and have trace bits set on them. For example:

```
      ⎕TRACE 'CALC' ⍝LINES 5, 25, AND 55 ARE TRACED
5 25 55
```

If the argument is empty or contains a value other than the well-formed name of an unlocked operation, APL returns an empty Boolean vector.

For more details about ⎕*TRACE*, see the *VAX APL User's Guide*.

Possible Errors Generated

Dyadic Form

9 *RANK ERROR (NOT MATRIX DOMAIN)*

9 *RANK ERROR (NOT VECTOR DOMAIN)*

15 *DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)*

15 *DOMAIN ERROR (INCORRECT TYPE)*

15 *DOMAIN ERROR (NOT AN INTEGER)*

27 *LIMIT ERROR (INTEGER TOO LARGE)*

Monadic Form

9 *RANK ERROR (NOT MATRIX DOMAIN)*

10 *LENGTH ERROR*

15 *DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)*

15 *DOMAIN ERROR (INCORRECT TYPE)*

⎕*TRAP* Trap Expression

Type

System Variable

Form

⎕*TRAP* ← *terminal-type*
current-value ← ⎕*TRAP*

Value Domain

Type	Character
Shape	Vector Domain
Depth	0 or 1 (simple)
Default	' '

Result Domain

Type	Character
Rank	1
Shape	Vector
Depth	1 (simple)

Description

⎕*TRAP* allows you to override a system response with a user-defined response. The value of ⎕*TRAP* is a character vector representing an APL expression. This expression is executed (in ⍎ fashion) when any of the events listed in Appendix A are signaled during the execution of a user-defined operation. This includes an attention signal, an abort input signal, or a stop bit (⎕*STOP*). For information on error handling, see the *VAX APL User's Guide*.

You can set ⎕*TRAP* as a global variable or localize it in an operation. When an error occurs during execution of a user-defined operation, APL searches for the most local ⎕*TRAP*. If ⎕*TRAP* is set to anything other than the empty vector (the default value), APL executes it in the envirnoment of the operation where the error occurred. For example:

```
      ∇ R;⎕TRAP
[1]   ⎕TRAP← '→L'
[2]   A←5
[3]   B←0
[4]   C←A÷B
[5]               ⍝DIVISION BY 0 IS DOMAIN ERROR
[6]   L: 'TRAPPED ERROR, THEN CONTINUED'
[7]   'EXECUTED LAST LINE'  ∇
      R
TRAPPED ERROR, THEN CONTINUED
EXECUTED LAST LINE
      )SI
      ⎕ERROR
 15 DOMAIN ERROR (DIVISION BY ZERO)
R[4]  C←A÷B
       ∧
```

The following example shows what happens when ⎕TRAP is not set:

```
      ∇G
[1]   A←5
[2]   B←0
[3]   C←A÷B ∇
      G
 15 DOMAIN ERROR (DIVISION BY ZERO)
G[3]  C←A÷B
       ∧
      )SI
G[3] *
```

If execution of an operation's ⎕TRAP expression does not transfer control to a new statement, the operation becomes suspended. If such an operation is a locked operation, APL cuts back the state indicator to the first unlocked operation and then signals *DOMAIN ERROR* (*UNSUCCESSFUL TRAP IN LOCKED FUNCTION*).

Because a ⎕TRAP expression can call an operation, you may want to localize ⎕TRAP in the called operation and set ⎕TRAP to '' to avoid unwanted loops.

Possible Errors Generated

9 *RANK ERROR* (*NOT VECTOR DOMAIN*)

15 *DOMAIN ERROR* (*ENCLOSED ARRAY NOT ALLOWED*)

15 *DOMAIN ERROR* (*INCORRECT TYPE*)

15 *DOMAIN ERROR* (*UNSUCCESSFUL TRAP IN LOCKED FUNCTION*)

⎕*TS* Time Stamp

Type

Niladic System Function

Form

current-time/date ← ⎕*TS*

Result Domain

Type	Integer
Rank	1
Shape	7
Depth	1 (simple)

Description

⎕*TS* (time stamp) returns a vector (in base 10 format) representing the current time and date. This vector is known as a time stamp and contains the current year, month, day, hour, minute, second, and millisecond.

For example:

```
      ⎕TS     ⍝21-NOV-90 11:31:55.134
1990 11 21 11 31 55 134
```

Possible Errors Generated

None.

⎕*TT* Terminal Type

Type

System Variable (session)

Form

⎕*TT* ← *terminal-type*
integer-scalar ← ⎕*TT*

Value Domain

Type	Near-integer
Shape	Singleton
Depth	0 or 1 (simple)
Value	0 through 19
Default	Determined when APL is invoked.

Result Domain

Type	Integer
Rank	0
Shape	⍳ 0 (scalar)
Depth	0 (simple scalar)

Description

⎕*TT* (terminal type) indicates the type of terminal being used for the current APL session. When you invoke APL, you specify the terminal type in an initialization stream or in response to the terminal designator prompt.

⎕*TT* allows you to change the terminal type during an APL session. The following table shows the possible values and meanings for ⎕*TT*:

⎕*TT* **Values**

Value	Meaning
1	APL COMPOSITE terminal
2	TTY-type terminal
3	Digital VK100 (GIGI) terminal (key-paired)
4	Digital LA-type terminal (key-paired)
5	APL/ASCII key-paired terminal
6	APL/ASCII bit-paired terminal
7	Digital VAXstation running VWS(composite)
8	Digital VT102 (key-paired)
9	Digital VT220 (key-paired)
10	Digital VT240 (key-paired)
11	Tektronix 4013 terminal (key-paired)
12	Tektronix 4015 terminal (key-paired)
13	HDSAVT (key-paired)
14	HDS201 (key-paired)
15	HDS221 (key-paired)
16	Digital VT320 (key-paired)
17	Digital VT330 (key-paired)
18	Digital VT340 (key-paired)
19	Digital VAXstation running DECwindows (key-paired)

You can query for the current ⎕*TT* value by entering ⎕*TT* without assigning a value. APL responds with the current value. For example:

```
$apl/term=decterm /silent=all
      ⎕TT
19
```

If you specify APL as your terminal designator when you first invoke APL, ⎕*TT* is set to `5`. You can change the value of ⎕*TT* by assigning it a valid terminal type. For example:

```
      ⍳10                  ⍝TERMINAL DESIGNATOR IS APL
1 2 3 4 5 6 7 8 9 10
      ⎕TT ← 2
      .io10 "The terminal type is now TTY
1 2 3 4 5 6 7 8 9 10
```

If you change the value of ⎕*TT*, APL may send an escape sequence to the terminal to change its character set. This escape sequence is the same as the one that is sent after you identify the terminal when invoking APL.

If the value of ⎕*TT* is currently `9`, `10`, `16`, `17`, `18` or `19` and you leave the APL environment temporarily (with a)*DO*,)*EDIT*, or)*PUSH* command), you should be careful about changing your terminal type while at the operating system level. If you return to APL with a different terminal type, or if the font files for the APL character set are not available, APL signals an error. (You also get an error if you restricted access to those files while at the DCL level.)

Note that ⎕*TT* is a session variable; that is, its value is not saved with the workspace and ⎕*TT* is not reset by the execution of a)*CLEAR* command (see Chapter 3). However, it can be localized in user-defined operations.

Possible Errors Generated

```
9  RANK ERROR (NOT VECTOR DOMAIN)

10  LENGTH ERROR (NOT SINGLETON)

15  DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

15  DOMAIN ERROR (ERROR ACTIVATING IMAGE)

15  DOMAIN ERROR (FONT FILE COULD NOT BE OPENED)

15  DOMAIN ERROR (INCORRECT TYPE)

15  DOMAIN ERROR (NEGATIVE INTEGER NOT ALLOWED)

15  DOMAIN ERROR (NOT AN INTEGER)

15  DOMAIN ERROR (PARAMETER OUT OF RANGE)

27  LIMIT ERROR (INTEGER TOO LARGE)
```

⎕*UL* User Load

Type

Niladic System Function

Form

pid ← ⎕*UL*

Result Domain

Type	Integer
Rank	0
Shape	⍳ 0 (scalar)
Depth	0 (simple scalar)

Description

The result of ⎕*UL* (user load) represents the user's process identification number (PID). VMS assigns you a unique PID each time you log in to the system. If you log off and then log back in to VMS, your PID (and thus, your ⎕*UL* value) probably will have changed. For example:

```
      ⎕UL
93
```

You can use the following expression to convert a PID from decimal, as it appears in the APL environment, to hexadecimal, as it appears in the DCL environment:

```
      '01234567890ABCDEF'[⎕IO+(8⍴16)⊤⎕UL]
00000055
```

Possible Errors Generated

None.

⎕*VERSION* Interpreter and Workspace Version

Type

Niladic System Function

Form

version-info ← ⎕*VERSION*

Result Domain

Type	Character
Rank	1
Shape	Vector
Depth	1 (simple)

Description

⎕*VERSION* returns a two-row character vector, with each row followed by a <CR><LF>. The first row identifies the version of the interpreter under which the current workspace was saved; the second row identifies the version of the interpreter that is currently running. The display is in the form *lv.u-edit*, where *l* is the support letter, *v* is the version number, *u* is the update number, and *edit* is the edit number. For example:

```
      ⎕VERSION
V3.2-834
V3.2-834
```

Possible Errors Generated

None.

⎕ *VI* Validating Input

Type

Monadic System Function

Form

valid/invalid-number ← ⎕ *VI character-vector*

Argument Domain

Type	Character
Shape	Vector domain
Depth	1 (simple)

Result Domain

Type	Boolean
Rank	1
Shape	Vector
Depth	1 (simple)

Implicit Arguments

⎕*NG* (controls negative number recognition)

Description

⎕*VI* determines the valid numbers in a character argument. ⎕*VI* examines fields in the argument that are delimited by one or more spaces, tabs, or a carriage return (optionally followed by a line feed), and returns a Boolean vector that contains a 1 in each position corresponding to a field containing a valid number, and a 0 in each position corresponding to an invalid number. If the argument is empty, ⎕*VI* returns an empty array.

⎕*VI* is often used in conjunction with ⎕*FI* and the compression function (see Section 1.3.1) to select the valid numbers from a character string; ⎕*VI* produces the left argument of the compression function, and ⎕*FI* produces the right argument. For example:

```
   A←'1.5 3 A ¯5 3.. 1.0E +1 ¯3'
   ⎕VI A
1 1 0 1 0 0 0 1
   ⎕FI A
1.5 3 0 ¯5 0 0 0 ¯3
   (⎕VI A)/⎕FI A
1.5 3 ¯5 ¯3
```

Recognition of negative numbers in the ⎕VI argument depends upon the value of the system variable ⎕NG. If ⎕NG equals 1 (the default), negative numbers in the ⎕VI argument must begin with the high minus sign (¯) to be recognized. If ⎕NG equals 0, numbers preceded by a minus sign (-) are recognized as negative numbers. If ⎕NG equals 2, negative numbers are preceded by an APL "+" symbol. (APL "+" prints as an ASCII "-" so ⎕NG_2 can be used to handle negative numbers in strings that will be read or written in ASCII.) For example:

```
   ⎕NG←1            ⍝¯ MEANS NEGATIVE
   X←'66 G ¯7 +9 -4'
   ⎕VI X
1 0 1 0 0
   ⎕NG←0            ⍝- MEANS NEGATIVE
   ⎕VI X
1 0 0 0 1
   ⎕NG←2            ⍝+ MEANS NEGATIVE
   ⎕VI X
1 0 0 1 0
```

Note that the case where ⎕NG is 0 may be useful when you use APL to interpret data created by other languages, specifically those that do not use the high minus sign (¯).

Possible Errors Generated

```
9  RANK ERROR (NOT VECTOR DOMAIN)

15  DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)

15  DOMAIN ERROR (INCORRECT TYPE)
```

⎕*VPC* Vector Process Control

Type

System Variable (session)

Form

⎕*VPC* ← *session variable*
integer-scalar ← ⎕*VPC*

Value Domain

Type	Non-negative near-integer
Shape	Singleton
Depth	0 or 1 (simple)
Default	Determined when APL is invoked.

Result Domain

Type	Integer
Rank	0
Shape	⍳ 0 (scalar)
Depth	0 (simple scalar)

Description

⎕*VPC* determines the threshold at which vector processing is used. A value of 0 indicates that vector processing is not used; a value of 1 indicates that the vector processing is always used.

When you invoke a session, APL determines whether a vector processor is available. ⎕*VPC* is set to 0 if no vector processor is available; it is set to the non-negative, near-integer default if a vector processor is available. You can also specify the value for ⎕*VPC* when you invoke an APL session with the qualifier /⟦NO⟧VECTOR=n (see the section on initializaiton streams in the *VAX APL User's Guide*).

If a vector processor is not present, setting ⎕*VPC* to a non-zero value results in the use of the vector processer emulator. The vector processer emulator may be useful for testing. The vector processer emulator should not be used for applications because APL's performance with the emulator is usually poorer than APL's performance with the scalar processor.

⎕*VPC* is a session variable; that is, its value is not saved with the workspace, and ⎕*VPC* is not reset by the execution of a)*CLEAR* command (see Chapter 3). However, it can be localized in user-defined operations.

You can query for the current ⎕*VPC* value by entering ⎕*VPC* without assigning a value. APL responds with the current value. For example:

```
      ⎕VPC
30
```

Controlled testing can help you identify the threshold at which the increased overhead of running the vector processer is compensated for by increased performance.

Possible Errors Generated

```
9  RANK ERROR (NOT SINGLETON)

15  DOMAIN ERROR (NEGATIVE NUMBER NOT ALLOWED)

15  DOMAIN ERROR (NOT AN INTEGER)

15  DOMAIN ERROR (VECTOR PROCESSOR NOT AVAILABLE)
```

⎕*VR* Visual Representation

Type

Monadic System Function

Form

⎕*VR* {*value* | *object-name*}

Argument Domain

Type	Any
Shape	Any
Depth	Any

Result Domain

Type	Character
Rank	Any
Shape	Any
Depth	Any

Implicit Arguments

⎕*DC* (Controls display of enclosed arrays)
⎕*PP* (Controls precision of numeric constants)

Description

⎕*VR* returns a character array of the visual representation of an APL object. The argument is either a numeric array of any rank or a character array with its shape in the vector domain.

If the argument is numeric, ⎕*VR* works the same way as monadic ⍕: the numeric array is formatted into a character array that looks as the numeric array would appear when displayed by APL (which is dependent on the ⎕*PP* setting).

If the argument is type character, its value must represent the name of an APL object. If the character argument represents a variable or label, the result is the same as for numeric arguments: APL formats the variable or label value as a character array, making it look as it would appear when displayed by APL.

If the character argument represents a user-defined operation, ⎕ *VR* returns a character vector that is similar to the canonical representation of the operation. Specifically, the visual representation of a user-defined operation, *F*, is the operation definition displayed by the editor command ∆*F*[⎕]∆. The result starts and ends with a ∆ character, and each line begins with a line number surrounded by square brackets and ends with a <CR><LF>.

If the character argument represents a derived function such as an operand of a user-defined operator, ⎕ *VR* returns a character representation of it. For example, if the operand is plus (+), then ⎕ *VR* returns the string '+'; if the operand is outer product (+.⋆), then ⎕ *VR* returns '+.⋆'; and if the operand is ⎕ *CR*, then ⎕ *VR* returns '⎕*CR*'.

If the argument to ⎕ *VR* is empty, the result is an empty character vector.

Examples:

```
                          ⍝CREATE FUNCTION FRTH
      ∇Z ← A FRTH B; X
[1]   L: Z ← A + B ∇
      )FNS
FRTH
      ⎕←V←2 3⍴⍳6 ⍝CREATE AND DISPLAY V
1 2 3
4 5 6
      ⍴V
2 3
      ⎕←A←⎕VR 'V'
1 2 3
4 5 6
      ⍴A
2 5
      ⎕←B←⎕VR 'FRTH'
      ∇Z ← A FRTH B; X
[1]    L: Z ← A + B
      ∇
      ⍴B
54
```

Possible Errors Generated

9 *RANK ERROR (NOT VECTOR DOMAIN)*

15 *DOMAIN ERROR*

15 *DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)*

15 *DOMAIN ERROR (OPERATION LOCKED)*

⎕*WA* Workspace Available

Type

Niladic System Function

Form

available-space ← ⎕*WA*

Result Domain

Type	Integer
Rank	0
Shape	⍳ 0 (scalar)
Depth	0 (simple scalar)

Description

⎕*WA* (workspace available) returns an integer scalar representing an estimate of the amount of available storage space, in bytes, in the active workspace. This value allows you to determine the maximum amount by which your workspace can increase. APL obtains the value by subtracting the current data-segment size from the maximum data-segment size which is the current)*MAXCORE* setting (see Chapter 3). Thus, the value returned by ⎕*WA* may be greater than the amount of available memory or the amount of memory allocated by APL. For example:

```
      ⎕WA
516980
```

Possible Errors Generated

None.

⎕*WAIT* Limiting Time on Read Functions

Type

Ambivalent System Function (dyadic form is quiet)

Form

current-timelimit ← ⎕*WAIT* *chans*
⍳0 ← *timelimit* ⎕*WAIT* *chan*

Monadic Argument Domain

Type	Near-integer
Shape	Vector domain
Depth	0 or 1 (simple)
Value	¯999 through 999 (but not 0)

Dyadic Left Argument Domain

Type	Near-integer
Shape	Singleton
Depth	0 or 1 (simple)
Value	¯1 through 255 (seconds)

Dyadic Right Argument Domain

Type	Near-integer
Shape	Singleton
Depth	0 or 1 (simple)
Value	¯999 through 999 (but not 0)

Result Domain

Type	Integer
Rank	1
Shape	Vector
Depth	1 (simple)

Description

Dyadic ⎕*WAIT* specifies the amount of time you want APL to wait when it tries to read a shared record that is locked by another user.

When you set a waiting period, APL waits even if you specified the */READONLY*:*NOLOCKS* qualifier when you assigned the file to a channel with ⎕*ASS* (*NOLOCKS* normally causes a read to happen without waiting).

The left argument (*timelimit*) determines the time limit; it has the following meanings:

Value of Time Limit	Meaning
¯1	Don't wait, return immediately
0	Wait indefinitely (this is the default)
n	Wait for *n* seconds

Monadic ⎕*WAIT* queries the system for the current time limits associated with individual channel numbers.

For each channel number in the argument, monadic ⎕*WAIT* returns a value between ¯1 and 255 that can have the following meanings:

Value Returned	Current Time Limit
¯1	Don't wait
0	Wait indefinitely
n	Wait for *n* seconds

⎕*WAIT* is described in the *VAX APL User's Guide* along with other file I/O information.

Possible Errors Generated

Monadic Form

1 *FILE NOT FOUND* (*FILE NOT FOUND*)

10 *LENGTH ERROR* (*NOT VECTOR DOMAIN*)

15 *DOMAIN ERROR* (*ENCLOSED HETEROGENEOUS ARRAY NOT ALLOWED*)

15 *DOMAIN ERROR* (*INCORRECT TYPE*)

15 *DOMAIN ERROR* (*INVALID CHANNEL NUMBER*)

15 *DOMAIN ERROR* (*NOT AN INTEGER*)

27 *LIMIT ERROR* (*INTEGER TOO LARGE*)

Dyadic Form

1 *FILE NOT FOUND* (*FILE NOT FOUND*)

10 *LENGTH ERROR* (*NOT SINGLETON*)

15 *DOMAIN ERROR* (*INCORRECT TYPE*)

15 *DOMAIN ERROR* (*ENCLOSED HETEROGENEOUS ARRAY NOT ALLOWED*)

15 *DOMAIN ERROR* (*NOT AN INTEGER*)

15 *DOMAIN ERROR* (*PARAMETER OUT OF RANGE*)

15 *DOMAIN ERROR* (*INVALID CHANNEL NUMBER*)

15 *DOMAIN ERROR* (*CHANNEL NOT ASSIGNED*)

27 *LIMIT ERROR* (*INTEGER TOO LARGE*)

33 *IO ERROR* (*TIMEOUT PERIOD EXPIRED*)

⎕*WATCH* Monitoring Variable Changes

Type

Ambivalent System Function

Form

currentmode ← ⎕WATCH *object-name*
success/failure mode-number ⎕WATCH *object-names*

Monadic Argument Domain

Type	Character
Shape	Vector domain or one-row matrix
Depth	0 or 1 (simple)

Dyadic Left Argument Domain

Type	Near-integer
Shape	Vector domain
Depth	0 or 1 (simple)

Dyadic Right Argument Domain

Type	Character
Shape	Matrix domain
Depth	0 or 1 (simple)

Result Domain

Type	Integer
Rank	1 or 2
Shape	Vector or matrix
Depth	1 (simple)

Description

⎕WATCH is a debugging tool that allows you to monitor changes in the value of APL objects. APL either displays or signals information when a watched object is referenced or modified. You can set watch modes on variable and label names and most system variables and niladic system functions. You cannot set watch modes on nonniladic system functions, ill-formed identifiers, user-defined operations, or group names. A watch mode remains in effect when the watched object occurs in a locked operation; the watch bit is not reset, as is the case with ⎕STOP and ⎕TRACE.

A modification occurs any time a value is explicitly assigned to a variable (niladic system functions and labels cannot be modified). A reference occurs any time an object is referenced for its value.

In display mode, APL sends information to the current output and continues execution of the operation where the reference or modification occurred. In signal mode, APL signals an error (trappable with ⎕TRAP) and suspends the operation.

Implicit in the use of the ⎕WATCH command are the ⎕L and ⎕R system variables. Each time a modification occurs on a watched object, APL reassigns the values of these variables: ⎕L contains the name of the changed object; ⎕R contains the previous value of the changed object. The object contains the new value.

Dyadic ⎕WATCH enables watchpoints on one or more objects. The right argument specifies the objects you want to watch. Each row contains the name of one niladic system function or one variable (which can be a defined or undefined variable), a system variable, or a label. You cannot watch the following system variables: ⎕R, ⎕L, or ⎕ERROR. If the right argument is empty, or if it contains the name of an object that cannot be watched, APL returns an empty Boolean vector.

The left argument determines the watch mode: either display or signal mode. In addition, you can specify the watch mode for either modifications or references.

There are six watch modes:

Mode	Meaning
2	Signal if modified
3	Display if modified
4	Signal if referenced
5	Display if referenced
6	Signal if modified or referenced
7	Display if modified or referenced

The result of dyadic ⎕*WATCH* is a Boolean vector indicating whether the watch mode was set for the specified variables. Each position in the vector corresponds to a row of the right argument. A 1 indicates that the watch mode was successfully set; a 0 indicates that the watch mode was not set. For example:

```
      B ← 3 5 ρ 'BABELSABLECABLE'
      B
BABEL
SABLE
CABLE
      4 ⎕WATCH B
1 1 1
```

To clear the watch mode associated with an object or objects, use dyadic ⎕*WATCH* with an empty left argument. The result is a Boolean vector indicating whether the watch mode was turned off. Each position in the vector corresponds to a row of the right argument. A 1 indicates the watch mode was successfully turned off; a 0 indicates the watch mode was not turned off. For example:

```
      '' ⎕WATCH 3 5 ρ 'BABELSABLECABLE'
1 1 1
```

In signal mode, you can trap a reference or modification with ⎕*TRAP*. APL signals a primary message and one of three secondary messages. The signals have the following form:

```
113 WATCH POINT ACTIVATED (VARIABLE HAS BEEN REFERENCED)

113 WATCH POINT ACTIVATED (VARIABLE HAS BEEN MODIFIED)
```

```
113 WATCH POINT ACTIVATED (VARIABLE HAS BEEN MODIFIED BY INDEX)
```

In display mode, APL displays information in different forms depending on whether the event is a reference or a modification. For a reference, the display form is as follows:

function-name [*line* ◇ *statement*] *object-name*
OLD NAME CLASS: *nc* *SHAPE*: *rho-vector*
value

For a modification, the display form is as follows:

function-name [*line* ◇ *statement*] *object-name*
OLD NAME CLASS: *nc* *SHAPE*: *rho-vector*
value
NEW NAME CLASS: *nc* *SHAPE*: *rho-vector*
value

Note that if *value* is an enclosed array, APL does not display the value. Instead, APL displays a message indicating that *value* is enclosed.

If the operation that contains the reference or modification is locked, APL displays the name of the object with a protected del (⍫) symbol (there is no line number).

Monadic ⎕*WATCH* returns information indicating the current watch mode for the object specified in the right argument. The right argument must have at most one row, which means you must query for the watch mode one object at a time.

The result of monadic ⎕*WATCH* is a one-element integer vector (unless the argument is empty, in which case APL returns ⍳ 0). There are seven possible values which indicate the following watch modes:

Mode	Meaning
1	Object not being watched
2	Signal if modified
3	Display if modified
4	Signal if referenced

Mode	Meaning
5	Display if referenced
6	Signal if modified or referenced
7	Display if modified or referenced

When ⎕*WATCH* is set in mode 2 or 6 on a name that is the left argument of a strand assignment, the signal is delayed until APL has completed all the assignments. If there is more than one watched name in the left argument, APL only signals information on the last (rightmost) one. (For more information, see the strand assignment sections in Chapter 1.)

Some events do not activate a ⎕*WATCH* signal or display (immediate mode events, for example). In addition, a watchpoint is not activated when the following occur:

- A variable is used as an output parameter in a call to an external function.
- An object becomes shadowed by an operation invocation.
- An object becomes unshadowed by an operation termination.
- A variable is included in the argument of any of the following commands: ⎕*EX*,)*ERASE*,)*COPY*,)*PCOPY*, ⎕*QCO*, and ⎕*QPC*.

When you enable ⎕*WATCH*, the watchpoint is set on the most local version of the specified objects. When a watched object becomes shadowed, APL saves the current ⎕*WATCH* definition and restores it when the object becomes unshadowed. Labels are always local to an operation and are defined only when the operation is being executed. To watch the referencing of a label, you must enable ⎕*WATCH* within the context of the operation (either inside the operation or in immediate mode while the operation is suspended or pending). (The *VAX APL User's Guide* has more information on debugging operations.) An example of this behavior follows:

```
    X←5              ⍝DEFINE GLOBAL X
    3 ⎕WATCH 'X'     ⍝ENABLE WATCHPOINT ON X
1
    ∇SHAG1;X         ⍝DEFINE LOCAL X
[1] ⎕←X←1           ⍝SIMPLE ASSIGNMENT
[2] ∇
    SHAG1            ⍝CALL TO SHAG1, LOCAL X SHADOWS GLOBAL X
1
                     ⍝NO ⎕WATCH EVENT OCCURS
```

This same behavior occurs when a local variable becomes shadowed by a more local variable:

```
      ∇SHAG1;X              ⍝RE-WRITE SHAG1
[1]   ⎕←X←1
[2]   3 ⎕WATCH 'X'         ⍝ENABLE WATCHPOINT ON LOCAL X
[3]   SHAG2                ⍝CALL SHAG2
[4]   ∇
      ∇ SHAG2;X             ⍝ADD ANOTHER LOCAL X
[1]   ⎕←X←2                ⍝SIMPLE ASSIGNMENT
[2]   ∇
      SHAG1                 ⍝EXECUTE SHAG1
1            (Value of local X in  SHAG1)
2            ( SHAG2 local X shadows  SHAG1 local X)
```

Examples:

```
      F ← 5                 ⍝DEFINE GLOBAL F
      3 ⎕WATCH 'F'          ⍝ENABLE DISPLAY IF MODIFIED
1
      ∇FOO                  ⍝DEFINE FUNCTION FOO
[1]   F ← ⍳9               ⍝SIMPLE ASSIGNMENT MODIFIES F
[2]   F                    ⍝LINE 2 REFERENCES F
[3]   ∇
      FOO                        ⍝EXECUTE FOO
FOO[1] F
NAME CLASS: 2 SHAPE:
5
NEW NAME CLASS: 2 SHAPE: 9
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
      5 ⎕WATCH 'F'          ⍝REPLACE PREVIOUS WATCHPOINT AND
1
                            ⍝ENABLE DISPLAY IF REFERENCED
      FOO
FOO[2] F
OLD NAME CLASS: 2 SHAPE: 9
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
      2 ⎕WATCH 'F'      ⍝REPLACE PREVIOUS WATCHPOINT AND
1
                            ⍝ENABLE SIGNAL IF MODIFIED
      F ← 5      ⍝NO SIGNAL FOR INTERACTIVE ASSIGNMENT
      FOO
113 WATCH POINT ACTIVATED (VARIABLE HAS BEEN MODIFIED)
FOO[1]      F ← ⍳9               ⍝SIMPLE ASSIGNMENT MODIFIES F
       ^
      )SI
FOO[1] *
      ⎕L
F
      F
1 2 3 4 5 6 7 8 9
      ⎕R
5
```

Possible Errors Generated

Monadic Form

9 *RANK ERROR* (*NOT MATRIX DOMAIN*)

10 *LENGTH ERROR*

15 *DOMAIN ERROR* (*ENCLOSED ARRAY NOT ALLOWED*)

15 *DOMAIN ERROR* (*INCORRECT TYPE*)

Dyadic Form

9 *RANK ERROR* (*NOT MATRIX DOMAIN*)

9 *RANK ERROR* (*NOT VECTOR DOMAIN*)

10 *LENGTH ERROR*

15 *DOMAIN ERROR* (*ENCLOSED ARRAY NOT ALLOWED*)

15 *DOMAIN ERROR* (*INCORRECT TYPE*)

15 *DOMAIN ERROR* (*INVALID WATCH MODE*)

15 *DOMAIN ERROR* (*NOT AN INTEGER*)

27 *LIMIT ERROR* (*INTEGER TOO LARGE*)

⎕XQ Executing Expressions

Type

Monadic System Function (sometimes quiet)

Form

result ← ⎕XQ *apl-expression*

Argument Domain

Type	Any
Shape	Any
Depth	Any

Result Domain

Type	Any
Rank	Any
Shape	Any
Depth	Any

Description

⎕XQ executes the expression represented by its argument as if that expression were entered in immediate mode or included in a user-defined operation. For example, the expressions ⍳5 and ⎕XQ'⍳5' return the same result:

```
      ⍳5
1 2 3 4 5
      ⎕XQ'⍳5'
1 2 3 4 5
```

The argument can be a numeric (any shape) or a character array (vector domain). If the argument to ⎕XQ is numeric, APL returns the value of the argument):

```
      ⎕XQ 55
55
```

APL treats a <CR><LF> in the argument as a statement separator as if it were input from the terminal, so multiple lines are allowed. The ⎕*XQ* function always returns a value: either the value of the last statement evaluated in its argument, or, if the last statement has no value, an empty array. For example:

```
      D←⎕XQ'5+4
      3+2
      6'
9
5
      D
6
```

The ⎕*XQ* function can execute system commands or invoke the function editor. For example:

```
      ⎕XQ'∇R←FOO
A←2
B←3
R←A+B
∇
'
      FOO
5
      ⎕XQ')FNS'
FOO
      ⎕XQ')VARS'
A B
      ⎕XQ')ERASE ',⎕XQ')VARS'      ⍝ERASE ALL VARIABLES
      A
  11 VALUE ERROR
      A
       ^
```

Pendent ⎕*XQ* functions are indicated by the '⎕*XQ*' characters in the state indicator. For example:

```
      ⎕XQ ')SI'
⎕XQ
```

Note that quiet functions are still quiet when executed, provided that ⎕*XQ* is the leftmost function in the statement:

```
      ⎕XQ'Z←1'
      ,⎕XQ'Z←1'
1
```

When the argument is empty and numeric, the result is an empty numeric vector (⎕*XQ* ⍳0 ←→ ⍳0). For example:

```
      ⎕XQ ⍳0
```

When the argument is empty and character, the result is an empty character vector ('' ≡ ⎕*XQ* '') if a value is required by the expression. For example:

```
      ⎕XQ ''                  ⍝QUIET, NO OUTPUT
      A←⎕XQ ''
```

If APL encounters an error while evaluating the ⎕*XQ* function's argument, it does not signal an error; instead, it stops evaluating the argument and returns an empty array whose shape is 0 *n*, where *n* is a number indicating the error that was encountered (see Appendix A for a complete description of all APL errors). The complete text of the error message is placed in ⎕*ERROR*. For example:

```
      E←⎕XQ'5+5
3+2,
4'
10
      E
                    (APL outputs a blank line)
      ρE
0 7
      ⎕ERROR
   7 ⎕XQ SYNTAX ERROR (RIGHT ARGUMENT TO FUNCTION MISSING)
      3+2,
        ^
```

If you enter the attention signal while the ⎕*XQ* function is executing, APL stops executing the argument to ⎕*XQ*, and ⎕*XQ* returns an empty array whose shape is 0 18. Then, the normal order of execution continues.

The ⎕*XQ* function cannot execute a branch statement; if one is entered, the branch is not taken, and the result of the ⎕*XQ* function, if needed, is an empty vector. Thus, ⎕*XQ* never alters the flow of control within an operation.

Possible Errors Generated

```
9  RANK ERROR (NOT VECTOR DOMAIN)
```

3 VAX APL System Commands

VAX APL provides a wide variety of system commands that allow you to examine or change the state of the system. For example, you can do the following:

- Clear, save, or name the active workspace
- Load or copy a workspace from a secondary storage device
- List workspace, variable, and user-defined operation names
- Display the status of user-defined operations and local variables in the workspace
- Set the index origin, the maximum number of significant digits, and the output line width
- End an APL session

Unlike system functions and system variables, system commands are not considered part of the APL language.

System commands are particularly useful in function-definition mode, because they are executed immediately instead of being executed when the operation is executed. Thus, you can change the APL environment without exiting from the function editor. For example, if the terminal print width is set at 50, and you display an operation line that is 60 characters long, you could use the `)WIDTH` system command to change the print width so that the operation line is displayed on one line.

Note that by giving system commands as arguments to the `⎕XQ` system function, you can use the commands within user-defined operations, rather than having them execute immediately.

3.1 System Command Form

APL system commands begin with a right parenthesis, as shown in the following form:

)*white command-name* ⟦ *space white arguments* ⟧

The right parenthesis is a required part of the system command name. *white* is optional white space; that is, zero or more spaces or tabs. *Space* is a required blank space. *Arguments* may or may not be allowed. See the individual descriptions of the system commands for more details.

You can abbreviate a command name to its shortest unique form. Some system commands take required or optional parameters; when you include them, you must separate the individual items of the parameter list with at least one space or tab. If a system command that takes no arguments is followed by nonwhite space, or if an argument is invalid, APL signals *INCORRECT PARAMETER*.

The following examples show the form of several system commands:

```
)CONTINUE
)CONT
)CONTIN
)SAVE MYWORK
)WS40 A B C VAR6 N
```

The first three examples invoke the same system command,)*CONTINUE*; note that the first four letters of each of the command names are the same. In the fourth example, *MYWORK* is an argument to the)*SAVE* system command. The fifth example shows a)*COPY* command that takes a series of arguments.

3.2 System Command Categories

There are two broad categories of system commands:

- Query commands obtain information
- Action commands that change the state of a workspace or the operating environment

The action commands can be further categorized into the following logical groups, which are described in the following sections:

- Query/change commands find out about or change the state of the environment
- APL action commands manipulate APL objects in a workspace
- System action commands terminate or interrupt an APL session

- Workspace manipulation commands manipulate workspaces

There are some system commands that specifically affect APL I/O; they are described in detail in the *VAX APL User's Guide*. Table 3–1 summarizes the system commands.

3.2.1 Query System Commands

The query system commands return information about the current state of the session, the active workspace, or the APL system or environment. The query system commands follow:

```
)FNS
)GRP
)GRPS
)NMS
)OPS
)VARS
)SI
)SINL
)SIS
)VERSION
)CHARGE
)OWNER
)LIB
)HELP
```

3.2.2 Query/Change System Commands

The query/change system commands are both action and query commands; that is, they can return information about the present state of the APL environment, or they can be used with an optional parameter to change the state of the environment. The `)ORIGIN` command, for instance, can either return or change the index origin setting.

In the following example, `)ORIGIN` is used first as an action command; it sets the index origin to `0` and reports that the previous setting was `1`. Then, `)ORIGIN` is used as a query command; it reports that the current setting of the index origin is `0`:

```
      )ORIGIN 0
WAS 1
      )ORIGIN
0
```

Each of the query/change commands can be thought of as displaying, or changing and displaying, a system variable. There are two types of system variables: workspace and session.

Workspace variables are associated with a particular workspace; that is, they are saved and loaded with the workspace. The system commands associated with the workspace variables are as follows:

```
)WSID ⟦wsname⟧ ⟦/PASSWORD⟦: ⟦pw⟧⟧⟧
)PASSWORD ⟦⟦/PASSWORD ⟦ : ⟦pw⟧⟧⟧ | pw⟧
)ORIGIN ⟦n⟧
)DIGITS ⟦n⟧
)WIDTH ⟦n⟧
```

Session variables are associated with the current APL session; they do not change when the current workspace is changed, and they cannot be saved with a workspace. The system commands associated with the session variables follow:

```
)MAXCORE ⟦n⟧
)MINCORE ⟦n⟧
```

Note that the system variables ⎕GAG, ⎕TLE, ⎕TT and ⎕VPC are also session variables.

3.2.3 APL Action System Commands

The APL action system commands cause some action to take effect within the current workspace. The APL action system commands are as follows:

```
)CLEAR
)EDIT object-name /qualifiers
)ERASE ⟦/FNS⟧ ⟦/GRPS⟧ ⟦/OPS⟧ ⟦/VARS⟧ list
)GROUP group-name ⟦group-member-list⟧
)SIC
)STEP ⟦n⟧ ⟦/SILENT⟧ ⟦/INTO | /OVER⟧
```

3.2.4 System Commands that Initiate System Action

This section describes the system commands that terminate or interrupt an APL session or initiate some other program.

You can exit from APL in a variety of ways:

- Returning to the DCL command level
- Terminating the APL session, optionally returning to the DCL command level
- Terminating the session and saving the active workspace

- Interrupting the session and running other programs while at the DCL command level, eventually returning to APL

The following system commands are in this category of commands that initiate system action:

```
)MON
)OFF
)CONTINUE
)PUSH
)ATTACH
)DO
```

3.2.5 Workspace Manipulation System Commands

The workspace manipulation system commands preserve, make active, and delete workspaces. They also copy objects from workspaces to the currently active workspace. The workspace manipulation system commands are as follows:

```
)LOAD
)XLOAD
)SAVE
)COPY
)PCOPY
)DROP
```

3.3 System Command Reference

The following sections describe the APL system commands in alphabetical order. Each description indicates the general category of the command: query, query/change, apl action, system action, or workspace manipulation.

Table 3–1 lists all the system commands and gives a brief description of their uses.

Table 3–1 System Commands

Command	Meaning
)*ATTACH*	Temporarily suspends the APL session and returns control to a specified process
)*CHARGE*	Displays a record of activity for the current APL session

(continued on next page)

Table 3–1 (Cont.) System Commands

Command	Meaning
)CLEAR	Replaces active workspace with clear workspace
)CONTINUE	Saves active workspace and exits APL
)COPY	Copies global objects from another workspace
)DIGITS	Displays or changes the number of significant digits to be displayed
)DO	Executes a VMS command; returns output to APL
)DROP	Deletes workspaces or files from a directory-structured device
)EDIT	Edits a global object with the VAXTPU editor
)ERASE	Erases the named global object from the current workspace
)FNS	Displays an alphabetical list of global function names
)GROUP	Collects named objects into a group
)GRP	Lists members of a group
)GRPS	Displays an alphabetical list of group names
)HELP	Displays information about APL features
)INPUT	Diverts input to a device other than your terminal
)LIB	Displays names of workspaces or files on a directory-structured device
)LOAD	Retrieves a workspace from secondary storage
)MAXCORE	Displays or changes the setting for maximum workspace size
)MINCORE	Displays or changes the setting for minimum workspace size
)MON	Returns you to operating system command level
)NMS	Displays all names in the symbol table
)OFF	Ends current APL session
)OPS	Displays an alphabetical list of global operator names
)ORIGIN	Displays or changes index origin
)OUTPUT	Diverts output from your terminal to another device
)OWNER	Displays information about the creation of the current workspace
)PASSWORD	Displays or changes the workspace password
)PCOPY	Same as)COPY but protects names already in use

(continued on next page)

Table 3–1 (Cont.) System Commands

Command	Meaning
)*PUSH*	Temporarily suspends the APL session, returning control to the operating system
)*SAVE*	Saves a copy of the active workspace
)*SI*	Displays workspace state indicator
)*SIC*	Clears workspace state indicator
)*SINL*	Displays workspace state indicator, local symbols for each user-defined operation, and argument to pending execute functions
)*SIS*	Displays workspace state indicator, currently executing line, and argument to pending execute functions
)*STEP*	Executes lines of a function one at a time
)*VARS*	Displays an alphabetical list of global variables
)*VERSION*	Displays the APL version numbers for the workspace and interpreter
)*WIDTH*	Displays or changes the terminal line width
)*WSID*	Displays or changes workspace name; optionally changes workspace password
)*XLOAD*	Retrieves a workspace from secondary storage without executing ⎕*LX*

)*ATTACH* Interacting with Other Processes

Type

Action System Command

Form

)*ATTACH* {*process-name*}

Qualifiers

/PARENT
Specifies that you want to attach to the first process established in the current job.

Description

Note that you must specify either */PARENT* or the *process-name*, but you cannot specify both.

)*ATTACH* interrupts the APL session and attaches to a process that already exists within your current job. The APL session is not terminated when you use)*ATTACH*. To return to APL, you can use the DCL command ATTACH on the process name of the APL process. When you return to the interrupted APL session, program execution resumes at the point after the execution of the)*ATTACH* command.

Examples:

```
      ⎕GAG ← 2   ⍝BROADCASTS WILL BE DISPLAYED IN APL CHARACTERS
                 ⍝EXECUTE MAIL IN A SUBPROCESS NAMED MAILPROC
      )PUSH/PROCESSNAME=MAILPROC MAIL

MAIL>ATTACH/PARENT
                          ⍝WE ARE BACK IN APL
                          ⍝GO READ THE NEW MAIL
      )ATTACH MAILPROC
 You have 1 new message.

MAIL>read/new
     #1          8-NOV-1999 15:15:15:41
 From: APLVAX::USER2
 To: USER1
 CC:
 Subj: Pizza today?

 Do you want to have pizza for lunch today?
```

```
MAIL>attach/parent
                             ⍝WE ARE BACK IN APL
      )DO SHOW PROCESS/SUBPROCESSES

8-NOV-1990 15:39:04.42   User: USER1     Process ID:   00000067
                         Node: APLVAX    Process name: ⍝USER←2⍝

There are 2 processes in this job:

  ←TWA2:
    USER1←2 (*)
                             ⍝USER1←1 IS THE PROCESS FOR DO)
      )OFF
TWA2:  THURSDAY 8-NOV-1990 15:39:05.01
CONNECTED 00:00:39.53  CPU TIME 00:00:01.09
6 STATEMENTS  1 OPERATIONS
306 PAGE FAULTS  410 BUFFERED IO  125 DIRECT IO
$show process/subprocesses

8-NOV-1990 15:47:58.25   User: USER1   Process ID:    0000005A
                         Node: APLVAX   Process name:  "_TWA4:"

There are 3 processes in this job:

  _TWA4: (*)
    USER1_1
    MAILPROC
```

Possible Errors Generated

22	*INCORRECT PARAMETER (MISSING ARGUMENT)*
22	*INCORRECT PARAMETER (UNRECOGNIZED QUALIFIER KEYWORD)*
114	*ERROR PROCESSING ATTACH (ATTACH REQUEST REFUSED)*
114	*ERROR PROCESSING ATTACH (INVALID LOGICAL NAME)*
114	*ERROR PROCESSING ATTACH (NONEXISTENT PROCESS)*
249	*EXTRANEOUS CHARACTERS AFTER COMMAND*
383	*PARENT QUALIFIER REPEATED*

)*CHARGE* Displaying Accounting Information

Type

Query System Command

Form

```
)CHARGE
```

Description

)*CHARGE* displays a record of activity during the current APL session and includes the following:

- Your terminal identification
- Current time and date
- Length of time connected to APL
- Amount of computer CPU time used inside APL
- Number of APL operations executed
- Number of page faults while inside APL
- Number of buffered I/O and number of direct I/O while inside APL

For example:

```
      )CHARGE
SYS$INPUT:  WEDNESDAY 14-NOV-1990 16:03:22.16
CONNECTED 00:00:01.33  CPU TIME 00:00:00.40
0 STATEMENTS  0 OPERATIONS
160 PAGE FAULTS  20 BUFFERED IO  15 DIRECT IO
```

Possible Errors Generated

```
22 INCORRECT PARAMETER (EXTRANEOUS CHARACTERS AFTER COMMAND)
```

)*CLEAR* Clearing the Active Workspace

Type

APL Action System Command

Form

```
)CLEAR
```

Description

)*CLEAR* empties the active workspace by erasing all variables, groups, and user-defined operations; resetting all workspace variables (but not session variables) to their default values; closing all open files; and clearing the state indicator.

The clear workspace has the following characteristics:

- Contains no user-defined operations, groups, or variables
- Has an index origin (⎕*IO*) of 1
- Has an output line length (⎕*PW*) determined by the operating system width specification
- Has a comparison tolerance value (⎕*CT*) of 1*E*¯15
- Has a random link value (⎕*RL*) of 695197565
- Has an empty character array as the value for ⎕*LX*, ⎕*TRAP*, ⎕*ERROR*, ⎕*L*, and ⎕*R*
- Has the automatic save feature turned off (⎕*AUS* is 0)
- Outputs the negative sign (¯) in TTY mode as .NG (⎕*NG* is 1)
- Displays both primary and secondary error messages when an error occurs (⎕*TERSE* is 0)
- Has a workspace and interpreter version that are the same (the lines returned by ⎕*VERSION* match)
- Displays numbers with ten significant digits (⎕*PP*)
- Has a clear symbol table and state indicator
- Has the name *CLEAR WS*
- Has an empty password

- Requests quad input with the message ⎕: followed by a <CR><LF> and six blanks (⎕*SF*)
- Has a default ⎕*DC* value of `(¯1 1 0 2) ''`, which means that there are no boxes around enclosed arrays

Note that APL gives you a clear workspace when you begin a work session, unless you have a *CONTINUE* workspace in your default device and directory area, or unless you use an initialization stream to specify a workspace to be loaded. Also note that)*CLEAR* clears only the active workspace; it has no effect on workspaces you have saved with the)*SAVE* system command.

When the)*CLEAR* command completes execution, APL displays the message *CLEAR WS*. For example:

```
      )CLEAR
CLEAR WS
```

)*MINC*,)*MAXC*, ⎕*GAG*, ⎕*TLE*, and ⎕*TT* are not affected by)*CLEAR*.

Possible Errors Generated

```
22 INCORRECT PARAMETER (EXTRANEOUS CHARACTERS AFTER COMMAND)
```

)*CONTINUE* Saving the Workspace and Ending the Session

Type

System Action System Command

Form

)*CONTINUE* ⟦*HOLD* | *LOGOUT*⟧

Default in Clear Workspace

HOLD

Description

)*CONTINUE* works the same way as the)*OFF* system command, except that before ending the session)*CONTINUE* saves the active workspace in your default device and directory area under the name *CONTINUE.APL*. If files named *CONTINUE.APL* already exist in your directory, the new *CONTINUE* workspace will have a version number that is one greater than the next most recent version.

The *HOLD* parameter (the default) returns you to DCL command level after ending the APL session. The *LOGOUT* parameter logs you off the system after ending the APL session. The)*CONTINUE* command prints the same message that the)*SAVE* command prints, followed by the same summary information that)*OFF* displays. For example:

```
      )CONTINUE HOLD
WEDNESDAY 28-NOV-1990 16:04:29.46 15 BLKS
SYS$INPUT:  WEDNESDAY 28-NOV-1990 16:04:29.90
CONNECTED 00:00:01.62  CPU TIME 00:00:00.42
0 STATEMENTS  0 OPERATIONS
191 PAGE FAULTS  24 BUFFERED IO  21 DIRECT IO
```

If a *CONTINUE* workspace exists in your default area when you begin an APL session, it is loaded as your active workspace, unless you specify a different workspace in an APL initialization stream, or unless the workspace had a password when it was saved. If the *CONTINUE* workspace is saved with a password, APL signals *WORKSPACE LOCKED* when the APL session begins. You can still load the workspace by executing the command)*LOAD CONTINUE /PASSWORD*:*pw*.

Note that the name of a *CONTINUE* workspace that is loaded is not *CONTINUE*; the name is the one the workspace had when it was saved. The load message displayed when APL is invoked identifies what the name was when the workspace was saved. For example:

```
      )LOAD CONTINUE
SAVED WEDNESDAY 28-NOV-1990 16:04:29.46 15 BLKS WAS EXAMPLE
      )WSID
EXAMPLE
```

Note

APL does not delete your *CONTINUE* workspace after it is loaded. A particular *CONTINUE* workspace in your default area may be loaded as your active workspace each time you invoke APL, not just the first time that you invoke APL after the workspace was created. If you do not want the *CONTINUE* workspace to be loaded, you must explicitly delete it from your default area, or specify a different workspace in an APL initialization stream.

Possible Errors Generated

15 *DOMAIN ERROR(EXTRANEOUS CHARACTERS AFTER COMMAND)*

22 *INCORRECT PARAMETER (UNRECOGNIZED QUALIFIER KEYWORD)*

)COPY Copying Objects from a Workspace

Type

Manipulation System Command

Form

)COPY wsname ⟦*list*⟧

Qualifiers

/PASSWORD⟦**:**⟦***pw***⟧⟧
Specifies the password associated with the stored workspace.

/CHECK
Causes APL to examine the workspace for possible corruption (damage to the internal structure of the workspace). If damage is detected, a message is displayed and APL tries to recover as much information as possible from the workspace and continue the copy. The recovered workspace may be missing APL variables, user-defined operations, individual lines of user-defined operations, and other APL objects that were damaged. The user must determine what named objects have been removed from the workspace.

Description

)COPY retrieves global user-defined operations, global variables, and groups from a stored workspace (*wsname*) and places them into your active workspace. If there is a password associated with the stored workspace, you must include it in the command string.

You can copy all the named objects in a workspace or a subset of them; *list* identifies the specific objects to be copied. When you specify a list of objects, you can use the ⋆ and ÷ wildcards. If you omit the *list* parameter, all user-defined operations, variables, and groups are copied. *)COPY* does not transfer local values for variables and functions, nor does it copy the state indicator, channel assignments, or any system variable such as the print width, index origin, or print precision.

If your active workspace contains objects with the same name as those in the copied workspace, *)COPY* replaces the global (but not the local) values in your active workspace with the copied ones. For example, if *B* is a variable in the active workspace with a global value of 10 and a local value of 5, and the workspace being copied has a variable *B* with a global value of 20, after *)COPY* is executed the active workspace will have a variable *B* with a global value of

20 and a local value of 5. A suspended or pendent operation, or an operation still being defined in the active workspace is not replaced, and an operation being created in the workspace being copied is not copied.

When you copy a group, all members of the group are copied along with their values. However, if a member of a group is itself a group, APL copies only the group name and not the value. Thus, for example, suppose the group GROUP1 has as members the variables A and B, and the group GROUP2. Also suppose that GROUP2 has as members the variables C and D. Then, if you copy GROUP1, you copy the values of A and B, but only the name of GROUP2, not the values of C and D.

The)COPY command displays the same message as the)LOAD command. Note that the size printed in this message is the size (in disk pages) of the active workspace after execution of the)COPY command completes. If the list to be copied contains an object that is not in the specified workspace, APL returns the message NOT FOUND:, followed by a list of the objects (separated by tabs) that were not found. The objects that were found are still copied, however.

The)COPY command performs the same operation as the ⎕QCO system function (see Chapter 2), but ⎕QCO does not display messages to confirm that the copy was successful.

Examples:

```
      )COPY AVER
SAVED WEDNESDAY 28-NOV-1990 16:20:42.14 24 BLKS
      )COPY AVER B
SAVED WEDNESDAY 28-NOV-1990 16:20:42.14 24 BLKS
      )COPY AVER G
SAVED WEDNESDAY 28-NOV-1990 16:20:42.14 24 BLKS
NOT FOUND: G
```

Possible Errors Generated

```
22 INCORRECT PARAMETER

22 INCORRECT PARAMETER(ILL FORMED NAME)

27 LIMIT ERROR(ARGUMENT STRING IS TOO LONG)

83 DAMAGED WORKSPACE HAS BEEN CORRECTED (SOME SYMBOLS MAY HAVE
   BEEN ERASED)
```

)*DIGITS* Output Precision

Type

Query/Change System Command

Form

)*DIGITS* ⟦*n*⟧

Default in Clear Workspace

10

Description

)*DIGITS* displays or changes the value of the print precision system variable (⎕*PP*).

The print precision (*n* in the form) is the number of significant digits displayed in APL floating-point output; it can be an integer from 1 to 16. It does not affect the precision of internal calculations or the display of integers. APL rounds off any number that has more digits than the current setting.

Executing the)*DIGITS* command in change mode has the same effect as assigning a value to the ⎕*PP* system variable (see Chapter 2).

Examples:

```
      )DIGITS
10
      1.23456789123456789
1.234567891
      )DIG 5
WAS 10
      1.23456789123456789
1.2346
      )DIG 2
WAS 5
      1.23456789123456789
1.2
```

Possible Errors Generated

```
22 INCORRECT PARAMETER (EXTRANEOUS CHARACTERS AFTER COMMAND)

22 INCORRECT PARAMETER (ILL FORMED NUMERIC CONSTANT)

22 INCORRECT PARAMETER (PARAMETER OUT OF RANGE)
```

)DO Executing a DCL Command

Type

System Action System Command

Form

)DO *command-string*

Qualifiers

/LOWERCASE
Refers to how you want)DO to translate any output from the execution of the command string. Without this qualifier, the)DO command converts any ASCII lowercase letters to uppercase unless you are using VT102, VT220, VT240, VT320, VT330, VT340, DECterm, VS, or TTY mnemonic mode, or unless the output is the argument to execute (either ⎕XQ or ⍎). Use the /LOWERCASE qualifier if you do not want this conversion to occur.

/NOKEYPAD
Specifies that you do not want the keypad characteristics of the current process to be available to the new subprocess. The default is that the characteristics are available.

/NOLOGICALS
Specifies that you do not want the logical name table from the current process to be available to the new subprocess. The default is that the table is available.

/NOSYMBOLS
Specifies that you do not want the global and local symbol table (defined at the DCL level) from the current process to be available to the new subprocess. The default is that the symbol table is available.

Description

)DO interrupts the APL session and creates a VMS subprocess, putting you at the DCL command level without terminating the APL session. Unlike the)PUSH command,)DO attempts to recover any output resulting from the execution of the command string.

With)DO, you must always include a command string (do not enclose the string in quotation marks); VMS creates a subprocess, executes the command specified, and then returns to APL when the execution completes. For example, the following executes the DCL command SHOW TIME and returns its output to APL:

```
      DOTIME←⎕XQ')DO SHOW TIME'
      DOTIME
  8-NOV-1990 16:25:21
```

The command string must be no longer than 132 characters (after translation to ASCII), not including leading white space (spaces or tabs before the argument begins), but including all other white space within the argument.

Any output written to SYS$OUTPUT or SYS$ERROR is retrieved by APL. See the *VMS DCL Dictionary* for a description of SYS$OUTPUT and SYS$ERROR.

While you are at DCL command level, your terminal is in ASCII rather than APL mode, and your terminal characteristics (such as output line width) revert to the system settings. When you return to APL, the APL character set is restored, and your ⎕PW setting is the same as it was before you executed the)DO command (although the default for ⎕PW changes if you changed your system terminal width). However, other terminal characteristics you may have changed at command level (for example, the ⎕GAG setting, or the ability to input lowercase characters) remain changed.

Possible Errors Generated

```
22 INCORRECT PARAMETER (LOWERCASE QUALIFIER REPEATED)

22 INCORRECT PARAMETER (MISSING ARGUMENT)

22 INCORRECT PARAMETER (NOLOGICALS QUALIFIER REPEATED)

22 INCORRECT PARAMETER (NOSYMBOLS QUALIFIER REPEATED)

22 INCORRECT PARAMETER (NOKEYPAD QUALIFIER REPEATED)

73 SUBPROCESS ERROR (COMMAND BUFFER OVERFLOW
   SHORTEN EXPRESSION OR COMMAND LINE)
```

)DROP Deleting Stored Workspaces or Files

Type

Workspace Manipulation System Command

Form

)DROP *file-spec*

Description

)DROP can delete any file for which you have the necessary protection privileges. You can delete one, several, or all the files on a device and directory.

If you do not include a device and directory, your default device and directory are assumed; however, you must always include a file name, file type, and version number. You may use a wildcard designator to substitute for the version number or for all or part of the file name or file type. In the following example, all files on the default device and directory that begin with the letter *S* are deleted. APL prints the file specification for each file dropped. For example:

```
      )DROP *.LIS;*
÷DELETE-I-FILDEL, DEV1:[APLGRP]DOC.LIS;1 deleted (3 blocks)
÷DELETE-I-FILDEL, DEV1:[APLGRP]SAMPLE.LIS;2 deleted (3 blocks)
÷DELETE-I-FILDEL, DEV1:[APLGRP]SAMPLE.LIS;1 deleted (3 blocks)
÷DELETE-I-TOTAL, 3 files deleted (9 blocks)
```

The maximum length of the *)DROP* command argument is 121 characters (after translation to ASCII), not including leading white space (spaces and tabs before the argument begins), but including all other white space within the argument.

Executing the *)DROP* command is equivalent to executing the DCL command DELETE/LOG. For more details about the DELETE command, see the *VMS DCL Dictionary*.

Possible Errors Generated

```
22 INCORRECT PARAMETER (LINE TOO LONG TO TRANSLATE)
```

```
22 INCORRECT PARAMETER (MISSING ARGUMENT)
```

)*EDIT* Editing with VAXTPU

Type

APL Action System Command

Form

)*EDIT* *objectname*

Qualifiers

⟦*NO*⟧/*COMMAND*
Allows you to specify an initialization file to VAXTPU. The value for *filespec* is a VMS file specification. If you omit the /*COMMAND* qualifier, or if you do not specify a *filespec* value, VAXTPU uses the file specification assigned to the logical name TPUINI as a default.

⟦*NO*⟧/*DISPLAY*
Tells VAXTPU that you are using a support ANSI CRT terminal. This is the default. You should specify /*DISPLAY* only during an interactive session. /*NODISPLAY* tells VAXTPU that your are not using a supported terminal. You should use this qualifier only when you run VAXTPU procedures in batch mode.

/*EXECUTE*⟦***:tpucommand***⟧
Allows you to specify a VAXTPU command string that you want to execute after the editor finishes any command or section files. Note, however, that VAXTPU does not execute the /*EXECUTE* qualifier when either the command or section file contatins a QUIT or EXIT command.

The value for *tpucommand* is a character string containing one or more VAXTPU statements that you want VAXTPU to execute. It should not contain any non-ASCII APL characters or embedded <CR><LF>s. You do not have to place quotation marks around the string, but if quotation marks are necessary for the VAXTPU operation, they must be balanced. The maximum length of the string is 100 characters, and it must be terminated by the end of the line or by another qualifier. If you do not specify *tpucommand*, APL ignores the /*EXECUTE* qualifier.

The /*EXECUTE* qualifier is particularly useful when you run APL and)*EDIT* in batch mode. For example, you can set up an error-checking routine to handle situations where the VAXTPU file is too large to return to the APL environment. If you create a VAXTPU procedure called CHECKMESBUFFER that checks the VAXTPU message buffer for a *WORKSPACE FULL* error, you can call the procedure with the */EXECUTE* qualifier:

```
)EDIT FOO/EXECUTE:CHECKMESBUFFER
```

When you first invoke VAXTPU, the section and command files run, and the */EXECUTE* qualifier calls the CHECKMESBUFFER procedure. The procedure does nothing because there is currently no *WORKSPACE FULL* error message in the VAXTPU message buffer. However, if the editing session ends and the file is too large, APL reinvokes VAXTPU. This time the CHECKMESBUFFER procedure detects the error message and handles it accordingly.

/LC
Determines whether the line numbers of a user-defined operation appear in the VAXTPU editor. This qualifier is useful because it allows you to view the current line numbers associated with the lines of the operation. APL ignores the */LC* qualifier for objects that are not operations.

Once the file is in the editor, the line numbers no longer determine the organization of the lines in the operation. When the file returns to the APL environment, APL assigns new line numbers based on the order that it reads the records from the VAXTPU editor. If you add new lines to the operation, you do not have to include any line numbers.

When you do not specify */LC*, APL generates the canonical representation of the operation in the VAXTPU editor. (The canonical representation does not include line numbers.)

/MODE⟦***:mode***⟧
Allows you to determine the input/output mode for the data moving between the AP and VAXTPU environments. The value for *mode* is the integer 2 or 3, and represents ⍞ and ⍌, respectively. If you omit the */MODE* qualifier, or if you do not specify a *mode* value, APL uses */MODE*:2 as a default.

Note that you must specify *MODE*:3 if you have embedded line feeds in an object and want them to remain intact in the file that VAXTPU returns to APL. However, be aware that ⍌ turns off APL overstrike and TTY mnemonic translation.

/NC⟦***:nc***⟧
Determines the name class of the object you are editing. This qualifier is useful only when you intend to create a new object in the editor. The value for *nc* is an integer that specifies the name class of the new object, which has the following characteristics:

Name Class	Data Type	Shape
2	Character	Vector
3	Function	Not applicable
4	Operator	Not applicable

When you omit the /*NC* qualifier, or when you specify /*NC* without a value, APL uses the name class of the object named by *objectname* as the default. If *objectname* specifies a currently undefined object, APL uses 2 as the default value.

/*NG*⟦**:*ng***⟧
Determines how VAXTPU displays the APL high minus sign (¯), which is used to indicate a negative number. This qualifier only affects the representation of numeric variables; numeric values within operations are not affected.

/*NG* is the equivalent of ⎕*NG* and accepts the same values (0, 1, or 2). When *ng* is 1, negative numbers are preceded by the high minus sign (¯). When *ng* is 0, negative numbers are preceded by the minus sign (-). When *ng* is 2, negative numbers are preceded by the APL plus sign (+). (The setting 2 is used when reading or writing ASCII files; the ASCII minus sign translates to the APL plus sign. See the discussion of ⎕*NG* in the *VAX APL Reference Manual* for more information.)

The default setting for /*NG* is the current setting for ⎕*NG*.

/*PP*⟦**:*pp***⟧
Determines the print precision of noninteger numeric values sent to VAXTPU. /*PP* is the equivalent of ⎕*PP* and accepts the same values (1 to 16). If you omit the /*PP* qualifier, APL uses the current value of ⎕*PP* as the default. If you specify /*PP* but do not include a value, APL uses the maximum allowable value (16).

/*PW*⟦**:*pw***⟧
Determines how APL segments an object for output to the temporary file in the VAXTPU editor. The value for *pw* specifies the maximum number of characters in a single line of output. /*PW* is the equivalent of ⎕*PW* and accepts the same values (35 to 2044). However, the greatest value that VAXTPU will accept from APL is 900. If you specify a /*PW* value greater than 900, APL selects 900 by default. If you do not specify /*PW*, or if you specify /*PW* without a value, then the default setting is ⎕*PW* or 900, whichever is smaller.

VAXTPU truncates any record that has a length greater than 900. To avoid losing data, APL forms records in the following manner when creating the temporary file:

Functions	APL breaks records between each line
Matrices	APL breaks records between each row
Vectors	APL breaks records at each <CR> <LF>

When you enter VAXTPU, some line wrapping may occur, depending on the setting used for */PW*. This could cause unexpected changes in the edited object, and may result in an error when you attempt to end the editing session. To avoid confusion, APL places a warning message in the VAXTPU message buffer as you enter the editing session: LINE WRAP HAS OCCURED. The semantics for line wrapping are as follows:

- If */PW* is not specified, APL wraps records with length >900
- If */PW* is specified, APL wraps records with length >900⌊⎕*PW*
- If */PW=pw* is specified, APL wraps records with length >900⌊ *pw*

⟦*NO*⟧*/SECTION*⟦*:filespec*⟧
Allows you to specify a section file to VAXTPU. The value for *filespec* is a VMS file specification. If you omit the */SECTION* qualifier, or if you do not specify a *filespec*, VAXTPU uses the file specification assigned to the logical name TPUSECINI as a default. If you desire to use the EDT emulation mode, specify */SECTION:EDTSECINI*.

***/TERMINAL*⟦*:termtype*⟧**
Determines the terminal type you want to use during the *)EDIT* session. The values for *termtype*, and the character sets that they represent are as follows:

Terminal Type	Character Set
TTY	*TTY*
KEY	*KEY*
BIT	*BIT*
COMPOSITE	*COMPOSITE*
*VT*102	*BIT*
*VT*220	*COMPOSITE*
*VT*240	*COMPOSITE*
*VT*320	*COMPOSITE*

Terminal Type	Character Set
*VT*330	*COMPOSITE*
*VT*340	*COMPOSITE*
*HDS*201	*COMPOSITE*
*HDS*221	*COMPOSITE*
VS	*COMPOSITE*
DECTERM	*COMPOSITE*

If you omit the /*TERMINAL* qualifier, or if you do not specify the *termtype* value, APL uses the current terminal type as the default.

Description

The)*EDIT* system command allows you to edit global APL objects with the VAXTPU editor. You can edit user-defined operations and variables. You cannot edit enclosed arrays, and you cannot modify an operation that is suspended or pendent. (If you edit a suspended or pendent operation, APL puts an appropriate message in the VAXTPU message buffer, and you must end the VAXTPU session with a QUIT command.)

When you invoke)*EDIT*, APL creates a temporary file containing the object you want to edit and then invokes VAXTPU. When you exit VAXTPU, APL reads the edited file from VAXTPU into the workspace. Note that APL returns you to the VAXTPU editor if an error occurs as the file reenters the workspace.

For more information about VAXTPU, see the *VAX Text Processing Utility Manual.*

Note that)*EDIT* is also described in the *VAX APL User's Guide.*

Possible Errors Generated

```
5  DEFN ERROR (OPERATION SUSPENDED OR PENDENT)

15  DOMAIN ERROR (ERROR ACTIVATING IMAGE)

111  EDIT COMMAND ERROR (xx QUALIFIER REPEATED)

111  EDIT COMMAND ERROR (ARGUMENT TO xx IS OUT OF RANGE)

111  EDIT COMMAND ERROR (BAD ARGUMENT TO xx)
```

```
111 EDIT COMMAND ERROR (EDIT COMMAND UNAVAILABLE DURING FUNCTION
  DEFINITION)

111 EDIT COMMAND ERROR (ENCLOSED ARRAY NOT ALLOWED)

111 EDIT COMMAND ERROR (EXECUTE QUALIFIER ARGUMENT IS TOO LONG)

111 EDIT COMMAND ERROR (ILL FORMED NUMERIC CONSTANT)

111 EDIT COMMAND ERROR (ILL FORMED NUMERIC MATRIX)

111 EDIT COMMAND ERROR (ILLEGAL NAME CLASS)

111 EDIT COMMAND ERROR (INCORRECT PARAMETER)

111 EDIT COMMAND ERROR (MISSING ARGUMENT)

111 EDIT COMMAND ERROR (OPERATION LOCKED)

111 EDIT COMMAND ERROR (UNRECOGNIZED QUALIFIER KEYWORD)

111 EDIT COMMAND ERROR (VOLUME TOO LARGE)
```

)*ERASE* Erasing Global Names

Type

APL Action System Command

Form

)*ERASE* *list*

Qualifiers

/FNS
Limits the name class of the objects to functions.

/VARS
Limits the name class of the objects to variables.

/GRPS
Limits the name class of the objects to groups.

/OPS
Limits the name class of the objects to operations.

Description

)*ERASE* deletes the APL objects named in the list; it undefines global user-defined functions and operations, erases global variables, and disperses groups and erases their members.

When you specify *list*, you can use the ⋆ and ÷ wildcards. You can use the */FNS*, */GRPS*, */OPS*, and */VARS* qualifiers in conjunction with wildcards to limit the name class of the objects being erased.

You cannot erase pendent or suspended operations, nor can you erase labels or other local names. If you are inside the ∆ editor, you cannot erase the function being edited.

If a member of the named group is itself a group, the group name is erased, but the members of the subgroup remain intact. For example:

```
      )CLEAR
CLEAR WS
      )LOAD TRIG_CIRCLE
SAVED THURSDAY 8-NOV-1990 15:08:07.52 12 BLKS
      )FNS
ARC COS     DIAM    RADIUS  SIN     TAN
                          ⍝TRIG CONTAINS THE GROUP CIRCLE
      )GRP TRIG
SIN COS     TAN     CIRCLE
                          ⍝SHOW MEMBERS OF GROUP CIRCLE
      )GRP CIRCLE
ARC RADIUS  DIAM
      )ERASE TRIG
      )GRP TRIG
 22 INCORRECT PARAMETER (NOT A GROUP)
      )GRP TRIG
               ^
                          ⍝ERASING TRIG ERASED GROUPNAME CIRCLE
      )GRP CIRCLE
 22 INCORRECT PARAMETER (NOT A GROUP)
      )GRP CIRCLE
                 ^
                          ⍝MEMBERS OF CIRCLE NOT ERASED
      )FNS
ARC DIAM    RADIUS
```

If a specified object cannot be erased, either because such an operation is illegal or because the object is undefined, the following message is displayed:

`NOT ERASED: list of objects`

The objects are separated by tabs. There is no message when `)ERASE` is successful.

Note that `)ERASE` leaves a slot in the symbol table for the erased name (symbol). Although you erase a symbol, the slot in the symbol table still exists. If you reuse a name that was in the symbol table, APL places it in the same slot where it was before. If you do a `)COPY` of the workspace into a `CLEAR WS`, APL rebuilds the workspace, thus erasing the slot as well as the symbol.

Examples:

```
      ⎕←B←2 3 4
2 3 4
      )ERASE B
      B
 11 VALUE ERROR
      B
      ∧
      ∇ R←F
[1]                          ⍝F CANNOT BE ERASED
[2]   )ERASE F
NOT ERASED: F
[2]    ∇
      )FNS G⋆
G2  GI
      )VARS G⋆
G3  G4
                          ⍝USE QUALIFIER TO LIMIT WILDCARD
      )ERASE /FNS G⋆
                          ⍝FUNCTIONS G1 AND G2 ARE GONE
      )FNS
                          ⍝VARIABLES G3 AND G4 STILL DEFINED
      )VARS
G3  G4
```

Possible Errors Generated

```
22 INCORRECT PARAMETER (INVALID KEYWORD OR QUALIFIER)

22 INCORRECT PARAMETER (ILL FORMED NAME)

27 LIMIT ERROR (ARGUMENT STRING IS TOO LONG)
```

)FNS Displaying a List of Functions

Type

Query System Command

Form

)FNS ⟦*start-string*⟦*stop-string*⟧⟧

Qualifiers

/WSID: ***wsname***⟦*/PASSWORD*: ***pw***⟧
Allows you to specify a nonactive workspace. If the nonactive workspace was saved with a password, you must also specify the */PASSWORD* qualifier.

Description

)FNS displays a list of the global names used as user-defined function names in a workspace. By default, APL displays the list from the currently active workspace.

The optional string parameters identify starting and stopping points for the list. When you specify the string parameters, you can use the ⋆ and ÷ wildcards. The objects are listed in ⎕*AV* order, separated by tabs. Each output line in the list begins in column one.

Note that the wildcard determines the *start-string*. There is no wildcard for the *stop-string*.

If you use *)FNS* with no parameters, APL displays all the global function names in the workspace:

```
      )LOAD FNS
SAVED THURSDAY  8-NOV-1990 17:12:11.52 41 BLKS
      )FNS
ALPH    HILB    INVRS   INVT   LSQ
```

If you include just one argument, APL uses Z̲ as the default for the *stop-string*:

```
      )LOAD OPERS
SAVED THURSDAY 8-NOV-1990 18:06:12.76 12 BLKS
      )FNS IN
INVRS       INVT    LSQ
      )FNS INV INV
INVRS       INVT
```

To obtain a list of all user-defined function names that begin with a given prefix, use the prefix for both arguments or use a wildcard:

```
      )LOAD OPERS
SAVED THURSDAY  8-NOV-1990 18:06:12.76 12 BLKS
      )FNS INV*
INVRS        INVT
```

Possible Errors Generated

1 *FILE NOT FOUND (FILE NOT FOUND)*

22 *INCORRECT PARAMETER (EXTRANEOUS CHARACTERS AFTER COMMAND)*

22 *INCORRECT PARAMETER (FILE SPECIFICATION IS MISSING)*

22 *INCORRECT PARAMETER (INVALID KEYWORD OR QUALIFIER)*

22 *INCORRECT PARAMETER (NOT A LETTER)*

57 *FILE DOES NOT CONTAIN A WORKSPACE*

)GROUP Defining or Dispersing a Group

Type

APL Action System Command

Form

)GROUP group-name ⟦*group-member-list*⟧

Description

)GROUP collects APL objects together under a single name. The objects can be variables, user-defined operations, and other group names. When you specify the objects, you can use the ⋆ and ÷ wildcards.

The *)GROUP* command is often used with the *)COPY* and *)PCOPY* commands. After collecting a set of operations and variables under one group name, you can specify the name in a *)COPY* or *)PCOPY* command to copy the entire collection at one time.

In the following example, the functions and variables named *INT*, *FUTVAL*, and *PRESVAL* are collected into a group named *FINANCIAL*:

```
)GROUP FINANCIAL INT FUTVAL PRESVAL
```

To add a new member to an existing group, you must list the group name as an item in the member list. Thus, the variable *TAX* is added to the group named *FINANCIAL* as follows:

```
)GROUP FINANCIAL TAX FINANCIAL
```

To disperse a group, specify the group name without a group member list. The group name will no longer be defined, but the individual members of the group will still exist under their original names. The following command disperses the group *FINANCIAL*:

```
)GROUP FINANCIAL
```

A group name is always global; you cannot localize it. For example:

```
      A←1
      B←2
      ∇F;C
[1]   R←⎕XQ 'GROUP C A B')
[2]   ∇
      F
      )GRPS
C
```

Here, the)*GROUP* command executed inside the function *F* created a global group name *C*, even though *C* was included in the function's local symbol list.

Possible Errors Generated

```
22 INCORRECT PARAMETER (EXTRANEOUS CHARACTERS AFTER COMMAND)

22 INCORRECT PARAMETER (ILL FORMED NAME)

22 INCORRECT PARAMETER (MISSING ARGUMENT)

24 NOT GROUPED, NAME IN USE
```

)*GRP* Displaying the Members of a Group

Type

Query System Command

Form

)*GRP* *group-name*

Qualifiers

/*WSID*: ***wsname*** ⟦/*PASSWORD*: ***pw***⟧
Specifies a nonactive workspace. If the nonactive workspace is saved with a password, you must also specify the /*PASSWORD* qualifier.

Description

)*GRP* displays the names of the objects associated with the group name. The names are listed in the order in which they are entered into the group and are separated by tabs.

For example:

```
      )GROUP APLGRP LEE PETER STAN DAVE ERIC CHIP CHRIS SHOTA
      )GRP APLGRP
LEE PETER   STAN    DAVE    ERIC    CHIP    CHRIS   SHOTA
```

Possible Errors Generated

1 *FILE NOT FOUND (FILE NOT FOUND)*

22 *INCORRECT PARAMETER (EXTRANEOUS CHARACTERS AFTER COMMAND)*

22 *INCORRECT PARAMETER (ILL FORMED NAME)*

22 *INCORRECT PARAMETER (INVALID KEYWORD OR QUALIFIER)*

22 *INCORRECT PARAMETER (MISSING ARGUMENT)*

22 *INCORRECT PARAMETER (NOT A GROUP)*

57 *FILE DOES NOT CONTAIN A WORKSPACE*

)GRPS Displaying a List of Groups

Type

Query System Command

Form

)GRPS ⟦*start-string*⟦*stop-string*⟧⟧

Qualifiers

/WSID: ***wsname***⟦*/PASSWORD*: ***pw***⟧
Specifies a nonactive workspace. If the nonactive workspace was saved with a password, you must also specify the */PASSWORD* qualifier.

Description

)GRPS displays a list of group names in a workspace. By default, APL displays the list from the currently active workspace.

When you specify the string parameters, you can use the ⋆ and ÷ wildcards. The names are listed in ⎕*AV* order, separated by tabs. Each output line in the list begins in column 1.

Note that the wildcard determines the *start-string*. There is no wildcard for the *stop-string*.

If you use *)GRPS* with no parameters, APL displays all the group names in the workspace. For example:

```
      )GRPS
ALPH    HILB    INVRS   INVT   LSQ
```

If you include just one argument, APL uses Z̲ as the default for the second string:

```
      )GRPS IN
INVRS INVT LSQ
      )GRPS INV INV
INVRS INVT
```

To get a list of all group names that begin with a given prefix, use the prefix for both arguments or use a wildcard.

```
      )GRPS INV⋆
INVRS INVT
```

Possible Errors Generated

1 *FILE NOT FOUND* (*FILE NOT FOUND*)

22 *INCORRECT PARAMETER* (*EXTRANEOUS CHARACTERS AFTER COMMAND*)

22 *INCORRECT PARAMETER* (*INVALID KEYWORD OR QUALIFIER*)

22 *INCORRECT PARAMETER* (*NOT A LETTER*)

57 *FILE DOES NOT CONTAIN A WORKSPACE*

)*HELP* Obtaining Help on the VAX APL Language

Type

Query System Command

Form

)*HELP* ⟦*string*⟧

Qualifiers

/*LIBRARY*: ***filespec***
Specifies a Help library other than the default APL Help library. This feature allows you to write your own Help libraries and reference them through the APL)*HELP* facility. If you want to make your help library the default (and thus avoid specifying the /*LIBRARY* qualifier each time you invoke)*HELP*), you can define the logical name APL$HELP: as the value for *filespec*.

The /*LIBRARY* qualifier must follow directly after the)*HELP* command, and you must specify the colon or equal sign and the VMS file specification. If you specify a file that does not exist, APL signals *ERROR PROCESSING HELP* (*ERROR OPENING AS INPUT*).

Description

)*HELP* provides controlled access to the APL Help facility via the VMS Help librarian.

The APL Help library is a file associated with the VMS logical name APL$HELP:. You can define that logical name if you want your own help library to be the default. If APL$HELP: is not defined, VAX APL looks for a file named SYS$HELP:VAXAPL.HLB, which is placed on your system during installation.

The APL Help Library contains the actual text of the help topics and is organized into multiple levels. For example, ⎕*ASS* is a secondary level topic under *QUAD-NAMES*, which is a primary level topic.

You can gain access to the primary level topics by entering the name of a primary key as the string parameter. Each of these topics contains explanatory text and a menu of secondary level topics. The primary keys include the following.

Primary Key	**Topic**
Help	General information and menus of other topics
Error-numbers	Error messages beginning with a numeric string
Symbols	⎕*AV* characters
Qualifiers	Qualifiers beginning with the slash (/) symbol
Quad-names	System functions and variables beginning with the quad (⎕) symbol
System-commands	System commands beginning with the right parenthesis ()) symbol

To gain access to a secondary level topic, you can enter the name of a primary key followed by a space and the name of a secondary key. Use the following form:

)*HELP* *primary-key secondary-key*

In many cases, you can omit the primary-key parameter and obtain help directly from a secondary level; if you specify a system command, system function, system variable, qualifier name, or error number, APL generates the primary key for you and uses your string as the secondary key. (Note that APL generates primary keys only when you use the default APL Help library.) For example, you can enter)*HELP* ⎕*MBX* and receive information on ⎕*MBX* without enterring)*HELP QUAD-NAMES* ⎕*MBX*. The following table describes how APL translates secondary key entries:

INPUT	**TRANSLATION**
Secondary Key	**Primary Key**
null-string	Help
numeric-string	Error-numbers
atomic-vector-character	Symbols
string	Qualifiers
⎕*string*	Quad-names
) *string*	System-commands

APL also performs translations in other instances where the first character (not including blanks) following the)*HELP* command is a character from the following table. (These translations do not occur when you have specified /*LIBRARY*.)

INPUT Secondary Key	TRANSLATION Primary and Secondary Keys
.	Symbols period
$	Symbols dollar
÷	Symbols divide
!	Symbols shriek
' '	Symbols '
@	Symbols at sign
⍝	Symbols lamp
?	Symbols question mark
(	Symbols left parenthesis

Aside from the instances described above where APL recognizes a string and generates the appropriate primary and secondary keys, APL assumes that the string you enter is a primary key followed by optional subkeys separated by blanks. For example,)*HELP ARITHMETIC-FUNCTIONS* provides a description of arithmetic functions and a menu of subtopics on which you could obtain help. Entering)*HELP ARITHMETIC-FUNCTIONS FACTORIAL* provides information on the factorial function.

Once APL determines a primary key, it translates the key and all related subkeys from ⎕*AV* characters to TTY mnemonics using ⍞ mode; this produces keys in a format understood by the Help facility, which then locates the appropriate text. This text is then translated from TTY mnemonics to ⎕*AV* characters, converted into uppercase, and then sent to the appropriate output destination by APL. (The text is not converted into uppercase in two instances: when your terminal is a VT102, VT220, VT240, VT320, VT330, VT340, DECterm, VS, or is in TTY mode; and when you execute)*HELP* with ⎕*XQ* or ⍎ .)

When you request information that currently exists within the APL Help Library, the output appears in the following form:

```
key1
   key2
      key3

     help text

     additional help text (if any)
```

When you request information that currently does not exist within the APL Help Library, the output appears in the following form:

SORRY, NO DOCUMENTATION ON xxx

ADDITIONAL INFORMATION AVAILABLE ON ...

Where *xxx* is the string you specified when you invoked)*HELP*, and . . . indicates a menu of available help topics.

APL treats the *string* parameter as a prefix when it locates a topic in the APL Help Library. For example,)*HELP* ⎕*L* finds the help file text for all *QUAD-NAMES* beginning with ⎕*L*.

The APL Help facility accepts wildcards in the form of the ellipsis (. . .) and pairs of star characters (⋆⋆). (A single ⋆ character returns information on the ⋆ symbol.) For example:

Command	**Meaning**
)*HELP* ⋆⋆	Returns text on all primary key levels.
)*HELP* ⋆⋆ . . .	Returns all text on all levels.
)*HELP* *key-name* . . .	Returns all text on the primary key (*key-name* and all its subkeys).

Note that you cannot use the ellipsis on secondary (or lower) keys:

```
      )HELP
HELP

  The )HELP command provides you with controlled access to the VMS
  HELP librarian to obtain help on various topics related to the
  VAX APL language.  APL looks for the file associated with the
  logical name APL$HELP:.  If that is not defined, it looks for
  SYS$HELP:VAXAPL.HLB. This system command accepts terms familiar to
  APL as keys into the APL help library and returns a character vector
  (help text) with embedded Carriage Returns Line Feeds.


 Additional information available:

 APL-applications     APL-command-line     Arithmetic-Functions  Axis
 Comments   Editor    Error-Numbers   Execute-only
 File-System          Function-Names       Glossary    Help       Indexing
 Logical-Functions    Operators  Quad-Names Qualifiers
 Relational-Functions Specification-Function          Statements Symbols
 System-Commands Terminal-Input-Output Terminal-Support

      )HELP +
SYMBOLS

  +

      Plus   TTY mnemonic is  +
        To obtain help on monadic + type )HELP ARITHMETIC-FUNCTIONS CONJUGATE

        To obtain help on dyadic + type )HELP ARITHMETIC-FUNCTIONS ADD

         )HELP 6

    ERROR-NUMBERS

      0006  LABEL ERROR

        Improper use of a colon was detected, or an improper variable
        name was entered as a label.

        Secondary error messages:

        (DUPLICATE LABEL)

        (NAME IN USE)

        An attempt was made to use the same identifier for both a label and
        a local symbol or argument.

        (OPERATION SUSPENDED, PENDENT, OR MONITORED)

        An attempt was made to change a label definition in a suspended, pendent,
        or monitored operation.


         ⍝NOTE THAT THE ARGUMENT TO )HELP IS TREATED AS A PREFIX
         ⍝THERE IS MORE THAN ONE QUAD-NAME THAT STARTS WITH ⎕L
         )HELP ⎕L

    QUAD-NAMES

      ⎕L

        ⎕L - Watched Variable Name
        Type:   System Variable
        Forms:  ⎕L ← any-value
                variable-name ← ⎕L
        Value Domain:
                       Type:   Any
                       Shape:  Any
                       Depth:  Any
        Result Domain:
                       Type:     Character (any when set by user)
                       Rank:     1 (vector) (any when set by user)
                       Shape:    Vector (any when set by user)
                       Depth:    1 (simple vector) (any when set by user)
                       Default:  ''

        A variable that is used implicitly by ⎕WATCH. ⎕L
        contains a character vector showing the name of a watched
        variable that has changed.  ⎕L is set implicitly by the
        system when a variable changes, but can also be set by the user.


       Additional information available:

       Errors


    QUAD-NAMES

      ⎕LC
```

```
     ⎕LC - Line Counter
     Type:  Niladic System Function
     Form:  current-line-number ← ⎕LC
     Result Domain:
                    Type:   Integer
                    Rank:   1 (vector)
                    Shape:  Vector
                    Depth:  1 (simple vector)
     Default Value: Empty

     Vector of line numbers in the state indicator;
     most recently suspended operation appears first.

     Typing →⎕LC restarts the most recently suspended
     operation at the beginning of the line where execution
     was stopped.


    Additional information available:

    Errors


QUAD-NAMES

  ⎕LX

     ⎕LX - Latent Expression
     Type:   System Variable
     Forms:  ⎕LX ← character-vector
             current-value ← ⎕LX
     Value Domain:
                    Type:     Character
                    Shape:    Vector domain
                    Depth:    0 or 1 (simple)
                    Default:  ''
     Result Domain:
                    Type:   Character
                    Rank:   1 (vector)
                    Shape:  Vector
                    Depth:  1 (simple vector)

     Causes expression to be executed automatically
     when workspace is loaded.

     The expression is not executed when you load the
     workspace with the )XLOAD system command.


    Additional information available:

    Errors

      ⍝THE NEXT EXAMPLE DEMONSTRATES THE /LIBRARY QUALIFIER
      )HELP /LIBRARY=DEV1:[APLGRP.LIBRARY]TEMP.HLB

     HELP

 This is a sample help file.  You can modify the VAX APL HELP
 function file or create additional help files.

 For help building library files, see the VMS LIBRARIAN
 REFERENCE MANUAL.

        Additional information available:

 APLHELP_File Assigning_Default_Library
 Library_Source_File Library_Utility
```

```
    )HELP /LIBRARY=DEV1:[APLGRP.LIBRARY]TEMP.HLB APLHELP
    APLHELP_File
Certain files may be modified after installation if desired.
VAXAPL.HLP, the source text of the VAX APL HELP function file,
is in SYS$LIBRARY.  You can add new text to the HELP library.
Refer to )HELP HELP HOW-TO-BUILD while inside VAX APL for
instructions on creating the help file.
```

Possible Errors Generated

```
112  ERROR PROCESSING HELP (ERROR OPENING AS INPUT)

112  ERROR PROCESSING HELP (ERROR PARSING ARGUMENT TO LIBRARY)

112  ERROR PROCESSING HELP (INVALID KEY)

112  ERROR PROCESSING HELP (TOO MANY HELP KEYS SPECIFIED)
```

) *INPUT* Diverting Input to Another Device

Type

Query/Change System Command

Form

) *INPUT* ⟦*filespec* ⟦/*character-set*⟧⟧

Qualifiers

/*LIST*
The query form of the) *INPUT* command. Use /*LIST* to list the names of the currently nested input files.

/*REVERT*
Cancels all nested input files and returns to your terminal as the source of input.

Description

) *INPUT* allows you to change the source of APL input from your terminal to other devices. Typically, you would select a file (*filespec*) to be the new source. *character-set* specifies that the file is to be read in a character set other than the one you designated for your terminal when you invoked APL. The possible values are /*TTY*, /*KEY*, /*BIT*, /*COMPOSITE* and /*APL*.

If no arguments are used,) *INPUT* cancels the current input stream and returns to the previous input stream on the list.

The) *INPUT* system command is also described in the *VAX APL User's Guide* along with other I/O information.

Possible Errors Generated

1 *FILE NOT FOUND* (*FILE NOT FOUND*)

14 *DEPTH ERROR* (*TOO MANY DIVERTED INPUTS*)

22 *INCORRECT PARAMETER* (*EXTRANEOUS CHARACTER AFTER COMMAND*)

22 *INCORRECT PARAMETER* (*INVALID CHARACTER SET QUALIFIER*)

35 *INVALID FILE SPECIFICATION* (*WILDCARDS NOT ALLOWED IN FILE SPECIFICATION*)

)*LIB* Listing Workspace Names

Type

Query System Command

Form

)*LIB* ⟦*file-spec*⟧

Description

)*LIB* displays a list of workspace or file names located in the area specified.

If you omit *file-spec*,)*LIB* lists all the files on your default device and directory area that have the file type .APL (the default for workspace names). If you use *file-spec*, APL lists the names of all selected files, not just workspaces, on the selected device and directory.

You can identify a particular file, or use the wildcard characters ⋆ and ÷, to substitute all or part of the file name or file type; for the version number, only ⋆ is a valid wildcard character. In the following example, this command lists all files on the default device and directory that have a file name beginning with the letter *W*:

```
        )LIB W⋆.⋆;⋆
```

The following command lists the names of all files on the default device in the directory [USER.APL]:

```
        )LIB [USER.APL]
```

The file names in the list begin in column 1 and are separated by a Carriage Return Line Feed. The list of file names is preceded by a line identifying the device and directory, and the list is followed by a line that tells how many files were listed. For example:

```
    )LIB

Directory DEV1:[APLGRP]

ALPHA.APL;1
CHAR.APL;1
GEORGE.APL;1
PRIME.APL;1
```

```
Total of 4 files.
      )SAVE WS40
THURSDAY 29-NOV-1990 16:54:45.31 3 BLKS
      )LIB

DEV1:[APLGRP]
ALPHA.APL;1
CHAR.APL;1
GEORGE.APL;1
PRIME.APL;1
WS40.APL;1

Total of 5 files.
      )SAVE WS40.VAR
THURSDAY 29-NOV-1990 16:54:45.84 3 BLKS
      )LIB WS40.*

Directory DEV1:[APLGRP]

WS40.APL;1
WS40.VAR;1

Total of 2 files.
      )LIB *.*

Directory DEV1:[APLGRP]

ALPHA.APL;1
CHAR.APL;1
GEORGE.APL;1
LIS←WOR←1.AAS;1
LIS←WOR←1EX.OUT;1
LIS←WOR←1TMP.AAS;1
PRIME.APL;1
WRITE←EXAMPLE.COM;7
WS40.APL;1
WS40.VAR;1

Total of 10 files.
```

Note that when you execute the)LIB command with no argument, APL passes the following command string to VMS for execution:

```
DIRECTORY/COLUMNS=1/HEADING/TRAILING *.APL;*
```

If you include an argument with)*LIB*, that argument is substituted for **.APL;** in the command string passed to VMS. The argument may be no more than 95 characters long (after translation to ASCII), not including leading white space (spaces or tabs before the argument begins), but including all other white space within the argument. For example:

```
      )LIB/PROTECTION WS40.*
Directory DEV1:[APLGRP]

WS40.APL;1            (RWED,RWED,RE,)
WS40.VAR;1            (RWED,RWED,RE,)

Total of 2 files.
```

The)*LIB* command uses the DCL command DIRECTORY. This is true even if you have a symbol definition for DIRECTORY that has different qualifiers. For more details about the DCL command DIRECTORY, see the *VMS DCL Dictionary*.

Possible Errors Generated

```
22  INCORRECT PARAMETER (LINE TOO LONG TO TRANSLATE)
```

)LOAD Retrieving a Workspace

Type

Workspace Manipulation System Command

Form

)LOAD wsname

Qualifiers

/PASSWORD **[:[*pw*]]**
If you use a password when the workspace is saved, you must specify it when you perform a load operation or APL will not retrieve the workspace. A */PASSWORD* or */PASSWORD:* specification that is not followed by a password is ignored.

/CHECK
The optional */CHECK* qualifier causes APL to examine the workspace for possible corruption (damage to the internal structure of the workspace). If damage is detected, a message is displayed and APL tries to recover as much information as possible from the workspace and continues the load. The recovered workspace may be missing APL variables, user-defined operations, individual lines of user-defined operations, and other APL objects that were damaged. The user must determine what named objects have been removed from the workspace. You must use the *)SAVE* command if you want to maintain an undamaged version of the recovered workspace.

Description

)LOAD makes *wsname* the active workspace by replacing, and thus destroying, the contents of the currently active workspace. If the workspace named by *wsname* is larger than the current *)MAXCORE* setting, APL increases the setting and loads the workspace.

The file specification you give for *wsname* must include at least a file name. APL will assume default values for the rest of the specification; that is, it assumes the file type *.APL*, the current user device and directory, and an empty password.

When you load a workspace, the)*LOAD* system command responds by displaying the word *SAVED*, followed by the time and date when the workspace was saved, followed by the size (in disk pages) of the newly active workspace. If the newly active workspace contains a suspended operation, APL also prints a star. If the newly active workspace is larger than the current setting for *MAXCORE*, APL prints the message *NEW MAXCORE IS nnnP*, where *nnn* is the new size of maxcore, and *P* indicates that the size is measured as pages of memory. The)*WSID* value of the loaded workspace is the value you specified for *wsname*.

The ⎕*QLD* system function (see Chapter 2) performs the same operation as the)*LOAD* command, but does not display the verifying messages.

The verifying messages for the)*LOAD* and)*SAVE* system commands are very similar. The only difference is that when you load a workspace, the message includes the word *SAVED*. You can use this difference to distinguish between a workspace that has just been loaded and a workspace that has just been saved. For example:

```
      ∇FRY
[1]   M ← ⎕XQ 'SAVE ROAR')
[2]   'MESSAGE IS ' ; M
[3]   ∇
      )WSID FRY
WAS CLEAR WS
      FRY
MESSAGE IS THURSDAY 8-NOV-1990 17:12:33.14 16 BLKS
      )LOAD ROAR
MESSAGE IS SAVED THURSDAY 8-NOV-1990 17:12:33.14 16 BLKS
      M
SAVED THURSDAY 8-NOV-1990 17:12:33.14 16 BLKS
```

In this case, the user executes the)*SAVE* command from within the function *FRY*. APL saves the workspace and assigns the verifying message of the)*SAVE* command to the variable *M*. Next, the user loads the workspace. APL immediately continues execution of the function and assigns the verifying message of the)*LOAD* command to the variable *M*.

When you load a workspace that was saved inside ⎕ input, APL removes the pendent ⎕ input from the state indicator stack. If the ⎕ input was executed from within an operation, APL suspends the operation after removing the pendent ⎕ input from the stack.

If the ⎕*LX* system variable (see Chapter 2) has a value in the workspace, it is executed when the workspace is loaded, unless the top of the state indicator stack contains an execute function, or unless the workspace was saved in function-definition mode (if it was, you remain in function-definition mode after the workspace is loaded). If the workspace was saved inside ⎕ input, the ⎕*LX* expression is executed only after APL removes the pendent ⎕ input from the state indicator stack.

Examples:

```
      )LOAD WS35
SAVED THURSDAY  8-NOV-1990 17:12:11.52 41 BLKS
      )LOAD SYS$SCRATCH:TICTAC
SAVED THURSDAY  8-NOV-1990 17:11:59.28 41 BLKS
                            ⍝JEN HAS A SUSPENDED OPERATION
      )LOAD JEN
SAVED THURSDAY  8-NOV-1990 17:04:23.38 42 BLKS*
      )CLEAR
CLEAR WS
      ⎕LX ← '''USE APL_LASER PRINTER'''
      )LOAD SQUARE
SAVED THURSDAY  8-NOV-1990 17:03:11.46 11 BLKS
      SQUARE
      )WSID ROOT
WAS SQUARE
      )SAVE ROOT
THURSDAY  8-NOV-1990 17:27:40.51 12 BLKS
      )LOAD ROOT
SAVED THURSDAY  8-NOV-1990 17:27:40.51 12 BLKS*
```

Possible Errors Generated

```
22 INCORRECT PARAMETER

22 INCORRECT PARAMETER (INVALID KEYWORD OR QUALIFIER)
```

)*MAXCORE* Determining the Maximum Workspace Size

Type

Query/Change System Command

Form

)*MAXCORE* ⟦*n*⟧

Default in Clear Workspace

1024P / 1048576P

Description

As an action command,)*MAXCORE* changes the current setting for the maximum workspace size to the value specified (*n*) and displays the previous)*MAXCORE* setting. As a query command,)*MAXCORE* returns the current maximum workspace size and the system maximum workspace size.

You may not set the current maximum workspace size to a value smaller than the amount currently in use, or to a value less than 40, which is the minimum value of *n* for all new workspaces. Also, you may not set the maximum size to a value larger than the system maximum. Note that, depending on your system resources, you may not have access to the maximum amount of memory.

Although the number of pages (*P*) appears in the display, you do not type *P* in the command string. For example:

```
      )MAXC
1024P/ 1048576P
      )MAXC 2000
WAS 1024P/ 1048576P
      )MAXC
2000P/ 1048576P
```

Possible Errors Generated

```
22  INCORRECT PARAMETER (EXTRANEOUS CHARACTERS AFTER COMMAND)

22  INCORRECT PARAMETER (ILL FORMED NUMERIC CONSTANT)

22  INCORRECT PARAMETER (PARAMETER OUT OF RANGE)
```

)MINCORE Determining the Minimum Workspace Size

Type

Query/Change System Command

Form

)MINCORE ⟦*n*⟧

Default in Clear Workspace

40P

Description

)MINCORE displays or changes the current minimum workspace size (in pages). As an action command, *)MINCORE* changes the current setting for the minimum workspace size to the value specified (*n*) and displays the previous setting. Legal values for the *)MINCORE* setting are 0 through the current *)MAXCORE* value. As a query command, *)MINCORE* returns the current minimum workspace size.

The *)MINCORE* system command is useful in dealing with large arrays or in performing operations that require large amounts of intermediate storage. Such operations can make the workspace continually expand, thus slowing the processing and fragmenting of the workspace. You can improve system efficiency by using *)MINCORE* to ensure that a reasonable amount of memory is available at the beginning of the operation.

Generally, the *)MINCORE* setting does not change when you load a workspace. However, depending on the characteristics of the loaded workspace, the *)MINCORE* setting may be greater than the amount of available memory. In this case, *)MINCORE* is reset to the default when the *)LOAD* succeeds (no error is signaled).

Examples:

```
      )MINC
32P
      )MINC 100
WAS 32P
```

Possible Errors Generated

```
22  INCORRECT PARAMETER (EXTRANEOUS CHARACTERS AFTER COMMAND)

22  INCORRECT PARAMETER (ILL FORMED NUMERIC CONSTANT)

22  INCORRECT PARAMETER (PARAMETER OUT OF RANGE)
```

)*MON* Returning to Operating System Command Level

Type

System Action System Command

Form

```
)MON
```

Description

)*MON* returns control to operating system command level. It does not save the active workspace, but if ⎕*AUS* is set (see Chapter 2), the workspace is automatically saved. The)*MON* command does not close open files, but it does flush the file buffers.

When you use)*MON* to leave APL, you can return to APL by typing the DCL command CONTINUE. If you intend to return to APL, be careful not to destroy your memory image while you are at DCL level. This situation could occur if you issue a command that runs a program.

The)*MON* command has limited value because most DCL commands do run a program and thus will destroy the APL image. If you want to return to the DCL command level to run other programs, you should use the)*PUSH* or)*DO* command. If you want to return to another process, you should use the)*ATTACH* command.

Possible Errors Generated

```
22  INCORRECT PARAMETER (EXTRANEOUS CHARACTERS AFTER COMMAND)
```

)*NMS* Displaying Names in the Symbol Table

Type

Query System Command

Form

)*NMS* ⟦*start-string*⟦*stop-string*⟧⟧

Qualifiers

/*WSID*: ***wsname***
Allows you to list the names in a nonactive workspace. *wsname* specifies the workspace.

/*PASSWORD*: **⟦*pw*⟧**
Specifies the password used when the nonactive workspace was saved.

Description

)*NMS* displays all the names in the symbol table in ⎕*AV* order. The result is a combination of the same information obtained by the)*VARS*,)*FNS*,)*OPS*, and)*GRPS* commands.

By default, APL displays the names from the currently active workspace. The optional /*WSID* qualifier allows you to specify a nonactive workspace. If the nonactive workspace was saved with a password, you must also specify the /*PASSWORD* qualifier.

The optional string parameters identify starting and stopping points for the list. When you specify the string parameters, you can use the ⋆ and ÷ wildcards. The objects are listed in ⎕*AV* order, separated by tabs. Each output line in the list begins in column 1.

Note that the wildcard determines the *start-string*. There is no wildcard for the *stop-string*.

The following name classes are possible:

Value	Meaning
0	Name not in use
2	Variable name
3	Function name
4	Operator name
¯4	Group name

A symbol has a name class of 0 when it has no value. Such a symbol may be listed in the symbol table because it is currently referenced in a user-defined operation (either a function or operator) or was previously used and has since been erased. For example:

```
      )LOAD NAMES
SAVED THURSDAY 8-NOV-1990 17:44:43.63 15 BLKS
      )GRPS
LAB1        RECAP   REPLACE VERTICAL
      )VARS
A   B       LAT     VET
      )FNS
ADD
      )NMS
A.2 ADD.3   B.2     DC.0    DONE.0  DOWN.0  LAB1.¯4 LAT.2
NEW.0       OLD.0   OUT.0   RECAP.¯4        REPLACE.¯4       TEST.0
UP.0    VERTICAL.¯4 VET.2
```

If you use)NMS with no parameters, APL displays all the symbols in the workspace. If you use the optional string parameters, you can specify a particular section from the list of symbol names. For example, all the names starting with B through those starting with LA. If you include just one argument, APL uses Z̲ as the default for the second string. To get a list of all symbol names that begin with a given prefix, use the prefix for both arguments or use a wildcard. For example:

```
      )LOAD NAMES
SAVED THURSDAY 8-NOV-1990 17:44:43.63 15 BLKS
      )NMS B LA
B.2 DC.0    DONE.0  DOWN.0  LAB1.¯4 LAT.2
      )NMS R
RECAP.¯4    REPLACE.¯4      TEST.0  UP.0    VERTICAL.¯4     VET.2
      )NMS VE VE
VERTICAL.¯4 VET.2
      )NMS VE*
VERTICAL.¯4 VET.2
```

Possible Errors Generated

1 *FILE NOT FOUND (FILE NOT FOUND)*

22 *INCORRECT PARAMETER (EXTRANEOUS CHARACTERS AFTER COMMAND)*

22 *INCORRECT PARAMETER (FILE SPECIFICATION IS MISSING)*

22 *INCORRECT PARAMETER (INVALID KEYWORD OR QUALIFIER)*

22 *INCORRECT PARAMETER (NOT A LETTER)*

57 *FILE DOES NOT CONTAIN A WORKSPACE*

)*OFF* Terminating the APL Session

Type

System Action System Command

Form

)*OFF* ⟦*HOLD* | *LOGOUT* ⟧

Default in Clear Workspace

HOLD

Description

)*OFF* terminates an APL session.

If you specify the *HOLD* parameter (*HOLD* is the default), APL terminates your session and returns you to DCL command level. If you specify the *LOGOUT* parameter, APL not only terminates your session, but logs you off the system.

The)*OFF* command destroys the currently active workspace, deletes the ⎕*AUS* file, closes all open files, and resets the terminal characteristics to the system settings. When you use)*OFF*, you cannot return to APL by enterring the DCL command CONTINUE.

The)*OFF* command displays several lines of information before terminating the session. The lines contain the following:

- Your terminal identification
- Current time
- Current date
- Length of time connected to APL
- Amount of computer CPU time used inside APL
- Number of APL statements executed
- Number of APL operations executed
- Number of page faults while inside APL
- Number of buffered IO and number of direct IO while inside APL

Examples:

```
      )OFF LOGOUT
SYS$INPUT:  THURSDAY  8-NOV-1990 17:48:59.32
CONNECTED 00:00:00.98  CPU TIME 00:00:00.37
0 STATEMENTS  0 OPERATIONS
170 PAGE FAULTS  21 BUFFERED IO  9 DIRECT IO
      )OFF HOLD
SYS$INPUT:  THURSDAY  8-NOV-1990 17:50:13.15
CONNECTED 00:00:00.98  CPU TIME 00:00:00.38
0 STATEMENTS  0 OPERATIONS
154 PAGE FAULTS  21 BUFFERED IO  9 DIRECT IO
$
```

Possible Errors Generated

```
22 INCORRECT PARAMETER (EXTRANEOUS CHARACTERS AFTER COMMAND)

22 INCORRECT PARAMETER (UNRECOGNIZED QUALIFIER KEYWORD)
```

)*OPS* Displaying a List of Operators

Type

Query System Command

Form

)*OPS* ⟦*start-string*⟦*stop-string*⟧⟧

Qualifiers

/*WSID*: ***wsname***
Allows you to list the user-defined operators defined in a nonactive workspace. *wsname* specifies the nonactive workspace name.

/*PASSWORD*: ***pw***
Specifies the password used when the nonactive workspace was saved.

Description

)*OPS* displays a list of the global names used as user-defined operator names in a workspace. By default, APL displays the list from the currently active workspace. The optional /*WSID* qualifier allows you to specify a nonactive workspace. If the nonactive workspace was saved with a password, you must also specify the /*PASSWORD* qualifier.

The optional string parameters identify starting and stopping points for the list. When you specify the string parameters, you can use the ⋆ and ÷ wildcards. The objects are listed in ⎕*AV* order, separated by tabs. Each output line in the list begins in column 1.

Note that the wildcard determines the *start-string*. There is no wildcard for the *stop-string*.

If you use)*OPS* with no parameters, APL displays all the global operator names in the workspace:

```
      )OPS
ALPH    HILB    INVRS    INVT    LSQ
```

If you include just one argument, APL uses Z̲ as the default for the second string:

```
      )OPS IN
INVRS        INVT     LSQ
      )OPS INV INV
INVRS        INVT
```

To obtain a list of all user-defined operator names that begin with a given prefix, use the prefix for both arguments or use a wildcard:

```
      )OPS INV*
INVRS        INVT
```

Possible Errors Generated

1 *FILE NOT FOUND (FILE NOT FOUND)*

22 *INCORRECT PARAMETER (EXTRANEOUS CHARACTERS AFTER COMMAND)*

22 *INCORRECT PARAMETER (FILE SPECIFICATION IS MISSING)*

22 *INCORRECT PARAMETER (INVALID KEYWORD OR QUALIFIER)*

22 *INCORRECT PARAMETER (NOT A LETTER)*

57 *FILE DOES NOT CONTAIN A WORKSPACE*

)ORIGIN Determining the Index Origin

Type

Query/Change System Command

Form

)ORIGIN ⟦*n*⟧

Default in Clear Workspace

1

Description

)ORIGIN displays or changes the setting of the index origin (⎕*IO*).

The index origin (*n* in the form) can be either 0 or 1; its setting determines whether the values of an array are indexed beginning with position 0 or 1.

Executing the *)ORIGIN* in change mode has the same effect as assigning a value to the ⎕*IO* system variable (see Chapter 2).

Examples:

```
      ι5
1 2 3 4 5
      )ORIGIN 0
WAS 1
      ι5
0 1 2 3 4
      )ORIGIN
0
```

Possible Errors Generated

22 INCORRECT PARAMETER (EXTRANEOUS CHARACTERS AFTER COMMAND)

22 INCORRECT PARAMETER (ILL FORMED NUMERIC CONSTANT)

22 INCORRECT PARAMETER (SYSTEM VARIABLE VALUE MAY ONLY BE 0 OR 1)

)OUTPUT Diverting Output to Another Device

Type

Query/Change System Command

Form

)OUTPUT ⟦*filespec* ⟦*/character-set*⟧

Qualifiers

/APPEND
Specifies that you want to add data to the end of an existing file. If you specify *filespec* without the */APPEND* qualifier, APL creates a new file.

/DISPOSE: { KEEP | DELETE | PRINT | SUBMIT | PRINTDELETE | SUBMITDELETE}
Specifies whether the value of *filespec* is a temporary or permanent file. */DISPOSE:KEEP*, which is the default, means the file is permanent; */DISPOSE:DELETE* means the file will be deleted when it is closed. */DISPOSE:PRINT* sends the file to a print queue (SYS$PRINT) when the file is closed, and */DISPOSE:SUBMIT* sends the file to a batch queue (SYS$BATCH) when the file is closed. *PRINTDELETE* and *SUBMITDELETE* send the file to the appropriate queue and then delete the file when the job is finished.

/LIST
This is the query form. Allows you to display the diverted output file on one line and SYS$OUTPUT (the VMS name for your default output stream) on the next line (or just SYS$OUTPUT, if output is not being diverted).

/REVERT
Causes the return of system output from the diverted destination to your terminal. This is the same as using *)OUTPUT* with no qualifiers.

/SHADOW
Allows you to display the diverted output on your terminal as well as sending it to a file. Otherwise, no system output is displayed except for system prompts and echoed input.

If you want to begin shadowing output that is already diverted, you can reenter the original *)OUTPUT* command and add the */SHADOW* and */APPEND* qualifiers. If you want to discontinue shadowing while keeping the same diverted output stream, you can reenter the original *)OUTPUT* command with the omission of the /SHADOW qualifier and the addition of the /APPEND qualifier. (You can change any of the original arguments or qualifiers at this time. If you omit information that you specified in the original)OUTPUT command, APL selects any default values that may be applicable. For example, output diverted from an APL terminal with the /TTY qualifier defaults to the APL character set if you do not reenter the /TTY qualifier.)

Description

)OUTPUT allows you to change the destination of output to a device other than your terminal. Typically, you send the output to a file or to another terminal. If the output is sent to a file, you can specify that you want to write the diverted output in a character set other than the one you designated for your terminal when you invoked APL. The possible values for *character-set* are /TTY, /KEY, /BIT, /COMPOSITE, and /APL.

)OUTPUT with no arguments or qualifiers causes the system output to return from the diverted destination to your terminal. This is the same as using the /REVERT qualifier.

When you use)OUTPUT, the output file has the appearance of a normal terminal display containing input lines and the resulting output. However, at your terminal the display is different. The only output that APL generates at your terminal is echoed input and APL prompts. APL echoes any input, whether it comes from your terminal or a file, and APL displays the usual 6-space prompt to signal the completion of a task. In fact, all APL-generated prompts (such as the ⎕SF prompt and function editor prompts) are still displayed at the terminal. If you want to see a normal display at your terminal, use the /SHADOW qualifier (see below).

Note that)OUTPUT files cannot be nested.

If you enter either a weak or strong attention signal while output is being diverted from your terminal, APL responds by displaying output on your terminal as well as in the diverted stream, just as if you had specified /SHADOW.

The)OUTPUT system command is also described in the *VAX APL User's Guide* along with other I/O information.

Possible Errors Generated

22 *INCORRECT PARAMETER* (*EXTRANEOUS CHARACTERS AFTER COMMAND*)

22 *INCORRECT PARAMETER* (*INVALID KEYWORD OR QUALIFIER*)

22 *INCORRECT PARAMETER* (*REDUNDANT KEYWORD OR QUALIFIER*)

33 *IO ERROR* (*INVALID WILDCARD OPERATION*)

)OWNER Displaying Information About Workspace Creation

Type

Query System Command

Form

```
)OWNER
```

Description

)OWNER displays information about the active workspace at the time it was created. A workspace is created when it is saved. The clear workspace is created when the)CLEAR system command is given.

The result of the)OWNER appears in the following form:

CREATED ON *day dd-mmm-yyyy hh:mm:ss.tt* BY *name [uic]* AT *dev:* WITH *lv.u-edit*

> *day dd-mmm-yyyy hh:mm:ss.tt* is the day, date and time of creation
> *name* is the user name of the creator
> *uic* is the user identification code of the creator
> *dev:* is the terminal device name used to create the workspace
> *lv.u-edit* is the version of APL used to create the workspace (see the description of ⎕VERSION in Chapter 2)

Examples:

```
      )CLEAR
CLEAR WS
      )OWNER
CREATED ON THURSDAY  8-NOV-1990 18:11:49.32 BY
 [APLGRP,USER] AT  WITH V3.2-834
      )SAVE USER1WS
THURSDAY  8-NOV-1990 18:11:49.51 3 BLKS
      )LOAD USER1WS
SAVED THURSDAY  8-NOV-1990 18:11:49.51 3 BLKS
      )OWNER
CREATED ON THURSDAY  8-NOV-1990 18:11:49.51 BY
 [APLGRP,USER] AT  WITH V3.2-834
```

Possible Errors Generated

```
22 INCORRECT PARAMETER (EXTRANEOUS CHARACTERS AFTER COMMAND)
```

)*PASSWORD* Workspace Password

Type

Query/Change System Command

Form

)*PASSWORD* ⟦*pw*⟧

Default in Clear Workspace

Empty

Qualifiers

/*PASSWORD* **:⟦*pw*⟧**
Specifies the password associated with the active workspace.

Description

)*PASSWORD* displays or changes the password associated with the active workspace.

APL passwords are eight characters long, and the password you supply must be a valid APL identifier. Passwords longer than eight characters are truncated on the right; passwords with fewer than eight characters are padded on the right with blanks.

If you do not change the password, the form of the)*PASSWORD* display is as follows:

/*PASSWORD*:*pw*

If you do change the password, the form of the)*PASSWORD* display is as follows:

WAS /*PASSWORD*:*pw*

If the password is empty, the display is one of the following:

/*PASSWORD*: *WAS* /*PASSWORD*:

When you use)*PASSWORD* to change the password, you can specify the new password directly after the)*PASSWORD*, or you can specify it following /*PASSWORD*. For example:

```
      )PASSWORD SESAME
WAS /PASSWORD:
                ⍝PASSWORD WILL BE TRUNCATED TO 8 CHARACTERS
      )PASSWORD /PASSWORD:OPENSESAME
WAS /PASSWORD:SESAME
      )PASSWORD /PASSWORD:
WAS /PASSWORD:OPENSESA
      )PASSWORD
/PASSWORD:
```

To load or copy objects from a workspace with a nonblank password, you must include the password for the workspace in the command string.

Possible Errors Generated

```
22  INCORRECT PARAMETER (EXTRANEOUS CHARACTERS AFTER COMMAND)

22  INCORRECT PARAMETER (ILL FORMED NAME)
```

)*PCOPY* Copying from a Workspace with Protection

Type

Workspace Manipulation System Command

Form

)*PCOPY* *wsname* ⟦*list*⟧

Qualifiers

/*CHECK*
The optional /*CHECK* qualifier causes APL to examine the workspace for possible corruption (damage to the internal structure of the workspace). If damage is detected, a message is displayed and APL tries to recover as much information as possible from the workspace and to continue the copy. The recovered workspace may be missing APL variables, user-defined operations, individual lines of user-defined operations, and other APL objects that were damaged. The user must determine what named objects have been removed from the workspace.

/*PASSWORD* **⟦ :⟦*pw*⟧⟧**
Specifies the password used when *wsname* was saved.

Description

)*PCOPY* (protected copy) is the same as the)*COPY* system command, except that)*PCOPY* does not replace objects in the active workspace with objects of the same name in the copy workspace. Instead, APL returns the message *NOT COPIED*: along with the names of the objects affected.

When copying groups, the)*PCOPY* command does not copy any members of the group that have the same name in both workspaces. If the group name itself is the same as a group name in the active workspace,)*PCOPY* does not copy the group name, nor any members of the group.

The)*PCOPY* system command performs the same operation as the ⎕*QPC* system function (see Chapter 2), but ⎕*QPC* does not display messages to verify the success of the copy.

Example:

```
      )PCOPY VARS A T
SAVED TUESDAY  6-NOV-1990 18:21:58.41 13 BLKS
      FOUND: T
NOT COPIED: A
```

Possible Errors Generated

22 *INCORRECT PARAMETER*

22 *INCORRECT PARAMETER (ILL FORMED NAME)*

27 *LIMIT ERROR (ARGUMENT STRING IS TOO LONG)*

83 *DAMAGED WORKSPACE HAS BEEN CORRECTED (SOME SYMBOLS MAY HAVE BEEN ERASED)*

)PUSH Interacting with Operating System Programs

Type

System Action System Command

Form

)PUSH ⟦*command-string*⟧

Qualifiers

/NOKEYPAD
Specifies that you do not want the keypad characteristics of the current process to be available to the new subprocess. The default is that the characteristics are available.

/NOLOGICALS
Specifies that you do not want the logical name table from the current process to be available to the new subprocess. The default is that the table is available.

/NOSYMBOLS
Specifies that you do not want the global and local symbol table (defined at the DCL level) from the current process to be available to the new subprocess. The default is that the symbol table is available.

/NOTIFY
Determines whether VMS broadcasts a message to your current process when the new subprocess completes or aborts. If you are executing the *)PUSH* command from the batch mode, you cannot use the */NOTIFY* qualifier. Note that you can use */NOTIFY* only when you specify the */nowait* qualifier.

/PROCESS: ***process-name***
Specifies the name for the new subprocess. If you do not use */NOWAIT*, and you use the DCL command ATTACH rather than the LOGOUT command to return to APL, you can later use the *)ATTACH* command with the process name that you specify.

/NOWAIT
Allows you to create a detached subprocess. When you specify */NOWAIT*, control returns to APL when the subprocess begins execution, and the subprocess continues to execute in the background. Note that if the subprocess uses any

terminal I/O, it becomes mixed with any terminal I/O used by your current process.

Description

)*PUSH* interrupts the APL session, creates a VMS subprocess, and puts you at DCL command level without terminating the APL session. You can perform any operation at the DCL command level; when you are finished, you can return to APL at the point you left off.

Note that if you use an invalid qualifier when you specify)*PUSH*, APL sends the unrecognized characters to the subprocess command level along with the command string.

If you want to display the process name of any subprocess owned by the current process, use the DCL command SHOW PROCESS/SUBPROCESSES (for more details, see the *VMS DCL Dictionary*).

When you use)*PUSH* without a command string, you remain at DCL command level until you enter LOGOUT to return to APL. When you use)*PUSH* with a command string (do not enclose the string in quotation marks), VMS executes the command string and then automatically returns control to APL. The command string must be no longer than 132 characters (after translation to ASCII), not including leading white space (spaces or tabs before the argument begins), but including all other white space within the argument.

For example, entering)*PUSH* and the DCL command SHOW TIME, and then LOGOUT has the same effect as entering)*PUSH* with the command string *SHOW TIME*:

```
      )PUSH

$SHOW TIME
  23-NOV-1990 13:32:42
$LOGOUT
  Process USER1 logged out at 23-NOV-1990 13:33:13
      )PUSH SHOW TIME
  23-NOV-1990 13:33:41
```

While you are at DCL command level, your terminal is in ASCII rather than APL mode, and your terminal characteristics (such as output line width) revert to the system settings. When you return to APL, the APL character set is restored, and your ⎕*PW* setting is the same as it was before you executed the)*PUSH* command (although the default for ⎕*PW* changes if you changed your system terminal width (see Chapter 2 for details). However, other terminal characteristics you may have changed at DCL command level (for example, the ⎕*GAG* setting, or the ability to input lowercase characters) remain changed.

APL makes no attempt to recover the output from any of the work you do at DCL command level. For example:

```
      TIME ← ⍎')PUSH SHOW TIME'

   23-NOV-1990 13:40:34
      11 VALUE ERROR (REQUIRED VALUE NOT SUPPLIED BY EXECUTE)
      TIME ← ⍎')PUSH SHOW TIME'
             ∧
```

Here, APL executes the)*PUSH* command, and VMS displays the result of the SHOW TIME command. But APL does not recover the output and cannot assign the value to the variable *TIME*.

For more details about VMS subprocesses, see the *VMS DCL Dictionary*.

Possible Errors Generated

```
22 INCORRECT PARAMETER (ILLEGAL ASCII CHARACTER)

22 INCORRECT PARAMETER (MISSING ARGUMENT)

22 INCORRECT PARAMETER (NOKEYPAD QUALIFIER REPEATED)

22 INCORRECT PARAMETER (NOLOGICALS QUALIFIER REPEATED)

22 INCORRECT PARAMETER (NOSYMBOLS QUALIFIER REPEATED)

22 INCORRECT PARAMETER (NOTIFY QUALIFIER REPEATED)

22 INCORRECT PARAMETER (NOWAIT QUALIFIER REPEATED)

22 INCORRECT PARAMETER (PROCESS NAME QUALIFIER REPEATED)

73 SUBPROCESS ERROR (COMMAND BUFFER OVERFLOW---SHORTEN EXPRESSION
   OR COMMAND LINE)
```

)SAVE Saving a Copy of the Active Workspace

Type

Workspace Manipulation System Command

Form

)SAVE ⟦*wsname*⟧

Qualifiers

/CHECK
The optional */CHECK* qualifier causes APL to examine the workspace for possible corruption (damage to the internal structure of the workspace). When */CHECK* is specified on *)SAVE*, APL checks for possible damage before saving the current workspace on disk. If there is damage, APL signals an error and aborts the execution of *)SAVE*. If this occurs, use *)SAVE* without */CHECK* to save the damaged workspace; use *)LOAD* with */CHECK* to recover as much as possible from the damaged workspace and determine what APL objects have been lost from the damaged workspace. You must use the *)SAVE* command if you want to maintain an undamaged version of the recovered workspace.

/MAXLEN⟦:***n***⟧
The optional */MAXLEN* qualifier allows you to specify the maximum record length, *n* (in bytes), to be used to save the workspace. If you omit */MAXLEN* (or specify it without an argument), APL uses the value of *⎕DML* (see Chapter 2) as the maximum record length.

/PASSWORD ⟦ :⟦***pw***⟧⟧
The *)SAVE* system command allows you to specify a password for your workspace. The default is an empty password (eight blanks). If you save a workspace that has a password – either one you specify with *)SAVE* or one specified earlier by the *)PASSWORD* or *)WSID* command—you have to specify the password when you load or copy the workspace.

Description

)SAVE saves a copy of the active workspace in a file specified by *wsname*. If you omit *wsname*, the file is saved with the name currently returned by *)WSID*.

The)SAVE command displays the time and date the workspace is saved, the number of disk blocks required to store the workspace, and the workspace identification (either the name currently returned by)WSID, or the name you specify as *wsname*).)SAVE appends a star (⋆) to the message if the saved workspace contains a suspended operation.

When you save a workspace, you have the option of saving it under its current name—the name returned by)WSID—or renaming it. However, APL does not save a workspace under a name that already exists in your storage area, unless the)WSID is that name. If you specify a new name with the)SAVE command, you not only store your active workspace under that name, but you also change the name of the currently active workspace to the new name specified.

If your current)WSID is the same as a workspace you have already saved, APL creates a new version of the file. Both the old and new files are available on the appropriate storage device; however, the new file is considered the current version and has a version number one greater than that of the old file.

APL does not save a clear workspace. If your workspace is clear, you must first give it a name with the)WSID command, or you must use the)SAVE command with a workspace name.

If you specify a password using)PASSWORD or)WSID, but then save the workspace using the *wsname* parameter, the workspace is saved with an empty password (unless you specify a new one with)SAVE). For example:

```
      )WSID MYWS/PASSWORD:SESAME
WAS EXAMPLE
                          ⍝CHANGE THE WSID
      )SAVE MYWS
THURSDAY  8-NOV-1990 19:42:58.52 15 BLKS
      )WSID
MYWS
                          ⍝PASSWORD CHANGED TO EMPTY
      )PASSWORD
/PASSWORD:
                          ⍝NO NEED FOR A PASSWORD WHEN LOADING
      )LOAD MYWS
SAVED THURSDAY  8-NOV-1990 19:42:58.52 15 BLKS
```

You can save a workspace while there is a suspended operation on the top of the SI stack. When you load the workspace, the operation is still suspended; it does not continue automatically. You can cause an automatic startup by using the ⎕LX system variable (see Chapter 2).

If you execute a)SAVE command within an operation, for example, with ⎕XQ ')SAVE', APL saves the workspace, displays the time and date, and continues executing the operation. The next time you load that workspace, APL displays a slightly different message (see)LOAD for details) and begins the session by executing that particular operation after the ⎕XQ ')SAVE'. It does not execute ⎕LX, because ⎕LX does not execute if the loaded workspace is in function-definition mode, or if the operation at the top of the state indicator stack is pendent.

Examples:

```
      )CLEAR
CLEAR WS
      )WSID
CLEAR WS
      )SAVE
 60 WS NOT SAVED, THIS WS IS CLEAR WS
      )SAVE
      ∧
      )WSID WS30
WAS CLEAR WS
      )SAVE
THURSDAY  8-NOV-1990 19:46:08.95 3 BLKS WS30
      )WSID WS10
WAS WS30
      )SAVE
THURSDAY  8-NOV-1990 19:46:09.24 3 BLKS WS10
      )WSID WS30
WAS WS10
      )SAVE WS10
 60 WS NOT SAVED, THIS WS IS WS30
      )SAVE WS10
      ∧
      )WSID WS35
WAS WS30
```

Possible Errors Generated

```
22  INCORRECT PARAMETER (EXTRANEOUS CHARACTERS AFTER COMMAND)

22  INCORRECT PARAMETER (INVALID KEYWORD OR QUALIFIER)

59  WS NOT SAVED, THIS WS IS CLEAR WS
```

)*SI* Displaying the State Indicator

Type

Query System Command

Form

```
)SI
```

Description

)*SI* displays the state indicator of the active workspace. The state indicator contains the status of the execution of user-defined operations, quad input requests, and execute functions.

For user-defined operations, APL displays the operation name followed by, within brackets, the line and statement numbers at which the operation stopped executing. No statement number is displayed if the statement at which execution stopped is the first or only statement on the line. If a statement number is displayed, it is separated from the line number by a diamond (⋄) character. A star following the bracketed line and statement number indicates that the operation is currently suspended; no star indicates that the operation is pendent. For example:

```
      )SI
F[2] *
G[3⋄2]
```

In this example, the pendent operation *G* stopped executing at statement 2 on line [3] and is currently waiting for operation *F*, which was suspended at line [2].

Pendent quad input requests are indicated by a ⎕ character. For example:

```
      A←⎕
⎕:
      )SI
⎕
⎕:
```

First, the)*SI* display shows that the quad input request is pendent; then, APL displays ⎕: to reprompt for quad input.

Pendent execute functions are indicated by the ⎕*XQ* or ⍎ characters. For example:

```
      ⎕XQ ')SI'
⎕XQ

      )SI
                        (There is no output)
```

In this example, the)*SI* display indicates that an execute function is pendent. This occurs because APL executes expressions from right to left, and the output from)*SI* is displayed before the execute function is considered to have completed. Afterwards, the state indicator is clear.

The order of display in the)*SI* list is significant; the operation or quad input request that was most recently active is listed first, the next most recent request is listed second, and so on.

Locked operations in the state indicator are flagged with a ⍫ character, and no line number is displayed.

You can clear individual operations from the state indicator by using the branch function (→) to restart or terminate suspended operations, or you can use the system function ⎕*RESET* or)*SIC* to clear the state indicator entirely. When the state indicator is clear,)*SI* returns no result. For more information, see the *VAX APL User's Guide*

Possible Errors Generated

```
22  INCORRECT PARAMETER (EXTRANEOUS CHARACTERS AFTER COMMAND)
```

)*SIC* Clearing the State Indicator

Type

APL Action System Command

Form

```
)SIC
```

Description

)*SIC* clears the state indicator. Once cleared, the state indicator shows no suspended operations and no pending quad input requests or execute functions. After you use)*SIC*, the)*SI*,)*SINL*, and)*SIS* system commands do not return a value. The)*SIC* system command behaves in the same manner as the ⎕*RESET* system function (see Chapter 2), and they can be used interchangeably.

)*SIC* does not return a value.

Possible Errors Generated

```
22  INCORRECT PARAMETER (EXTRANEOUS CHARACTERS AFTER COMMAND)
```

)*SINL* Displaying the State Indicator and Local Symbols

Type

Query System Command

Form

```
)SINL
```

Description

)*SINL* displays the same information as)*SI*. This includes the status of the execution of user-defined operations, quad input requests, and execute functions. In addition,)*SINL* lists the local symbols of each operation, and displays the argument expression of any pending execute function. Local symbols in locked operations (flagged with a ⍫ character) are not displayed.

For example:

```
      )SINL
F[2] * R A B
G[3⋄2] * T C A D
```

Here, the pendent operation *G* has the local symbols *T*, *C*, *A*, and *D*. The suspended operation *F* has the local symbols *R*, *A*, and *B*.

When the state indicator is clear,)*SINL* returns no result.

Possible Errors Generated

22 *INCORRECT PARAMETER (EXTRANEOUS CHARACTERS AFTER COMMAND)*

)*SIS* Displaying the State Indicator and Executing Lines

Type

Query System Command

Form

)*SIS*

Description

)*SIS* displays the same information as)*SI*. In addition, it displays the line that is currently being executed and the argument expression of any pendent execute functions.

For example:

```
      )SI
F[2] *
⍎
G[3⋄2]
      )SIS
F[2] *       B÷0
⍎ F X
G[3⋄2] X←A×2 ⋄ Y←⍎ 'F X'
```

Here, the function *F* is suspended at line [2] because of an invalid division by zero. The execute function that called *F* is pending, and its argument is displayed. Finally, the function *G* is pending, and its currently executing line, containing the execute function that calls *F*, is displayed.

Note that)*SIS* does not display the executing line of a locked operation.

When the state indicator is clear,)*SIS* returns no result.

Possible Errors Generated

```
22  INCORRECT PARAMETER (EXTRANEOUS CHARACTERS AFTER COMMAND)
```

)STEP Executing Lines of a Suspended Operation

Type

APL Action System Command

Form

)STEP ⟦*n*⟧

Qualifiers

/SILENT
Specifies that APL should not display the operation name and the current line that are at the top of the state indicator after the execution of the lines of the operation.

/INTO
Specifies that you want APL to step into any called operations.

/OVER
Specifies that you want APL to step over any called operations. This is the default setting.

Description

)STEP is a debugging feature that allows you to execute one or more lines of a suspended operation. The *)STEP* command is valid only when specified from immediate mode, and when there is a suspended operation on the top of the state indicator.

Examples:

```
      ∇FRILL
[1]   'FRILL LINE 1'
[2]   'FRILL LINE 2'
[3]   'FRILL LINE 3'
[4]   'FRILL LINE 4'
[5]   'FRILL LINE 5'
[6]   ∇
      ∇T
[1]   FRILL
[2]   'T LINE 2'
[3]   'T LINE 3'
[4]   'T LINE 4'
[5]   'T LINE 5'
[6]   ∇
      1 ⎕STOP 'T'
1
      T
  77 STOPSET
 T[1]  FRILL
   ^
      )SIS
T[1] *      FRILL
      )STEP 4 /INTO
FRILL LINE 1
FRILL LINE 2
FRILL LINE 3
FRILL[4]   'FRILL LINE 4'
      )SIC
      T
  77 STOPSET
T[1]   FRILL
  ^
      )SIS
T[1] * FRILL
      )STEP 4 /OVER
FRILL LINE 1
FRILL LINE 2
FRILL LINE 3
FRILL LINE 4
FRILL LINE 5
T LINE 2
T LINE 3
T LINE 4
T[5]   'T LINE 5'
```

```
      )SIC
      T
  77 STOPSET
T[1]   FRILL
  ^
      )SIS
T[1] * FRILL
      )STEP 4 /OVER/SILENT
FRILL LINE 1
FRILL LINE 2
FRILL LINE 3
FRILL LINE 4
FRILL LINE 5
T LINE 2
T LINE 3
T LINE 4
      )SIS
T[5] * 'T LINE 5'
```

Possible Errors Generated

22 *INCORRECT PARAMETER (EXTRANEOUS CHARACTERS AFTER COMMAND)*

22 *INCORRECT PARAMETER (INVALID KEYWORD OR QUALIFIER)*

22 *INCORRECT PARAMETER (PARAMETER OUT OF RANGE)*

46 *OPERATION INVALID IN THIS CONTEXT*

) *VARS* Displaying a List of Variables

Type

Query System Command

Form

) *VARS* ⟦*start-string*⟦*stop-string*⟧⟧

Qualifiers

/*WSID*: ***wsname***
Allows you to specify the nonactive workspace APL uses to develop the list.

/*PASSWORD*: ***pw***
Specifies the password used to save the nonactive workspace.

Description

) *VARS* displays a list of global names used as variable names in a workspace. By default, APL displays the list from the currently active workspace. The optional /*WSID* qualifier allows you to specify a nonactive workspace. If the nonactive workspace is saved with a password, you must also specify the /*PASSWORD* qualifier.

The optional string parameters identify starting and stopping points for the list. When you specify the string parameters, you can use the ⋆ and ÷ wildcards. The objects are listed in ⎕*AV* order, separated by tabs. Each output line in the list begins in column 1.

Note that the wildcard determines the *start-string*. There is no wildcard for the *stop-string*.

If you use) *VARS* with no parameters, APL displays all the global variable names in the workspace:

```
      )VARS
A   I       J       K       N
```

If you include just one argument, APL uses Z̲ as the second string:

```
      )VARS K
K   N
```

To get a list of all variable names that begin with a given prefix, use the prefix for both arguments or use a wildcard:

```
      )VARS
PETER       STAN    STEVE   STUART  THOMAS  WILLIAM
      )VARS ST ST
STAN        STEVE   STUART
      )VARS ST*
STAN        STEVE   STUART
```

Possible Errors Generated

1 *FILE NOT FOUND (FILE NOT FOUND)*

22 *INCORRECT PARAMETER (EXTRANEOUS CHARACTERS AFTER COMMAND)*

22 *INCORRECT PARAMETER (FILE SPECIFICATION IS MISSING)*

22 *INCORRECT PARAMETER (INVALID KEYWORD OR QUALIFIER)*

22 *INCORRECT PARAMETER (NOT A LETTER)*

57 *FILE DOES NOT CONTAIN A WORKSPACE*

)*VERSION* Displaying the APL Version Number

Type

Query System Command

Form

```
)VERSION
```

Description

)*VERSION* displays the APL version number under which the currently active workspace was last saved, followed by a Carriage Return Line Feed, followed by the current version of the APL interpreter and a trailing <CR><LF>.

The display is in the following form:

lv.u-edit

l is the support letter
v is the version number
u is the update number
edit is the edit number

For example:

```
      )VERS
V3.2-834
V3.2-834
```

Possible Errors Generated

```
22  INCORRECT PARAMETER (EXTRANEOUS CHARACTERS AFTER COMMAND)
```

)*WIDTH* Output Width

Type

Query/Change System Command

Form

)*WIDTH* ⟦*n*⟧

Default in Clear Workspace

System setting

Description

)*WIDTH* displays or changes the setting of the print width system variable (⎕*PW*) and toggles the video screen between 80- and 132-column mode on some terminals (see below). The print width (*n* in the form) is the number of characters that can appear in an output line. The legal values are the integers from 35 through 2048.

The)*WIDTH* system command does not affect the allowable length of input lines. However, it does affect the display of error messages. Lines in the error message that are longer than the print width are truncated; they are not wrapped to the next line. If truncating a line would prevent APL from displaying the point in the line where the error was discovered, APL cuts part of the beginning of the line from the display so that the error is visible.

Executing the)*WIDTH* system command in change mode has the same effect as assigning a value to the ⎕*PW* system variable (see Chapter 2).

When you use)*WIDTH* to set the print width to above or below 80 on some terminals, APL toggles the video screen between 80- and 132-column mode. For example, setting the width to 80 or less toggles the screen to 80-column mode. Setting the width to 81 or more toggles the screen to 132-column mode. The affected terminals are the VT220, VT240, VT320, VT330, VT340, VT102, DECTERM, HDSAVT, HDS201, and HDS221. Setting ⎕*PW* does not cause this behavior.

Note that APL uses two font files for the VT240, VT320, VT330, and VT340 support: one for 80-column and the other for 132-column mode. If you suspend the APL session and change the terminal width at DCL level, the screen will be in the new mode and APL will be in the previous mode when you return to APL. Use the appropriate value to)*WIDTH* to correct it.

Examples:

```
      )WIDTH
132
      A←'THIS IS A TEST OF THE PRINT WIDTH VARIABLE'
      A
THIS IS A TEST OF THE PRINT WIDTH VARIABLE
      )WIDTH 35
WAS 132
      A
THIS IS A TEST OF THE PRINT WIDTH V
       ARIABLE
```

Possible Errors Generated

```
22 INCORRECT PARAMETER (EXTRANEOUS CHARACTERS AFTER COMMAND)

22 INCORRECT PARAMETER (ILL FORMED NUMERIC CONSTANT)

22 INCORRECT PARAMETER (PARAMETER OUT OF RANGE)
```

)WSID Workspace Identification

Type

Query/Change System Command

Form

)WSID ⟦*wsname*⟧

Default in Clear Workspace

CLEAR WS with a blank password

Qualifiers

/PASSWORD **⟦ :⟦*pw*⟧⟧**
Specifies the password associated with the workspace.

Description

)WSID displays or changes the name of the active workspace. When you use *)WSID* to change the name of the active workspace, you must specify the *wsname* parameter.

You can use the password qualifier of *)WSID* to change the password associated with a workspace. When you use *)WSID* as an action command, the password is changed (but not displayed) either to the password you specify as the argument to */PASSWORD* or, if you do not specify a password, to the empty password (eight blanks). The password is never changed when you use *)WSID* as a query command (*)WSID* with no argument). For example:

```
      )WSID MYWS/PASSWORD:SESAME
WAS EXAMPLE
      )WSID
MYWS
      )PASSWORD
/PASSWORD:SESAME
      )WSID YOURWS
WAS MYWS
      )PASSWORD
/PASSWORD:
```

The file specification you give for *wsname* must include at least a file name. APL assumes default values for the rest of the specification; that is, it assumes the file type .*APL*, the current user device and directory, and an empty password.

For more information on ⎕*LX*, see Chapter 2.

Possible Errors Generated

```
22 INCORRECT PARAMETER

22 INCORRECT PARAMETER (INVALID KEYWORD OR QUALIFIER)
```

A
System Messages

If an error is detected during the evaluation of an expression, APL displays the following:

- An appropriate primary error message from the list included in this appendix
- The text of the line in which the error occurred
- A caret (∧) approximately underneath the point in the line at which the error was discovered

Often the primary error message is followed on the same line by a secondary error message that offers a more specific explanation of what caused the error. Secondary error messages are surrounded by parentheses. (If you do not want to see secondary error messages, set ⎕*TERSE* to 1.)

When an expression that produces an error is executed by the ⎕*XQ* function, the result returned is an empty array with the shape 0 *n*, where *n* is an *ERROR NUMBER*. For example, the APL error number for *VALUE ERROR* is 11, so when an expression that produces a *VALUE ERROR* is executed by the ⎕*XQ* function, the value returned is an empty array with the shape 0 11:

```
      C←⎕XQ 'A+B'
      ⍴C
0 11
      ⎕ERROR
 11 ⎕XQ VALUE ERROR
      A+B
       ∧
```

The following pages list the primary error messages and, when appropriate, explain what they mean and what you can do to correct the errors. Some of the secondary error messages that APL may display with the primary messages are also identified. In many cases, no explanation of secondary error messages is given, because the message is self-explanatory.

1 *FILE NOT FOUND*

Explanation: The requested workspace or file was not found in the specified disk area.

1 *FILE NOT FOUND* (*FILE NOT FOUND*)

2 *SYSTEM ERROR*

Explanation: An internal inconsistency was detected. Please report this error to your Digital software specialist.

3 *WORKSPACE FULL*

Explanation: The active workspace could not retain all the information requested, nor could it expand further. Erase unneeded objects, issue a)*MAXCORE* command to enlarge the workspace, or do a)*SAVE*,)*CLEAR*, and)*COPY* sequence on the needed information.

3 *WORKSPACE FULL* (*EXCESSIVE FRAGMENTATION*)

3 *WORKSPACE FULL* (*MAXCORE EXCEEDED*)

3 *WORKSPACE FULL* (*VIRTUAL MEMORY EXHAUSTED*)

4 *NOT A VALID SYSTEM IDENTIFIER*

Explanation: An attempt was made to use a system identifier that is not supported by this APL implementation.

5 *DEFN ERROR*

Explanation: Invalid syntax was detected in a line or command entered in function-definition mode.

5 *DEFN ERROR* (*CANNOT DELETE HEADER*)

5 *DEFN ERROR* (*EDIT COMMAND ILLEGAL IN QUAD FX ARGUMENT*)

5 *DEFN ERROR* (*EXPECTING A DOLLAR SIGN*)

5 *DEFN ERROR* (*EXPECTING A NUMBER*)

5 *DEFN ERROR* (*EXPECTING A NUMBER, OR RIGHT BRACKET*)

5 *DEFN ERROR* (*EXPECTING A NUMBER, QUAD, DELTA, OR JOT*)

5 *DEFN ERROR* (*EXPECTING A QUAD, OR RIGHT BRACKET*)

5 *DEFN ERROR (EXPECTING A QUAD)*

5 *DEFN ERROR (EXPECTING A RIGHT BRACKET)*

Explanation: An error was discovered while the function editor scanned an edit command string.

5 *DEFN ERROR (EXPECTING A STRING DELIMITER)*

Explanation: Did not find a delimiter for one of the search or replace strings for dollar sign editing.

5 *DEFN ERROR (ILL FORMED LINE NUMBER)*

5 *DEFN ERROR (ILL FORMED NUMERIC CONSTANT)*

5 *DEFN ERROR (LEFT BRACKET EXPECTED)*

5 *DEFN ERROR (LINE NUMBER OUT OF RANGE)*

Explanation: A line number greater than 9,999 was specified.

5 *DEFN ERROR (LINE NUMBER TRUNCATED)*

Explanation: More than five decimal digits were specified in a line number.

5 *DEFN ERROR (LOCAL SYMBOL EXPECTED)*

5 *DEFN ERROR (NAME IN USE)*

Explanation: An attempt was made to use the same identifier for both arguments of an operation, or for both a label and a local symbol or argument.

5 *DEFN ERROR (NEGATIVE INTEGER NOT ALLOWED)*

5 *DEFN ERROR (NO PREVIOUS SEARCH STRING)*

Explanation: The search string is empty and there was no previous use of dollar sign editing during this activation of the Del editor.

5 *DEFN ERROR (NO SYMBOL AFTER OPENING DEL)*

Explanation: The operation name was missing from the line entered.

5 *DEFN ERROR (NO SYMBOL AFTER RESULT ARROW)*

5 *DEFN ERROR (NOT A SYSTEM VARIABLE)*

Explanation: An attempt was made to localize a system function.

5 *DEFN ERROR (NOT AN INTEGER)*

Explanation: A print position parameter that is not an integer was entered in superedit mode.

5 *DEFN ERROR (NOT IN FUNCTION DEFINITION MODE)*

Explanation: An edit command was entered outside of function-definition mode. Edit commands are illegal in immediate mode except when used to display or edit the last executed input line.

5 *DEFN ERROR (OPERATION LOCKED)*

Explanation: An attempt was made to list or change a locked operation.

5 *DEFN ERROR (OPERATION SUSPENDED, PENDENT, OR MONITORED)*

Explanation: An attempt was made to edit a pendent or monitored operation, or an attempt was made to change the number of lines in a suspended operation or the definition of a local symbol in a suspended operation.

5 *DEFN ERROR (OPERATION SUSPENDED OR PENDENT)*

Explanation: For)*EDIT*, an attempt was made to end the VAXTPU session with an EXIT command when you are not allowed to modify the function.

5 *DEFN ERROR (RIGHT BRACE EXPECTED)*

Explanation: An error was discovered while the function editor scanned an operation header and found a left brace that was not balanced with a right brace.

5 *DEFN ERROR (RIGHT PARENTHESIS EXPECTED)*

5 *DEFN ERROR (RIGHT PARENTHESIS OR SYMBOL EXPECTED)*

5 *DEFN ERROR (SEMICOLON EXPECTED)*

5 *DEFN ERROR (SYMBOL EXPECTED)*

5 *DEFN ERROR (TOO MANY LINES IN OPERATION)*

Explanation: An attempt was made to close an operation that has more than 10,000 lines.

5 *DEFN ERROR (UNEXPECTED CHARACTER IN HEADER)*

6 *LABEL ERROR*

Explanation: Improper use of a colon was detected, or an improper variable name was entered as a label.

6 *LABEL ERROR (DUPLICATE LABEL)*

6 *LABEL ERROR (NAME IN USE)*

Explanation: An attempt was made to use the same identifier for both a label and a local symbol or argument.

6 *LABEL ERROR (OPERATION SUSPENDED, PENDENT, OR MONITORED)*

Explanation: An attempt was made to change a label definition in a suspended, pendent, or monitored operation.

7 *SYNTAX ERROR*

Explanation: Invalid syntax was detected, such as an operation call with missing arguments, or an unmatched parenthesis.

7 *SYNTAX ERROR (ILL FORMED NUMERIC CONSTANT)*

7 *SYNTAX ERROR (MISMATCHED DELIMITERS)*

7 *SYNTAX ERROR (MISSING ARGUMENT)*

7 *SYNTAX ERROR (MISSING LEFT ARGUMENT TO ASSIGNMENT)*

Explanation: There is no left argument to the specification function (←). For example: ←2 is incorrect.

7 *SYNTAX ERROR (MISSING OPERAND)*

7 *SYNTAX ERROR (NO DYADIC FORM OF FUNCTION)*

7 *SYNTAX ERROR (NO MONADIC FORM OF FUNCTION)*

Explanation: Inner product and outer product are dyadic.

7 *SYNTAX ERROR (OPERATOR HAS NO OPERANDS)*

7 *SYNTAX ERROR (UNBALANCED DELIMITER)*

7 *SYNTAX ERROR (BRANCH NOT ALLOWED IN MIDDLE OF AN EXPRESSION)*

Explanation: The branch (→) function was used when it was not the principal function of a statement.

7 SYNTAX ERROR (DEPTH ERROR)

Explanation: Either there are too many nested parentheses or brackets, or the expression is too complex for APL to parse.

7 SYNTAX ERROR (ILLEGAL CHARACTER IN EXPRESSION)

Explanation: An internal ⎕AV code appeared outside of a literal or comment.

7 SYNTAX ERROR (NO DYADIC FORM OF DERIVED FUNCTION)

Explanation: Scan, reduction, expansion, compression, and replication all derive monadic functions.

7 SYNTAX ERROR (NO MONADIC FORM OF DERIVED FUNCTION)

Explanation: Inner and outer product both derive dyadic functions.

7 SYNTAX ERROR (NON-NILADIC FUNCTION HAS NO ARGUMENTS)

Explanation: An ambivalent, dyadic, or monadic user-defined operation was invoked without any arguments.

7 SYNTAX ERROR (NOT IN FUNCTION DEFINITION MODE)

Explanation: An editing command was entered at the beginning of a line in immediate mode.

7 SYNTAX ERROR (SUBSCRIPT NOT ALLOWED)

Explanation: An attempt was made to index something that does not have a value.

7 SYNTAX ERROR (WRONG NUMBER OF ARGUMENTS TO USER FUNCTION)

Explanation: A monadic user-defined operation was invoked with two arguments.

8 ERROR RETURNING FROM EXTERNAL ROUTINE

8 ERROR RETURNING FROM EXTERNAL ROUTINE (DOMAIN ERROR)

Explanation: A conversion failed when data returned to the workspace.

8 ERROR RETURNING FROM EXTERNAL ROUTINE (ILLEGAL ASCII CHARACTER)

Explanation: A conversion to ASCII failed as character data returned to the workspace.

8 *ERROR RETURNING FROM EXTERNAL ROUTINE (LENGTH ERROR)*

Explanation: A Varying string (*/TYPE:VT*) returned to the WS is bigger than it was when it was passed to the external routine. (It is allowed to be smaller or the same size.)

9 *RANK ERROR*

Explanation: The ranks of two operands did not conform.

9 *RANK ERROR (ITEMS NOT SCALAR OR ALL THE SAME RANK)*

Explanation: The items of the right argument of disclose (⊃) are neither scalars nor of matching rank.

9 *RANK ERROR (ITEMS NOT SINGLETON OR ALL THE SAME RANK)*

Explanation: The items of *B* must be either singletons or of matching rank.

9 *RANK ERROR (LEFT ITEM NOT VECTOR DOMAIN)*

Explanation: Either the left argument or an item in the left argument to pick (⊃) is not a singleton and its rank is greater than 1.

9 *RANK ERROR (MUST BE VECTOR)*

Explanation: The value and each item in the value, must be vectors.

9 *RANK ERROR (NOT A SCALAR, VECTOR, OR MATRIX)*

Explanation: The rank of an argument to ⌹, ⍒ or ⍋ is greater than 2.

9 *RANK ERROR (NOT MATRIX DOMAIN)*

9 *RANK ERROR (NOT SINGLETON)*

Explanation: Deal, and the ⎕*WAIT* and ⎕*DL* functions accept only single numbers as an argument.

9 *RANK ERROR (NOT VECTOR DOMAIN)*

Explanation: An argument or value is not a singleton and its rank is greater than 1.

9 *RANK ERROR (NUMERIC PRIMARY KEY MUST BE SINGLETON)*

Explanation: A numeric key for a keyed file must be a singleton.

9 *RANK ERROR (RANKS DIFFER BY MORE THAN ONE)*

Explanation: The arguments, after singleton extension to catenate or rotate differ in rank by more than one.

10 *LENGTH ERROR*

Explanation: The shapes of two operands did not conform.

10 *LENGTH ERROR* (*ARGUMENT MUST BE* 1 *OR* 2 *ELEMENTS*)

Explanation: ⍇, ⍈, ⎕*CIQ*, and ⎕*COQ* may have at most two items in their right argument.

10 *LENGTH ERROR* (*ARGUMENT STRING IS TOO LONG*)

Explanation: The left argument to dyadic ⍋ or ⍒ is greater than 256 characters along any one axis.

10 *LENGTH ERROR* (*DATA TYPE EXCEEDS DATA LENGTH*)

Explanation: The data type specified for ⍇ file input or the ⎕*CIQ* function is incompatible with the length of the left argument.

10 *LENGTH ERROR* (*DATA TYPE MISSING*)

Explanation: The data type parameter in the right argument to ⎕*CIQ* is required in this case.

10 *LENGTH ERROR* (*DISPLAY CONTROL ITEM WRONG LENGTH*)

Explanation: The first item must have length 4. The second item can either be empty or have length 8.

10 *LENGTH ERROR* (*DISPLAY CONTROL VECTOR MUST BE TWO ITEMS*)

Explanation: The value must have length 2.

10 *LENGTH ERROR* (*ILLEGAL EMPTY ARGUMENT*)

Explanation: An empty argument was used with ⎕*FMT*, ⎕*MAP*, ⎕*QCO*, ⎕*QLD*, ⎕*QPC*, or ⎕*SIGNAL*.

10 *LENGTH ERROR* (*INDEX LESS THAN INDEX ORIGIN*)

Explanation: An index is less than the current setting of ⎕*IO*.

10 *LENGTH ERROR* (*INDEX OUT OF RANGE*)

Explanation: For pick (⊃), an element of the left argument exceeds the length of the corresponding axis of an item of the right argument.

10 *LENGTH ERROR* (*ITEM COUNT MISMATCH*)

Explanation: If the number of variable names specified in the right argument to ⎕*PACK*, is not equal to the number of packets contained in the left argument.

10 *LENGTH ERROR (KEY VALUE TOO LARGE FOR KEY SIZE)*

Explanation: For */KY* files.

10 *LENGTH ERROR (LEFT ARGUMENT LENGTH GREATER THAN RIGHT ARGUMENT DEPTH)*

Explanation: For pick (⊃), the length of the left argument is greater than the depth of the right argument.

10 *LENGTH ERROR (LEFT ITEM LENGTH NOT EQUAL TO SELECTED ITEM RANK)*

Explanation: For pick (⊃), the length of an item of the left argument does not match the rank of the selected item at the corresponding depth of the right argument.

10 *LENGTH ERROR (LEFT LENGTH NOT EQUAL TO RIGHT RANK)*

Explanation: For ↑, ↓, or dyadic ⍉, where no axis has been specified and the right argument *B* is not a scalar and its rank is not equal to the length of the left argument *A*.

10 *LENGTH ERROR (LENGTHS OF INNER AXES DO NOT MATCH)*

Explanation: For Base, the length of the last axis of the left argument *A* is not equal to the length of the first axis of the right argument *B*, and neither axis is 1. For Inner product, after singleton extension, the left argument last axis length must equal the right argument first axis length.

10 *LENGTH ERROR (NOT SINGLETON)*

Explanation: The value is not a single item. Dyadic ⍈ (for */AS* files), *?*, and the numeric system variables require a single item for their argument or value. For example, the following is incorrect: ⎕*IO*←⍳ 3

10 *LENGTH ERROR (NUMBER OF ROWS MUST MATCH)*

Explanation: The number of rows in the arguments to dyadic ⌹ must match.

10 *LENGTH ERROR (SHAPES OFF AXIS DO NOT MATCH)*

Explanation: For Catenate and Rotate, after singleton extension, the shape of the left argument must match the shape of the right argument except along the specified axis.

10 *LENGTH ERROR (THERE ARE FEWER ROWS THAN COLUMNS)*

Explanation: The number of rows in the right argument to ⌹ must be greater than or equal to the number of columns.

10 *LENGTH ERROR (TOO MANY ELEMENTS IN KEY SPECIFICATION)*

Explanation: An attempt was made to include elements other than *value*, *key-num*, *tech*, and *key-type* between the brackets of your file output expression.

11 *VALUE ERROR*

Explanation: A variable name was used and has not been assigned a value, or a user-defined operation that should return a value was executed and it did not return a value.

11 *VALUE ERROR (BRANCH HAS NO RESULT)*

Explanation: A branch (→) expression was used as a response to ⎕ input.

11 *VALUE ERROR (FUNCTION DOES NOT RETURN A RESULT)*

11 *VALUE ERROR (FUNCTION RESULT UNDEFINED)*

11 *VALUE ERROR (NO VALUE TO ASSIGN)*

Explanation: There is no right argument to the specification function (←). For example: *T*← is incorrect.

11 *VALUE ERROR (REQUIRED VALUE NOT SUPPLIED)*

11 *VALUE ERROR (SUBSCRIPTED NAME IS UNDEFINED)*

Explanation: In the form *A*[*K*]←*B*, *A* is not a defined name.

14 *DEPTH ERROR*

Explanation: For ⎕*FMT* there are more than eight nested parentheses in *A*.

14 *DEPTH ERROR (LEFT ARGUMENT DEPTH GREATER THAN* 2)

Explanation: The items in *A* must be simple (vectors or singletons).

14 *DEPTH ERROR (TOO MANY DIVERTED INPUTS)*

Explanation: Files were nested to a depth greater than 10 with)*INPUT*.

15 *DOMAIN ERROR*

Explanation: The values given for the arguments were outside of the function domain. For ⎕*OM*, the argument is not Boolean and is nonempty.

15 *DOMAIN ERROR (BUFFER OVERFLOW)*

15 *DOMAIN ERROR (CANNOT MODIFY SELECTIVE ASSIGNMENT TARGET)*

Explanation: The variable being assigned to cannot be modified by the expression forming the left argument of the selective assignment. For example: `((ρA←ι2)⌽A)←'AB'` is incorrect.

15 *DOMAIN ERROR (CANNOT SIGNAL EOF)*

Explanation: An attempt was made to use 75 as the right argument to ⎕*SIGNAL*. ⎕*SIGNAL* does not accept 75 as a right argument.

15 *DOMAIN ERROR (CHANNEL NOT ASSIGNED)*

Explanation: The value in the right argument does not refer to an assigned channel. An attempt was made to use ⎕*WAIT* or ⎕*REWIND* on an unassigned channel.

15 *DOMAIN ERROR (CHANNEL NOT ASSIGNED TO A KEYED FILE)*

Explanation: The file associated with the channel number is not a */KY* file.

15 *DOMAIN ERROR (CHARACTER KEY TOO LONG OR NOT IN VECTOR DOMAIN)*

Explanation: For */KY* files.

15 *DOMAIN ERROR (CONFLICTING QUALIFIERS SPECIFIED)*

Explanation: More than one of the following qualifiers was specified in the argument to ⎕*ASS*: */READONLY*, */WRITEONLY*, or */UPDATE.*

15 *DOMAIN ERROR (DATA TYPE MUST BE UNSPECIFIED OR ZERO)*

Explanation: For ⎕*CIQ*.

15 *DOMAIN ERROR (DELETION NOT ALLOWED)*

Explanation: A sequential delete was attempted for a */KY* or */AS* file.

15 *DOMAIN ERROR (DIVISION BY ZERO)*

Explanation: Division by zero is attempted.

15 *DOMAIN ERROR (DUPLICATE FMT QUALIFIER)*

Explanation: A qualifier is used more than once with a particular format phrase.

15 *DOMAIN ERROR (DUPLICATE FMT STANDARD SUBSTITUTION CHARACTER)*

Explanation: A substitute for a standard symbol character was specified more than once.

15 *DOMAIN ERROR* (*EMPTY FMT STRING PARAMETER NOT ALLOWED*)

Explanation: The O, R, or S qualifier string is empty.

15 *DOMAIN ERROR* (*ENCLOSED/HETEROGENEOUS ARRAY NOT ALLOWED*)

Explanation: The argument is not a simple, homogeneous array. For ⎕*DC*, the first item must be a simple homogeneous array. The second item, if not empty, must be simple.

15 *DOMAIN ERROR* (*ENCLOSED VALUE REQUIRED*)

Explanation: The value must be an enclosed array.

15 *DOMAIN ERROR* (*ERROR ACTIVATING IMAGE*)

Explanation: For ⎕*MAP*, the shared image named by *B* does not exist. For ⎕*TT*,)*EDIT*, or the initialization stream, there is an attempt to enter VT220, VT240, VT320, VT330, VT340 or DECterm mode when SYS$SYSTEM:APLSHR is not accessible. APL can signal this error when you invoke APL with the /*TERMINAL* qualifier, when you use)*EDIT* with the /*TERMINAL* qualifier, when you use)*EDIT* with an HDS201 or HDS221 terminal, or when you set ⎕*TT*.

15 *DOMAIN ERROR* (*ERROR PARSING ARGUMENT TO BLOCK SIZE*)

Explanation: An error was discovered when parsing the /*BLOCKSIZE* qualifier in the argument to ⎕*ASS*.

15 *DOMAIN ERROR* (*ERROR PARSING ARGUMENT TO BUFFER COUNT*)

Explanation: An error was discovered when parsing the /*BUFFERCOUNT* qualifier in the argument to ⎕*ASS*.

15 *DOMAIN ERROR* (*ERROR PARSING ARGUMENT TO CCONTROL*)

Explanation: An invalid value was specified for the /*CCONTROL* qualifier in the argument to ⎕*ASS*.

15 *DOMAIN ERROR* (*ERROR PARSING ARGUMENT TO DEFAULT FILE SPEC*)

Explanation: An error was discovered when parsing the /*DEFAULTFILE* qualifier in the argument to ⎕*ASS*.

15 *DOMAIN ERROR* (*ERROR PARSING ARGUMENT TO DISPOSE*)

Explanation: An error was discovered when parsing the /*DISPOSE* qualifier in the argument to ⎕*ASS*.

15 *DOMAIN ERROR* (*ERROR PARSING ARGUMENT TO EVENT FLAG*)

Explanation: An error was discovered when parsing the */EFN* qualifier in the argument to ⎕*ASS*.

15 *DOMAIN ERROR* (*ERROR PARSING ARGUMENT TO KEY SPECIFICATION*)

Explanation: An error was discovered when parsing the */KY* qualifier in the argument to ⎕*ASS*.

15 *DOMAIN ERROR* (*ERROR PARSING ARGUMENT TO MAXLEN*)

Explanation: An error was discovered when parsing the */MAXLEN* qualifier in the argument to ⎕*ASS*.

15 *DOMAIN ERROR* (*ERROR PARSING ARGUMENT TO PROTECTION*)

Explanation: An error was discovered when parsing the */PROTECTION* qualifier in the argument to ⎕*ASS*.

15 *DOMAIN ERROR* (*ERROR PARSING ARGUMENT TO RECORD TYPE*)

Explanation: An error was discovered when parsing the */RECORDTYPE* qualifier in the argument to ⎕*ASS*.

15 *DOMAIN ERROR* (*EXTRANEOUS CHARACTERS AFTER COMMAND*)

Explanation: There are characters other than spaces following the command.

15 *DOMAIN ERROR* (*FILE IS ASSIGNED WRITE ONLY*)

Explanation: The file associated with the channel number cannot be rewound because it was assigned with the */WRITEONLY* qualifier.

15 *DOMAIN ERROR* (*FILE SPECIFICATION IS MISSING*)

Explanation: There is no file specification or default file specification in the argument to ⎕*ASS*.

15 *DOMAIN ERROR* (*FMT DECORATION OR LITERAL STRING TOO LONG*)

Explanation: A text string in the left argument consists of more than 255 characters.

15 *DOMAIN ERROR* (*FMT RIGHT ARGUMENT DOES NOT MATCH FORMAT PHRASE*)

Explanation: The data type of a value in the right argument does not match the type called for by a format phrase specification in the left argument.

15 *DOMAIN ERROR (FONT FILE COULD NOT BE OPENED)*

Explanation: For ⎕*TT* or the initialization stream, there is an attempt to enter VT220, VT240, VT320, VT330 or VT340 mode when the APL font file is not accessible. Possibly, the file does not exist or is associated with a protection code that does not allow access.

15 *DOMAIN ERROR (FUNCTION HAS NO FILL ITEM)*

Explanation: Either each (¨) or outer product (∘.*f*) was applied with a user-defined function to an empty argument.

15 *DOMAIN ERROR (FUNCTION HAS NO IDENTITY ELEMENT)*

Explanation: The inner axes of an inner product or the reduction axis is empty and there is no identity element for the left operand function.

15 *DOMAIN ERROR (FUNCTION MISSING)*

Explanation: For ⎕*MAP*, if *function-name* is not present or if it is followed by any attributes.

15 *DOMAIN ERROR (ILL FORMED FMT PARAMETER)*

Explanation: An invalid numeric parameter (such as a negative sign with no number) was found.

15 *DOMAIN ERROR (ILL FORMED NAME)*

Explanation: For ⎕*MAP*, if the left argument has a formal parameter that contains illegal characters, or if the right argument has a value for the */ENTRY* or */VALUE* qualifier that contains illegal characters.

15 *DOMAIN ERROR (ILLEGAL ASCII CHARACTER)*

15 *DOMAIN ERROR (ILLEGAL CHARACTER IN FMT LEFT ARGUMENT)*

Explanation: An invalid character appears in the left argument of ⎕*FMT*.

15 *DOMAIN ERROR (ILLEGAL COMPOSITE CHARACTER)*

15 *DOMAIN ERROR (ILLEGAL DATA TYPE CONVERSION)*

15 *DOMAIN ERROR (ILLEGAL DEC MULTINATIONAL CHARACTER)*

15 *DOMAIN ERROR (ILLEGAL EMPTY ARGUMENT)*

15 *DOMAIN ERROR (ILLEGAL FMT FORMAT PHRASE)*

Explanation: A letter in the left argument of ⎕*FMT* does not represent a valid format phrase or qualifier.

15 *DOMAIN ERROR* (*ILLEGAL FMT G FORMAT PHRASE PATTERN CHARACTER*)

Explanation: An invalid character was found in a type G format phrase pattern string.

15 *DOMAIN ERROR* (*ILLEGAL FMT LITERAL STRING DELIMITER*)

Explanation: A decorator or literal string delimiter was invalid.

15 *DOMAIN ERROR* (*ILLEGAL FMT S QUALIFIER SYMBOL*)

Explanation: The first symbol of a substitution pair is not ⋆, ., ,, 0, 9, *Z*, or @.

15 *DOMAIN ERROR* (*ILLEGAL ISO* 8*BIT CHARACTER*)

15 *DOMAIN ERROR* (*ILLEGAL LEFT ARGUMENT TO ASSIGNMENT*)

Explanation: An element of *A* is not an undefined or variable name.

15 *DOMAIN ERROR* (*ILLEGAL MODE*)

15 *DOMAIN ERROR* (*ILLEGAL NAME CLASS*)

Explanation: For ⎕*PACK*, the right argument is not a variable. For assignment (←), the left argument is neither a variable nor an undefined name.

15 *DOMAIN ERROR* (*ILLEGAL SELECTIVE ASSIGNMENT FUNCTION*)

Explanation: The function *f* is not one of the allowed selection functions.

15 *DOMAIN ERROR* (*ILLEGAL USE OF FMT QUALIFIER*)

Explanation: The specified qualifier and format phrase are incompatible.

15 *DOMAIN ERROR* (*INCORRECT PARAMETER*)

Explanation: A parameter in the left argument to ⎕*MAP* is incorrect.

15 *DOMAIN ERROR* (*INCORRECT TYPE*)

Explanation: An argument is non-empty and is either numeric, when it should be character, or character when it should be numeric. For example, the following is incorrect: ⎕*IO*←'*G*'

15 *DOMAIN ERROR* (*INDEX LESS THAN INDEX ORIGIN*)

Explanation: An element of an argument is less than the current setting of ⎕*IO*.

15 *DOMAIN ERROR* (*INDEX OUT OF RANGE*)

Explanation: An element of the left argument exceeds the length of the corresponding axis of an item of the right argument.

15 *DOMAIN ERROR* (*INTEGER OVERFLOW*)

15 *DOMAIN ERROR* (*INVALID CHANNEL NUMBER*)

Explanation: A channel number is not between ¯ 999 and 999 or is 0.

15 *DOMAIN ERROR* (*INVALID CIQ HEADER*)

15 *DOMAIN ERROR* (*INVALID EXTERNAL DATA TYPE*)

15 *DOMAIN ERROR* (*INVALID FILE SPECIFICATION*)

Explanation: There is an error in the shared image file specification in the right argument of ⎕*MAP*.

15 *DOMAIN ERROR* (*INVALID FUNCTION IN SELECTIVE ASSIGNMENT*)

Explanation: The principal function or functions in the left argument is ineligible for use with selective assignment. For example: (*A*+*B*) ← 3 is incorrect.

15 *DOMAIN ERROR* (*INVALID HEADER TYPE*)

Explanation: An incorrect header type was specified for ⎕*COQ* or ⎕*CIQ*.

15 *DOMAIN ERROR* (*INVALID KEYED FILE PURE DATA TYPE*)

Explanation: For /*KY* files.

15 *DOMAIN ERROR* (*INVALID LENGTH IN PACK HEADER*)

Explanation: The first item of the value in the left argument to ⎕*PACK* must equal the length of the left argument.

15 *DOMAIN ERROR* (*INVALID OBJECT IN INDEXED ASSIGNMENT*)

15 *DOMAIN ERROR* (*INVALID OBJECT IN SELECTIVE ASSIGNMENT*)

Explanation: The first object inside the parentheses of selective assignment must be a variable name. For example: (1↑ 2) ← 3 is incorrect.

15 *DOMAIN ERROR* (*INVALID OBJECT IN STRAND ASSIGNMENT*)

15 *DOMAIN ERROR (INVALID PACK HEADER)*

Explanation: The length of the left argument to ⎕*PACK* must be greater than or equal to 8. The shortest possible packed data has four elements for the ⎕*PACK* header and 4 elements for the shortest ⎕*COQ* header.

15 *DOMAIN ERROR (INVALID RANK IN PACK HEADER)*

Explanation: The value of the third element in the left argument to ⎕*PACK* must equal 1 (1 means the packed data is a vector).

15 *DOMAIN ERROR (INVALID RHO VECTOR IN PACK HEADER)*

Explanation: The length of the left argument to ⎕*PACK* must equal the value of the fourth element in the left argument plus 4.

15 *DOMAIN ERROR (INVALID TYPE IN PACK HEADER)*

Explanation: The value of the second element in the left argument to ⎕*PACK* must equal 1 (1 means the type is integer).

15 *DOMAIN ERROR (INVALID WATCH MODE)*

Explanation: An incorrect mode was specified for ⎕*WATCH*.

15 *DOMAIN ERROR (KEY OF REFERENCE OUT OF RANGE OR NOT A NUMERIC SINGLETON)*

Explanation: An attempt was made to specify a key of reference that is not a numeric singleton or that is less than 0 or greater than 254 (inclusive).

15 *DOMAIN ERROR (KEY NOT FOUND IN TREE)*

Explanation: For ⎕*MAP*, if the left argument specifies an entry point that does not exist in the shared image.

15 *DOMAIN ERROR (LEFT ARGUMENT NOT DENSE FROM INDEX ORIGIN)*

Explanation: For dyadic ⍉, the left argument is not a dense sequence beginning at ⎕*IO*.

15 *DOMAIN ERROR (MISSING FMT FORMAT PHRASE SEPARATOR)*

Explanation: A format phrase separator (such as a comma or parenthesis) was expected but not supplied.

15 *DOMAIN ERROR (MISSING FMT FORMAT PHRASE/QUALIFIER CHARACTER)*

Explanation: A format phrase or qualifier was expected but not supplied.

15 *DOMAIN ERROR* (*MISSING FMT FORMAT PHRASE/QUALIFIER PARAMETER*)

Explanation: No string was included with a decorator or an S format phrase; no number was included where a width or decimal parameter was required; or no number was included with a K or W qualifier.

15 *DOMAIN ERROR* (*MISSING LITERAL STRING IN FMT LEFT ARGUMENT*)

Explanation: The text string parameter was missing from a decorator.

15 *DOMAIN ERROR* (*NAME IN USE*)

Explanation: For ⎕*MAP*, if the name specified for *function-name* is already defined as an object other than a function.

15 *DOMAIN ERROR* (*NEGATIVE INTEGER NOT ALLOWED*)

15 *DOMAIN ERROR* (*NEGATIVE NUMBER NOT ALLOWED*)

Explanation: The value of the argument is less than 0.

15 *DOMAIN ERROR* (*NO DIGIT SELECTOR IN FMT G FORMAT PHRASE PATTERN*)

Explanation: A type G format phrase pattern does not contain at least one 9 or one Z, or a character that is substituted for a 9 or a Z.

15 *DOMAIN ERROR* (*NO FMT EDITING FORMAT PHRASE*)

Explanation: The left argument of ⎕*FMT* does not contain at least one value editing format phrase, that is, at least one of type A, I, E, F, G, or Y.

15 *DOMAIN ERROR* (*NOT A LETTER*)

Explanation: A nonletter was used as the left argument to ⎕*NL*.

15 *DOMAIN ERROR* (*NOT A SYSTEM VARIABLE*)

Explanation: The argument is a quad name but not a system variable.

15 *DOMAIN ERROR* (*NOT A VALID SYSTEM IDENTIFIER*)

15 *DOMAIN ERROR* (*NOT AN EXTERNAL FUNCTION*)

Explanation: For ⎕*MAP*, if the argument names an illegal identifier, a system identifier, a name with no value, or a name that is not an external function.

15 *DOMAIN ERROR* (*NOT AN INTEGER*)

Explanation: An argument is not a near-integer. For example, the following is incorrect: ⎕*IO*←2.5

15 *DOMAIN ERROR (OPERATION SUSPENDED, PENDENT, OR MONITORED)*

15 *DOMAIN ERROR (PARAMETER OUT OF RANGE)*

Explanation: An attempt was made to use an unavailable value as the argument. For ⎕*DC*, Elements 1 and 2 of the first item can only be ¯ 1, 0, or 1. For ⎕*FMT*, the repetition count, field width, number of decimal places or significant digits, column position, scale factor, or exponent size is out of range.

15 *DOMAIN ERROR (REDUNDANT KEYWORD OR QUALIFIER)*

Explanation: A keyword or qualifier was repeated in the argument to ⎕*ASS*.

15 *DOMAIN ERROR (RIGHT ARGUMENT IS LESS THAN LEFT)*

Explanation: For dyadic *?*.

15 *DOMAIN ERROR (RIGHT ARG TOO DEEPLY NESTED)*

Explanation: The right argument to ⎕*FMT* is not a vector domain of simple arrays.

15 *DOMAIN ERROR (SEMICOLON LIST NOT ALLOWED)*

Explanation: A semicolon list was used as an argument to a primitive function.

15 *DOMAIN ERROR (SINGULAR MATRIX)*

Explanation: For ⌹, division by 0 is attempted.

15 *DOMAIN ERROR (SYSTEM VARIABLE MUST BE* 0 *OR* 1 *OR* 2 *OR* 3)

Explanation: The value of ⎕*GAG* must be 0, 1, 2, or 3.

15 *DOMAIN ERROR (SYSTEM VARIABLE VALUE MAY ONLY BE* 0 *OR* 1)

Explanation: ⎕*IO*, ⎕*NG*, ⎕*TERSE*, ⎕*TIMEOUT*, or ⎕*TLE* accept only 0 or 1.

15 *DOMAIN ERROR (TIMEOUT READ UNSUPPORTED FOR CURRENT VALUE OF QUAD TT)*

Explanation: An attempt was made to set ⎕*TIMELIMIT* while the current value of ⎕*TT* indicates a VT220, VT240, VT320, VT330, VT340 or DECterm terminal.

15 *DOMAIN ERROR (UNBALANCED PARENS IN FMT LEFT ARGUMENT)*

Explanation: The parentheses in the left argument of ⎕*FMT* are not nested properly.

15 *DOMAIN ERROR (UNBALANCED TEXT DELIMITER IN FMT LEFT ARGUMENT)*

Explanation: The closing delimiter for a text string was not compatible with the opening delimiter.

15 *DOMAIN ERROR (UNPAIRED SYMBOL IN FMT S QUALIFIER)*

Explanation: The length of the standard symbol substitution string is not even.

15 *DOMAIN ERROR (UNRECOGNIZED SEARCH MODE)*

Explanation: For */KY* files.

15 *DOMAIN ERROR (UNSUCCESSFUL TRAP IN LOCKED FUNCTION)*

Explanation: An error occurred while executing the trap expression in a locked function.

15 *DOMAIN ERROR (WIDTH TOO SMALL)*

Explanation: The width parameter for dyadic ⍕ is too small to accommodate the data.

15 *DOMAIN ERROR (WILDCARDS NOT ALLOWED IN FILE SPECIFICATION)*

Explanation: Wildcards are not allowed in the right argument to ⎕*MAP*.

16 *UNBALANCED DELIMITER*

Explanation: An input line has unbalanced parentheses, or the argument to the execute function contains unbalanced quotation marks or ∆ characters.

17 *EDIT ERROR*

Explanation: An improper character editing request was entered.

17 *EDIT ERROR (COLUMN POSITION OUT OF RANGE)*

Explanation: The print position number that was entered for superedit was greater than the page width, or was negative.

17 *EDIT ERROR (EXPECTING A RIGHT BRACKET)*

Explanation: An attempt was made to delete the line number during line editing.

17 *EDIT ERROR (ILL FORMED LINE NUMBER)*

`17` *EDIT ERROR (ILLEGAL CHARACTER IN LINE EDIT COMMAND)*

Explanation: The command that was entered included a character other than a letter, digit, /, space, or backspace.

`17` *EDIT ERROR (LEFT BRACKET MISSING)*

`17` *EDIT ERROR (LINE EDITING NOT ALLOWED IN EXECUTE)*

`17` *EDIT ERROR (NONEXISTENT LINE)*

`17` *EDIT ERROR (PREVIOUS INPUT LINE EMPTY)*

`17` *EDIT ERROR (OPERATION SUSPENDED, PENDENT, OR MONITORED)*

Explanation: An attempt was made to make an illegal change to a suspended, pendent, or monitored operation.

`18` *ATTENTION SIGNALED*

Explanation: The attention signal was detected during operation execution.

`19` *DEVICE DOES NOT EXIST*

Explanation: An invalid device specification was entered.

`20` *DEVICE NOT AVAILABLE*

Explanation: The requested device has already been assigned to another process.

`21` *INCORRECT COMMAND*

Explanation: A system command was entered improperly.

`21` *INCORRECT COMMAND (AMBIGUOUS ABBREVIATION)*

Explanation: Not enough characters of a system command were entered to distinguish it from other commands.

`21` *INCORRECT COMMAND (MISSING SYSTEM COMMAND)*

Explanation: A right parenthesis was entered at the beginning of a line and was not followed by a known system command.

`21` *INCORRECT COMMAND (NO SUCH SYSTEM COMMAND)*

`22` *INCORRECT PARAMETER*

Explanation: Invalid syntax was specified for a recognized system command.

`22 INCORRECT PARAMETER (ARGUMENT STRING IS TOO LONG)`

Explanation: The argument entered for `)DO` or `)PUSH` was more than 2096 keystrokes.

`22 INCORRECT PARAMETER (CURRENT WORKSPACE CLEARED)`

Explanation: APL failed to load the requested workspace.

`22 INCORRECT PARAMETER (EXTRANEOUS CHARACTERS AFTER COMMAND)`

Explanation: Extra characters were entered after all the required parameters for a system command.

`22 INCORRECT PARAMETER (ILL FORMED NAME)`

Explanation: In the argument to `)ERASE` or `)GROUP`.

`22 INCORRECT PARAMETER (ILL FORMED NUMERIC CONSTANT)`

Explanation: A numeric argument to a system command was entered improperly.

`22 INCORRECT PARAMETER (ILLEGAL ASCII CHARACTER)`

Explanation: An illegal character was used in the argument to `)PUSH`.

`22 INCORRECT PARAMETER (ILLEGAL NAME CLASS)`

Explanation: A label or system object was used in the argument to `)GROUP`.

`22 INCORRECT PARAMETER (INVALID CHARACTER SET QUALIFIER)`

Explanation: An invalid qualifier was used in the argument to `)INPUT` or `)OUTPUT`.

`22 INCORRECT PARAMETER (INVALID KEYWORD OR QUALIFIER)`

Explanation: An invalid keyword or qualifier was used in the argument to `)INPUT`, `)OUTPUT`, `)SAVE`, or `)STEP`.

`22 INCORRECT PARAMETER (LINE TOO LONG TO TRANSLATE)`

Explanation: The argument entered for `)DROP` or `)LIB` was greater than approximately 2048 keystrokes.

`22 INCORRECT PARAMETER (LOWERCASE QUALIFIER REPEATED)`

Explanation: An invalid repetition of `/LOWERCASE` was used in the argument to `)DO` or `)PUSH`.

22 *INCORRECT PARAMETER (MISSING ARGUMENT)*

Explanation: An argument was not supplied for a system command that should have one.

22 *INCORRECT PARAMETER (NOKEYPAD QUALIFIER REPEATED)*

Explanation: An invalid repetition of */NOKEYPAD* was used in the argument to *)DO* or *)PUSH.*

22 *INCORRECT PARAMETER (NOLOGICALS QUALIFIER REPEATED)*

Explanation: An invalid repetition of */NOLOGICALS* was used in the argument to *)DO* or *)PUSH.*

22 *INCORRECT PARAMETER (NOSYMBOLS QUALIFIER REPEATED)*

Explanation: An invalid repetition of */NOSYMBOLS* was used in the argument to *)DO* or *)PUSH.*

22 *INCORRECT PARAMETER (NOT A GROUP)*

Explanation: An attempt was made to display the contents of a nongroup.

22 *INCORRECT PARAMETER (NOT A LETTER)*

Explanation: The argument to *)NMS*, *)VARS*, *)FNS*, or *)GRPS* was not a letter.

22 *INCORRECT PARAMETER (NOTIFY QUALIFIER REPEATED)*

Explanation: An invalid repetition of */NOTIFY* was used in the argument to the *)PUSH* command.

22 *INCORRECT PARAMETER (NOWAIT QUALIFIER REPEATED)*

Explanation: An invalid repetition of */NOWAIT* was used in the argument to the *)PUSH* command.

22 *INCORRECT PARAMETER (PARAMETER OUT OF RANGE)*

Explanation: The numeric argument entered for a system command was outside the legal range of values for the command. The ranges are:

For *)DIGITS*, 1 to 16
For *)WIDTH*, 35 to 2048
For *)MAXCORE*, the *)MINCORE* value to 1048576
For *)MINCORE*, 0 to the *)MAXCORE* value
For *)SAVE/MAXLEN*, 512 to 2048

22 *INCORRECT PARAMETER (PARENT QUALIFIER REPEATED)*

Explanation: In the)*ATTACH* command.

22 *INCORRECT PARAMETER (PROCESS NAME QUALIFIER REPEATED)*

Explanation: In the)*PUSH* command.

22 *INCORRECT PARAMETER (REDUNDANT KEYWORD OR QUALIFIER)*

Explanation: A keyword or qualifier was repeated in the argument to)*OUTPUT*,)*STEP*, or ⎕*ASS*.

22 *INCORRECT PARAMETER (SYSTEM VARIABLE VALUE MAY ONLY BE* 0 *OR* 1)

Explanation: In the)*ORIGIN* command.

22 *INCORRECT PARAMETER (UNRECOGNIZED QUALIFIER KEYWORD)*

22 *INCORRECT PARAMETER (WILDCARDS NOT ALLOWED IN FILE SPEC)*

Explanation: A wildcard was used in the name of a workspace identifier.

23 *WORKSPACE LOCKED*

23 *WORKSPACE LOCKED (INCORRECT PASSWORD)*

23 *WORKSPACE LOCKED (WORKSPACE HAS NO PASSWORD)*

Explanation: An incorrect password (or none at all) was given to access a workspace that was saved with a password.

24 *NOT GROUPED, NAME IN USE*

25 *EXECUTE ERROR*

Explanation: APL signaled an error while executing the argument to the ⍎ execute function.

27 *LIMIT ERROR*

Explanation: The result of the operation exceeded some implementation limit; for example, if the argument array to (⎕*FX* has more than 65535 columns.

27 *LIMIT ERROR (ARGUMENT STRING IS TOO LONG)*

Explanation: The length of an argument cannot be greater than 255 keystrokes.

27 *LIMIT ERROR (ARGUMENT TOO LARGE)*

Explanation: The argument to ⎕*SF* was greater than 255 keystrokes.

27 *LIMIT ERROR (ARGUMENT TOO LONG)*

Explanation: For (⎕*MAP*, if *A* contains more than 255 formal parameters (including the result).

27 *LIMIT ERROR (AXIS TOO LONG)*

27 *LIMIT ERROR (DELAY VALUE TOO LARGE)*

Explanation: The delay specified for (⎕*DL* was larger than approximately 3.4E11 milliseconds.

27 *LIMIT ERROR (FLOATING OVERFLOW)*

Explanation: Arithmetic overflow has occurred.

27 *LIMIT ERROR (INPUT LINE TOO LONG)*

27 *LIMIT ERROR (INTEGER TOO LARGE)*

Explanation: A value is greater than the largest allowable integer.

27 *LIMIT ERROR (PARAMETER OUT OF RANGE)*

Explanation: One of the parameters in the left argument of dyadic ⍕ is less than ¯127 or greater than 127.

27 *LIMIT ERROR (RANK TOO LARGE)*

27 *LIMIT ERROR (VOLUME TOO LARGE)*

Explanation: The result of a primitive function has more elements than the implementation can accomodate.

28 *AXIS RANK ERROR (NOT VECTOR DOMAIN)*

Explanation: The specified axis number argument ([*K*]) is not a singleton and its rank is greater than 1.

29 *AXIS LENGTH ERROR*

Explanation: The specified axis number argument has more than one item.

29 *AXIS LENGTH ERROR (ARGUMENT RANK AND AXIS INCOMPATIBLE)*

29 *AXIS LENGTH ERROR (NOT SINGLETON)*

Explanation: The axis argument is not a singleton.

29 *AXIS LENGTH ERROR (LEFT ARGUMENT HAS WRONG LENGTH)*

Explanation: The length of the axis argument to ↑ or ↓ does not match the length of the left argument.

30 *AXIS DOMAIN ERROR*

Explanation: The specified axis argument value was not a nonnegative integer (except in the case of laminate, which accepts floating-point numbers greater than ¯1), or the specified function was not in the domain of the axis operator.

30 *AXIS DOMAIN ERROR (ARGUMENT RANK AND AXIS INCOMPATIBLE)*

Explanation: The axis argument or an element of the axis argument is greater than the rank of the argument with the largest rank.

30 *AXIS DOMAIN ERROR (AXIS LESS THAN INDEX ORIGIN)*

Explanation: The axis argument is less than ⎕*IO*.

30 *AXIS DOMAIN ERROR (INCORRECT TYPE)*

Explanation: The axis argument is not a number.

30 *AXIS DOMAIN ERROR (NOT AN INTEGER)*

Explanation: The axis argument is not a near-integer.

30 *AXIS DOMAIN ERROR (SEMICOLON LIST NOT ALLOWED)*

Explanation: There is a semicolon inside the brackets that surround the axis argument .

30 *AXIS DOMAIN ERROR (AXES NOT IN CONTIGUOUS ASCENDING ORDER)*

Explanation: The axis argument elements must be in contiguous ascending order for Ravel.

30 *AXIS DOMAIN ERROR (DUPLICATE AXIS NUMBER)*

Explanation: An axis argument element was specified more than once.

30 *AXIS DOMAIN ERROR (ENCLOSED ARRAY NOT ALLOWED)*

Explanation: The axis argument must be a simple homogeneous array.

30 *AXIS DOMAIN ERROR (INCORRECT OPERATION)*

Explanation: An operation was specified that was not one of the following: Ravel, Catenate/Laminate, Reverse, Rotate, Expand, Scan, Replicate/Compress, Reduce, Monadic Grade up/down, Take, Drop.

30 *AXIS DOMAIN ERROR (RIGHT ARGUMENT HAS WRONG RANK)*

Explanation: An axis argument value was specified that is greater than the rank of the right argument.

31 *PROTECTION VIOLATION*

Explanation: The protection assigned to the workspace you specified prohibits the access you requested.

31 *PROTECTION VIOLATION (INSUFFICIENT PRIVILEGE OR FILE PROTECTION VIOLATION)*

32 *INVALID SIMULTANEOUS ACCESS*

Explanation: More than one user tried to save the same workspace simultaneously, or a user tried to access a nonshared file that is already in use.

32 *INVALID SIMULTANEOUS ACCESS (FILE CURRENTLY LOCKED BY ANOTHER USER)*

33 *IO ERROR*

33 *IO ERROR (INVALID WILDCARD OPERATION)*

Explanation: For *)OUTPUT*, a wildcard was specified in place of a value for *filespec*.

33 *IO ERROR (NULL PRIMARY KEY)*

Explanation: An attempt was made to specify an empty key value.

33 *IO ERROR (SEQUENTIAL DELETE OPERATION IS NOT ALLOWED FOR KY FILES)*

Explanation: An attempt was made to omit the entire key specification.

34 *COMPONENT ERROR*

Explanation: An attempt was made to read a component that cannot be read.

34 *COMPONENT ERROR (COMPONENT CROSSES CELL BOUNDARY)*

34 *COMPONENT ERROR (COMPONENT IS DAMAGED)*

34 *COMPONENT ERROR (RECORD NOT A COMPONENT)*

35 *INVALID FILE SPECIFICATION*

35 *INVALID FILE SPECIFICATION (WILDCARDS NOT ALLOWED IN FILE SPECIFICATIONS)*

Explanation: Wildcards are invalid in the file specifications for *)INPUT* and *)OUTPUT*.

36 *INDEX RANK ERROR*

Explanation: The rank of the index and the argument are not compatible.

36 *INDEX RANK ERROR (CANNOT INDEX A SCALAR)*

37 *INDEX LENGTH ERROR*

Explanation: In the form *A[K]←B B* is not a singleton and its shape does not conform to the shape of the selected items of *A*.

37 *INDEX LENGTH ERROR (INDEX OUT OF RANGE)*

38 *INDEX DOMAIN ERROR*

38 *INDEX DOMAIN ERROR (INCORRECT TYPE)*

Explanation: An attempt was made to enter an index array that does not consist of nonnegative integers.

38 *INDEX DOMAIN ERROR (INDEX LESS THAN INDEX ORIGIN)*

38 *INDEX DOMAIN ERROR (NOT AN INTEGER)*

Explanation: A value of the axis argument is not a near-integer.

39 *NO SUCH DIRECTORY*

40 *OPERATOR DOMAIN ERROR (ARRAY OPERAND NOT ALLOWED)*

Explanation: An array was specified as an operand to an each (¨) or dot (.) operator.

41 *NO ROOM ON FILE STRUCTURE OR QUOTA EXCEEDED*

Explanation: The specified file structure was full, or the disk allocation was exceeded. In the latter case, files must be deleted from the user's disk area before more files can be added.

42 *DEVICE IS WRITE-LOCKED*

Explanation: The specified device (usually a magnetic tape) was physically write-protected.

43 *SYSTEM RESOURCES EXHAUSTED*

Explanation: The system ran out of space to perform certain functions for the user. See the system manager at your installation.

44 *ERROR INVOKING EXTERNAL ROUTINE*

Explanation: An error occurred while trying to map an external routine or process the actual arguments before executing the external routine.

44 *ERROR INVOKING EXTERNAL ROUTINE (DOMAIN ERROR)*

Explanation: One of the following situations has occurred:

- The data leaving the workspace cannot be converted to the data type expected by the external routine (for example, numbers could not be converted to */TYPE:T*).
- A conversion failed as data passed from the workspace to the external routine.

44 *ERROR INVOKING EXTERNAL ROUTINE (EXTRANEOUS CHARACTERS AFTER COMMAND)*

Explanation: Unrecognized input, such as an undefined or repeated qualifier, appeared at the end of the command.

44 *ERROR INVOKING EXTERNAL ROUTINE (ILL FORMED NAME)*

Explanation: The actual parameter specified for either the */ACCESS:OUT* or */ACCESS:INOUT* qualifier is not a valid APL name.

44 *ERROR INVOKING EXTERNAL ROUTINE (ILLEGAL ASCII CHARACTER)*

Explanation: A conversion to ASCII failed as character data (*/TYPE:T* or */TYPE:VT* left the workspace.

44 *ERROR INVOKING EXTERNAL ROUTINE (ILLEGAL NAME CLASS)*

Explanation: The actual parameter specified for either the */ACCESS:OUT* or */ACCESS:INOUT* qualifier is defined, but is not a variable.

44 *ERROR INVOKING EXTERNAL ROUTINE (INCORRECT PARAMETER)*

Explanation: One of the following situations has occured:

- The actual parameter specified for either the */ACCESS:OUT* or */ACCESS:INOUT* qualifier is currently undefined and is */TYPE:Z*. The parameter must either be defined so an unconverted value can be passed or undefined with a known data type, not */TYPE:Z*

- The actual argument is missing when the formal paramter is specified with the */MECHANISM:IMMEDIATE* qualifier.

44 *ERROR INVOKING EXTERNAL ROUTINE (INCORRECT TYPE)*

Explanation: The actual paramter specified for either the */ACCESS:OUT* or */ACCESS:INOUT* qualifier is not a character.

44 *ERROR INVOKING EXTERNAL ROUTINE (LENGTH ERROR)*

Explanation: One of the following situations has occurred:

- The actual argument has a length greater than 4 bytes when ⎕*MAP* was specified with the */MECHANISM:IMMEDIATE* qualifier.
- The actual argument has a length greater than 2+2*16 when dyadic ⎕*MAP* was specified with the */MECHANISM:DESCRIPTOR* qualifier.
- A complex data type is being passed an odd number of items (APL requires two numbers to form each complex number).
- The length of a Varying sTring (*/TYPE:VT*) is greater than ¯1+2*16.

44 *ERROR INVOKING EXTERNAL ROUTINE (NOT VECTOR DOMAIN)*

Explanation: The actual parameter specified for either the */ACCESS:OUT* or */ACCESS:INOUT* qualifier is not in the vector domain.

44 *ERROR INVOKING EXTERNAL ROUTINE (NOT SINGLETON)*

Explanation: The actual argument is not a singleton (as it should be) when dyadic (⎕*MAP* is specified with the */MECHANISM:IMMEDIATE* qualifier.

44 *ERROR INVOKING EXTERNAL ROUTINE (WRONG NUMBER OF ARGUMENTS TO USER FUNCTION)*

Explanation: More actual arguments were specified than there are formal parameters defined in the formal parameters of the external routine.

45 *SIGNAL FROM EXTERNAL ROUTINE*

Explanation: An external routine signaled the error that is the secondary error message.

46 *OPERATION INVALID IN THIS CONTEXT*

Explanation: An attempt was made to use)*STEP* when there was no suspended operation.

47 *OUTPUT LINE TOO LONG*

47 *OUTPUT LINE TOO LONG (BUFFER OVERFLOW)*

Explanation: A line editing sequence created a line that was too long to fit in the I/O buffer.

47 *OUTPUT LINE TOO LONG (PAGE WIDTH EXCEEDED)*

Explanation: A line editing sequence created a line longer than the page width limit.

48 *INPUT LINE TOO LONG*

48 *INPUT LINE TOO LONG (ARGUMENT STRING IS TOO LONG)*

Explanation: The argument to)*HELP* was longer than APL's input buffer.

49 *FILE CONTAINS A DAMAGED WORKSPACE*

Explanation: The file specified by)*LOAD*,)*COPY*, or)*PCOPY* contains a damaged workspace.

49 *FILE CONTAINS A DAMAGED WORKSPACE (CURRENT WORKSPACE CLEARED)*

Explanation: An attempt was made to load a file that contains a damaged workspace. The current workspace is cleared.

50 *CHARACTER ERROR*

Explanation: The user entered an illegal overstruck character.

50 *CHARACTER ERROR (ILLEGAL CHARACTER IN EXPRESSION)*

Explanation: An internal (⎕*AV* code was included outside of a literal or comment.

50 *CHARACTER ERROR (ILLEGAL OVERSTRIKE)*

51 *INPUT ABORTED*

Explanation: The user entered the abort signal to escape from quad, quote quad, or quad del input.

52 *FUNCTION EDITING ABORTED*

Explanation: The user entered the abort signal to escape from the function editor.

53 *LINE EDITING ABORTED*

Explanation: The user entered the abort signal to escape from character editing mode.

54 *INTERNAL ERROR SAVING WORKSPACE*

Explanation: An internal inconsistency was detected. Please notify your Digital software specialist.

55 *NOT A RANDOM ACCESS DEVICE*

56 *INCORRECT MODE FOR DEVICE*

Explanation: The I/O mode for the operation requested was improper for the chosen device.

57 *FILE DOES NOT CONTAIN A WORKSPACE*

Explanation: An attempt was made to load or copy a file that does not contain an APL workspace.

57 *FILE DOES NOT CONTAIN A WORKSPACE (CURRENT WORKSPACE CLEARED)*

58 *DATA TRANSMISSION ERROR*

Explanation: A data transmission error was detected during input or output. This message is usually associated with a nonrecoverable device error.

59 *FILE ALREADY EXISTS WITH GIVEN NAME*

Explanation: An attempt was made to save a workspace with the same file name as an existing file that is not a workspace.

60 *WS NOT SAVED, THIS WS IS* ***wsname***

Explanation: An attempt was made to save a workspace with the same file name as an existing workspace, without first making that same name the workspace identification (returned by)*WSID*). This error message is to prevent inadvertent overwriting of previously saved workspaces.

62 *NOT A DIRECTORY STRUCTURED DEVICE*

63 *FILE ASSIGNED READ ONLY*

64 *CHANNEL NOT ASSIGNED*

Explanation: The channel specified in a file operation was not previously associated with a file via a ⎕*ASS* system function.

65 *CHANNEL CANNOT DO BOTH INPUT AND OUTPUT*

Explanation: An attempt was made to do both input and output to a channel assigned to a sequentially organized file.

66 *NOT AN INPUT DEVICE*

Explanation: The user tried to perform input from an output-only device, such as a line printer.

67 *NOT AN OUTPUT DEVICE*

Explanation: The user tried to perform output from an input-only device, such as a card reader.

68 *END OF FILE ENCOUNTERED*

Explanation: A sequential read operation was attempted when there was no next record or component and when the channel was assigned with */SIGNAL*.

69 *RECORD NOT FOUND*

Explanation: A random read operation was attempted on a nonexistent record or component when the channel was assigned with */SIGNAL*.

71 *DEVICE ERROR*

Explanation: A file operation attempted to use a mode that is improper for the device specified in the associated ⎕*ASS* function.)

72 *SYSTEM SERVICE FAILURE*

73 *SUBPROCESS ERROR*

73 *SUBPROCESS ERROR* (*COMMAND BUFFER OVERFLOW - SHORTEN EXPRESSION OR COMMAND LINE*)

74 *BLOCK TOO BIG*

Explanation: A data-transfer error occurred during I/O. Specifically, the last read attempted to read a block of data that was too large.

75

Explanation: The end of the file was reached when */SIGNAL* was not being used. No message is printed and execution continues.

76 *RESULT ERROR (BRANCH HAS NO RESULT)*

Explanation: Branch was used with ⎕ input.

77 *STOPSET*

Explanation: The operation was suspended because a stop bit was set for the current line.

78 *END OF TAPE*

Explanation: The end of a reel of magnetic tape was reached.

79 *SYSTEM FUNCTION ILLEGAL IN EXECUTE*

Explanation: The (⎕*BREAK* system function was used in the argument to the execute function.

80 *RETURN TO CALLER OF THIS IMAGE*

Explanation: The right argument to (⎕*SIGNAL* was 80.

81 *BROADCAST RECEIVED*

Explanation: A broadcast was received when (⎕*GAG* was set to 3.

82 *CHANNEL NUMBER IS NOT AVAILABLE*

83 *DAMAGED WORKSPACE HAS BEEN CORRECTED*

83 *DAMAGED WORKSPACE HAS BEEN CORRECTED (SOME SYMBOLS MAY HAVE BEEN ERASED)*

Explanation: A workspace, which previously contained corrupted data, was loaded with the */CHECK* qualifier.

86 *FILE IS ASSIGNED WRITE ONLY*

100 *HI FILE READ ERROR*

Explanation: An error occurred while reading the file specified by the /HI qualifier on an APL command line or in an initialization file.

101 *INITIAL WORKSPACE NOT FOUND*

Explanation: The workspace that was specified on the APL command line or in the initialization file was not found by APL.

102 *VECTOR PROCESSOR NOT AVAILABLE*

103 *ERROR IN INITIALIZATION FILE*

Explanation: APL detected an error while processing the parameters in the initialization file identified by the logical name APL$INIT.

104 *NEGATIVE THRESHOLD WITH VECTOR QUALIFIER NOT ALLOWED*

105 *ERROR INITIALIZING CONSOLE CHANNEL*

106 *ERROR INITIALIZING WORKSPACE ENVIRONMENT*

108 *FATAL INITIALIZATION ERROR*

109 *FATAL ERROR SETTING UP CLEAR WORKSPACE*

110 *ERROR READING INPUT FILE*

111 *EDIT COMMAND ERROR*

111 *EDIT COMMAND ERROR* (*xx QUALIFIER REPEATED*)

Explanation: For) *EDIT*, the same qualifier was specified more than once. *xx* is the name of the repeated qualifier.

111 *EDIT COMMAND ERROR* (*ARGUMENT TO xx IS OUT OF RANGE*)

Explanation: For) *EDIT*, a numeric value that is outside the acceptable range was specified for a qualifier. *xx* is the name of the qualifier.

111 *EDIT COMMAND ERROR* (*BAD ARGUMENT TO xx*)

Explanation: For) *EDIT*, an invalid value was specified for a qualifier. *xx* is the name of the qualifier.

111 *EDIT COMMAND ERROR* (*CANNOT EDIT SYSTEM SYMBOL*)

111 *EDIT COMMAND ERROR* (*EDIT COMMAND UNAVAILABLE DURING FUNCTION DEFINITION*)

111 *EDIT COMMAND ERROR* (*ENCLOSED ARRAY NOT ALLOWED*)

Explanation: An attempt was made to edit an enclosed array.

111 *EDIT COMMAND ERROR* (*EXECUTE QUALIFIER ARGUMENT IS TOO LONG*)

Explanation: For /*EXECUTE*, the string specified for *tpucommand* is too long.

111 *EDIT COMMAND ERROR* (*ILL FORMED NUMERIC CONSTANT*)

Explanation: For) *EDIT*, there is nonnumeric data (data unacceptable to ⎕*VI*) inside a numeric array that is returning from VAXTPU.

111 *EDIT COMMAND ERROR* (*ILL FORMED NUMERIC MATRIX*)

Explanation: For) *EDIT*, a record or records in the matrix returning from VAXTPU have either more or fewer values than the number of values in the first record.

111 *EDIT COMMAND ERROR* (*ILLEGAL ASCII CHARACTER*)

111 *EDIT COMMAND ERROR (ILLEGAL NAME CLASS)*

Explanation: For */NC*, either a value other than 2, 3, or 4 was specified, or the specified value does not match the current name class value for *objectname*.

111 *EDIT COMMAND ERROR (INCORRECT PARAMETER)*

Explanation: For *)EDIT*, an unknown parameter was specified.

111 *EDIT COMMAND ERROR (MISSING ARGUMENT)*

Explanation: For *)EDIT*, an attempt was made to edit a system function or variable.

111 *EDIT COMMAND ERROR (OPERATION LOCKED)*

Explanation: For *)EDIT*, an attempt was made to edit a locked function.

111 *EDIT COMMAND ERROR (OPERATION SUSPENDED, PENDENT, OR MONITORED)*

111 *EDIT COMMAND ERROR (UNBALANCED DELIMITER)*

111 *EDIT COMMAND ERROR (UNRECOGNIZED QUALIFIER KEYWORD)*

111 *EDIT COMMAND ERROR (UNSUPPORTED TERMINAL TYPE)*

111 *EDIT COMMAND ERROR (VOLUME TOO LARGE)*

112 *ERROR PROCESSING HELP*

112 *ERROR PROCESSING HELP (INVALID KEY)*

112 *ERROR PROCESSING HELP (TOO MANY HELP KEYS SPECIFIED)*

112 *ERROR PROCESSING HELP (ERROR OPENING AS INPUT)*

Explanation: The file that was specified as the argument to the *)HELP* command did not exist.

112 *ERROR PROCESSING HELP (ERROR PARSING ARGUMENT TO LIBRARY)*

Explanation: The value for *filespec* on the */LIBRARY* qualifier was either not specified or specified incorrectly.

113 *WATCH POINT ACTIVATED*

113 *WATCH POINT ACTIVATED (VARIABLE HAS BEEN MODIFIED)*

113 *WATCH POINT ACTIVATED* (*VARIABLE HAS BEEN MODIFIED BY INDEX*)

113 *WATCH POINT ACTIVATED* (*VARIABLE HAS BEEN REFERENCED*)

114 *ERROR PROCESSING ATTACH*

Explanation: An error occurred when APL attempted to process the)*ATTACH* command.

114 *ERROR PROCESSING ATTACH* (*ATTACH REQUEST REFUSED*)

Explanation: The value specified for *process-name* is the name of a nonexistent process.

114 *ERROR PROCESSING ATTACH* (*NONEXISTENT PROCESS*)

114 *ERROR PROCESSING ATTACH* (*INVALID LOGICAL NAME*)

115–499 are reserved for VAX APL

500–999 are for user-defined error messages.

Explanation: For more information, see ⎕*SIGNAL*.

Glossary

abort input signal

A technique for escaping to immediate mode when APL is waiting for input. Different terminals form the abort input signal differently. Consult the index to find more information on this subject.

ambivalent function

A function that may be monadic or dyadic, depending on how many arguments are supplied when it is invoked.

APL terminal

A terminal that has an APL keyboard, that is, a terminal that can be set up to use the APL key-paired (typewriter-paired), APL bit-paired, or APL COMPOSITE character set.

argument

An array that is manipulated by a function. APL functions take zero, one, or two arguments.

array

Any number (including 0 or 1) of items treated as a unit.

assignment

A method for associating a name with an array.

atomic vector

An array, returned by the system function ⎕*AV*, that contains all the characters in the APL character set.

attention signal

A technique for suspending the execution of an operation and escaping to immediate mode. The weak attention signal (formed by pressing (Ctrl/C) once) means suspend execution of the current operation after executing the current statement, and return control to immediate mode. The strong attention signal (formed by pressing (Ctrl/C)) twice, means suspend the current operation as soon as possible, even in the middle of the statement, and return control to immediate mode.

axis

A dimension along which items in an array are arranged.

Boolean

A numeric item that has the value 0 or 1.

branch

Within a user-defined operation, a change in the normal order of statement execution.

canonical representation

A character matrix with rows consisting of the original lines of a user-defined operation.

channel

The logical path through which the APL file system interacts with external files and mailboxes.

character-editing mode

While in function-definition mode, a mode of editing in which you can edit individual characters in a line.

command line

The line that contains the DCL command APL. You enter the command line in response to the DCL prompt ($).

comment

Ignored characters appearing to the right of (and on the same line as) the ⍝ symbol; you can place a comment at the end of a line containing APL statements or on a separate line.

comparison tolerance

An amount used by APL when it calculates how much two numbers can differ and still be considered equal. The system variable ⎕CT contains the comparison tolerance used by APL.

component

In an external file, a record that contains an APL object.

constant

An item whose value is literally the constant itself.

dense sequence

For some functions, APL requires that an argument of nonnegative integers must form a dense sequence, beginning at ⎕IO. This means that the smallest element in the argument must be ⎕IO, and that an integer *N* from the argument domain may be included only if *N*-1 is also included. For example, if the argument domain is the integers from 1 to 3, the arguments 2 1 3, 1 2 2, and 1 1 1 form dense sequences, but the arguments 1 3 1 and 3 2 3 do not.

depth

The degree of nesting of an array.

derived function

A function that results from the combination of an operator and its operand or operands.

domain

The permissible type, shape, and values of a function's argument arrays or the permissible objects of an operator's operands.

dummy argument

In the header of a user-defined operation, an identifier that serves as a placeholder for the actual argument, operand, or result that is supplied when the operation is called.

dyadic function

A function that takes both a left and a right argument.

enclosed array

An array that includes one or more arrays.

empty array

An array that has a type and shape but no items. The length of the array along at least one axis is 0.

error trapping

Techniques to find and react to errors that occur during the execution of user-defined operations.

event flag

A shareable indicator, accessible through the APL file system, intended to aid in synchronizing access to shared files or mailboxes.

execute-only APL

The DCL command APL/EXECUTE_ONLY [*parameters*] invokes the run-time support version of VAX APL called QAPL. QAPL can execute applications written in VAX APL but does not contain the features to develop applications. QAPL can be copied to any valid VMS system free of charge.

expression

An identifier or constant standing alone, a function or operator and its arguments, or an expression enclosed in parentheses.

external data

Data created outside of APL.

external routine

A routine (not written in APL) that exists outside the APL environment. APL can call library routines and other external routines that support the VAX Procedure Calling and Condition Handling Standard. APL cannot call VMS system services routines.

fill element

A scalar data element (either a space or a 0) inside a fill item.

fill item

An array (consisting of spaces, zeros, or a combination of both) that APL inserts into another array. The shape and contents of a fill item are based on the prototype of the array that APL is using as a model for the array being built. Fill items are used by Take, Replicate, Expand, Disclose, and ⎕*BOX*.

function

An operation that applies to arrays and produces an array as a result.

function-definition mode

An operating mode in which the lines of APL you enter are not executed immediately but rather are stored for later execution. Function-definition mode begins when you type a ∇ and ends when you type a second ∇ or ⍫. This mode is used when creating user-defined functions and operators.

global symbol

A symbol that has the same value inside and outside a user-defined operation.

header

The initial line of a user-defined operation. See operation header for more information.

heterogeneous array

An array that contains both character and numeric data.

high minus

The symbol (¯) used to represent the negative sign in APL.

homogeneous array

An array that contains either character or numeric data, but not both.

identifier

A variable name, label name, group name, or user-defined operation name. See also system identifier.

identity element

A value (if one exists) to a dyadic function which, when used as one argument to the function, does not change the value of the other argument. For example, for any identity element *i* applied to a dyadic function *f* and an argument *a*, *a* does not change: *i f a*←→*a*

identity function

A function that APL applies to the prototype of an array when performing the reduction (*f*/B) of an axis that has length zero. Note that the inner product (*f.g*) derived functions imply the use of reduction. The identity function is applied to the prototype of the argument array in place of the specified function.

immediate mode

An APL operating mode in which lines are executed immediately after they are entered.

index

A notation used to specify the position of items within an array that you want to reference. The index appears immediately to the right of an array and consists of two brackets enclosing values that correspond to axes in the array. Index is synonymous with subscript.

indexed assignment

The assignment of values to selected items of a variable. The indexed variable is positioned to the left of the assignment arrow (←), and the index specifies the items in the array where the assignment is applied. Indexed assignment is synonomous with subscripted assignment or indexed specification.

index origin

The starting point for the index values of an array. The index origin may be 0 or 1. The system variable ⎕*IO* contains the current index origin value.

indexing

The use of an index to access particular items from an array.

initialization file

A file, referenced by the VMS logical name APL$INIT, that contains parameters that are processed when APL is initialized.

initialization stream

Either the DCL command line that invokes APL, or the initialization file referenced by the VMS logical name APL$INIT. Either or both of these streams may contain parameters to be processed when APL is initialized.

integer

Any of the positive and negative integers, or zero.

internal data

Data stored in one of the four APL internal data type formats.

key

A field defined by its location and length within each record and used to sort the records. At least one key, called the primary key, must be defined for a keyed file. Optionally, additional keys, called alternate keys, may be defined.

key of reference

The specific key used in a sequential or random read of a keyed file.

keyed file

A file in which records are organized by fields, called keys, inside the records. The VAX RMS term is indexed sequential file organization (ISAM). The keys of the file define the order in which the records are retrieved; you can retrieve records sequentially by one of the sorted orders or randomly by one of the record's key values. A keyed file must contain at least one key.

label

An identifier associated with a line in a user-defined operation.

latent expression

A character vector representing an APL expression; the expression is associated with a workspace and is automatically executed when the workspace is loaded. The system variable ⎕*LX* contains the value of the workspace's latent expression.

line

The statement or statements you enter beginning after an APL input prompt and ending when you press Return to enter the line.

local symbol

A symbol that has significance only during the execution of a particular user-defined operation.

locked operation

An operation definition that cannot be changed or displayed.

logical name

A symbolic name for any portion or all of a file specification.

mailbox

A virtual device useful for sending messages to other processes.

matrix

An array consisting of any number of items arranged along two axes, commonly called rows and columns.

matrix domain

A matrix, vector, or singleton.

monadic function

A function that takes one argument.

monitored operation

A user-defined operation that has some of its lines being monitored via ⎕*MONITOR*.

multikey file

A file in which records are organized by fields, called keys, inside the records. The RMS term is indexed sequential file organization (ISAM). The keys of the file define the order in which the records are retrieved: you can retrieve records sequentially by one of the sorted orders or randomly by one of the record's sort values. A multikey file must contain at least one key.

near-integer

A numeric item whose floor is equal to its ceiling; this includes all numbers sufficiently close to an integer as determined by the APL comparison tolerance.

nested array

A synonym for enclosed array.

next record pointer

An internal mechanism that keeps track of the next record to be processed by a sequential input function.

niladic function

A function that takes no arguments.

non-APL terminal

A terminal that does not have an APL keyboard. On such a terminal, APL characters must be represented by ASCII mnemonics.

nonnegative integer

Any of the positive integers or zero.

operation

Either a function or an operator. Occasionally, operation refers to a mathematical action (such as the addition operation) or to an action taken by the APL interpreter.

operation body

The executable lines of APL that appear in a user-defined operation definition.

operation header

The first line you enter when you define an operation. It names the operator; indicates whether the operation returns a value; indicates whether the operator is monadic or dyadic; indicates the use of an axis argument; and identifies the operation's local symbols.

operator

An operation that is applied to either arrays, or functions, or both and produces a derived function as a result. In VAX APL, there are user-defined operators and primitive operators.

operator sequence

A sequence of functions and operators whose result is a derived function.

overstruck character

An APL character formed by combining two other APL characters. For example, the ⍇ symbol is formed with the ⎕ and ← symbols. Different terminal types form overstrikes in different ways. Some terminals allow you to enter the first character, use Backspace, and then enter the second character on top of the first. Other terminals allow you to use a Compose Character key (or Ctrl/D) and then to enter the two characters. On these terminals, only the resulting overstrike character is displayed.

panic exit

A technique for immediately suspending the execution of an operation and giving control to the operating system. The panic exit is formed by pressing Ctrl/Y once. After a panic exit, you can return to where you left off by executing the DCL command CONTINUE. If you enter the panic exit while an operation is executing, the operation is suspended; if you then enter CONTINUE, the operation resumes execution at the point where it was interrupted.

pendent operation

A user-defined operation that has called another operation and is waiting for that operation to return.

pervasive operation

An operation that acts at all depths (levels of nesting) of an array.

PID

Process Identification, an integer value that uniquely identifies a VMS process.

positive integer

The integers greater than zero.

print precision

The maximum number of significant digits displayed in floating-point output. The system variable ⎕PP contains the current print precision value.

print width

The maximum number of characters that APL can display on a terminal output line. The system variable ⎕PW contains the current print width value.

process

The basic entity scheduled by VMS software that provides the context in which an image executes.

process identification

An integer value that uniquely identifies a VMS process.

prototype

An array that APL uses to determine the shape and contents of fill items. The prototype of an array *B* has the same shape as the first item of *B* and has character blanks and zeros in positions corresponding to characters and numbers, respectively, in the first item of *B*.

pure data record

A record that is a vector of values, with none of the embedded format information that APL includes within component data records.

quiet function

A function that does not return a value unless one is needed; that is, a value is returned only if it is not the leftmost function.

random link

The current value used by the APL random number generator. The system variable ⎕*RL* contains the current random link value.

range

The permissible type, shape, and values of a function's result array.

rank

The number of axes along which an array's items are arranged.

recursive operation

A user-defined operation that calls itself.

reshape

A function used to change the number of an array's axes or to change the length of one or more of its axes.

row-major order

An ordering of the items of an array so that the last subscript value varies most rapidly. For example, the row-major order of a 2 by 3 matrix would be [1;1], [1;2], [1;3], [2;1], [2;2], [2;3].

scalar

A rank 0 array (an array with no axes) containing a single numeric or character or enclosed item.

scalar extension

An implicit operation that reshapes a scalar argument to match the shape of a non-scalar argument.

scalar product

An implicit operator that applies a dyadic scalar function over each corresponding pair of items in the two arguments.

selective assignment

A method for replacing selected items of an array.

shadow

The act of localizing a name when a user-defined operation is activated so that the old value of the name is saved and the name becomes undefined in the context of the newly activated user-defined operation. The old value of the name is restored when the user-defined operation exits to its calling environment.

shape

The way an array's items are arranged; specifically, a numeric vector that describes the length of each of the array's axes.

signal

A term often used in the description of what APL does when it detects an error; APL signals an error.

simple array

An non-enclosed array whose depth is less than 2.

simple scalar

A scalar that contains only a single character or number.

singleton

A one-item array of any rank (includes scalars).

singleton extension

An implicit operation that is applied to a dyadic scalar function when one or both of the function's arguments are singletons. This implicit operator reshapes the singleton argument to match the shape of the nonsingleton argument, allowing the single value from the singleton to be applied to each item of the other argument. When both arguments are singletons, the argument with the smaller rank is reshaped to match the rank of the other singleton.

specification

A method for associating a name with an array.

state indicator

A vector that reports the status of user-defined operations, quad input requests, and execute functions.

statement

One or more expressions executed as a unit.

stop bit

A setting associated with a line in an operation definition that causes the operation to be suspended before the line is executed.

strand

Two or more juxtaposed arrays (including scalars) which form a vector. Also known as vector notation.

strand assignment

The process of associating a strand of values with a set of names.

subprocess

A process created by and subordinate to another process. The subprocess shares the resources of the creating process.

subscript

A notation used to specify the position of items within an array that you want to reference. The subscript appears immediately to the right of an array and consists of two brackets enclosing values that correspond to axes in the array. Subscript is synonymous with index.

subscripted assignment

An assignment that modifies only the items that are specified by an index list. Subscripted assignment is synonymous with indexed assignment or subscripted specification.

suspended operation

A user-defined operation that has stopped executing but still has lines of APL to be processed.

symbol table

A data structure inside the APL interpreter. The symbol table keeps track of the names of all objects in a workspace.

system identifier

Any system-provided name that always begins with the quad (⎕) symbol. System identifier refers to system variables and functions.

Index

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-343-4040 before placing your electronic, telephone, or direct mail order.

Electronic Orders

To place an order at the Electronic Store, dial 800-DEC-DEMO (800-332-3366) using a 1200- or 2400-baud modem. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

Your Location	Call	Contact
Continental USA, Alaska, or Hawaii	800-DIGITAL	Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061
Puerto Rico	809-754-7575	Local Digital subsidiary
Canada	800-267-6215	Digital Equipment of Canada Attn: DECdirect Operations KAO2/2 P.O. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6
International	————	Local Digital subsidiary or approved distributor
Internal[1]	————	USASSB Order Processing - WMO/E15 *or* U.S. Area Software Supply Business Digital Equipment Corporation Westminster, Massachusetts 01473

[1] For internal orders, you must submit an Internal Software Order Form (EN-01740-07).

Do Not Tear - Fold Here and Tape

digital™

No Postage
Necessary
If Mailed
in the
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Information Products
PK03-1/D30
129 PARKER STREET
MAYNARD, MA 01754-9975

Do Not Tear - Fold Here