APL System Functions

Basic Information

System functions are features that are available in any workspace. Their names begin with a quad (⎕) symbol. The rest of the name is case insensitive.

System functions are an extended set of basic operations that APL64 provides. They share many of the properties of APL primitive functions. You can incorporate them into user-defined functions, and most return an explicit result that you can use in subsequent operations.

System functions can be niladic (no arguments), monadic (one argument), dyadic (two arguments), or ambivalent (monadic/dyadic). Typically, they:

- provide information about the session, the active workspace, and the objects in it
- manipulate functions, variables, and arrays
- provide methods for handling workspaces and files
- assist in debugging programs
- provide an interface to the operating system or non-APL programs.

Contents

⎕a Roman Alphabet Upper Case Glyphs

Purpose:
Returns the uppercase Roman alphabet characters (glyphs).

Syntax: result ← ⎕a

Result:
result is a character vector.

Example:

```
      ⎕a
 ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

⎕acbd .Net AppContext.BaseDirectory

Purpose:
Reports the path of the running executable file or the WRE dll file.

Syntax: result ← ⎕acbd

Result:
⎕acbd returns the .Net AppContext.BaseDirectory path for the application.

Effect:
For the APL64 developer version the ⎕acbd value is the same as the ⎕exepath value.

For a WRE runtime application, the ⎕acbd value is the path of the running WRE dll file. The ⎕acbd path is generally not the location on the end user's workstation where the WRE executable has been installed.

- The ⎕acbd path is determined by the end user's workstation credential and the name of the 'Runtime Executable Name' specified when the WRE was created by the APL64 programmer. For example:
- C:\Users\user_Name\AppData\Local\Temp\.net\app_Name\GUID, where user_Name depends on the workstation user, app_Name depends on the WRE executable file name and GUID is a system-determined GUID.

⎕aes AES Encryption and Decryption

Purpose: The ⎕AES system function supports the encryption and decryption of text using the AES cryptographic algorithm, implemented in the Microsoft System.Security.Cryptography toolkit.

⎕AES Actions

DecryptString

Decrypt a byte[] array using previously obtained AES key and iv values using the AES algorithm.

Syntax: string ← ⎕AES 'DecryptString' keyAndIv srcBytes

srcBytes is a byte[] array which was obtained by encrypting text using the AES algorithm string is the decrypted text as an APL64 scalar string. If desired, use the APL64 monadic > function to convert the result string to a character vector.

EncryptString

Encrypt a string using previously-obtained AES key and iv values using the AES algorithm.

Syntax: byte[] ← ⎕AES 'EncryptString' keyAndIv srcText

srcText is an APL64 character scalar, character vector or scalar string value.

Bytes[] is an array of bytes which represent the encrypted srcText

GetKeyAndIv

Obtain an AES key and iv values to encrypt and decrypt text using the AES algorithm. The key length is 256.

Syntax: keyAndIv ← ⎕AES 'GetKeyAndIv'

keyAndIv is a vector containing the generated key (byte[]) and iv (byte[]) values

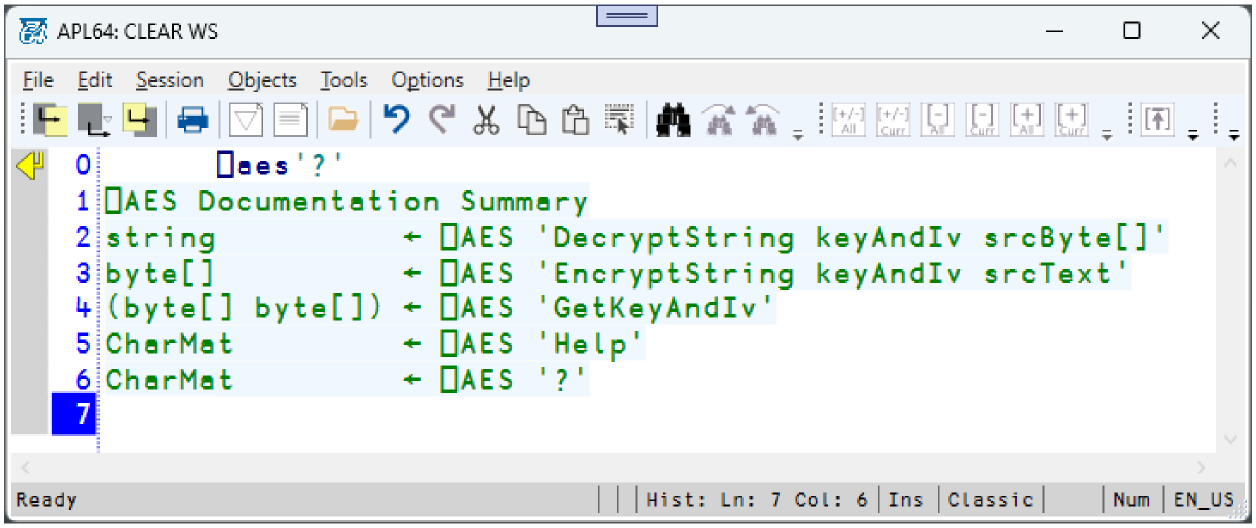
To achieve appropriate security of the encrypted text, the AES key and iv values must be maintained in a secure way.

Help

Obtain a summary of the ⎕AES documentation

Synonym: ?

Syntax: CharMat ← ⎕AES 'Help'


APL64: CLEAR WS
File Edit Session Objects Tools Options Help

```
      ⎕aes'?'
⎕AES Documentation Summary
string             ← ⎕AES 'DecryptString keyAndIv srcByte[]'
byte[]             ← ⎕AES 'EncryptString keyAndIv srcText'
(byte[] byte[])    ← ⎕AES 'GetKeyAndIv'
CharMat            ← ⎕AES 'Help'
CharMat            ← ⎕AES '?'
```

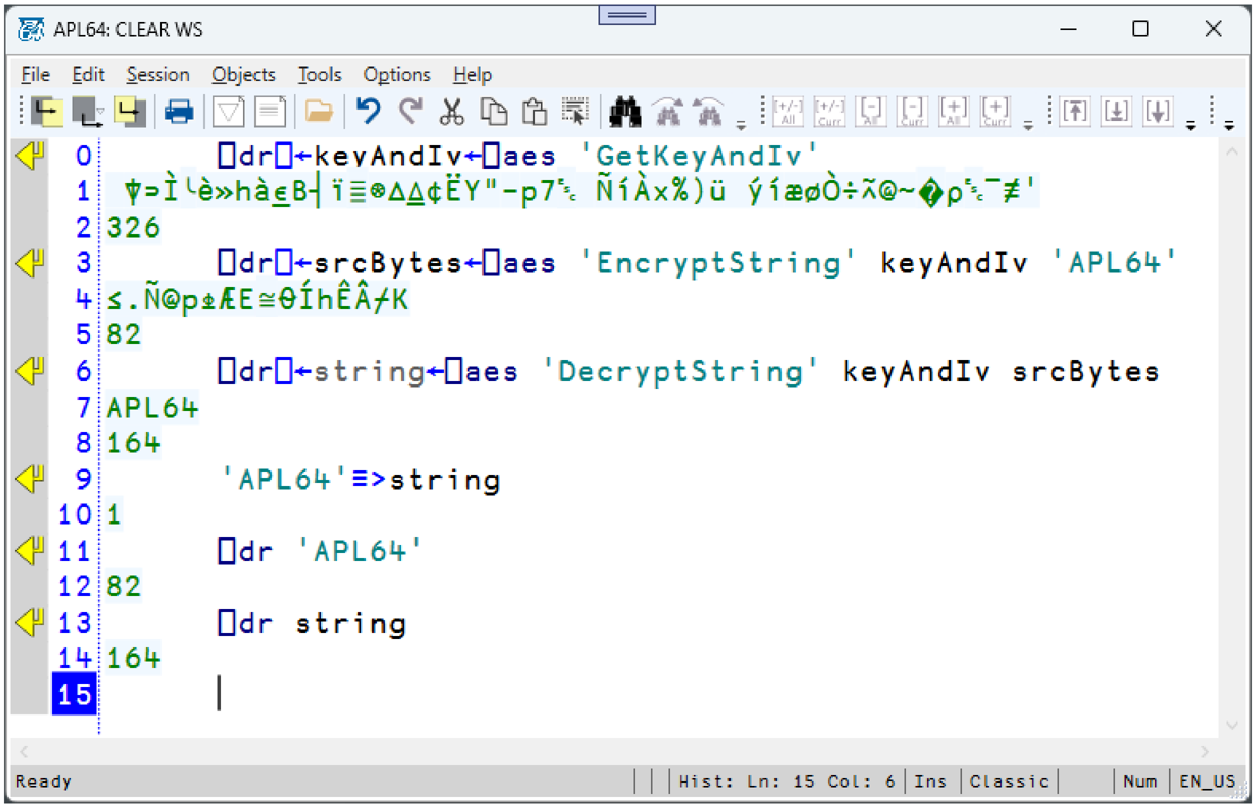
Ready
Hist: Ln: 7 Col: 6 Ins Classic Num EN_US

Examples

Example #1 - Encrypt and decrypt some text

```
⎕dr⎕←keyAndIv←⎕aes 'GetKeyAndIv'
⎕dr⎕←srcBytes←⎕aes 'EncryptString' keyAndIv 'APL64'
```

```
⎕dr⎕←string←⎕aes 'DecryptString' keyAndIv srcBytes
'APL64'≡>string
⎕dr 'APL64'
⎕dr string
```

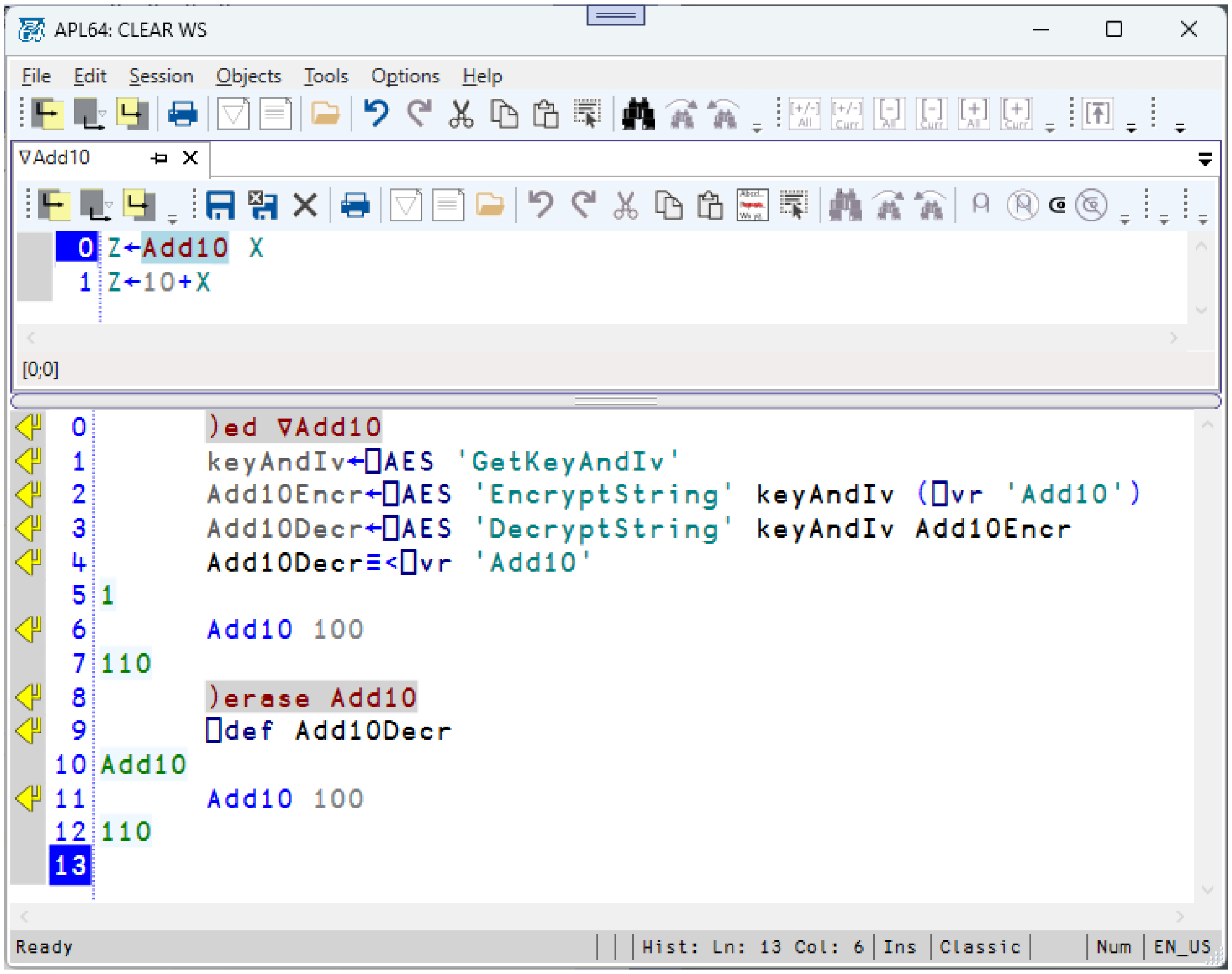


```
        ⎕dr⎕←keyAndIv←⎕aes 'GetKeyAndIv'
 ψ⊃Ì╰è»hà⍷B┤ï≣⍟∆⍙¢ËY"-p7␚ ÑíÀx%)ü ýíæøÒ÷⍨@~�ρ␚¯≢'
326
        ⎕dr⎕←srcBytes←⎕aes 'EncryptString' keyAndIv 'APL64'
≤.Ñ@p⍭ÆE≅θÍhÊÂ⍲K
82
        ⎕dr⎕←string←⎕aes 'DecryptString' keyAndIv srcBytes
APL64
164
        'APL64'≡>string
1
        ⎕dr 'APL64'
82
        ⎕dr string
164
```

Ready | Hist: Ln: 15 Col: 6 | Ins | Classic | Num | EN_US

Example #2 - Encrypt and decrypt an APL64 programmer-defined function

```
⎕def 'Z←Add10 X' 'Z←10+X'
)ed ∇Add10
keyAndIv←⎕AES 'GetKeyAndIv'
Add10Encr←⎕AES 'EncryptString' keyAndIv (⎕vr 'Add10')
Add10Decr←⎕AES 'DecryptString' keyAndIv Add10Encr
Add10Decr≡<⎕vr 'Add10'
Add10 100
)erase Add10
⎕def Add10Decr
Add10 100
```



```
Z←Add10 X
Z←10+X
```

[0;0]

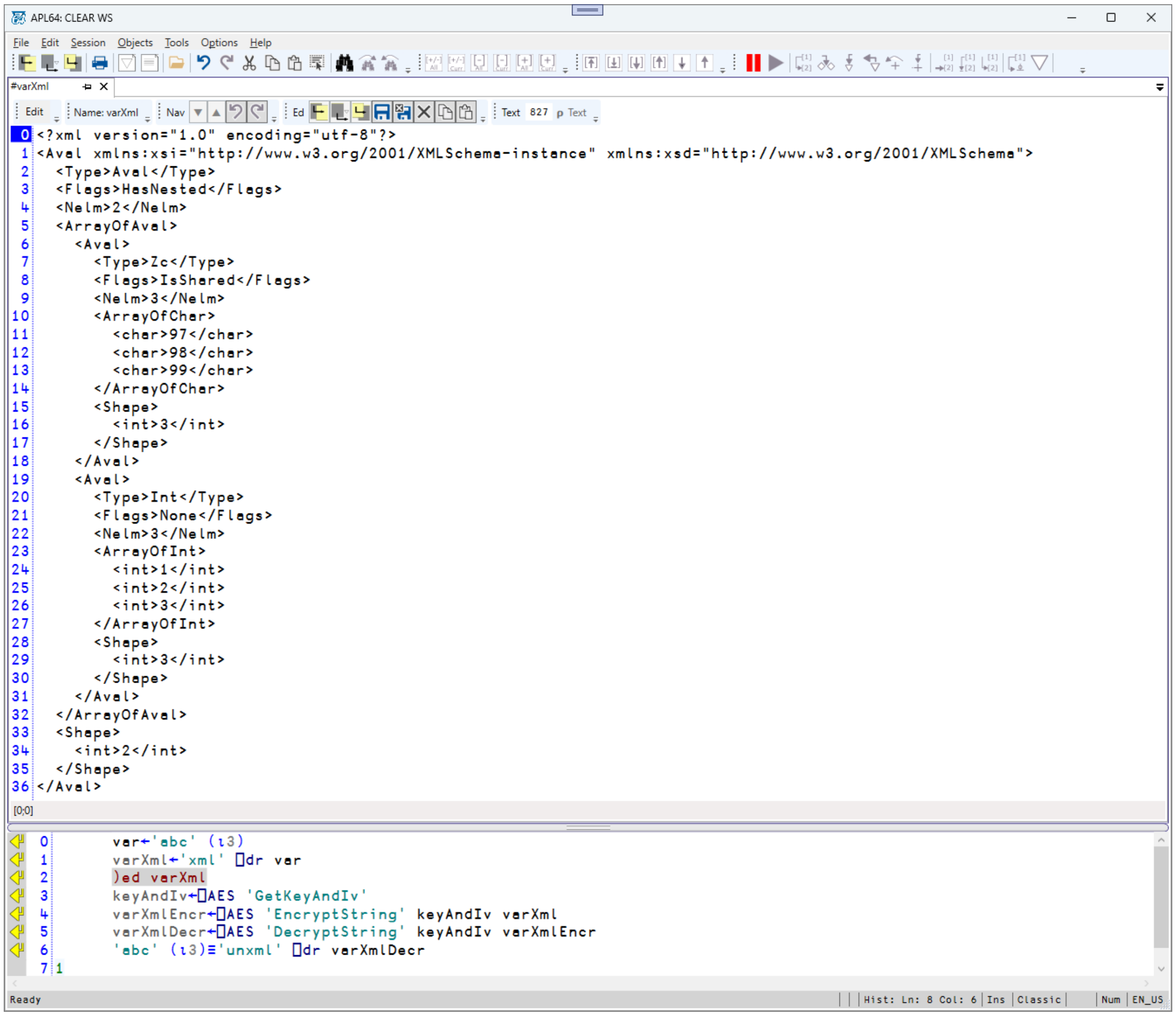
```
      )ed ∇Add10
      keyAndIv←⎕AES 'GetKeyAndIv'
      Add10Encr←⎕AES 'EncryptString' keyAndIv (⎕vr 'Add10')
      Add10Decr←⎕AES 'DecryptString' keyAndIv Add10Encr
      Add10Decr≡<⎕vr 'Add10'
1
      Add10 100
110
      )erase Add10
      ⎕def Add10Decr
Add10
      Add10 100
110
```

Ready | Hist: Ln: 13 Col: 6 | Ins | Classic | Num | EN_US

Example #3 - Encrypt and decrypt an APL64 variable value

Since encryption and decryption apply to text, the xml-format text representation of the APL64 variable is obtained using the APL64 ⎕DR system function and then the xml-format text is encrypted and decrypted.

```
var←'abc' (ι3)
varXml←'xml' ⎕dr var
)ed varXml
keyAndIv←⎕AES 'GetKeyAndIv'
varXmlEncr←⎕AES 'EncryptString' keyAndIv varXml
varXmlDecr←⎕AES 'DecryptString' keyAndIv varXmlEncr
('abc' ι3)≡⎕←'unxml' ⎕dr varXmlDecr
```



```
<?xml version="1.0" encoding="utf-8"?>
<Avel xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Type>Avel</Type>
  <Flags>HasNested</Flags>
  <Nelm>2</Nelm>
  <ArrayOfAvel>
    <Avel>
      <Type>Zc</Type>
      <Flags>IsShared</Flags>
      <Nelm>3</Nelm>
      <ArrayOfChar>
        <char>97</char>
        <char>98</char>
        <char>99</char>
      </ArrayOfChar>
      <Shape>
        <int>3</int>
      </Shape>
    </Avel>
    <Avel>
      <Type>Int</Type>
      <Flags>None</Flags>
      <Nelm>3</Nelm>
      <ArrayOfInt>
        <int>1</int>
        <int>2</int>
        <int>3</int>
      </ArrayOfInt>
      <Shape>
        <int>3</int>
      </Shape>
    </Avel>
  </ArrayOfAvel>
  <Shape>
    <int>2</int>
  </Shape>
</Avel>
```

[0;0]

```
      var←'abc' (⍳3)
      varXml←'xml' ⎕dr var
      )ed varXml
      keyAndIv←⎕AES 'GetKeyAndIv'
      varXmlEncr←⎕AES 'EncryptString' keyAndIv varXml
      varXmlDecr←⎕AES 'DecryptString' keyAndIv varXmlEncr
      'abc' (⍳3)≡'unxml' ⎕dr varXmlDecr
1
```

Ready | Hist: Ln: 8 Col: 6 | Ins | Classic | Num | EN_US

⎕ai Accounting Information

Purpose:
Returns current accounting information.

Syntax: result ← ⎕ai

Result:
The result is a four-element numeric vector containing:
[1] Your user number
[2] Number of seconds since the start of this APL session
[3] Reserved for future enhancement
[4] Reserved for future enhancement

⎕ai relies on the operating system clock for time measurement. User number is 1 by default; you can set it in the .INI file or override the default on the command line when you start APL64.

Caution:
⎕ai, as described here, is specific to this APL64 system. The length and definition of each item of result may be different in other APL systems or in future releases of this system. When writing your own functions, do not rely on the result of ⎕ai being a four-element vector.

Example:
The following expression provides the hours, minutes, and seconds since starting the APL64 session:

```
      0 60 60 ⊤ ⎕ai[2]
1 13 25.75
```

⎕al Roman Alphabet Lower Case Glyphs

Purpose:
Returns the lowercase Roman alphabet characters (glyphs).

Syntax: result ← ⎕al

Result:
result is a character vector.

Example:

```
      ⎕al
abcdefghijklmnopqrstuvwxyz
```

⎕arbin Arbitrary Input

Note: ⎕arbin has been deprecated in APL64. Refer to the documentation on ⎕scom for a superior option.

⎕ask Ask Text Dialog

Purpose:
⎕ASK can be used to present a dialog with user defined text caption, text prompt and text default value.

Syntax: CharVector ← [TextCaption] TextPrompt ⎕ASK TextDefaultValue

Result:
CharVector is an APL64 character vector containing the user-edited default value.

Remarks:
⎕ASK provides a replacement for Quote-Quad input (⍞) system function.
To learn more about ⎕ASK, in the APL64 Developer version go to **Help | Developer Version GUI | Using ⎕ASKx System Functions**.

⎕askex Ask Text Dialog

Purpose:
⎕ASKEX can be used to present a dialog with user defined text caption, text prompt and text default value.

Syntax: Value ← [TextCaption] TextPrompt ⎕ASKEX TextDefaultValue

Result:
Value is a two-element vector:

Element #	Element Description
1	User-entered text
2	Interpreter evaluation of user-entered text

Remarks:
⎕ASKEX provides a replacement for Quad input (⎕) system function.
To learn more about ⎕ASKEX, in the APL64 Developer version go to **Help | Developer Version GUI | Using ⎕ASKx System Functions**.

⎕askl Ask Integer Dialog

Purpose:
⎕ASKL can be used to present a dialog with user defined text caption, text prompt and text default value.

Syntax: IntegerValue ← [TextCaption] OptionsList ⎕ASKL NumericDefaultValue

Result:
IntegerValue is an APL64 numeric value containing the index (index origin zero) in the options list of the user-selected item. is a two-element vector.

Remarks:
To learn more about ⎕ASKL, in the APL64 Developer version go to **Help | Developer Version GUI | Using ⎕ASKx System Functions**.

⎕askn Ask Numeric Dialog

Purpose:
⎕ASKN can be used to present a dialog with user defined text caption, text prompt and text default value.

Syntax: NumericValue ← [TextCaption] TextPrompt ⎕ASKN NumericDefaultValue(s)

Result:
NumericValue is an APL64 vector (rank 1) value containing the user-edited value(s).

Remarks:
⎕ASKN can be used instead of quad input when only numeric results are permitted.

To learn more about ⎕ASKN, in the APL64 Developer version go to **Help | Developer Version GUI | Using ⎕ASKx System Functions**.

⎕at Attributes

Purpose:
Returns attributes of workspace objects or the fix time for a function.

Syntax:
```
result ←         code    ⎕at 'namelist'
result ← [scope] code ⎕at 'namelist'
```

Arguments:
namelist is a list of one or more identifiers; the argument can be a character matrix with one name per row; if the argument is a vector or scalar, it is treated as a one-row matrix.

code is an integer of value 1 through 4; however, codes 3 and 4 are reserved.

scope is an optional numeric argument that specifies the scope at which names are interpreted, where 1 = local (default); and 0 = global.

Result:
The explicit result is a matrix that has one row for each row of the argument.

If code is 1, each row of the result has three elements:

The first element is 1 if the object is a variable or a function with an explicit result; it is zero otherwise. The second element is 1 if the object is a monadic function; it is 2 if the object is a dyadic or ambivalent function; it is zero otherwise.

The third element is zero.

If code is 2, and the object is a function, result has the seven elements of the timestamp (⎕ts) when the function was last fixed by the function editor, ⎕def, ⎕defl, or ⎕fx. These elements are year, month, day, hour, minute, second, and millisecond. All seven elements are zero if the object is not a function.

⎕av Atomic Vector

Purpose:
Return a vector of all possible character values.

Syntax:
```
result ← ⎕av
```

Result:
The explicit result is a 256-element vector of all character values.
⎕av represents all possible characters - even those not directly available from the keyboard. You can index the explicit result and store the result in variables; however, you cannot display the entire result of ⎕av since several of its elements are terminal control characters.

Caution:
Do not rely heavily on the order in which the character set is mapped onto the elements in ⎕av, because the order is not the same in all APL systems.

Note: The new block comment glyphs ⍝ and ⍝ are not included in ⎕AV due to being limited to 256 glyphs.

Example:
In the example below, a translate table ALLCAPS is formed to translate character values from lowercase to uppercase.

```
      ⎕io←0
      ⎕av ⍳'ABC'
65 66 67
      ⎕av[65 66 67]
ABC
      OLD←'abc'
      ALLCAPS←⎕av
      IX←(⍳26)+⎕av⍳'a'
      ALPHA←'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
      ALLCAPS[IX]←ALPHA
      NEW←ALLCAPS[⎕av⍳OLD]
      NEW
ABC
```

⎕brotli Data Compression

Purpose:

Brotli is a lossless data compression algorithm developed by Google. It uses a combination of the general-purpose LZ77 lossless compression algorithm, Huffman coding and 2nd-order context modelling. Brotli is primarily used by web servers and content delivery networks to compress HTTP content, making internet websites load faster. A successor to gzip, it is supported by all major web browsers and has become increasingly popular, as it provides better compression than gzip.

The implementation of the Brotli algorithm in APL64 is an interface to the .Net System.IO.Compression toolkit. The APL64 ⎕BROTLI system function provides the tools to compress and decompress byte[] arrays. It is up to the APL64 programmer to select application-specific byte[] arrays to be compressed and decompressed. Any APL64 object which can be rendered as a byte[] array can be compressed and decompressed using ⎕BROTLI.

Syntax: result ← ⎕BROTLI ActionText [action arguments…]

Brotli Actions

Compress

Syntax: byte[] ← ⎕BROTLI 'Compress'   src(byte[]) compressionLevel(Int32)

src is an APL64 byte[] array variable with ⎕DR type 82.

compressionLevel	**Description**
0	Optimal

1	Fastest
2	No compression
3	Smallest size

Decompress

Syntax: result(byte[]) ← ⎕BROTLI ‘Decompress' src(byte[]

src is the result of a ⎕BROTLI compression action.

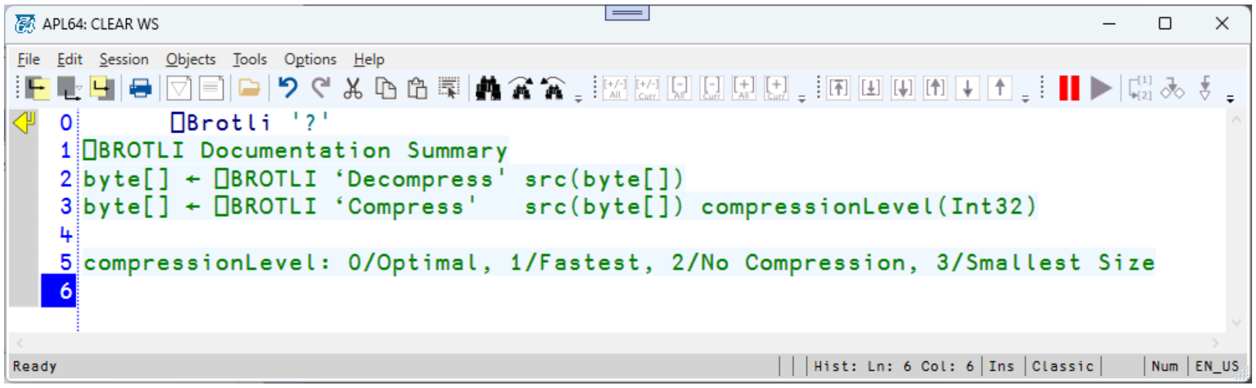
result is the byte[] array which was compressed.

Documentation Summary

Syntax: char[;] ← ⎕BROTLI ‘Help' or char[;] ← ⎕BROTLI ‘?'

Result: A rank-2, character array containing a summary of the ⎕BROTLI system function documentation.

```
⎕Brotli '?'
```



```
      ⎕Brotli '?'
⎕BROTLI Documentation Summary
byte[] ← ⎕BROTLI ‘Decompress' src(byte[])
byte[] ← ⎕BROTLI ‘Compress'   src(byte[]) compressionLevel(Int32)

compressionLevel: 0/Optimal, 1/Fastest, 2/No Compression, 3/Smallest Size
```

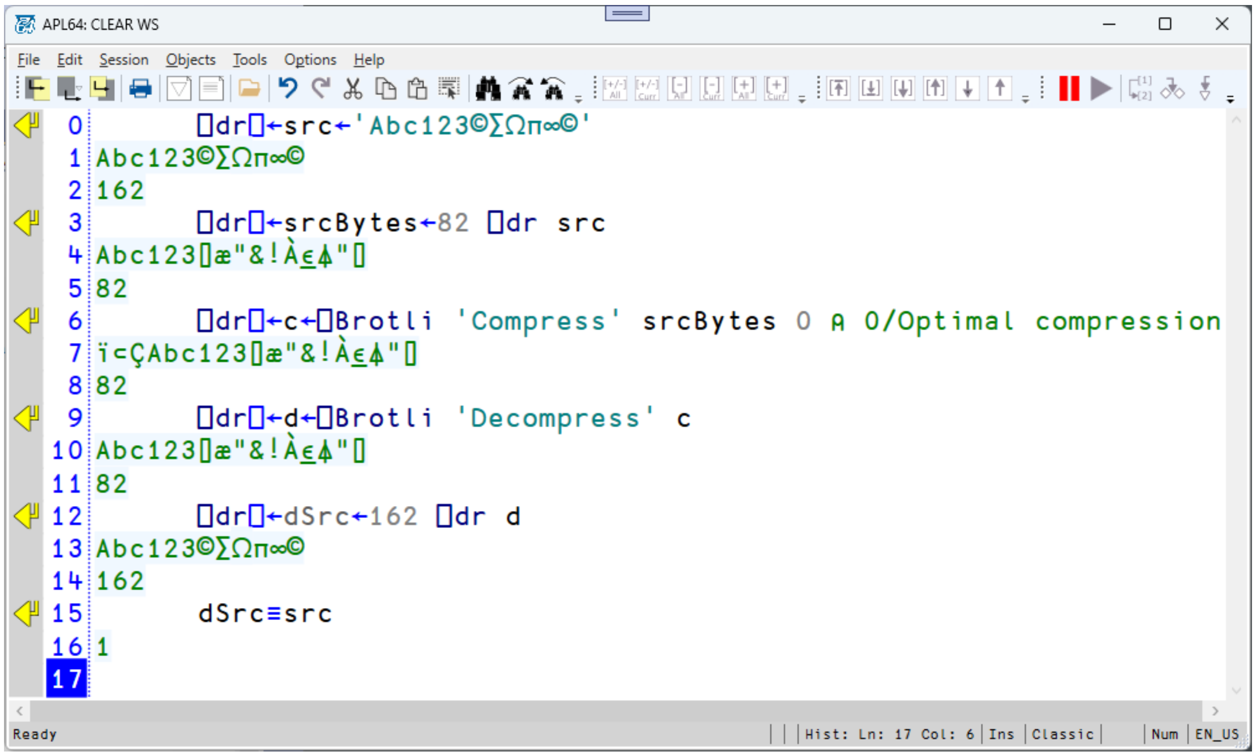
```
Ready                Hist: Ln: 6 Col: 6 Ins Classic   Num EN_US
```

Brotli Examples

Compress Unicode Text

In this example the Unicode text includes code points not in ⎕AV.

```
⎕dr⎕←src←'Abc123©∑Ωπ∞©'

⎕dr⎕←srcBytes←82 ⎕dr src

⎕dr⎕←c←⎕Brotli 'Compress' srcBytes 0 ⍝ 0/Optimal compression

⎕dr⎕←d←⎕Brotli 'Decompress' c

⎕dr⎕←dSrc←162 ⎕dr d
dSrc≡src
```

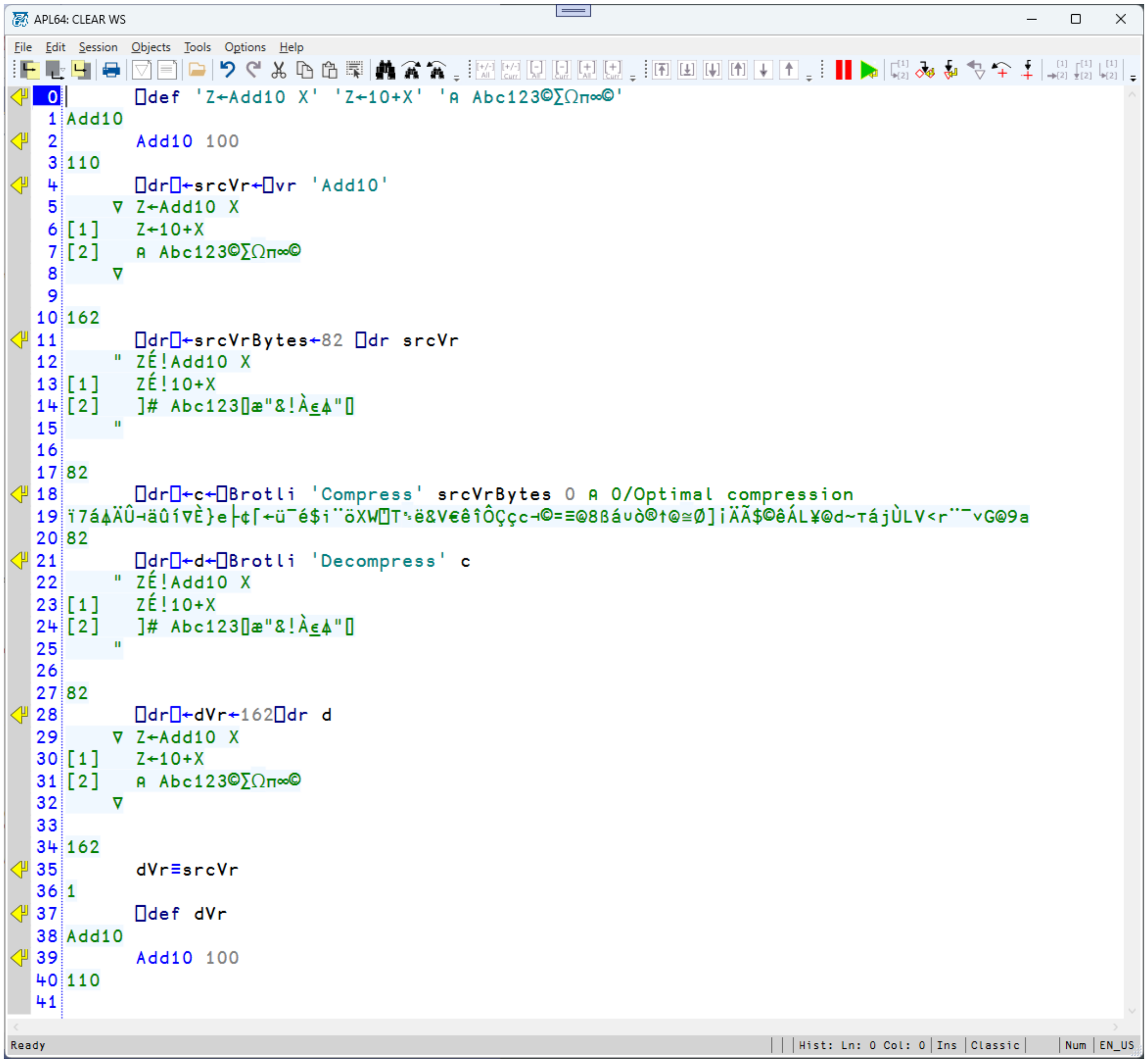


```
      ⎕dr⎕←src←'Abc123©∑Ωπ∞©'
Abc123©∑Ωπ∞©
162
      ⎕dr⎕←srcBytes←82 ⎕dr src
Abc123⌷æ"&!À∊⍋"⌷
82
      ⎕dr⎕←c←⎕Brotli 'Compress' srcBytes 0 ⍝ 0/Optimal compression
ï⊂ÇAbc123⌷æ"&!À∊⍋"⌷
82
      ⎕dr⎕←d←⎕Brotli 'Decompress' c
Abc123⌷æ"&!À∊⍋"⌷
82
      ⎕dr⎕←dSrc←162 ⎕dr d
Abc123©∑Ωπ∞©
162
      dSrc≡src
1
```

Compress an APL64 Function

In this example the ⎕VR of an APL64 programmer-defined function is converted to a byte[] array which is compressed and decompressed.

```
⎕def 'Z←Add10 X' 'Z←10+X' '⍝ Abc123©∑Ωπ∞©'
Add10 100

⎕dr⎕←srcVr←⎕vr 'Add10'

⎕dr⎕←srcVrBytes←82 ⎕dr srcVr

⎕dr⎕←c←⎕Brotli 'Compress' srcVrBytes 0 ⍝
0/Optimal compression

⎕dr⎕←d←⎕Brotli 'Decompress' c

⎕dr⎕←dVr←162⎕dr d

dVr≡srcVr

⎕def dVr
Add10
Add10 100
```



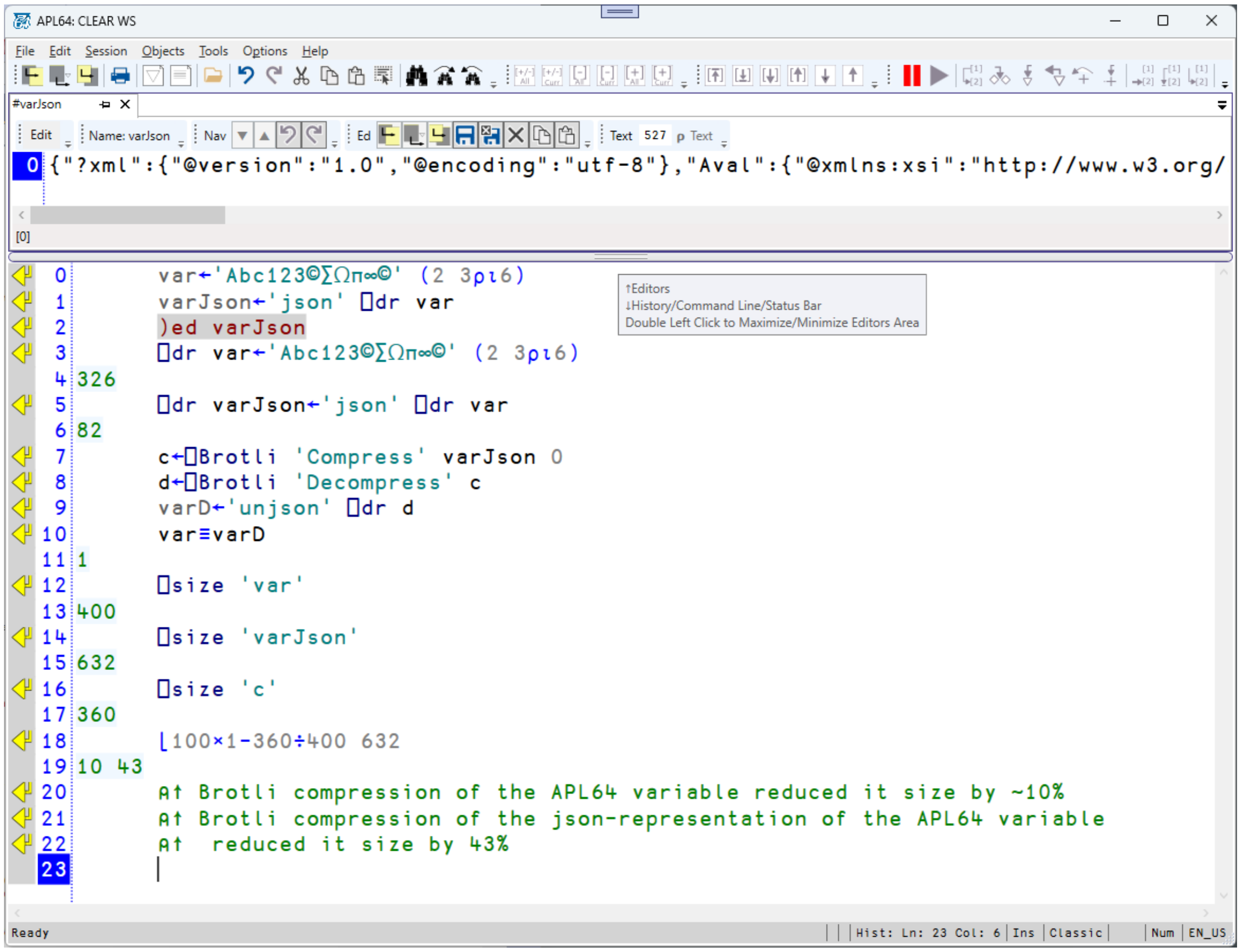
```
      ⎕def 'Z←Add10 X' 'Z←10+X' 'A Abc123©∑Ωπ∞©'
Add10
      Add10 100
110
      ⎕dr⎕←srcVr←⎕vr 'Add10'
    ∇ Z←Add10 X
[1]   Z←10+X
[2]   A Abc123©∑Ωπ∞©
    ∇

162
      ⎕dr⎕←srcVrBytes←82 ⎕dr srcVr
    " ZÉ!Add10 X
[1]   ZÉ!10+X
[2]   ]# Abc123⎕æ"&!À⍷⍋"⎕
    "

82
      ⎕dr⎕←c←⎕Brotli 'Compress' srcVrBytes 0 A 0/Optimal compression
ï7á⍋ÄÛ¬äûí∇È}e├¢⌈←ü¯é$i¨öXW⎕T⍀ë&V€êîÔÇçc⊣©=≡@8ßá∪ò®↑@≅Ø]¡ÄÃ$©êÁL¥@d~⊤ájÙLV<r¨¯∨G@9a
82
      ⎕dr⎕←d←⎕Brotli 'Decompress' c
    " ZÉ!Add10 X
[1]   ZÉ!10+X
[2]   ]# Abc123⎕æ"&!À⍷⍋"⎕
    "

82
      ⎕dr⎕←dVr←162⎕dr d
    ∇ Z←Add10 X
[1]   Z←10+X
[2]   A Abc123©∑Ωπ∞©
    ∇

162
      dVr≡srcVr
1
      ⎕def dVr
Add10
      Add10 100
110
```

Compress an APL64 Variable

In this example the json-representation of an APL64 programmer-developed variable is compressed and decompressed. In APL64 the json-representation of an APL64 programmer-developed variable is a byte[] array, even if the variable value contains Unicode characters not in ⎕AV.

```
var←'Abc123©∑Ωπ∞©' (2 3ρι6)
varJson←'json' ⎕dr var
)ed varJson
⎕dr var←'Abc123©∑Ωπ∞©' (2 3ρι6)

⎕dr varJson←'json' ⎕dr var

c←⎕Brotli 'Compress' varJson 0
d←⎕Brotli 'Decompress' c
```

```
varD←'unjson' ⎕dr d
var≡varD

⎕size 'var'

⎕size 'varJson'

⎕size 'c'

⌊100×1-360÷400 632
```



APL64: CLEAR WS
File Edit Session Objects Tools Options Help
#varJson
Edit Name: varJson Nav Ed Text 527 ρ Text
[0]
↑Editors
↓History/Command Line/Status Bar
Double Left Click to Maximize/Minimize Editors Area
Ready
Hist: Ln: 23 Col: 6 Ins Classic Num EN_US

```
0 {"?xml":{"@version":"1.0","@encoding":"utf-8"},"Aval":{"@xmlns:xsi":"http://www.w3.org/
```

```
      var←'Abc123©∑Ωπ∞©' (2 3ρι6)
      varJson←'json' ⎕dr var
      )ed varJson
      ⎕dr var←'Abc123©∑Ωπ∞©' (2 3ρι6)
326
      ⎕dr varJson←'json' ⎕dr var
82
      c←⎕Brotli 'Compress' varJson 0
      d←⎕Brotli 'Decompress' c
      varD←'unjson' ⎕dr d
      var≡varD
1
      ⎕size 'var'
400
      ⎕size 'varJson'
632
      ⎕size 'c'
360
      ⌊100×1-360÷400 632
10 43
      ⍝ Brotli compression of the APL64 variable reduced it size by ~10%
      ⍝ Brotli compression of the json-representation of the APL64 variable
      ⍝  reduced it size by 43%
```

⎕call Call Machine Language Routine

Note: ⎕call has been deprecated in APL64.

⎕cfappend Append to a Colossal Component File

Purpose:

Inserts a new component into a colossal component file.

Syntax: result ← value ⎕cfappend tieno

Arguments:
value is a value you want to append to a file; it can have any rank, shape, or datatype.

tieno is a positive integer file tie number for a currently tied file.
result is the number of the new component.

System Differences:
This system writes the new component into the smallest, sufficiently large, available hole in the file. Only if there is not a hole large enough to hold the new component is the size of the file increased.

⎕cfcreate Colossal Component File Create

Purpose:
Creates a new colossal component file.

Syntax: result ← 'fileid.ext' ⎕cfcreate tieno
result ← 'fileid.ext' ⎕cfcreate tieno tablesize
result ← 'fileid.ext' ⎕cfcreate tieno tablesize dirsize

Arguments:
fileid is a character vector file identifier with no restrictions on characters or format within APL; you must explicitly specify any file extension. If the name does not pass the operating system conditions for creating a file name, the system signals an error.

tieno is a positive integer file tie number or zero, which allows the system to pick a valid tie number.

tablesize, optional, specifies the size of the hole table, which is a list of empty space available for reuse. When this number is exceeded, the smallest hole in the table is overwritten with the next hole and that space is lost until the file is compacted. The default value, which takes three pages in memory, is 768. You can specify any number, but multiples of 256 may be more efficient. Very large hole tables may have a significant negative effect on performance and may also affect the number of simultaneous ties available on one machine, since memory is allocated for the hole table.

dirsize, optional, specifies the initial size of the component directory. When this number is exceeded, the directory is moved in the file and the size is doubled. The default value is 4096.

Result:
result is the tie number.

Syntax Differences:
You cannot specify a file size limit or a starting component number.

⎕cfdrop Drop Components from a Colossal Component File

Purpose:
Drops components from either (virtual) end of a colossal component file.

Syntax: ⎕cfdrop tieno n

Arguments:
tieno is a positive integer file tie number for a currently tied file.

n is the number of components you want to drop. If n is positive, the system zeroes the directory entries of the n lowest numbered components; if n is negative, it deletes the (|n) highest numbered directory entries.

⎕cfdup Duplicate a Colossal Component File

Purpose:
Creates a copy of a Colossal component file and, if possible, compacts it to occupy less disk space.

Syntax: 'fileid.ext' ⎕cfdup tieno
'fileid.ext' ⎕cfdup tieno tablesize

Arguments:
fileid is a character vector file identifier with no restrictions on characters or format within APL; you must explicitly specify any file extension. If the name does not pass the operating system conditions for creating a file name, the system signals an error.

tieno is a positive integer file tie number of a currently tied file.

tablesize, optional, specifies a new size for the hole table, which is the list of empty space available for reuse.

Effect:
⎕cfdup creates a new file with the specified name (fileid.ext) and copies all the current data from the file specified by tieno into it. In the process, it recovers unused space created by replacing components, which may allow the new file to occupy less disk space than the original file. The old file remains unchanged. You can use the same name as the file being duplicated for the new file. Using the same name replaces the file with the compacted version. The file must be exclusively tied to duplicate it with the same name.

Syntax Differences:
You cannot specify a file size limit or a starting component number.

⎕cferase Erase a Colossal Component File

Purpose:
Erases a colossal component file tied using ⎕cftie. The file must be exclusively tied.

Syntax: 'fileid.ext' ⎕cferase tieno

Arguments:
fileid is a character vector file identifier with no restrictions on characters or format within APL; you must explicitly specify any file extension. If the name does not pass the operating system conditions for a file name, the system signals an error.

tieno is a positive integer file tie number for the currently tied file; it must match fileid.ext.

⎕cfflush Flush Colossal File

Purpose:
Flush file buffers.

Syntax: ⎕cfflush tieno

Arguments:
tieno is a positive integer file tie number

Effect:
Clears buffers and causes any buffered data to be written to the file.

Example:

```
      ⎕cfflush 2
```

⎕cfhist Colossal Component File History

Purpose:
Provides historical information about an APL component file.

Syntax: result ← ⎕cfhist tieno

Arguments:
tieno is a positive integer file tie number of a currently tied file.

Result:
result is a three-row matrix containing information about the history of the file.

Row 1 contains the user number of the file owner and the timestamp of the file's creation in packed form and ⎕ts form.

Row 2 contains the user number and timestamp associated with the most recent change to the file.

Row 3 contains the user number and timestamp associated with component 0.

Note: See the description of ⎕cfrdci for detailed timestamp information.

⎕cfhold Hold Colossal File(s)

Purpose:
Coordinates file operations in shared colossal file systems.

Syntax: ⎕cfhold tieno

Arguments:
tieno is scalar or vector of positive integer file tie numbers of the files you wish to reserve for update.

Effect:
Provides an interlock scheme by which multiple users can coordinate file updates. Only one user can have the interlock at any one time. Each user who executes ⎕cfhold waits in a queue for a turn to have the interlock.

Invoking ⎕cfhold does not lock files; this is a cooperative system. While an interlock is set, the system delays other users in turn from completing execution of their ⎕cfhold operations but not from executing other file operations. The file system itself locks files during a write operation, so that it restricts conflicts. Thus, there is little point in using this function on a single file. If you have multiple files tied, and they are interdependent, you can hold a group of files to keep any of them from being written by another cooperating user until you have completed your operations.

⎕cfhold first releases any current interlocks that you have, and then, when it is your turn, sets an interlock on each designated file. The system does not set any interlocks while another user has an interlock set on any of the designated files; ⎕cfhold execution waits until all such other interlocks have been released.

The system releases all interlocks when the user who set them executes another ⎕cfhold, exits APL64, enters immediate execution mode, or signals a strong interrupt. You can release the interlock on an individual file without affecting other interlocks by untying or retying the file. An empty vector releases all interlocks and does not set any. The system does not release file interlocks when a program stops for quad (⎕) or quote-quad (⍞) input. Stopping for input when files are held can impose long delays on other users and should be avoided except when necessary.

Because of limitations of the operating system, the order in which users with contending file holds are granted access is not necessarily the same as the order in which they requested the file holds.

⎕cfinfo Colossal Component File Information

Purpose:
Returns information about the state of a colossal component file.

Syntax: result ← ⎕cfinfo tieno

Arguments:
tieno is a positive integer file tie number for a currently tied file.

Result:
The result is a nine-element floating point vector that contains the information shown below.

Element	**Description**
[1]	Not used by APL64. (See Note 1)
[2]	The version number of the tied file.
[3]	The number of slots available for entries in the current component directory.
[4]	The number of entries currently occupied.
[5]	The number of slots available for entries in the hole table.
[6]	The number of hole entries currently occupied.
[7]	The address of the first locking byte available for application use.
[8]	The number of application locking bytes available.
[9]	The active write synchronization value for the tied file (see ⎕cfsync for details).

Note: 1: The first element is not used by APL64 due to a security risk.

⎕cflock Lock Colossal File

Purpose:
Lock or unlock a colossal file.

Syntax: ⎕cflock tieno operation position length

Arguments:
tieno is a positive integer file tie number

operation is a Boolean value: 1 for locking the file, 0 for unlocking the file

position is a 64-bit integer specifying the beginning of the range to lock (the value of this argument must be equal to or greater than zero)

length is a 64-bit integer specifying the range to be locked (the value of this argument cannot be negative)

Effect:
Locking part of a file prevents other processes from reading from or writing to that part of the file. Unlocking a file allows access by other processes to all or part of a file that was previously locked.

Examples:

```
      ⎕cflock 2 1 240 1      ⍝ lock
      ⎕cflock 2 0 240 1      ⍝ unlock
```

⎕cfnames Names of Tied Colossal Component Files

Purpose:
Returns the file identifiers of all component files tied with ⎕cftie or ⎕cfstie.

Syntax: result ← ⎕cfnames

Result:
The result is a character matrix of file identifiers. Each row of ⎕cfnames is a complete path name, including drive letter and colon or UNC prefix, exactly as passed to the host operating system to open the file at the time it was tied. The rows of result have the same order as ⎕cfnums.

⎕cfnums Tie Numbers of Tied Colossal Component Files

Purpose:
Displays the tie numbers of all component files tied with ⎕cftie or ⎕cfstie.

Syntax: result ← ⎕cfnums

Result:
The result is a numeric vector of file tie numbers. The tie numbers are in the same order as the file names that ⎕cfnames reports; that is, the order in which they were tied. Note that ⎕cfnums does not include the tie numbers assigned by either the traditional or extended component file functions.

⎕cfrdci Colossal Component File Component Information

Purpose:
Returns information about one component of a colossal component file.

Syntax: result ←   ⎕cfrdci tieno compno
            result ← [selector] ⎕cfrdci tieno compno

Arguments:
selector is the optional left argument that specifies which elements of component information are to be returned. This is ⎕IO sensitive index of the element(s) to be returned. This argument may be omitted or specified as an integer scalar or vector.

tieno is a positive integer file tie number for a currently tied file.

compno is an integer scalar or vector that contains the component number of the component about which you want information.

Result:
The result is a 10-element floating point vector:
    The first element is the size of the component's value, in bytes.
    The second element is the user number of the person who last replaced or appended the component.
    The third element is the time that the component was last replaced or appended, in a packed-timestamp form given in microseconds since midnight 1 January 1900 UTC. The internal 64-bit calculations for this figure are accurate and will remain within bounds for many lifetimes (to the year 30,000). The floating-point representation may lose some significance.
    The remainder is the component timestamp in a seven-element unpacked form (year, month, date, hour, minute, second, millisecond).

If the selector is omitted and the second element of the right argument is a scalar, the result is a 10-element floating-point vector. If the second element of the right argument is a vector, then the result is a 10-column matrix with one row for each of the specified components.

If the selector is specified, its shape affects the shape of the result.

If the selector is a scalar and the second element of the right argument is a scalar, then the result will be a scalar. If the selector is a scalar and the second element of the right argument is a vector, then the result will be a vector. If the selector is a vector and the second element of the right argument is a scalar, then the result will be a vector. If the selector is a vector and the second element of the right argument is a vector, then the result will be a matrix.

System Differences:
The old system was based on local time; the new system uses Universal Time (Greenwich Mean Time). The old system always had zero for the milliseconds.

⎕cfread Read from a Colossal Component File

Purpose:
Reads a component of a colossal component file.

Syntax: result ← ⎕cfread tieno compno skipmd5

Arguments:
tieno is a positive integer file tie number for a currently tied file.

compno is a positive integer component number that you want to read, or zero for the special metadata component.

skipmd5 is a Boolean flag for skipping the md5 check during the read. Note: There is a risk involved in doing this. If the component has been damaged or mis-read, the MD5 sum protects against both data corruption and system failure. Skipping this check makes applications vulnerable to these issues. This option should be used with care.

Result:
result is the actual value of the component stored in the file.

⎕cfrename Rename a Colossal Component File

Purpose:
Changes the name of a colossal component file. The file must be exclusively tied.

Syntax: 'fileid.ext' ⎕cfrename tieno

Arguments:
fileid.ext is a character vector file identifier with no restrictions on characters or format within APL; you must explicitly specify any file extension. If the name does not pass the operating system conditions for a file name, the system signals an error.

tieno is the file tie number for the currently tied file.

System Differences:
You cannot rename a file and change its size as you could with the traditional ⎕frename function but not the ⎕xfrename function.

⎕cfreplace Replace a Component in a Colossal Component File

Purpose:
Changes the value of an existing component of a colossal component file.

Syntax: value ⎕cfreplace tieno compno

Arguments:
value is the value you want to store; it can be any rank, shape, or datatype.

tieno is a positive integer file tie number for a currently tied file.

compno is an integer component number of the component whose value you want to replace, or zero for the special metadata component.

System Differences:

When the system writes a new value, it uses the smallest sufficiently large hole in the file, if there is one. After it writes the entire value, it records the space where the old value was stored as a hole available for reuse. Performing a single replace of a component may increase the size of the file, but performing repeated replaces of a component with slightly larger-sized values will not cause repeated increases in file size (except under the most pathological conditions of non-adjacent component replaces and perverse sizes).

⎕cfsize Colossal Component File Size Information

Purpose:
Returns information about the size and contents of a colossal component file.

Syntax: result ← ⎕cfsize tieno

Arguments:
tieno is a positive integer file tie number for a currently tied file.

Result:
The result is a six-element floating point vector that contains the information shown below.

Element	**Description**
1	The number of the first component in the file.
2	The number of the next available component.
3	The physical storage (in bytes) that the file uses, including data and overhead.
4	Reserved for future use.
5	The amount of unused file space (gas) in the file.
6	The amount of unused file space lost by overwriting holes (hole table exceeded).

⎕cfstie Extended Share Tie a Colossal Component File

Purpose:
Ties a colossal component file for shared use.

Syntax: result ← 'fileid.ext' ⎕cfstie tieno

Note: See the description of ⎕cftie for the arguments and additional information.

⎕cfsync File Synchronization for a Colossal Component File

Purpose:
Requests the operating system to write pending data to the storage device, or sets the file synchronization value for the specified files.

Syntax:
```
                ⎕cfsync tieno
     synchmode  ⎕cfsync tieno
```

Arguments:
tieno is a vector of positive integer file tie numbers for currently tied files. If you invoke ⎕cfsync

monadically, it requests the system to synchronize the specified files immediately. The effect is a flushed buffer; on error, the operating system signals an error.

synchmode is a code that sets the active file synchronization code for all files listed in the right argument. The possible values are:

Code	**Description**
0	No automatic file synchronization. See Note 1.
1	File synchronization call issued when file is untied. This is the default value.
2	File synchronization call issued on completion of ⎕cfcreate, ⎕cfrename, ⎕cfappend, ⎕cfreplace, ⎕cfdrop, and ⎕cfdup. Equivalent to traditional system.
3	File synchronization call issued on every "logical" file write operation. See Note 2.
4	File synchronization call issued on every file write operation. See Note 2.

Notes:

1) If you use zero, the application must use monadic ⎕cfsync or rely on the operating system to synchronize the file.
2) Codes 3 and 4 are not currently implemented.
3) The value of the file synchronization code is returned as the 9th element of ⎕cfinfo.

System Differences:
In the traditional component file system, flushing file buffers on every file operation is automatic. In the colossal component file system, flushing file buffers is automatic only on file close. The user may override this state with the dyadic form of ⎕cfsync, or invoke specific flushes at critical points in the application using the monadic form.

⎕cftie Extended Tie a Colossal Component File

Purpose:
Ties a colossal component file.

Syntax:
```
result ← 'fileid.ext' ⎕cftie tieno
         'fileid.ext' ⎕cftie tieno [openmode]
                      ⎕cftie tieno
```

Arguments:
fileid.ext is a character vector file identifier with no restrictions on characters or format within APL; you must explicitly specify any file extension. If the name does not pass the operating system conditions for a file name, the system signals an error.

tieno is a positive integer file tie number or zero, which allows the system to pick a valid tie number. result is the tie number.

openmode is an optional valid integer DOS open mode.

The open mode allows access to colossal files by multiple users or processes. The open mode values consist of the sum of one code that indicates the type of access you want and a second code that indicates the type of access you grant to others while you have the file tied. If another user has opened the file first, the access you want must be compatible with the access granted by the other user, and the access you grant must be compatible with the access that user requested.

If you specify open mode explicitly, the supported values are 0, 2, 16, 18, 32, 34, 64, or 66. Thus, if you want to be able to read and write to a file but allow other users read-only access while you have the file tied, you will specify 34 for open mode.

Access Requested	**Access granted to other users**
0 = read access	16 = no access allowed (exclusive)
2 = read and write access	32 = read access allowed, write access denied
	64 = read and write access allowed

If a second user attempts to tie the file with an open mode that is incompatible with the open mode specified by the first user, the system signals an error. The question of compatibility involves both the access requested value and the access granted value. If the first user includes 16 in open mode, then no second user can tie the file. If the first user specifies 32 or 34, then the second user can request only read access; any value that includes 2 results in an error. If the first user specifies 64 or 66, the second user can request read and write access.

The second user must also specify an access granted value that is compatible with the first user's requested access. If the first user requested both read and write access, the second user must include 64. A second user cannot successfully specify exclusive access.

If you specify one of the access-requested values, but do not specify an access granted value (that is, open mode of 0 or 2), the effect for APL is the same as specifying exclusive access; another instance of APL cannot tie the file. If you do not specify open mode at all, the default value is 18.

Note that you can also trigger an error for access reasons unconnected with another user, for example, requesting write access on a file which the operating system considers to be read-only.

Monadic ⎕cftie takes a tie number as the right argument and returns a two-element integer vector indicating the file tie state. The value of the first element indicates what the current open mode of the file is. The second element indicates if the tie was shared tie or an exclusive tie. Shared ties show a value of 0, exclusive ties show a value of 1.

The optional left argument must be a valid colossal file identifier.

Note: The function does not support write-only access.

Effect:
The system ties a colossal file. By default, the file is exclusively tied. No other user can tie the file while it remains exclusively tied. File ties are "slippery;" that is, if a file is already tied to one tie number, you can tie that file to the same number or to another unused tie number without first untying the file. The argument designates the file tie number and, optionally, an open mode. The open mode allows multiple users or processes access to colossal files. The open mode values consist of the sum of one code that indicates the type of access you want and a second code that indicates the type of access you grant to others while you have the file tied. If another user has opened the file first, the access you want must

be compatible with the access granted by the other user, and the access you grant must be compatible with the access that the user requested.

When the open mode is not specified, the function first attempts to open the file with read and write access and denies access to others. Suppose the first attempt to open the file fails for a non-transient reason, such as access being denied, or network access being denied. In that case, the function will attempt to open the file again with read-only access and deny access to others.

Note: When the open mode is specified, the function attempts to open the file with the specified access. If the first attempt fails, the function doesn't use the alternate (read-only) access or open the file again. Instead, the system signals an error. To erase a file that is tied with read-only access, the file needs to be tied again with the exclusive open mode.

Access:
The file must exist, and the user must have the permission to tie the file. and the passnumber must match an appropriate one in the access matrix. The access code for ⎕cftie is 2.

Examples:

```
'PRIME NUMBERS.CF' ⎕cftie 37
'C:\APLW\FILES\MYFILE.CF' ⎕cftie 2
```

⎕cfuntie Untie Colossal Component Files

Purpose:
Unties one or more colossal component files.

Syntax: ⎕cfuntie tieno1 tieno2 tieno3 . . .tieno**n**

Argument:
tieno1 tieno2 tieno3 . . . tieno**n** are file tie numbers of currently tied files you want to untie.

Note: This function works only on new component files tied using ⎕cftie or ⎕cfstie. It does not affect any component files tied with either the traditional or extended file tie functions.

⎕cfversion Version Number of the Colossal Component File System

Purpose:
Displays the version number of the colossal component file system.

Syntax: result ← ⎕cfversion

Result:
The result is the version number of the current colossal component file system; that is, the system as implemented in the version of APLNow32.exe that you are using. This is not necessarily the version of any file you have tied.

⎕chdir Change Current Directory

Purpose:
Reports or changes the default directory on a specified disk drive, or changes the default disk drive.

Syntax:
```
result ←         ⎕chdir 'path'
result ←         ⎕chdir 'drive'
result ←         ⎕chdir ''
result ← ''      ⎕chdir 'path'
result ← [drive] ⎕chdir 'path'
result ← [drive] ⎕chdir 'drive'
result ← [drive] ⎕chdir ''
```

Arguments:
drive as the left argument is a single letter identifying the disk drive.

drive as the right argument is a two-element vector comprising the disk drive letter, followed by a colon (:).

path is a character vector that specifies the disk drive, followed by a colon (:) and a directory path name or a UNC (Universal Naming Convention) path and directory name. If you omit the disk drive letter, ⎕chdir applies to the current disk drive.

The optional left argument is a disk drive you want to make the default before performing the actions specified by the right argument. It normally is the same as the disk drive specified in the right argument, but it need not be.

If the right argument contains a disk drive letter followed by a colon (:) and a path name, ⎕chdir changes the default directory for that disk drive to the specified path. If the right argument is a disk drive letter followed by a colon, the explicit result indicates the default directory for that disk drive.

If the right argument contains a UNC (Universal Naming Convention) path and directory name, an empty left argument is required.

An empty right argument returns the current directory for the default disk drive.

Result:
If ⎕chdir changes the current directory, the explicit result is the current directory before the change was made. If the argument does not specify a new path, ⎕chdir returns the default directory for the disk drive.

Effect:
Monadic ⎕chdir performs the same function as the DOS command CD. It changes the working directory for the specified disk drive, but does not affect the default disk drive. Dyadic ⎕chdir changes the default disk drive.

Note: Unlike APL+Win, the current directory will not change when you perform certain actions, such as loading a new workspace. This is true whether you use a system command, a system function, or the File menu. Additionally, the [Session]NavigateDirs parameter to control whether to load a workspace without changing the current directory has been deprecated in APL64.

Examples:
Report the current directory on the default disk drive:

```
      ⎕chdir ''
C:\WINNT
```

Change the default disk drive to D and report the current directory on drive C:

```
      'D' ⎕chdir 'C:'
C:\WINNT
```

Report the current directory on the default disk drive:

```
      ⎕chdir ''
D:\APL
      )wslib
DATETIME
```

Change the current directory on disk drive D:

```
      ⎕chdir 'D:\APL\APLWIN'
D:\APL
      )wslib
EXAMPLE  MNP  UGLY SPACE
```

Change the current directory to a UNC path and directory name:

```
      '' ⎕chdir '\\server\share'
D:\APL\APLWIN
```

⎕cm Character Matrix

Purpose:
Returns a user-specified delimited character matrix result.

Syntax: result ← [separator] ⎕cm array

Argument:
array is a nested vector of character vectors or scalars.

separator is the optional left argument to specify the line separator sequence. This can be a character scalar or a non-empty character vector. For example, comma and semi-colon.

The overall array and each item and subitem in the array is processed in ravel order. But this is not to say it is the same as ⎕enlist. Each character matrix or higher rank item in the argument (including sub-items) gets an implicit newline for each row. Each nested sub-item gets at least one new row in the result also (and if it contains nested sub-items or character matrix or higher rank items, those also get newlines in the result).

In addition to allowing ⎕tcnl as a line separator in the argument, ⎕cm (as well as ⎕fx and ⎕cv) also recognize ⎕tclf or the two-character sequences (⎕tcnl,⎕tclf) and (⎕tclf,⎕tcnl) to represent one newline in the result. This means that the string "abc",⎕tcnl,⎕tcnl,"def" gives three lines in the result (one of them blank between "abc" and "def"). But the similar string "abc",⎕tcnl,⎕tclf,"def" will only produce two lines in the result because the ⎕tcnl,⎕tclf sequence is considered a single newline in argument. These rules make it easier to handle text files that may have originated in Windows, Linux, Unix, OSX, etc, with equal ease.

Note: ⎕cm will remove trailing blanks from lines.

Example:

```
     ⎕cm 'abc' (2 2ρ'DEFG') ('hello',⎕tcnl,'world!')
abc
DE
FG
hello
world!
     ρ⎕cm 'abc' (2 2ρ'DEFG') ('hi',⎕tcnl,'there')
5 5
```

Or even more complicated, we have the following:

```
     ⎕cm 3 2ρ 'abc' (2 2ρ'DEFG') ('hello',⎕tcnl,'world!')
abc
DE
FG
hello
world!
abc
DE
FG
hello
world!
     ρ⎕cm 3 2ρ 'abc' (2 2ρ'DEFG') ('hello',⎕tcnl,'world!')
10 6
     ':,' ⎕cm 'APL: IS, EASY'
APL
 IS
 EASY
```

⎕cmd Execute DOS Command

Purpose:
Executes the command given to it from the DOS environment with or without elevated Administrator permissions and returns to resume APL execution in the workspace.

Syntax:
```
result ←          ⎕cmd 'command'
result ← 'code'   ⎕cmd 'command'
result ←          ⎕cmd ''
```

Arguments:
command is a character singleton or vector of up to 8192 characters that represents the DOS command you want to execute.

code is a numeric scalar in the range of 0-7 that determines whether APL waits for DOS to complete and whether DOS opens a visible window:

Values	Description
0	APL continues
1	DOS executes the command; APL waits until DOS completes
2	APL continues without waiting; DOS executes with no window
3	DOS executes with no window; APL waits until DOS finishes
4	APL continues (elevated Administrator permission). See Note.
5	DOS executes the command; APL waits until DOS completes (elevated Administrator permission). See Note.
6	APL continues without waiting; DOS executes with no window (elevated Administrator permission). See Note.
7	DOS executes with no window; APL waits until DOS finishes (elevated Administrator permission). See Note.

Note: You should see the UAC prompt asking for permission to run the command with administrator permissions provided UAC prompting isn't disabled in Windows' User Account Control Settings.

If the right argument to ⎕cmd is an empty vector (⎕cmd ''), the system loads a copy of the DOS command interpreter and allows you to type any number of DOS commands from the keyboard. (Note that this is true regardless of the left argument you supply.) To return to APL, type: EXIT

Result:
The explicit result; always 0 0⍴''.

Effect:
⎕cmd creates a command session in which DOS runs the command either with or without opening a window. If a window is created, Windows destroys the window when the command terminates. If the left argument is missing or odd (either 1 or 3), APL is blocked while the operating system executes the command. The system loads from disk the DOS command processor file that the environment variable COMSPEC= specifies; this is normally cmd.exe.

To set this environment variable, use the command SET from DOS before entering APL64. You cannot change the COMSPEC= parameter during a ⎕cmd session, because such a session lasts only until you return from DOS back to APL.

Before the system loads the DOS command interpreter, it forces all the file buffers for tied files to disk. On return to APL, the system checks all tied files to ensure that they still exist. If an error occurs during the execution of a DOS command under ⎕cmd, DOS reports the error and returns control to APL64. The system does not record the DOS error in ⎕dm.

Warning:
Never rename a file that is tied by the suspended APL64 session. Files on your disk will be damaged if you write to a file that was tied by APL64 while being renamed within DOS.

Caution:
This command may not work under future Windows operating systems, as Microsoft continues to disavow DOS.

Examples:
Enter the DOS command level until you type "exit":

```
 ⎕cmd ''
```

Create a directory file.

```
 3 ⎕cmd 'DIR>TEMP.TXT'
```

⎕cn Character Nested Array

Purpose:
Normalizes any character array or scalar into a nested vector of character arrays.

Syntax: result ← [separator] ⎕cn array

Arguments:
array is any array of character arrays or scalars or strings.

separator is the optional left argument to specify the line separator sequence. This can be a character scalar or a non-empty character vector. For example, comma and semi-colon.

Result:
The result is a nested vector of character arrays.

Note: ⎕cn will remove trailing blanks from arrays.

Example:

```
  DISPLAY ('M' ('F',  ('C1' (3 3ρ'ABCDEFGHI'))),⊂'abc', ⊂'def')
+→-----------------------------------+
|   +→-------------+ +→------------+ |
| M |   +→-+ +→--+ | |       +→--+ | |
| - | F |C1| ↓ABC| | | a b c |def| | |
|   | - +--+ |DEF| | | - - - +---+ | |
|   |        |GHI| | +∊------------+ |
|   |        +---+ |                 |
|   +∊-------------+                 |
+∊-----------------------------------+

  DISPLAY ⎕CN ('M' ('F',  ('C1' (3 3ρ'ABCDEFGHI'))),⊂'abc', ⊂'def')
+→-------------------------------------------+
| +→+ +→+ +→-+ +→--+ +→--+ +→--+ +→--+ +→--+ |
| |M| |F| |C1| |ABC| |DEF| |GHI| |abc| |def| |
| +-+ +-+ +--+ +---+ +---+ +---+ +---+ +---+ |
+∊-------------------------------------------+
    DISPLAY ':,' ⎕cn 'APL: IS, EASY'
+→--------------------+
| +→--+ +→--+ +→----+ |
| |APL| | IS| | EASY| |
| +---+ +---+ +-----+ |
+ϵ--------------------+
```

⎕copy Copy from Saved Workspace

Purpose:
Copies APL functions and variables from a saved workspace into the active workspace.

Syntax:

```
result ←              ⎕copy 'wsid'
result ← 'namelist'   ⎕copy 'wsid'
```

Arguments:
wsid is the workspace name; the system assumes the default extension (.ws64). You can use the file extension .ws64 to load an APL64 workspace. You must use the file extension .w3 to load an APL+Win workspace. When copying an APL+Win workspace file name that includes one or more spaces, you must append a semi-colon to the long file name.

namelist is a character matrix of names of functions or variables with one name per row, or a character vector with each name separated by one or more blanks.

Result:
If you use ⎕copy without specifying namelist, then result is empty if the system successfully copied all the functions and variables of wsid. If one or more functions in wsid are suspended or pendent in the current workspace, result is a numeric vector containing an appropriate response code for each one that the system did not copy.

If you specify namelist, result contains a response code for each name in namelist. The result is an integer vector that represents the success or failure. The possible response codes are:

Response Code	**Explanation**
127	The name is a duplicate.
2	The system successfully copied a variable.
1	The system successfully copied a function.
0	The system did not find an object with the supplied name.
¯2	The object was too large to copy into the available free workspace.
¯3	The name is defined as a label; it cannot be changed. (See Note.)
¯4	There is insufficient space in the symbol table to copy this object, and the workspace is too full to expand the symbol table.
¯6	The amount of workspace available is too small to perform the copy.
¯7	The object was not copied due to an INTERFACE CONVERSION ERROR.

Note: The system recognizes a label in the active workspace only when a function is pendent or suspended. The system copies a label as a variable only if the function is pendent or suspended in the source workspace and the name does not exist as a label in a pendent or suspended function in the active workspace.

If an unanticipated error occurs, the system does not return a result.

Effect:
Copies functions and variables from the local environment of the specified workspace (wsid) into the local environment of the active workspace replacing any objects by the same name. This means that if you invoke ⎕copy within a running function or in the session in the active workspace while a function is suspended, and you copy a variable that is localized within that function, only the local value is created or replaced. Similarly, if the specified workspace was saved while a function was suspended, and that function localized the variable you copy, the local value is copied into the active workspace.

See the description of ⎕pcopy for a way to prevent replacement of existing objects. See the system command)copy for a way of copying from the global environment to the global environment.

Copying a function copies only its source form; the system discards all precompiled internal code and clears ⎕stop and ⎕trace settings in the function.

Note: You cannot copy a workspace that was saved in APL+DOS.

Example:
The example below shows that two of the four requested objects were successfully copied, and that the value of MTRX changed.

```
      )VARS
MTRX
      MTRX
1 2
3 4
      )SI
SUSPENDED[3]*

      'MTRX XXX DATA SUSPLABEL' ⎕copy 'WS3'
2 0 2 ¯3
      )VARS
DATA MTRX
      MTRX
CAT
DOG
RAT
```

⎕cpc Cross-Platform Component

Purpose:
Supports actions on an APL64 Cross-platform Component (CPC) in the APL64 developer version

Syntax: result ←   ⎕cpc argument
              result ← action   ⎕cpc argument

Arguments:
action is the action to be performed on object;
argument is the optional argument to the action.

Results:
result is a two-element variable where the first element is an integer: 0 = success, 1 = failure. The second element is a character vector describing the action result.

Note: Refer to the menu Help | Developer Version GUI | Create Cross-Platform Component for additional information on this system function.

⎕cpcinfo Cross-Platform Component Information

Purpose:
This system function determines the calling environment of an APL64 public function.

Effects:
The first element of the ⎕cpcinfo system function may be used to determine if an APL64 public function is running in a cross-platform component.

Element	Element Value
1	Boolean: Running in CPC

In future versions of APL64, additional elements may be added to the result of the ⎕cpcinfo system function.

Note: Refer to the menu Help | Developer Version GUI | Cross-Platform Component | Create Cross-Platform Component for additional information on this system function.

⎕cpcis Cross-Platform Component IntelliSense Scaffold

Purpose:
This system function creates an IntelliSense scaffold.

For detailed documentation use the Help | Developer Version GUI | Cross-Platform Component | Implement Cross-platform Component Intellisense menu item in the APL64 developer version.

⎕cr Canonical Representation of a Function

Purpose:
Returns the canonical representation of a function.

Syntax: result ← [scope] ⎕cr 'fnname'

Argument:
fnname is a character singleton or vector that contains the name of a function.

scope is an optional numeric argument that specifies the scope at which names are interpreted, where 1 = local (default); and 0 = global.

Result:
The result is a character matrix that contains the canonical representation of the most local definition of the function. The system uses blanks to pad all the lines except the longest.

If fnname is not the name of an unlocked function, result is an empty matrix (shape 0 0).

You can assign the result of ⎕cr to a variable and use it as the argument to ⎕def or ⎕fx to redefine the original function.

Example:

```
     ∇ TRI N;A
[1]   ⎕←A←,1
[2]   L1:→(N<ρA)ρ0 ⋄ ⎕←A←(0,A)+A,0
[3]   →L1
     ∇

    ρQ←⎕cr 'TRI'
4 28
    Q
TRI N;A
⎕←A←,1
L1:→(N<ρA)ρ0 ⋄ ⎕←A←(0,A)+A,0
→L1

    ⎕fx Q
TRI
```

⎕crl Canonical Representation of a Single Function Line

Purpose: Returns a character vector containing the canonical representation of a single line of a function.

Syntax: result ← [scope] ⎕crl 'fnname[n]'

Arguments:
fnname is a character vector or singleton that contains the name of a valid function;
n is a non-negative integer that represents a line number in the function;
scope is an optional numeric argument that specifies the scope at which names are interpreted, where 1 = local (default); and 0 = global.

Result:
The result is the canonical representation of line n of function fnname with a length matching that of line n (generally shorter than the width of ⎕cr 'fnname'). If n is zero, the result is the header of the function.

If fnname is a locked function or if n is greater than the number of lines in the function, the result is an empty vector.

The result is also an empty vector if the argument is ill-formed, the function does not exist, or n is not given.

Examples:

```
     ∇ FOO
[1]   ⎕←'THIS IS A TEST'
[2]   A←ι12
[3]   ⎕←A×3
     ∇

     ⎕crl 'FOO'
     ρ⎕crl 'FOO'
0
     ⎕crl 'FOO[2]'
A←ι12
     DD←⎕crl 'FOO[1]'
     DD
⎕←'THIS IS A TEST'
     -DD
THIS IS A TEST
```

⎕crlf Newline + Line feed

Purpose:
This system function represents the combined terminal control codes Newline (⎕tcnl) and Line feed (⎕tclf).

Effects:
The ⎕crlf system function returns a two-element vector consisting of (⎕tcnl,⎕tclf) and is a convenient left argument to ⎕cv for producing standard Windows delimited output. In C/C++/C# this is the "\r\n" sequence.

⎕crlpc Public Comment

Purpose:
Retrieves the public comment from a single line of a function. A public comment begins with ⍝' and can occur after executable code on a given line. ⎕crlpc operates even on locked functions, which allows users to retrieve documentation embedded in the code when they cannot see the code itself.

Syntax: result ← [scope] ⎕crlpc 'fnname[n]'

Arguments:
fnname is a character vector or singleton that contains the name of a function.

n is a non-negative integer that represents a line number in the function

scope is an optional numeric argument that specifies the scope at which names are interpreted, where 1 = local (default); and 0 = global.

Result:
The result is the public comment for line n of function fnname. If line n has no public comment or if n is greater than the number of lines in the function, result is an empty vector. It is also an empty vector if the argument is ill-formed or the function does not exist.

Example:
You can use ⎕crlpc to identify different versions of the same locked function, if you document the version number in a public comment.

```
    ⎕crlpc 'LOCKEDFN[1]'
⍝' VERSION 4 REVISED 07/15/91 BY SAM
```

⎕cse C# Script Engine

Purpose:
Allows an APL64 application system programmer to use the C# programming language to access the Microsoft .Net Framework and obtain the benefit of the Framework tools. For detailed documentation and the quick start guide, use the Help | APL Language | C# Script Engine Manual and Help | APL Language | C# Script Engine – Quick Start menu items in the APL64 developer version.

⎕csr C# Representation

Purpose:
Returns the name(s) or header(s) of the public functions in the current workspace as a character vector.

For detailed documentation use the Help | Developer Version GUI | Cross-Platform Component | Implement Cross-platform Component Intellisense menu item in the APL64 developer version.

⎕cv Character Vector

Purpose:
Returns a user-specified delimited character vector result.

Syntax:
```
result ←             ⎕cv array
result ← [separator] ⎕cv array
```

Arguments:
array is any array of character arrays or scalars or strings.

separator is the optional left argument to specify the line separator sequence. The default ⎕tcnl line separator can be overridden by specifying a left argument that is a character scalar or non-empty character vector. For example, comma and semi-colon. The separator is inserted between each line of output, but not to the beginning or end of the result.

Result:
The result is a character vector.

Note: ⎕cv will remove trailing blanks from lines.

Example:

```
    ⎕cv 'ONE' 'TWO' 'THREE'
ONE
TWO
THREE
    ⎕tcht ⎕cv 'ONE' 'TWO' 'THREE'
```

```
ONE   TWO   THREE
     ':' ⎕cv 'ONE' 'TWO' 'THREE'
ONE:TWO:THREE
     ':' ⎕tcht ⎕cv 'ONE' 'TWO' 'THREE'
ONE:   TWO:   THREE
```

⎕cwxx Debugger stepping-by-primitive functions

Purpose:
Provides a method to use the current values associated with a suspended function for debugging purposes. When you are suspended in the Debugger Pane, and you choose to step through the suspension by primitive, the system displays a subset of six values associated with the primitive at the point of suspension. The values displayed are also available in the subset of six corresponding system functions, which you can manipulate in immediate execution mode or use in another function for analyzing and debugging your application. Note, however, that these functions are defined only when a value exists. If you attempt to reference one of these functions that is not currently relevant, the system generates a VALUE ERROR. The six functions are:

- ⎕cwaxis holds the integer value of the specified axis along which the function is operating
- ⎕cwlarg holds the value of the left argument
- ⎕cwrarg holds the value of the right argument
- ⎕cwres holds the value of the result; this exists when stepping OVER by primitive
- ⎕cwsub holds the scalar or vector that represents a specified or implicit indexing subscript
- ⎕cwval holds the value that is to be assigned or has just been assigned to a variable.

When stepping by primitive, the debugger can stop on a primitive function, system function, user-defined function, or a function derived by an operator and its operands. In this context, primitive includes assignment and branch arrows and indexing into an array. The debugger also stops on full lines, at which time none of these functions has a value.

⎕d Digits glyphs

Purpose:
Returns the digits glyphs.

Syntax: result ← ⎕d

Domain:
result is a character vector.

Example:

```
    ⎕d
0123456789
```

⎕deb Delete Extra Blanks

Purpose:
Removes all extra blanks (leading, trailing, and multiple) from the character array or string. If the argument is a matrix, removes extra columns if entirely blanks.

Syntax: result ← [duplicate] ⎕deb text

Argument:
duplicate is the optional left argument to specify the duplicate character to be removed.
text is any array of character arrays or a string.

Result:
The result is any array of characters arrays or a string.

Remarks:
⎕deb is the replacement for the assembler function DEB in the ASMFNS.w3 workspace in APL+Win.

Example:

```
      'a b c'≡⎕deb ' a   b  c  '
      ('B' ⎕DEB 2 4ρ'ABBDEBBH')≡2 3ρ'ABDEBH'
      ('B' ⎕DEB 2 7ρ'BABBDBBBEBBHBB')≡2 3ρ'ABDEBH'
      'a b c'≡>⎕deb «  a   b  c  »
      «axb»≡«x»⎕DEB «xaxxbxx»
      'The car cost $45,450'≡⎕DEB '  The  car  cost  $45,450  '

1
1
1
1
1
1
```

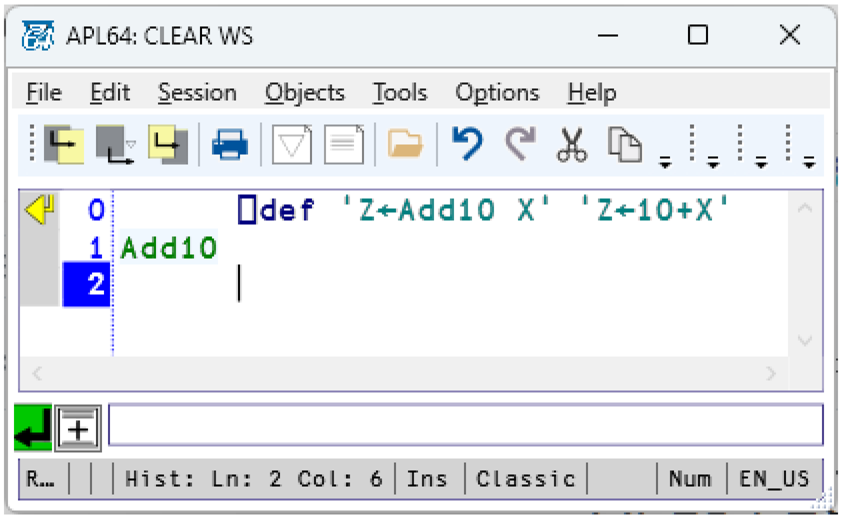
⎕def Function Definition

Purpose:
Defines a function from a character representation.

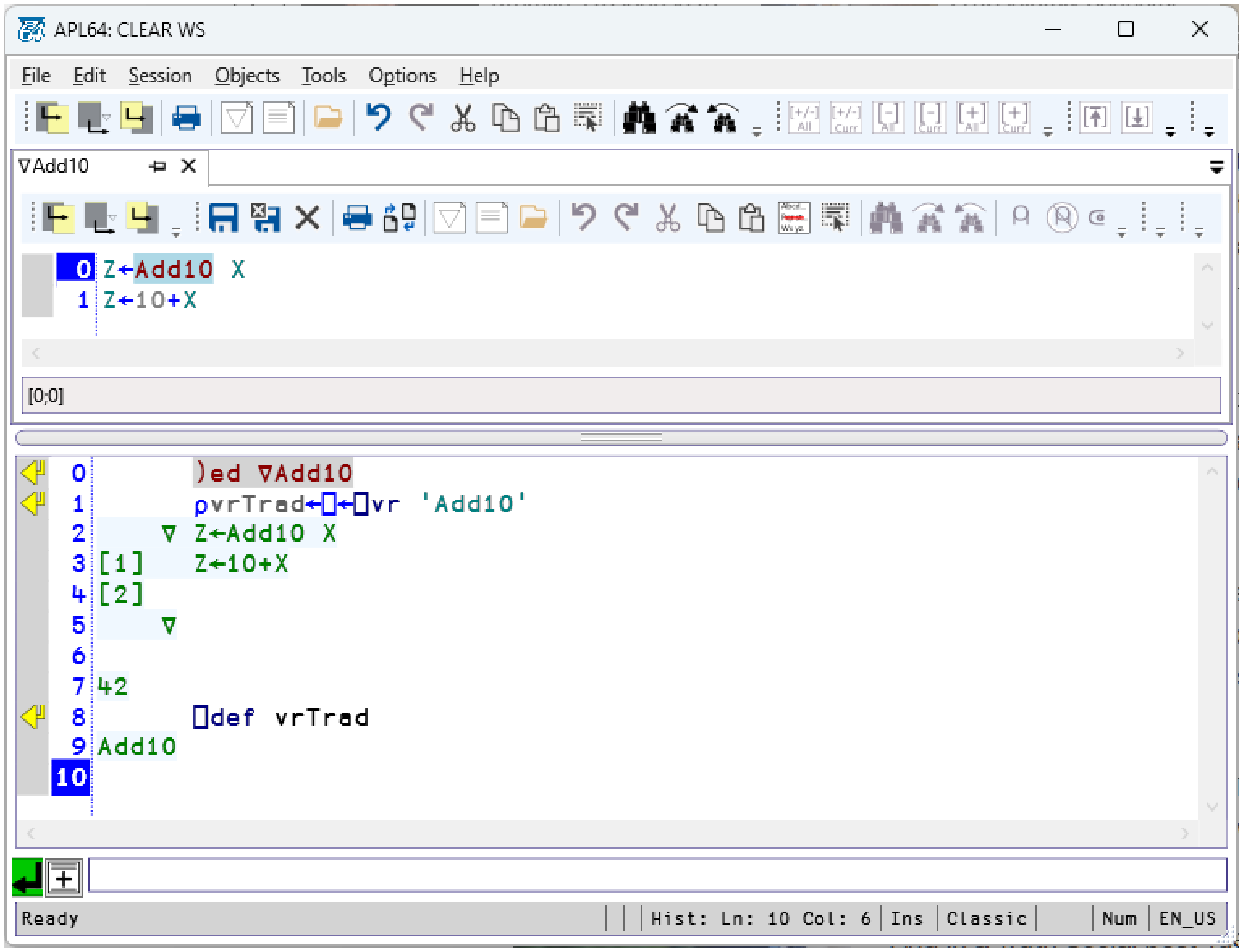
Syntax: result ←   ⎕def 'fnrep'
              result ← timestamp   ⎕def 'fnrep'

Argument:
fnrep is a character representation of an APL64 programmer-developed function. Several formats of fnrep are supported:

- fnrep is a nested vector of character vectors or scalars:

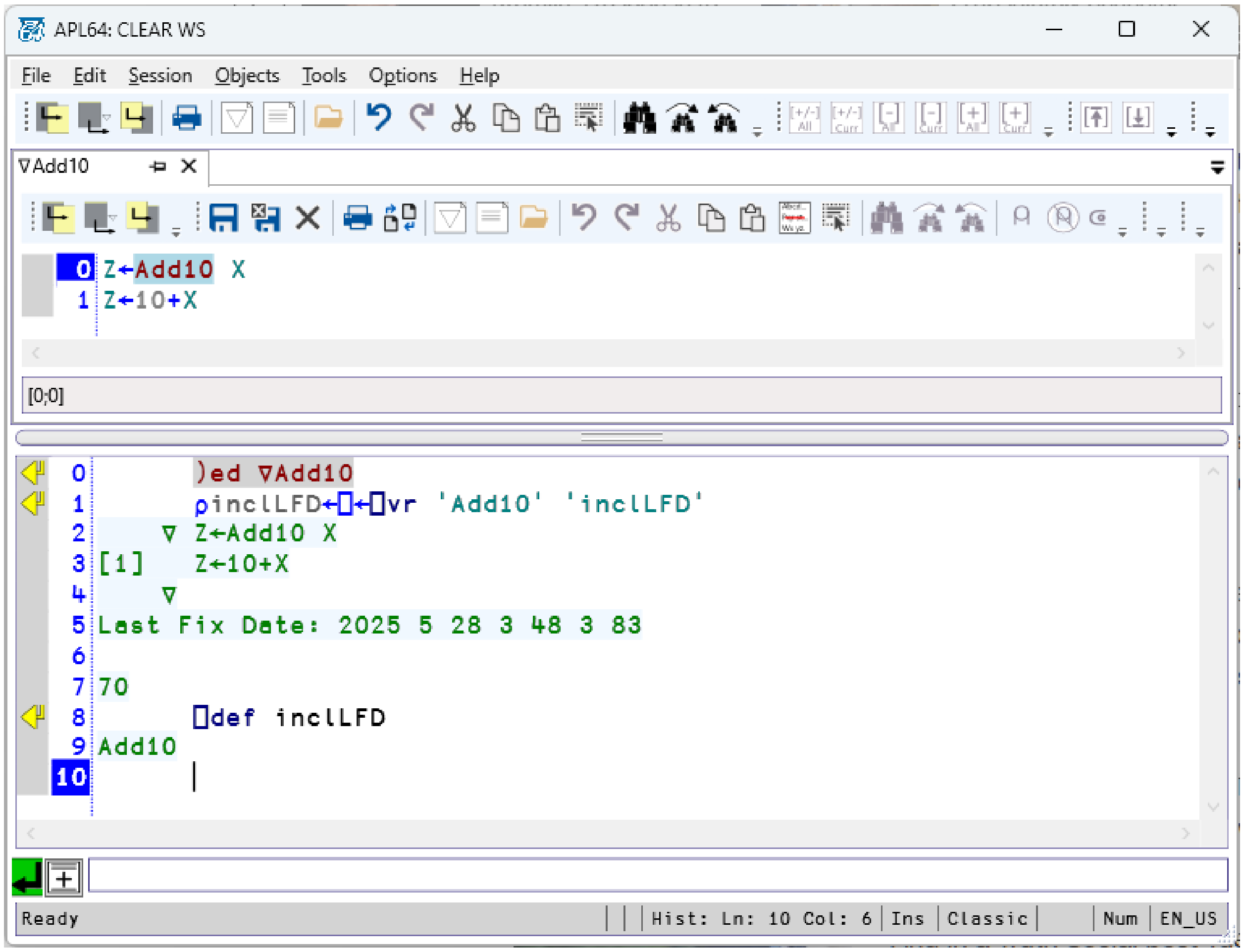

```
      ⎕def 'Z←Add10 X' 'Z←10+X'
Add10
```

- fnrep is a character vector in ⎕vr result format:
    - ⎕vr traditional format: If fnrep contains a leading or trailing ⍫, the function will be locked after it is defined.


```
∇Add10
[0] Z←Add10 X
[1] Z←10+X
```

```
      )ed ∇Add10
      ⍴vrTrad←⎕←⎕vr 'Add10'
    ∇ Z←Add10 X
[1]   Z←10+X
[2]
    ∇

42
      ⎕def vrTrad
Add10
```

- ⎕vr InclLFD format:



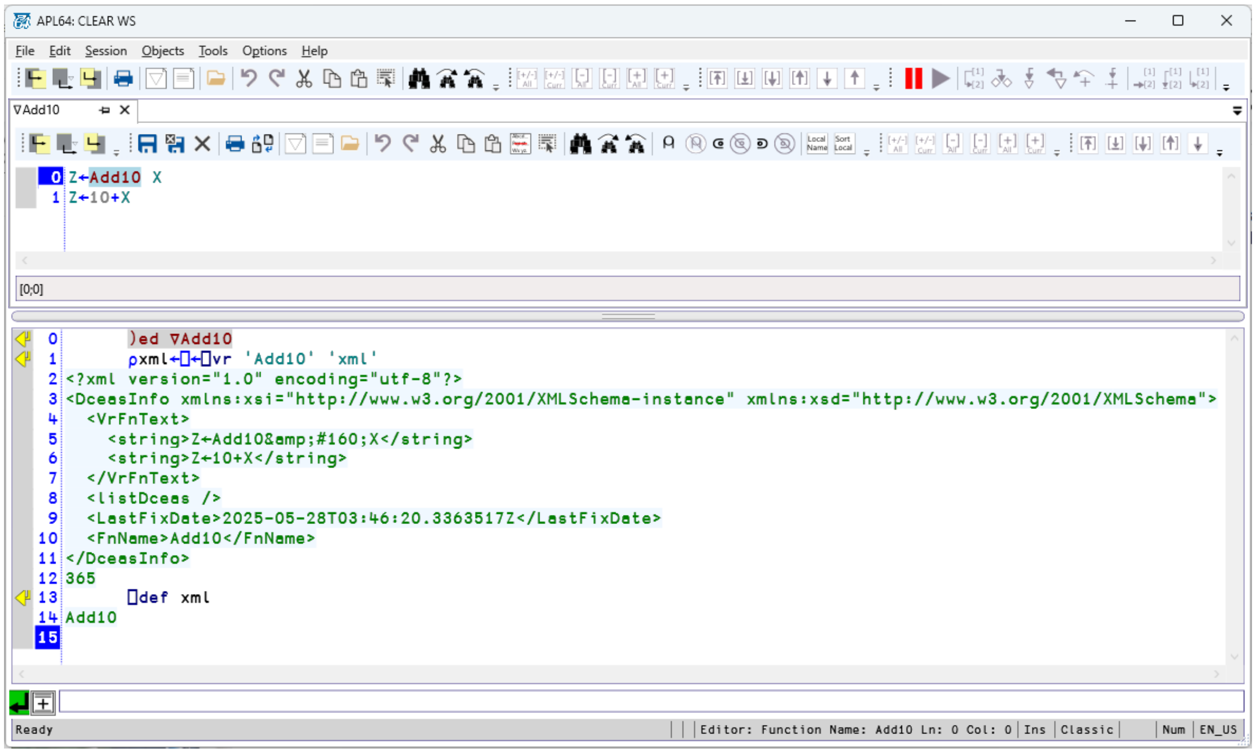
```
Z←Add10 X
Z←10+X
```

[0;0]

```
      )ed ∇Add10
      pinclLFD←⎕←⎕vr 'Add10' 'inclLFD'
    ∇ Z←Add10 X
[1]   Z←10+X
    ∇
Last Fix Date: 2025 5 28 3 48 3 83

70
      ⎕def inclLFD
Add10
```

Ready | Hist: Ln: 10 Col: 6 | Ins | Classic | Num | EN_US

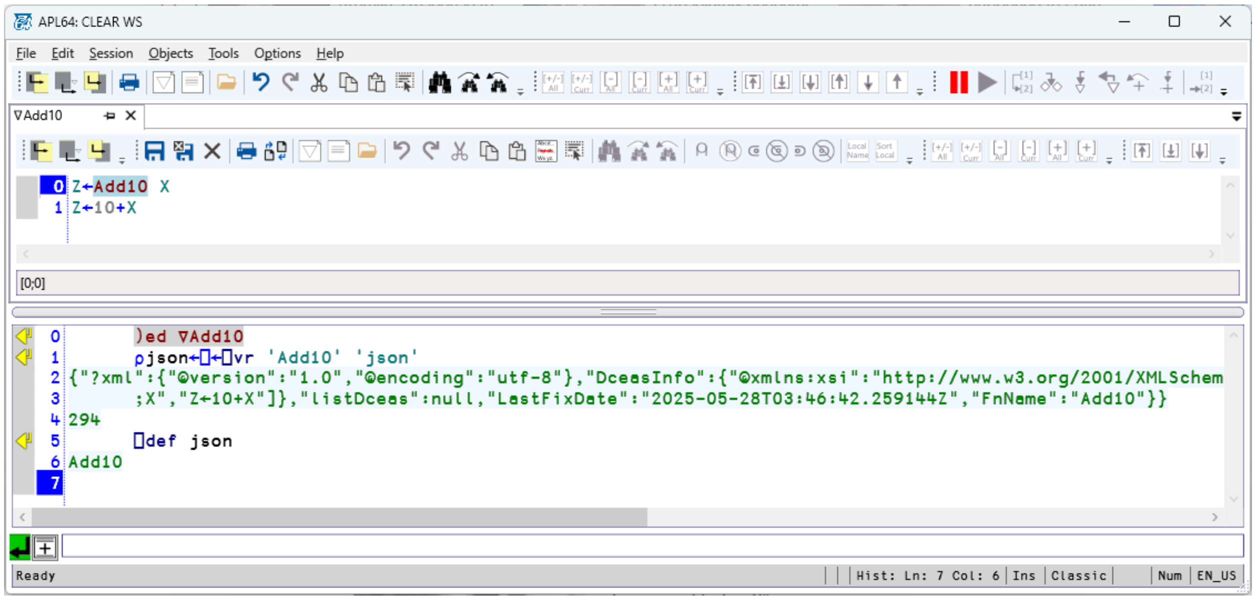
- ⎕vr XML format:



```
∇Add10
Z←Add10 X
Z←10+X
```

```
      )ed ∇Add10
      ⍴xml←⎕←⎕vr 'Add10' 'xml'
<?xml version="1.0" encoding="utf-8"?>
<DceasInfo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <VrFnText>
    <string>Z←Add10&amp;#160;X</string>
    <string>Z←10+X</string>
  </VrFnText>
  <listDceas />
  <LastFixDate>2025-05-28T03:46:20.3363517Z</LastFixDate>
  <FnName>Add10</FnName>
</DceasInfo>
365
      ⎕def xml
Add10
```

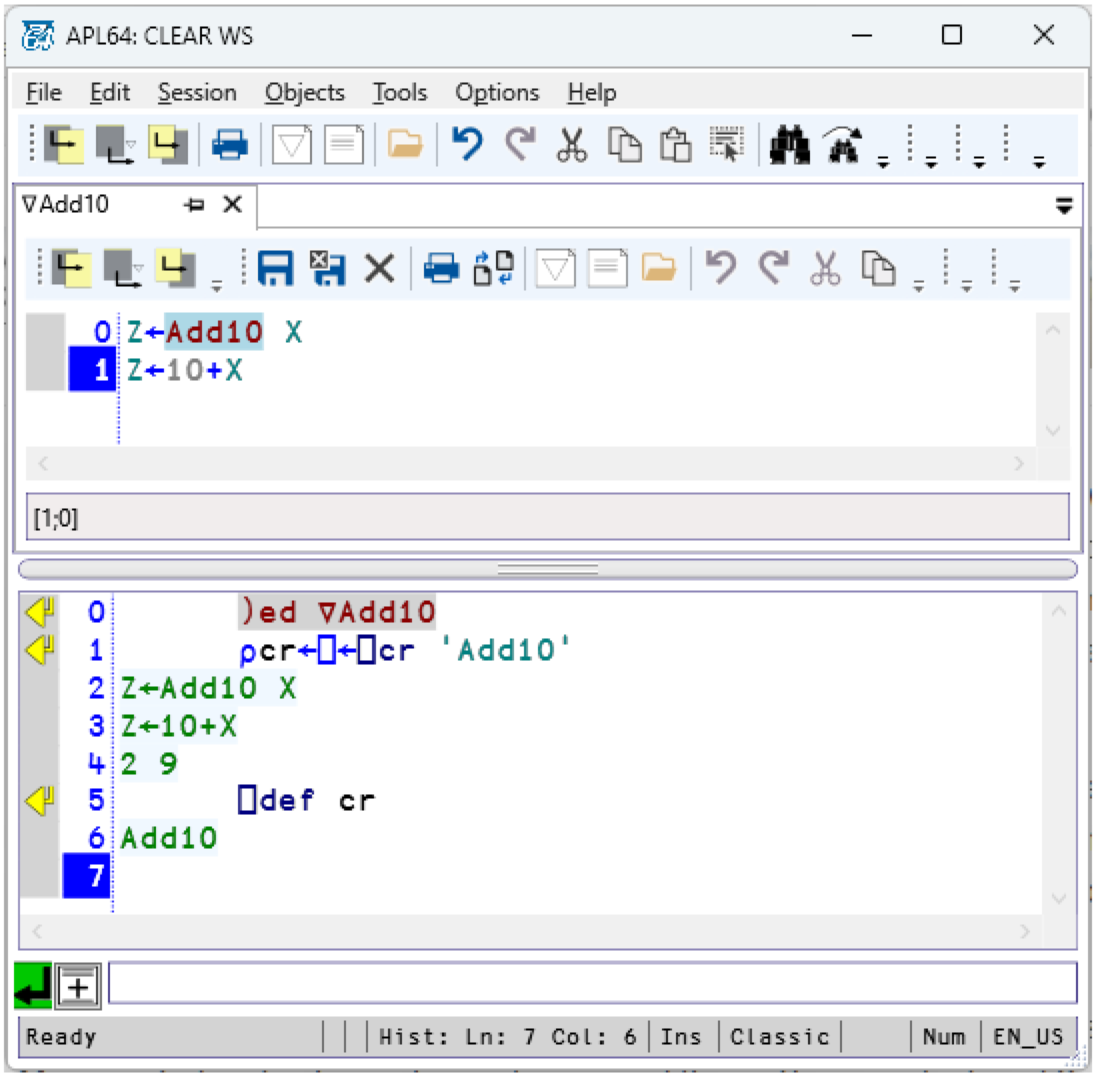
- ○ ⎕vr JSON format:



```
∇Add10
Z←Add10 X
Z←10+X
```

```
      )ed ∇Add10
      ⍴json←⎕←⎕vr 'Add10' 'json'
{"?xml":{"@version":"1.0","@encoding":"utf-8"},"DceasInfo":{"@xmlns:xsi":"http://www.w3.org/2001/XMLSchem
      ;X","Z←10+X"]},"listDceas":null,"LastFixDate":"2025-05-28T03:46:42.259144Z","FnName":"Add10"}}
294
      ⎕def json
Add10
```

- fnrep is a character matrix in ⎕cr result format:


APL64: CLEAR WS
File Edit Session Objects Tools Options Help
∇Add10
[1;0]
Ready | Hist: Ln: 7 Col: 6 | Ins | Classic | Num | EN_US

```
Z←Add10 X
Z←10+X
```

```
      )ed ∇Add10
      pcr←⎕←⎕cr 'Add10'
Z←Add10 X
Z←10+X
2 9
      ⎕def cr
Add10
```

timestamp is an arbitrary six- or seven-element vector in the range 1980 1 1 0 0 0 through 2048 1 19 3 14 7 (January 1, 1980 through January 19, 2048); if you specify the seventh element (milliseconds), it is ignored. Arguments outside this range produce a domain error, as do invalid dates and times. If you do not specify a left argument, the system uses the current date and time.

result:
If the function definition is successful, result is the name of the defined function. If not successful, result is a two-element numeric vector that contains information about the error (see Errors below).

Effect:
Defines a function of the appropriate name in the active workspace, unless an error condition occurs. If the name of the function defined corresponds to a local identifier in a currently executing, pendent, or suspended function, the newly defined function is local to that function and the system erases it when the function in which it is localized completes execution.

If the name of the function defined corresponds to the name of an existing function, the system replaces the existing function, discards precompiled internal code for the function, and removes ⎕stop or ⎕trace settings in the function. If the replacement text of the function is in the Xml or Json format and contains editing history, that editing history be included in the updated function definition.

There is always a timestamp associated with a function. If you do not specify a left argument, the system uses ⎕ts when it fixes the function. You can see the timestamp for a function by using the system function ⎕at with 2 as its left argument and the function name included in the right argument.

Comparison of ⎕def and ⎕fx:
⎕def and ⎕fx provide similar capabilities. ⎕def provides more options:

- ⎕def accepts both canonical (matrix) and visual (vector) representations of a function; ⎕fx accepts only the canonical representation.
- ⎕def can create a locked function; ⎕fx cannot.
- ⎕def indicates both the cause and the location of an error; ⎕fx indicates only the location.
- ⎕def indicates a WS FULL condition with an error code without halting execution. ⎕fx stops and signals an error.

Errors:
If the system recognizes an error condition during analysis of a character vector or matrix argument, it does not define the function but does not report an explicit error. Instead, the result of ⎕def is a two-element integer vector that contains information about the error.

First Element of ⎕def Error Result	
Code	**Error Description**
1	WS FULL: Function definition requires more workspace storage than is available
2	DEFN ERROR: Function is in use or header is ill-formed or the function name is in use as a variable or label
3-9	Reserved error codes

The second element of ⎕def error result indicates the ⎕io-dependent row of the function representation where the error begins.

⎕defl Single Function Line Editing

Purpose:
Edits a single line of the most local definition of an unlocked function.

Syntax:
```
result ← ⎕defl 'fnname[n]line'
result ← ⎕defl 'fnname[~n]'
result ← ⎕defl 'fnname[Δn]'
```

Arguments:
The argument must be a character scalar or vector.
fnname is the name of a function.

[n] is a line number.

line is the text of the line you want to insert or replace

[~n] or [Δn] is a line number or numbers you want to delete.

To insert a new line into the function named fnname, specify n as a decimal fraction between two existing lines, such as [3.5]. In this case, ⎕defl inserts line between lines 3 and 4. If n is greater than the number of lines in the function, ⎕defl inserts line at the end of the function.

To replace an existing line in the function named fnname, specify the line number n in brackets followed by the replacement text (line).

To delete a line from the function, specify a tilde (~) before n and omit line. You can delete multiple lines by specifying n as a vector, as in [~3 4 5].

Result:
If the operation is successful, result is a character vector that contains the name of the function. If the name of the function changes as a result of replacing line 0 of the function, the result is the name of the new function. If the operation is not successful, result is a numeric scalar that contains information about the error (see Errors below).

Effect:
Replaces, inserts or deletes the lines as requested by the syntax. The system automatically renumbers all lines following the point of insertion or deletion. The form of the argument to ⎕defl is the same for insertion and replacement. The effect depends upon the value of n relative to the line numbers of the function.

Errors:
If the system recognizes an error condition during analysis of a character vector or matrix argument, it does not define the function but does not report an explicit error. Instead, the result is an integer scalar that indicates the error type. The values are the same as the first element of the error type for ⎕def. If one of these errors occurs, the system does not change the function.

Example:

```
      ⎕VR 'TRI'
    ∇ TRI N;A
[1]   ⎕←A←,1
[2]   L1:→(N<ρA)ρ0 ⋄ ⎕←A←(0,A)+A,0 ⋄ →L1
    ∇

      ⎕defl 'TRI[1] A←,1'
TRI

      ⎕vr 'TRI'
```

```
   ∇ TRI N;A
[1]   A←,1
[2]   L1:→(N<ρA)ρ0 ⋄ ⎕←A←(0,A)+A,0 ⋄ →L1
   ∇
```

⎕div Quotient

Purpose:
Returns the quotient.

Syntax: result ← A ⎕div B

Arguments:
A and B are a numeric array or scalar.

Result:
The result is the quotient of A divided by B.

Remarks:
⎕div is the replacement for the assembler function DIV in the ASMFNS.w3 workspace in APL+Win.

Caution:
If an element of B is zero the corresponding result value is zero.

Example:

```
      0≡1.234 ⎕DIV 0
      1≡1.234 ⎕DIV 1.234
      X←¯1.234+ι100⋄(100ρ0)≡X ⎕DIV 0
      X←¯1.234+ι100⋄(100ρ1)≡X ⎕DIV X
      X←Y←¯1.234+ι100⋄Y[1 7 9]←0⋄(Y⎕DIV Y)≡X⎕DIV Y
      X←2 3ρι6⋄(2 3ρ0)≡X⎕DIV 2 3ρ0
      X←2 3ρι6⋄(2 3ρ1)≡X⎕DIV X
      X←2 3 4ρι×/2 3 4⋄(2 3 4ρ0)≡X⎕DIV 2 3 4ρ0
1
1
1
1
1
1
1
1
```

⎕dl Delay Execution

Purpose:
Delays execution.

Syntax: result ← ⎕dl seconds

Argument:
seconds is the length, in seconds, of the delay you wish to cause; a numeric singleton (may be fractional).

Result:
result is the actual delay in seconds; it may vary each time you use ⎕dl.

Effect:
Using the system clock, ⎕dl delays execution for the time requested. You can abort the delay with a weak interrupt, in which case result may be substantially less than seconds. If you want to see the value of result after an interrupt, you must assign it; it does not display in the session.

Example:

```
    ⎕dl 5
5.05
```

⎕dlb Delete Leading Blanks

Purpose:
Removes leading blanks from the character array or string.

Syntax: result ← [duplicate] ⎕dlb text

Argument:
duplicate is the optional left argument to specify the duplicate character to be removed.
text is any array of character arrays or a string.

Result:
The result is any array of characters arrays or a string.

Remarks:
⎕dlb may be used as replacement for the function DLB in the ASMFNS.w3 workspace in APL+Win.

Example:

```
    ⎕dlb ' The car cost $45,450 '
The car cost $45,450
   'The car cost $45,450 '≡⎕dlb ' The car cost $45,450 '
1
```

⎕dltb Delete Leading and Trailing Blanks

Purpose:
Removes leading and trailing blanks from the character array or string.

Syntax: result ← [duplicate] ⎕dltb text

Argument:
duplicate is the optional left argument to specify the duplicate character to be removed.
text is any array of character arrays or a string.

Result:
The result is any array of characters arrays or a string.

Remarks:
⎕dltb may be used as replacement for the function DLTB in the ASMFNS.w3 workspace in APL+Win.

Example:

```
    ⎕dltb ' The car cost $45,450 '
The car cost $45,450
    'The car cost $45,450'≡⎕dltb ' The car cost $45,450 '
1
```

⎕dr Data Representation

Purpose:
Reports the datatype of an array, interprets the values of an array as a specified datatype, converts an array from one datatype to another, reduces an APL array to a coded character vector that can be stored outside APL, or restores such a coded vector to the original APL value.

Syntax:

```
result ←                           ⎕dr array
result ←    datatypespec           ⎕dr array
codedvector ← 'wrapl'              ⎕dr anyvalue
anyvalue ← 'unwrapl'               ⎕dr codedvector
result ← 'deflate parameters'      ⎕dr array
result ← 'inflate parameters'      ⎕dr  array
result ← 'xml'                     ⎕dr  array
result ← 'unxml'                   ⎕dr  array
result ← 'json'                    ⎕dr  array
result ← 'unjson'                  ⎕dr  array
```

Arguments:
array is any APL array; with a numeric left argument, the system treats array as a sequence of values of the type indicated by the first element of the left argument.

datatypespec is a vector of one, two, or three elements that indicates how the system should treat or convert array. The first element specifies the datatype for interpreting the right argument. The second element is the datatype to which the system should convert the resulting values. Use a three-element left argument to convert to values that are not valid APL datatypes in APL64.

parameters is a keyword parameter to the inflate and deflate functions, which are optional, that specify the way the array is inflated and deflated.

xml and unxml are keyword parameters to convert to and from XML serialized data. The right argument you use with unxml as the left argument should be a vector created using xml as the left argument.

json and unjson are keyword parameters to convert to and from JSON serialized data. The right argument you use with unjson as the left argument should be a vector created using json as the left argument.

Character left arguments to ⎕dr may be either uppercase or lowercase but not mixed case. The right argument anyvalue you use with wrapl as the left argument can be any APL array. It can contain any values and can be a simple or nested, homogeneous or heterogeneous array.

The right argument you use with unwrapl as the left argument should be a vector created using wrapl as the left argument. If you use a vector not produced by using wrapl as the left argument, there may be unintended consequences.

Result:

Monadic ⎕dr

For the monadic form of ⎕dr, the explicit result is a scalar integer representing the datatype of the right argument. The possible datatype codes include:

Value	Description
11	1-bit Boolean
82	8-bit character
162	16-bit wide-character (See Note.)
164	.Net string
322	32-bit Unicode character array
323	32-bit integer
326	32-bit pointer
645	64-bit floating-point data, formatted and normalized according to the IEEE standard.
807	80-bit heterogeneous

Note: Most UTF-16 Unicode codepoints are represented by a single 16-bit character but others are represented by a pair of surrogate characters. See the following for details about surrogate characters: https://en.wikipedia.org/wiki/UTF-16.

Examples:

```
      X←(1.2) ('WORD') (⍳10) (⎕ucs 1234) («STRING»)
      X
 1.2 WORD  1 2 3 4 5 6 7 8 9 10  Ä STRING
      ]DISPLAY X
.→-------------------------------------------.
|                                            |
|     .→---..→-------------------.           |
| 1.2 |WORD||1 2 3 4 5 6 7 8 9 10| Ä  STRING |
|     '----''~-------------------' -         |
'ϵ-------------------------------------------'
      ⎕DR ¨X
645 82 323 322 164
```

Dyadic ⎕dr with a one-element left argument

With a numeric left argument, ⎕dr treats array as a sequence of values of the type indicated by the first element of the left argument.

For the dyadic form of ⎕dr with a singleton left argument, the system forms the explicit result by interpreting the bit pattern of the right argument according to the code of the left argument. The result contains the values that those bits represent; they may be different values than those that generated the right argument, but the bit pattern is the same. For example:

```
      323 ⎕dr 'abcd'
1684234849
      82 ⎕dr 1684234849
abcd
      11 ⎕dr 'b'
0 1 1 0 0 0 1 0
```

If there are not enough data to fill an integral number of elements in the specified datatype, a LENGTH ERROR occurs.

You can use dyadic ⎕dr with a singleton left argument, for example, to transmit floating-point numbers to an identical personal computer without loss of precision. Represent the floating-point numbers as character data, transmit the character data, and reinterpret them as floating-point numbers when transmission is complete. This technique avoids the loss of precision encountered in source-level transfer. After ⎕dr converts floating-point numbers to character data, you can store the character representation of the data on any computer, regardless of how that system represents floating-point numbers internally. However, this representation may not equate to the same data, or even be meaningful, if re-translated to numeric data on a different type of computer.

Examples:

```
      CHARS←'A B C D E F G H I J K L '
      ⍴CHARS
24
      FLOAT←645 ⎕dr CHARS
      ⍴FLOAT
3
      FLOAT
3.00213493E¯153 3.598811293E¯153 4.195487656E¯153
      82 ⎕dr FLOAT
A B C D E F G H I J K L
      ⎕dr 1 0 1
11
      ⎕dr ⊂⍳0
326
      ⎕io←0 ⋄ ⎕av ⍳ 82 ⎕dr 7
7 0 0 0
```

There are seven additional left arguments to ⎕dr, 'MD5', ‘SHA1’, ‘SHA256’, ‘SHA384’, ‘SHA512’,'I64' and 'F64'. This functionality was added because 64-bit integers are used in the colossal component file system, but it has broader uses. The argument 'MD5' converts a simple vector or scalar of any data type to an MD5 digest represented as a 4-element integer vector.

result ← 'MD5' ⎕dr array

where array is a simple array or scalar.

result is a 4-element integer vector for the right argument.

Use caution when running an MD5 checksum against a bit vector. Bit vectors are stored 8 values to the byte and MD5 works on byte data. Using MD5 against a bit vector that is not a multiple of 8 bits in length will give non-reproducible results on independently generated data.

You can convert result to an MD5 checksum 32-bit character vector with the function below:

```
    ∇ z ← ShowMD5 w;x;⎕io
[1]   ⎕io←0
[2]   x←⌽[0]256 256 256 256⊤,w
[3]   z←,(,[ 1 2]⍉'0123456789abcdef'[16 16 ⊤ x])
    ∇
```

The argument 'SHA1' converts a simple vector or scalar of any data type to an SHA1 digest represented as a 5-element integer vector.

result ← 'SHA1' ⎕dr array

where array is a simple array or scalar.

result is a 5-element integer vector for the right argument.

Use caution when running an SHA1 checksum against a bit vector.

The argument 'SHA256' converts a simple vector or scalar of any data type to an SHA256 digest represented as an 8-element integer vector.

result ← 'SHA256' ⎕dr array

where array is a simple array or scalar.

result is an 8-element integer vector for the right argument.

Use caution when running an SHA256 checksum against a bit vector.

The argument 'SHA384' converts a simple vector or scalar of any data type to an SHA384 digest represented as a 12-element integer vector.

result ← 'SHA384' ⎕dr array

where array is a simple array or scalar.

result is a 12-element integer vector for the right argument.

Use caution when running an SHA384 checksum against a bit vector.

The argument 'SHA512' converts a simple vector or scalar of any data type to an SHA512 digest represented as a 16-element integer vector.

result ← 'SHA512' ⎕dr array

where array is a simple array or scalar.

result is a 16-element integer vector for the right argument.

Use caution when running an SHA512 checksum against a bit vector.

The arguments 'I64' and 'F64' let you convert a floating-point number with an integral value to a 64-bit integer, which is represented as a pair of 32-bit integers, and a pair of 32-bit integers to a floating-point value.

result ← 'I64' ⎕dr float

where float is a scalar or vector of integer-valued numbers in floating point representation. The number must be within the integer tolerance (system fuzz); any fractional part of the number must be less than 1÷16 regardless of the magnitude of the number; and the value must be no larger than 2*52.

result is a pair of 32-bit integer values that taken together represent the bit pattern of a 64-bit integer number of the correct value in the little-endian representation of the host machine.

```
      big←123456789E7
      big
1.23456789E15
      'I64' ⎕dr big
1015601280 287445

      cube←.0000000001+64*3
      ⎕dr cube
645
      'I64' ⎕dr cube
262144 0

      result ← 'F64' ⎕dr bigInts
```

where bigInts is a vector of pairs of 32-bit integer values that taken together represent the bit pattern of a 64-bit integer number. The shape of the right argument must be an even number. The value of the 64-bit number must be less than 2*52.

result is a vector, half the length of the right argument, of floating-point numbers.

```
      'F64' ⎕dr 1015601280 287445
1.23456789E15
      ⍴'F64' ⎕dr 1015601280 287445
1

2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 ⊤ 287445
0 1 0 0 0 1 1 0 0 0 1 0 1 1 0 1 0 1 0 1
+/(2*32 34 36 38 39 41 45 46 50)
1.234566874E15
```

```
1.234566874E15+1015601280
1.23456789E15
```

Additionally, there are several internal changes that take care of particular problems users encountered, or that improve performance or usability.

Dyadic ⎕dr with a two-element left argument

With a two-element left argument, ⎕dr changes the bit pattern but attempts to preserve the values of the right argument. It interprets the bit pattern of the right argument according to the first element of the left argument and creates a new bit pattern that will represent those same values (if possible) according to the second element of the left argument.

The result of ⎕dr in this case is a two-element nested vector, where 1⊃result contains the converted values, and 2⊃result is a Boolean array of the same shape, whose elements are 1 if the conversion was valid. Conversion is valid if the system can convert the value to the given type with no loss of significance. If the system cannot convert the value, a 0 appears in 2⊃result and the system stores the fill item (0 or blank) for the value at the corresponding position in 1⊃result. You can convert a floating-point value to integer if it is within the APL integer tolerance of an integer value.

This statement tests an integer array to see if it can be converted to Boolean data (that is, if its values are all 0 or 1).

```
      ⊃1↓323 11 ⎕dr 3 0 1
0 1 1
```

⎕dr can create Boolean, character, 32-bit integer, and floating-point datatypes (codes 11, 82, 323, and 645); these codes are valid as the second element of a two-element left argument. You can also use them as the first element.

⎕dr cannot create the pointer or heterogeneous datatypes (codes 326 and 807); you cannot use these codes as the second element of a two-element left argument, but you can use them as the first element.

Examples:
The following strained example shows some manipulation of data to demonstrate the differences between bit pattern and value. Although the example does not even resemble something productive, the steps may be enlightening.

```
      i←323 ⎕dr 'abcd'
      i
1684234849
      j←323 645 ⎕dr i
      j
 1684234849  1
      k←⊃j[1]
      k
1684234849
      82 ⎕dr i
abcd
      82 ⎕dr k
@⍠Ø↑ÙA
```

```
      ρ 82 ⎕dr k
8

      ⎕io←0
      l←82 ⎕dr i
      m←82 ⎕dr k
      ⎕av ι l
97 98 99 100
      ⎕av ι m
0 0 64 152 216 24 217 65

      82 ⎕dr ↑(645 323 ⎕dr k)
abcd
```

Using the 16-bit integer datatype code

You can also interpret data as 16-bit integers; the code for this datatype is 163. APL64 does not have a 16-bit integer datatype, so you cannot use code 163 as the second element of a two-element left argument, nor will it be returned as the result of monadic ⎕dr. You can use it as the first element of a two-element left argument; for example:

```
    ⎕io←0
    CHARS←⎕av[0 29 2 0]
    (INTS VALID) ← 163 323 ⎕dr CHARS
    INTS
7424 2
    VALID
1 1
```

INTS is equivalent to 256 256 ⊥ ⎕av ι⌽2 2 ρCHARS

Dyadic ⎕dr with a three-element left argument

A three-element left argument always yields a character result, so the third element must be 82. This form generates data that are not valid APL datatypes, such as 16-bit integers. At present, the only possible value for the second element is 163. The statement below shows how to convert numeric data from this system into a form that, when written to file, can be read as 16-bit integers by APL+PC.

```
    (1⊃11 163 82 ⎕dr boolean) ⎕nappend ¯1
```

The system interprets the right argument as Boolean data and converts each bit into a 16-bit integer. The system returns the final result as character data with each pair of characters corresponding to one of the 16-bit integers.

Note: Not all eight-byte groups of data correspond to valid normalized IEEE floating-point numbers. ⎕dr with a two- or three-element left argument forces denormalized numbers to exact 0.0. ⎕dr treats infinites and NANs as failed conversions; they produce a LIMIT ERROR.

Encoding and restoring an array with ⎕dr

If you want to store or transmit an array of values, you can use ⎕dr to encode the entire array as a single vector of unreadable, binary data by using the character string 'wrapl' as the left argument. The resulting vector will generally contain non-displayable characters and will not display meaningfully. You should treat it as binary data and not as text data, if, for example, you want to transmit it.

When you retrieve the coded vector, you can use ⎕dr to restore the exact values of the array by using 'unwrapl' as the left argument. You cannot meaningfully do anything else with the coded vector.

Deflating and inflating an array with ⎕dr

⎕dr can be used to compress and expand data in APL arrays. This facility is generally referred to as deflation and inflation. For most data the deflated version will occupy less space than the original. Data may be saved in deflated format in files, or it may be transmitted over a network. It may later be inflated, which will reconstruct the original data. This feature provides both a means of conserving space, either in the workspace, or on file, and a way of reducing data transmission time over a network. Deflate and inflate provide lossless compression.

There are two ways of specifying the deflate and inflate functions: a character vector left argument to ⎕dr; or a nested vector left argument to ⎕dr.

The character vector form looks like:

```
y←'deflate compression_level=8 gzip_header' ⎕dr x

z←'inflate gzip_header' ⎕dr y
```

The nested vector form looks like:

```
y←'deflate' ('compression_level' 8) 'gzip_header' ⎕dr x

z←'inflate' 'gzip_header' ⎕dr y
```

There are two types of keyword parameters for deflate and inflate. One type is a name only (such as gzip_header). The other type is a name and a value (such as compression_level=8 or ('compression_level' 8)). The order of keyword names following deflate or inflate is irrelevant.

All names used in the left argument for deflate and inflate are case insensitive. There are two forms for the deflate and inflate keywords: a full name and an abbreviated name. Either name may be used. In the following descriptions, the abbreviated names are shown below the full names. Names that represent default states have no abbreviation, since they can just be left out.

The parameters for deflation are:

compression_level = n
complevel = n

compression_level specifies a time versus size tradeoff in the compression process. The allowable values are from 0 to 9. A value of zero indicates no compression. A value of 1 indicates maximum speed. A value of 9 indicates maximum compression. The default compression_level value is 6.

no_compression
nocomp

best_speed
bestspeed

best_compression
bestcomp

default_compression

These keywords are shortcuts for compression_level = n. no_compression means the same as compression_level = 0. best_speed means the same as compression_level = 1. best_compression means the same as compression_level = 9. default_compression means the same as compression_level = 6.

    memory_level = n
    memlevel = n

As compression may be done in multiple passes, the internal compression state is maintained in memory. The amount of memory consumed for this purpose is controlled by the memory_level parameter, which has a range of 1 to 9. memory_level =1 uses minimum memory but is slow and reduces compression ratio. memory_level =9 uses maximum memory for optimal speed. The default value is 8.

    window_size = n
    winsize = n

The deflate algorithm uses both Huffman encoding and the LZ77 algorithm to compress data. For an explanation of the deflate algorithm and references to both Huffman encoding and LZ77 see RFC 1951. The LZ77 portion of the deflate algorithm uses a moving window of previously seen data to search for matching sequences of data in the input stream being processed. The size of this window affects the efficiency of the final compression. This window may vary in size from 256 to 32768 bytes. The window_size parameter is specified as a number between 8 and 15 representing the base 2 exponent of the actual size of the window. The default value is 15.

    block_size = n
    blocksize = n

Deflation processes data in a loop, handing a fixed amount of data to the deflate code on each trip through the loop. In default operation, an entire APL vector will be processed in one trip. If desired, the block_size parameter may be used to specify the amount of data to be processed in multiple trips through the loop.

    wrapl = n

The wrapl parameter specifies whether the data is automatically converted to a wrapped array during the deflation process (see the 'wrapl' argument to ⎕dr). This parameter may be specified as wrapl = 1 (wrap the array) or wrapl = 0 (do not wrap the array). The default is 1. Any APL array may be deflated using wrapl = 1. Only APL character vectors may be deflated using wrapl = 0.

filtered_strategy
filtered

huffman_only_strategy
huffman

rle_strategy
rle

fixed_strategy
fixed

default_strategy

The deflate algorithm uses a combination of Huffman encoding and LZ77 matched string encoding. The strategy parameters are used to tune the compression algorithm to work better with differing data characteristics. Data containing shorter matching strings and a somewhat random distribution may compress better with the filtered_strategy. This forces more Huffman encoding and less string matching. The huffman_only_strategy will eliminate string matching. The rle_strategy (run length encoding) will limit string matching to strings of length one. rle_strategy is faster than full string matching and gives better results than huffman_only_strategy on PNG image data. fixed_strategy prevents the use of dynamic Huffman codes, allowing for a simpler decoder for special applications. In general, if you don't know your data matches well with one of these characteristics, the default_strategy is the best choice.

zlib_header

gzip_header
gziphead

no_header
nohead

The default header on deflated data is the zlib header. To specify the gzip header, use gzip_header. To specify no header, use no_header.

The parameters for inflation are:

wrapl = n

The wrapl parameter is used during inflation to indicate whether the data was wrapped during deflation or not. If wrapl = 0 was used during deflation, wrapl = 0 must also be used during inflation. If wrapl = 1 was used during deflation, wrapl = 1 may be used during inflation, but is not necessary since it is the default.

zlib_header

gzip_header
gziphead

no_header
nohead

The default header on deflated data is the zlib header. If the deflated data was produced with a gzip header, specify gzip_header. If the deflated data was produced with no header, specify no_header.

result_size_estimate = n
restimate = n

This parameter specifies an estimate of how big the inflated data will be. It is used in the initial allocation of space for the result. If more space is needed, the result allocation will be expanded incrementally. If less space is needed, the result will be truncated.

result_size_limit = n
rlimit = n

This parameter specifies a fixed amount of inflated data to be returned. Any excess data beyond this size is not returned. If the total data available is smaller, the result is truncated.

result_size_estimate and result_size_limit only apply to wrapl = 0 inflation. The two parameters are mutually exclusive. If you specify both, you will get an error (Zlib Error: Conflicting keywords specified). If you do not supply either, a value of 3 times the deflated data size will be used by default for result_size_estimate.

The deflate and inflate functions rely on the ZLIB compression library written by Jean-loup Gailly and Mark Adler.

The format of data compressed with these functions is described in RFC 1950, RFC 1951, and RFC 1952.

XML serializing an array with ⎕dr

If you want to store or transmit an array of values, you can use ⎕dr to encode the entire array as readable XML data by using the character string 'xml' as the left argument. The resulting vector will contain a simple text-based format representing the APL array.

When you retrieve the coded vector, you can use ⎕dr to restore the exact values of the array by using 'unxml' as the left argument.

Example:

```
      xml_data←'xml' ⎕dr 'abc' (1 2 3)
      xml_data
<?xml version="1.0"?>
<Aval xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Type>Aval</Type>
  <Flags>IsShared HasNested</Flags>
  <ArrayOfAval>
    <Aval>
```

```
      <Type>Zc</Type>
      <Flags>IsShared</Flags>
      <base64Binary>YWJj</base64Binary>
      <Nelm>3</Nelm>
      <Shape>
        <int>3</int>
      </Shape>
    </Aval>
    <Aval>
      <Type>Int</Type>
      <Flags>IsShared</Flags>
      <ArrayOfUnsignedInt>
        <unsignedInt>1</unsignedInt>
        <unsignedInt>2</unsignedInt>
        <unsignedInt>3</unsignedInt>
      </ArrayOfUnsignedInt>
      <Nelm>3</Nelm>
      <Shape>
        <int>3</int>
      </Shape>
    </Aval>
  </ArrayOfAval>
  <Nelm>2</Nelm>
  <Shape>
    <int>2</int>
  </Shape>
</Aval>
      'unxml' ⎕dr xml_data
 abc   1 2 3
      'unxml' ⎕dr 'abc' (1 2 3)
DOMAIN ERROR
[imm] 'unxml' ⎕dr 'abc' (1 2 3)
              ^
```

JSON serializing an array with ⎕dr

If you want to store or transmit an array of values, you can use ⎕dr to encode the entire array as readable JSON data by using the character string 'json' as the left argument. The resulting vector will contain a simple text-based format representing the APL array.

When you retrieve the coded vector, you can use ⎕dr to restore the exact values of the array by using 'unjson' as the left argument.

Example:

```
      json_data←'json' ⎕dr 'abc' (123)
      json_data
{"?xml":{"@version":"1.0","@encoding":"u
      tf-8"},"Aval":{"@xmlns:xsi":"http:
      //www.w3.org/2001/XMLSchema-instan
      ce","@xmlns:xsd":"http://www.w3.or
      g/2001/XMLSchema","Type":"Aval","F
      lags":"IsShared HasSimple HasNeste
      d","Nelm":"2","ArrayOfAval":{"Aval
      ":[{"Type":"Zc","Flags":"IsShared"
      ,"Nelm":"3","ArrayOfChar":{"char":
      ["97","98","99"]},"Shape":{"int":"
      3"}},{"Type":"Int","Flags":"IsShar
      ed","Nelm":"1","ArrayOfInt":{"int"
      :"123"},"Shape":null}]},"Shape":{"
      int":"2"}}}
      'unjson' ⎕dr json_data
abc 123
```

⎕drop Deletes a saved workspace from disk

Purpose:
Erase a saved workspace from disk storage under program control.

Syntax: result ← ⎕drop wsid

Argument:
wsid is the workspace identifier without the .ws64 file extension.

Effect:
Deletes the named workspace (wsid) from storage and displays the timestamp of the operation; does not affect the active workspace.

If the workspace does not exist, you receive a WS NOT FOUND message with the path and name of the workspace. If you do not have permission from the operating system to delete this file, the system displays a HOST ACCESS ERROR.
The combined use of ⎕ntie and ⎕nerase provide a similar capability.

Note: This system function is not applicable to APL+Win workspaces.

Result:
The timestamp of the operation.

Example:

```
      )wslib
UCMDUTIL.ws64  myws.ws64
      ⎕drop 'myws'
4/7/2022 10:11:59 PM
      )wslib
UCMDUTIL.ws64
```

⎕driveinfo Interface to .Net DriveInfo class

Purpose:
Returns information on a drive on the current machine.

Syntax: result ← ⎕driveinfo property drive

Argument:
property is a text vector that determines the information on the drive. The acceptable values are:

Value	**Meaning**
AvailableFreeSpace	The amount of available space on a drive, in bytes
DriveFormat	The name of the file system; e.g., NTFS or FAT32
DriveType	The drive type; e.g., CD-ROM, fixed, network, or removable
Help or ?	Help information
IsReady	Indicates whether a drive is ready; 1 is ready and 0 is not ready
Name	The name of the drive; e.g., C:\
RootDirectory	The root directory of a drive
TotalFreeSpace	The total amount of free space available on a drive, in bytes
TotalSize	The total size of storage on a drive, in bytes
VolumeLabel	The volume label of a drive

drive can be a disk drive letter, a root directory, or a root directory with a path name.
Note: A UNC (Universal Naming Convention) path and directory name are not supported.

Result:
The result depends on the property called.

Examples:

```
      ⎕driveinfo 'IsReady' 'c'
1
      ⎕driveinfo 'DriveFormat' 'c'
 NTFS
      ⎕driveinfo 'Name' 'c'
c:\
```

⎕dtb Delete Trailing Blanks

Purpose:
Removes trailing blanks from the character array or string.

Syntax: result ← [duplicate] ⎕dtb text

Argument:
duplicate is the optional left argument to specify the duplicate character to be removed.
text is any array of character arrays or a string.

Result:
The result is any array of characters arrays or a string.

Remarks:
⎕dtb may be used as replacement for the function DTB in the ASMFNS.w3 workspace in APL+Win.

Example:

```
   ⎕dtb' The car cost $45,450 '
 The car cost $45,450
  ' The car cost $45,450'≡⎕dtb' The car cost $45,450 '
1
```

⎕dtbr Delete Trailing Blank Rows

Purpose:
Removes trailing blanks rows from the character matrix.

Syntax: result ← ⎕dtbr matrix

Argument:
matrix is any array of character matrix.

Result:
The result is any array of character matrix.

Remarks:
⎕dtbr may be used as replacement for the function DTBR in the ASMFNS.w3 workspace in APL+Win.

Example:

```
      a←4 5⍴'abcde     fghij     '
      ]display a
.→----.
↓abcde|
|     |
|fghij|
|     |
'-----'
      ]display ⎕DTBR a
.→----.
↓abcde|
|     |
|fghij|
'-----'
```

⎕dtr Deep Transpose

Purpose:
Performs a deep transpose of an array. This function works like monadic transpose but recurses to each child element and transposes it as well.

Syntax: result ← ⎕dtr array

Argument:
array is any APL array; it can be simple or nested, homogeneous or heterogeneous.

Result:
result is an array with the same rank as array.

Example:

```
     x←1 2ρ'abcd' (2 3ρι6)
     x
abcd   1 2 3
       4 5 6
     ⎕dtr x
abcd

 1 4
 2 5
 3 6
```

⎕dyadic Dyadic Function

Purpose:
Returns the valence for the current function.

Syntax: result ← ⎕dyadic

Result:
The result of ⎕dyadic is 1 if the function is being called dyadically or 0 otherwise such as when called from immediate execution.

⎕edit Edit an Object

Purpose:
Invokes an edit session where you can modify a function, character vector, character matrix, or numeric variable. Enter edit mode with named object (unlocked function, character vector or matrix).

Syntax: ⎕edit objectName
⎕edit objectName [row col]
⎕edit objectName [row col] (winHeight winWidth winTop winLeft winMaximized)

Argument:
objectName is the name of the function, character variable, or numeric variable you want to edit. If no object with the specified name exists or object is an empty character vector, ⎕edit assumes the default edit session type. If you are creating a new object, you can specify the type of object you want to create by preceding the name with ∇ for a function or # for a variable.

row and col are the optional two-element integer array for specifying the initial row and column caret position in an edit session; (row col) are index origin 0. If this optional element is not provided or the object name does not refer to a function or the object name refers to a non-existent (function), the initial caret position in the editor is system determined.

winHeight, winWidth, winTop, winLeft and winMaximized are the optional five-element array for specifying the specifications for the independent floating modal editor session window. winHeight, winWidth, winTop, and winLeft specify the dimensions and position of the edit session window position, in 96th-inch units; dimensions are the height (winHeight) and width (winWidth) and position is the row (winTop) and column (winLeft) of the upper left corner of the edit session window relative to the Windows desktop. While this modal edit session is open, there will be no other interaction with the APL64 programmer session until the modal edit session is closed. winMaximized is a Boolean scalar that specifies whether the modal floating edit session window opens maximized or not; 0=No (Default) and 1=Yes.

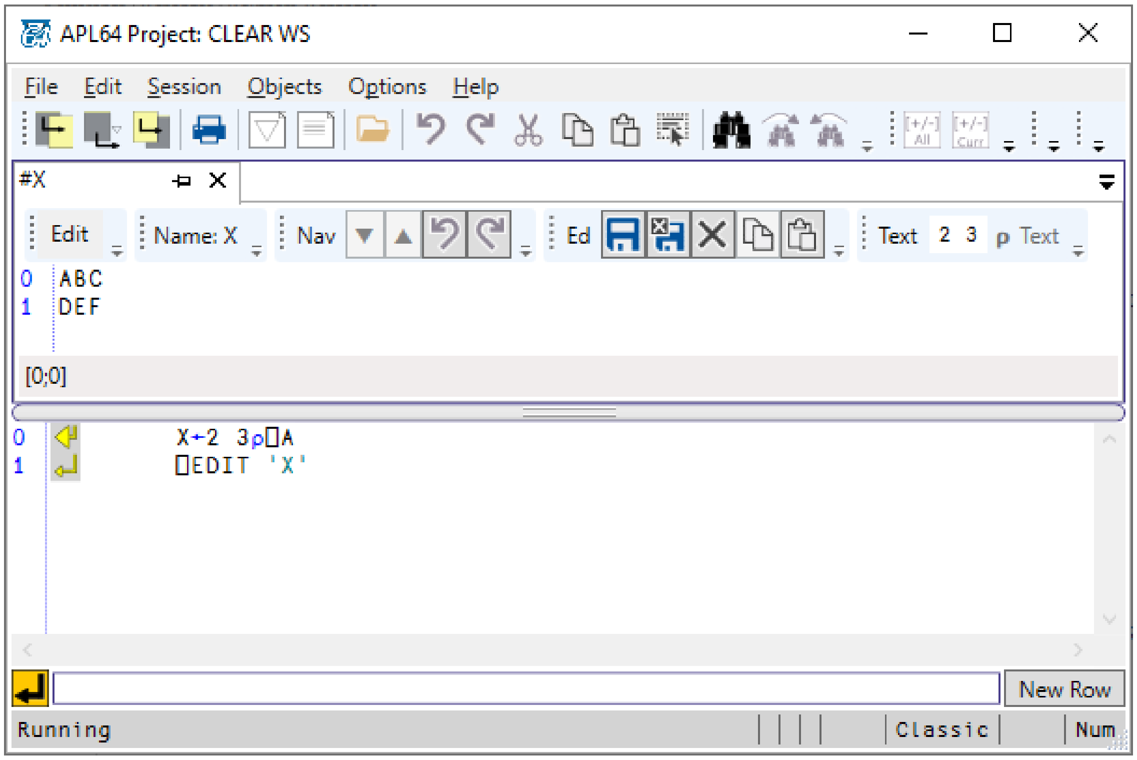
Remarks:
⎕EDIT is available only in the developer version of APL64 as the runtime version of APL64 does not provide a programmer GUI including object editors.

Use the ⎕EditEvents system function to subscribe to events associated with editing APL64 objects.

Examples:

Non-modal Variable Editor

```
X←2 3⍴⎕A
⎕EDIT 'X'
```

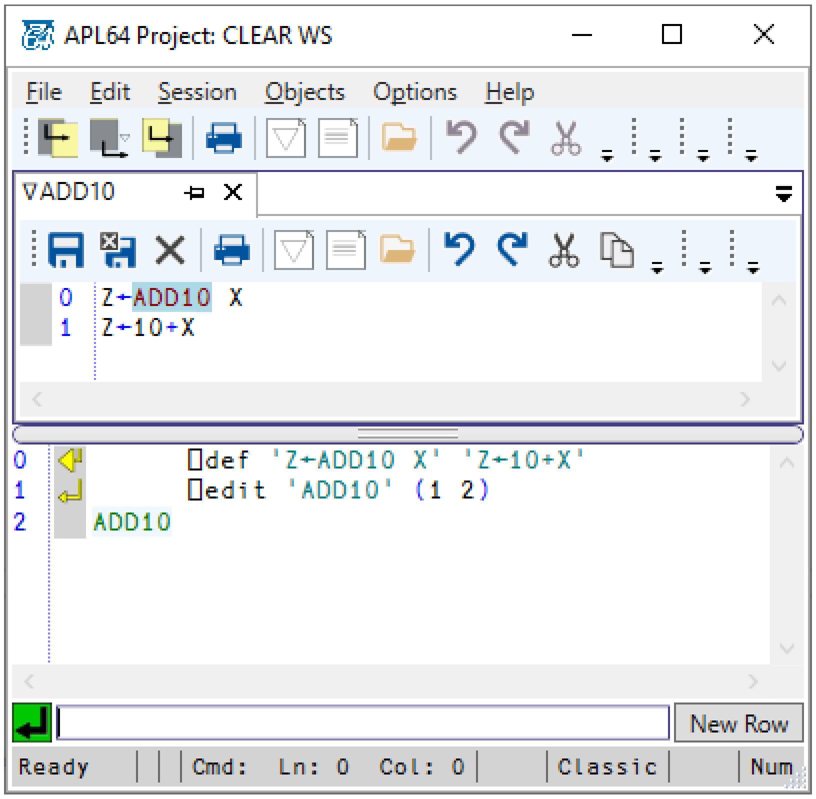


```
0 ABC
1 DEF
[0;0]
```

```
X←2 3⍴⎕A
⎕EDIT 'X'
```

Non-modal function editor

```
⎕def    'Z←ADD10     X'     'Z←10+X'
⎕edit 'ADD10' (1 2)
```

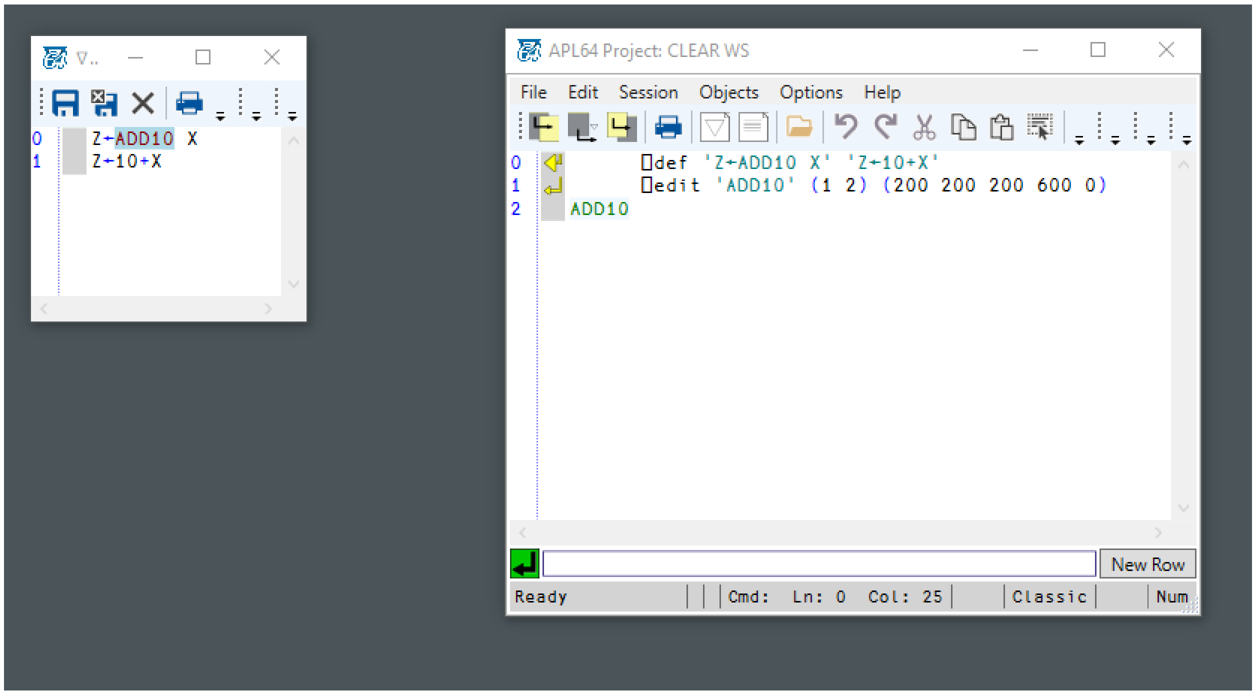


```
Z←ADD10 X
Z←10+X
```

```
      ⎕def 'Z←ADD10 X' 'Z←10+X'
      ⎕edit 'ADD10' (1 2)
ADD10
```

Modal function editor

```
⎕def 'Z←ADD10 X' 'Z←10+X'
⎕edit 'ADD10' (1 2) (200 200 200 600 0)
```

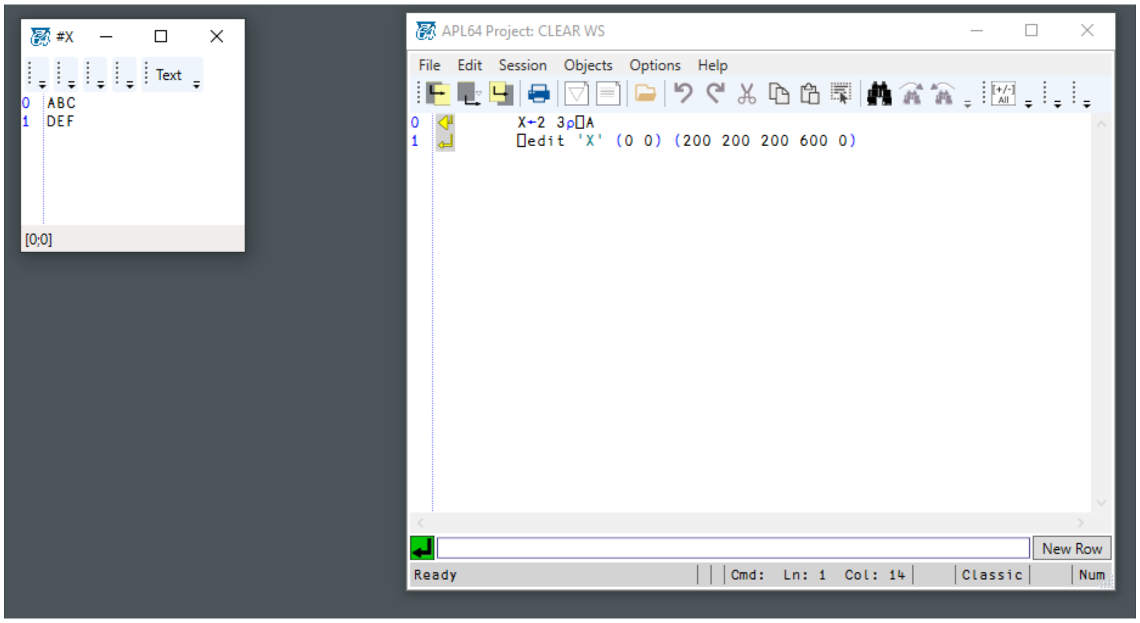


```
Z←ADD10 X
Z←10+X
```

```
⎕def 'Z←ADD10 X' 'Z←10+X'
⎕edit 'ADD10' (1 2) (200 200 200 600 0)
ADD10
```

Modal Variable Editor:

```
X←2 3ρ⎕A
⎕edit 'X' (0 0) (200 200 200 600 0)
```



```
ABC
DEF
```

```
X←2 3ρ⎕A
⎕edit 'X' (0 0) (200 200 200 600 0)
```

⎕editevents

Subscribe to APL64 developer GUI events associated with the editing of user defined functions or variables. For detailed documentation use **Help | Developer Version GUI | Using ⎕EditEvents and ⎕EditInfo**.

⎕editinfo

Obtain information about editing of user defined functions or variables in an APL64 developer version instance. For detailed documentation use **Help | Developer Version GUI | Using ⎕EditEvents and ⎕EditInfo**.

⎕edss Edits an object in a worksheet editor

Edits APL arrays in a worksheet editor. For detailed documentation use **Help | Developer Version GUI | Editing APL64 Objects | Using ⎕EDSS and)EDSS**.

⎕embrcs Extract Embedded Resources

Purpose:
Extracts embedded resources in the APL64 runtime executable.

Syntax: result ← ⎕embrcs action
⎕embrcs action rarg

Arguments:
action is a text vector that determines how the system function will operate; they are:

action	Derived From
?	Help; return syntax
EMBRCS	Embedded Resources
GEAMRNS	Get Entry Assembly Manifest Resource Names
GEAN	Get Entry Assembly Name
SRTF	Save Resource To File

rarg are the arguments to the action SRTF.

Result:
The result depends on the arguments you supply.

Effect:

⎕embrcs '?'
Returns a character matrix describing the syntax of the ⎕embrcs system function.

⎕embrcs 'GEAMRNS'
Returns a character matrix of the 'resource names' of the accessible resources in the executing assembly, i.e. the APL64 runtime executable. Each row of the result contains such a resource name. This action applies only to the APL64 'runtime' version.

⎕embrcs 'GEAN'
Returns the full name of the ‘Entry assembly’. In the APL64 ‘runtime’ version the entry assembly is the APL64 runtime executable.

⎕embrcs 'SRTF' 'resourceName' 'targetFilePath' 'boolOverwrite'
Accesses and writes the resource content specified by the ‘resourceName’ to the file specified by the ‘targetFilePath’. This action applies only to the APL64 ‘runtime’ version.

If the file specified by the ‘targetFilePath’ exists it will be overwritten only the ‘boolOverwrite’ is 1. If the file specified by the ‘targetFilePath’ does not exist, it will be created. If the file path of the target file does not exist it will be created. The end user’s workstation permissions for the target file must provide for read/write access.

Remarks:
The ⎕embrcs function applies only to the Windows Runtime Executable.

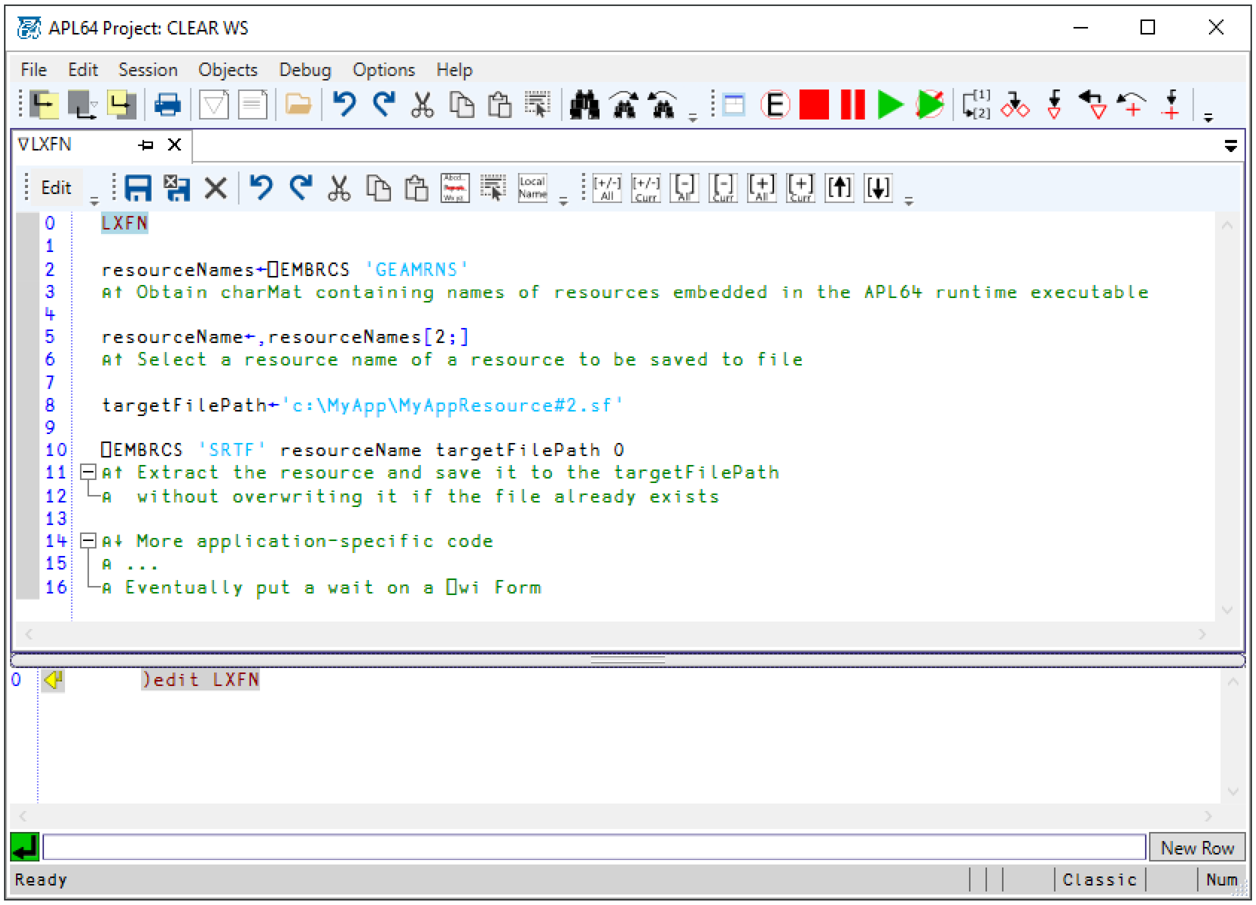
In APL64, a runtime application is a ‘runtime executable’ created via the APL64 menu **Session | Options | Create .Net Assembly | Create Runtime Executable** dialogue. A runtime executable is a Microsoft Windows exe-format file containing the APL64 runtime assembly, an APL64 workspace and any APL programmer- designated, application-specific files. An APL64 runtime executable provides for convenient deployment of an APL64-based application system to end users as a single, ready-to-run file.

The application-specific files contained in an APL64 runtime executable are the ‘embedded resources’ of the application. Any type of Microsoft Windows compatible file may be an embedded resource in an APL64 runtime executable. In the embedded state, a resource is not directly accessible by APL64 system functions such as ⎕fread, ⎕cfread or ⎕nread. Using the ⎕embrcs system function, an embedded resource in the currently-running APL64 runtime executable can be identified and saved to a file on the workstation so that it will be accessible by APL64 system functions.

Typically, the ⎕embrcs actions will be used in the ⎕lx function which commences the APL64 application’s processing, so that the application-specific files are accessible to that application system when it is run by the application system’s end user.

Examples:

#1: Typical use of ⎕embrcs in the ⎕lx function of an APL64 runtime application:



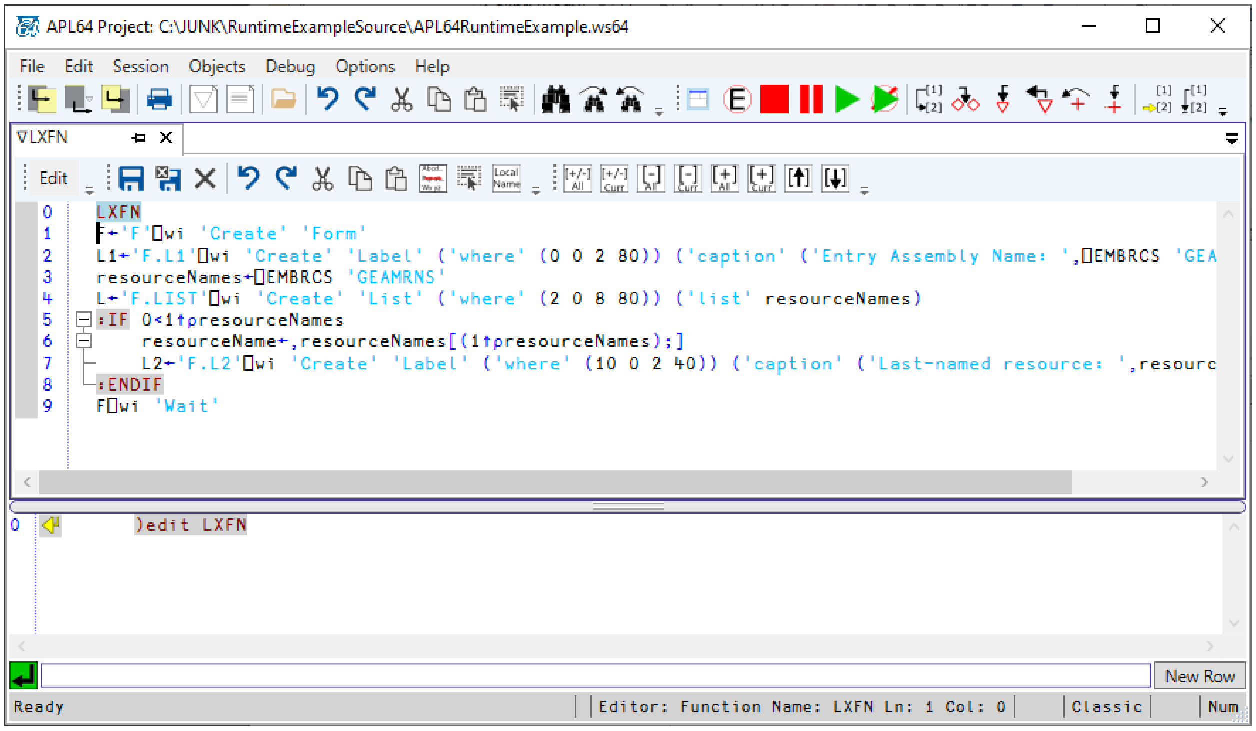
APL64 Project: CLEAR WS

```
∇LXFN

resourceNames←⎕EMBRCS 'GEAMRNS'
⍝⍝ Obtain charMat containing names of resources embedded in the APL64 runtime executable

resourceName←,resourceNames[2;]
⍝⍝ Select a resource name of a resource to be saved to file

targetFilePath←'c:\MyApp\MyAppResource#2.sf'

⎕EMBRCS 'SRTF' resourceName targetFilePath 0
⍝⍝ Extract the resource and save it to the targetFilePath
⍝  without overwriting it if the file already exists

⍝⍝ More application-specific code
⍝ ...
⍝ Eventually put a wait on a ⎕wi Form
```

```
      )edit LXFN
```

#2: Resource names are prefixed by the ‘entry assembly’ name:

This ⎕lx function displays the embedded resources names when the APL64 runtime executable runs:

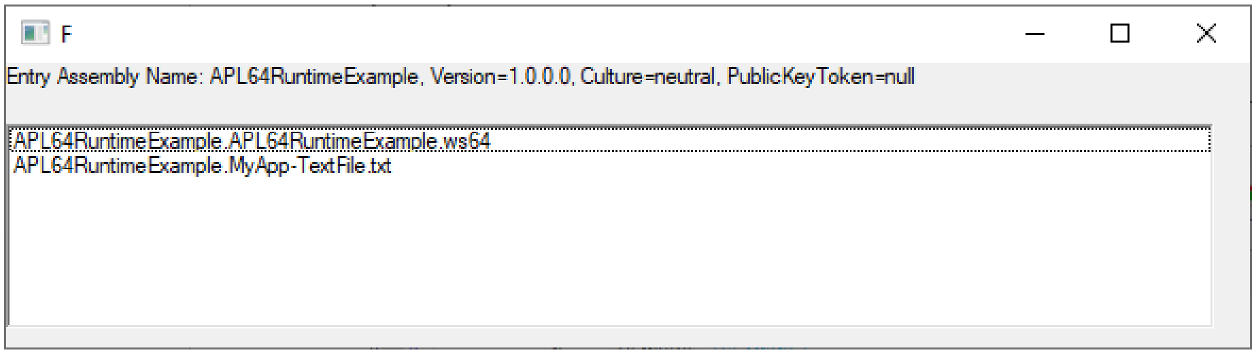

APL64 Project: C:\JUNK\RuntimeExampleSource\APL64RuntimeExample.ws64

```
LXFN
F←'F'⎕wi 'Create' 'Form'
L1←'F.L1'⎕wi 'Create' 'Label' ('where' (0 0 2 80)) ('caption' ('Entry Assembly Name: ',⎕EMBRCS 'GEA
resourceNames←⎕EMBRCS 'GEAMRNS'
L←'F.LIST'⎕wi 'Create' 'List' ('where' (2 0 8 80)) ('list' resourceNames)
:IF 0<1↑ρresourceNames
    resourceName←,resourceNames[(1↑ρresourceNames);]
    L2←'F.L2'⎕wi 'Create' 'Label' ('where' (10 0 2 40)) ('caption' ('Last-named resource: ',resourc
:ENDIF
F⎕wi 'Wait'
```

```
      )edit LXFN
```

Ready | Editor: Function Name: LXFN Ln: 1 Col: 0 | Classic | Num

The APL64 runtime workspace and an application-specific file are embedded in the runtime executable:


F

Entry Assembly Name: APL64RuntimeExample, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null

```
APL64RuntimeExample.APL64RuntimeExample.ws64
APL64RuntimeExample.MyApp-TextFile.txt
```

⎕em Error Message

Purpose:

Returns the error message part of ⎕dm.

Syntax: result ← ⎕em

Result:

The result is a character vector that contains just the error message part of ⎕dm. This provides a convenient way to reference just the error message without function context and a way to localize the error message.

⎕enlist Convert Array to Simple Vector

Purpose:

Converts any array into a simple vector. This function is a cover for the Enlist primitive function; you can use this system function at any Evolution Level.

Syntax: result ← ⎕enlist array

Argument:
array is any APL array; it can be simple or nested, homogeneous or heterogeneous.

Result:
The result is the array converted into a simple vector.

Effect:
Converts the argument into a simple vector. ⎕enlist does not preserve the rank, shape, or depth of the array.

Note: At Evolution Level 2, you can use monadic epsilon (∊) as Enlist. At Evolution Level 1, monadic epsilon had a different behavior.

Example:

```
      A←'MARY' (2 3ρι6) 0 'JOE'
      ]DISPLAY A
.→-------------------------.
| .→---. .→-----.   .→--. |
| |MARY| ↓ 0 1 2| 0 |JOE| |
| '----' | 3 4 5|   '---' |
|        '~-----'         |
'∊-------------------------'
      ]DISPLAY ⎕enlist A
.→---------------------.
|MARY 0 1 2 3 4 5 0 JOE|
'+---------------------'
```

⎕erase Erase Functions and Variables

Purpose:
Erases, if possible, the most local version of functions or variables in the workspace while under program control.

Syntax: result ← ⎕erase 'namelist'

Argument:
namelist is a list of function or variable names; it can be a character vector that contains the names separated by one or more blanks, or a character matrix with one identifier in each row.

Result:
The result is a character matrix in which each row contains the name of a defined function or variable

that was not erased. Objects that are undefined are not included in result. If the system erases everything in namelist or if namelist is empty, result is an empty matrix.

Effect:
Erases the functions and variables specified in namelist. An object might not be erased because the name is a label.

In APL64, ⎕erase can erase a suspended or executing function. In fact, a function can even erase itself. The name association with the function is broken, but the executing function does not actually disappear until it completes execution or is cleared from the state indicator stack.

Note: ⎕erase and ⎕ex provide similar capabilities. For maximum portability to other APL systems, use ⎕ex rather than ⎕erase.

Example:

```
     ρ⎕←⎕erase 'MYPROGRAM'
0 0
     ρ⎕vr 'MYPROGRAM'
0
```

⎕error Error Exception Signal

Purpose:
Generates a user-defined error exception.

Syntax: ⎕error 'message'

Argument:
message is a diagnostic message; that is, a character singleton or vector that contains the first line of the diagnostic message associated with the resulting error exception.

Effect:
⎕error provides two facilities:

- the ability of a function to signal an exception to the program from which it was called
- the ability to signal user-defined error exceptions.

When the system executes ⎕error, it returns the state indicator stack to the environment from which the function executing ⎕error was called. If the state indicator is empty or contains only one function when the system executes ⎕error, the error exception is signaled in the global environment.

If message is empty (''), ⎕error does not signal an exception; this permits conditional signaling of error exceptions with a statement of the form ⎕error condition /'message'.

See the Exception Handling chapter in this manual for more information.

Examples:
In the following function SQRT, ⎕error signals an error in the environment from which SQRT is called instead of within SQRT itself. It also detects a negative argument and generates an error message that is more informative than the DOMAIN ERROR report normally produced by the system.

```
   ∇R←SQRT A;⎕elx
[1]   ⎕elx←'⎕error ((⎕dmι⎕tcnl)-⎕io)↑⎕dm'
[2]   ⎕error (∨/,A<0)/'ARGUMENT NEGATIVE'
[3]   R←A*0.5
   ∇

    SQRT ¯1
ARGUMENT NEGATIVE
    SQRT ¯1
    ^
```

If SQRT is called from another function and a negative argument is supplied to SQRT, ⎕error signals an error in the calling function.

```
   ∇R←M RELMASS V;C
[1] ⍝ COMPUTES RELATIVISTIC MASS
[2] ⍝ OF A MOVING OBJECT
[3] ⍝ M "→ REST MASS; V "→ VELOCITY
[4] ⍝ C "→ SPEED OF LIGHT IN METERS/SEC
[5]   C←300000000
[6]   R←M÷SQRT 1-(V*2)÷C*2
   ∇

    1 RELMASS 2.9E8
3.905667329

    1 RELMASS 3.5E8
ARGUMENT NEGATIVE
RELMASS[5] R←M÷SQRT 1-(V*2)÷C*2
           ^
```

You can use the following technique to clear the result of ⎕dm, provided that the state indicator is clear and ⎕elx does not call ⎕error.

```
    ⎕error ' '
```

Since ⎕error reduces the state indicator stack by one function call, you can use it to move one level up in the state indicator for debugging purposes; for example:

```
    DRIVER
LENGTH ERROR
```

```
SUBROUTINE[1] Z←A+B×0,1↓A
              ^
    )SI
SUBROUTINE[1] *
PROCESS[7]
MAINFN[3]
DRIVER[5]
    ⎕error 'POP'
POP
PROCESS[7] SUBROUTINE
        ^
    )SI
PROCESS[7] *
MAINFN[3]
DRIVER[5]
```

You can now correct the argument (B) to SUBROUTINE and resume execution.

```
    B←(ρA)↑B ⋄ →⎕lc
```

⎕eval Evaluate Array

Purpose:
An alternative to using the execute primitive function, which may cause an internal stack overflow in APL64 when the argument character vector is a too large.

Syntax: result ← ⎕eval array

Argument:
array is a character singleton or vector that may only contain the following syntactic elements: strings, array of Unicode characters, numeric constants, character constants, strand notation for nesting, the monadic enclose primitive for nesting, the reshape primitive, and parentheses (for strand notation and grouping). Character constants may use either single or double quotes, and follow the normal rules for doubling quotes to embed quotes in the character string. Additionally, special characters (e.g., ⎕tcnl and ⎕tcht) may be embedded literally in the constant. The reshape primitive may only occur once at the beginning of an array or sub-array;
e.g., 7 ρ 3 ρ 'AB' is not allowed. Finally, the assignment is not allowed within the argument.

Result:
The result is the result generated by evaluating the array.

Caution:
⎕eval is not a general replacement for the execute primitive function.

Examples:

```
    A←⎕eval '3 4 5'
    A+1
4 5 6
```

⎕ex Erase Functions and Variables

Purpose:
Erases, if possible, the most local version of one or more functions or variables in the active workspace under program control.

Syntax: result ← ⎕ex 'namelist'

Argument:
namelist is a list of functions or variable names; it can be a character vector containing the names separated by one or more blanks, or a character matrix with one identifier in each row. The list cannot contain blank rows.

If ⎕ex produces a WS FULL or DOMAIN ERROR, nothing is erased.

Result:
The result is a Boolean vector with one element for each name in namelist. The result is 1 if the object was erased or undefined; the result is 0 if the object was not erased. An object might not be erased because the name is ill-formed or a label. If namelist is an empty vector, ⎕ex returns 0.

Effect:
Erases functions and variables specified in namelist. ⎕ex does not erase an identifier if it is a label, system function, or system variable.

Caution:
Some APL systems may restrict namelist to a character matrix.

Examples:

```
    TREE← 3 4 5
    TREE
3 4 5
    ⎕ex 'TREE'
1
    TREE
VALUE ERROR
    TREE
    ^
```

⎕exepath Path to current executable file

Purpose:
Returns the path where the APL64.exe file or WRE executable file was installed.

Syntax: path ← ⎕exepath

Domain:
Returns a character vector containing the path defined by .Net:
System.IO.Path.GetDirectoryName(Environment.GetCommandLineArgs().First()).

Effect:
For the APL64 developer version the ⎕exepath value is path of the APL64.exe file.

For a WRE runtime application, the ⎕exepath value is the path where the WRE executable (.exe) file was installed on the end-user's workstation.

Note: Compare the definitions and values of ⎕exepath and ⎕acbd.

⎕expand Expand an Array

Purpose:
Expands an array with fill items. This function provides the same behavior that is derived by the Expand operator with a variable as the left argument; you can use this system function at any Evolution Level.

Syntax: result ← boolvec ⎕expand array
result ← boolvec ⎕expand[i] array

Arguments:
array: any APL array; if the chosen dimension has length 1, arg is extended along the expansion axis to the number of positive elements in the operand (+/boolvec). This allows you to generate only fill elements for an array that is scalar or has length 1 along the chosen dimension and all zeroes in boolvec.
boolvec is a Boolean vector that specifies where you want to insert fill items in array; the sum of the elements must be the same length as the default or designated dimension of array.
i is a non-negative, integer-valued scalar that indicates the axis along which you want the function to expand the elements; the default is along the last dimension.

Result:
The result is the array, expanded by inserting a fill item for each corresponding 0 in boolvec.

Examples:

```
      0 0 1 0 1 ⎕expand 7 8
0 0 7 0 8
      1 0 1 0 1 ⎕expand 4
4 0 4 0 4
      0 0 ⎕expand 5
0 0
      0 0 ⎕expand[1] 1 4ρι2
 0 0 0 0
 0 0 0 0

      A←2 3ρ'ABCDEF'
```

```
ABC
DEF
    1 0 1 0 0 1 ⎕expand A
A B  C
D E  F
    1 0 1 ⎕expand[1] A
ABC

DEF
```

Note: At Evolution Level 1, there was a difference in the result for appropriate arguments between the expressions larg \¨ rarg and larg (\)¨rarg. The latter required a nested left argument, each item being a Boolean vector appropriate for the corresponding item of rarg. At Evolution Level 2, the syntax(\)¨ generates an EVOLUTION ERROR. Use ⎕expand¨ when you want to use Expand with different arguments for each item of a nested array.

```
    nest← (1 2) (3 4)
    1 0 1 ⎕expand nest
 1 2  0 0  3 4
    1 0 1 \ nest
 1 2  0 0  3 4
    1 0 1 \¨ nest
 1 0 2  3 0 4
    (1 0 1) (1 0 0 1) ⎕expand¨ nest
 1 0 2  3 0 0 4
    (1 0 1) (1 0 1 0 1) ⎕expand¨ (5 6) (7 8 9)
 5 0 6  7 0 8 0 9
```

⎕fappend Append to a Component File

Purpose:
Appends a new component to the end of a component file.

Syntax: result ← value ⎕fappend tieno
              result ← value ⎕fappend tieno [passno]

Arguments:
value is a value you want to append to a file; it can have any rank, shape, or datatype;

tieno is a positive integer file tie number;

passno is an integer passnumber.

The right argument must be an integer scalar or one- or two-element, integer vector that defines the file to which you want to append. It contains a valid positive tie number and optional valid passnumber. If you omit the passnumber, the system assumes it is 0.

Result:
The result is the number of the new component.

Effect:
Appends a new data component and component information (⎕frdci) to a file. Increases the disk space occupied by the file.

Access:
The file must be tied, the passnumber must match the one in effect, and you must have append access. The access code for ⎕fappend is 8.

Examples:
The first example places the visual representation of the function TRI in the next component of the file tied to 27 and captures the component number in the variable COMP. The next example appends the variable JANSALES to the end of the file tied to 33.

```
      COMP←(⎕vr 'TRI') ⎕fappend 27

      ⎕fsize 33
1 20 36412 100000 0
      JANSALES←48032 ⋄ JANSALES ⎕fappend 33
20
      ⎕fsize 33
1 21 36432 100000 0
```

⎕favail Availability of Component File System

Purpose:
Indicates availability of the component file system.

Syntax: result ← ⎕favail

Result:
The result is 1 if the component file system is available for use, 0 if it is not.

Note: In APL64, component files are accessed directly from Windows and are always available. ⎕favail is included for compatibility with other APL systems that use a file server, which may not be available at all times.

⎕fcreate Create Component File

Purpose:
Creates a new component file that is share-tied (⎕fstie) and not exclusively tied (⎕ftie).
(See also the extended file function ⎕xfcreate, which supersedes this function.)

Syntax:
```
'fileid'                ⎕fcreate tieno
'fileid size'           ⎕fcreate tieno
'fileid size/compno'    ⎕fcreate tieno
```

Arguments:
fileid is a file identifier;

size specifies the file size limit in bytes, expressed as numerals, up to 4294967295 (4 gigabytes);

compno is an integer that specifies the initial component number;

tieno is a positive integer file tie number.

The left argument must be a character scalar or vector that designates the file you want to create. It contains the file identifier (fileid) and, optionally, the file size limit (size) and starting component number (compno). The file name must be different from any others in that directory or library.

The optional size specifies a limit on the amount of space the file can occupy on disk. If you omit size, the default is 0, meaning the file has no limit on its size. You can change the file size limit with ⎕frename or ⎕fresize.

The optional compno specifies the starting component number for the new file. It must be an integer and follow a slash (/) in the argument. If you omit compno, the starting component number is 1. (If you omit size, you must include a space between fileid and the slash.)

The file tie number (tieno) must be a positive integer scalar or one-element vector. You must have no other file tied with the number.

Effect:
Creates a new file and ties it to the specified tie number.

Access:
No file access code is required for ⎕fcreate.

Examples:

```
      'TEXTFILE' ⎕fcreate 27
      'PRINTFILE 225000' ⎕fcreate 1
      'C:\APLII\TEMP' ⎕fcreate 99
      ⎕libd '12 C:\APLII\DATA'
      '12 DATA88' ⎕fcreate 98
```

Note: The system appends the default extension (.SF) to the fileid you provide. Using the traditional file functions, you are limited in names you can use. For new programming, the extended file functions provide more capability and flexibility.

⎕fdrop Drop Components from File

Purpose:
Drops components from either end of a component file.

Syntax: ⎕fdrop tieno n
⎕fdrop tieno n [passno]

Arguments:
tieno is a positive integer file tie number;

n is the number of components you want to drop;

passno is an integer passnumber.

The argument must be a two- or three-element integer vector that designates the tie number (tieno) of the file, the components you want to drop, and an optional passnumber. If you omit the passnumber, the system assumes it is 0.

Effect:
Drops components from a file. If n is positive, the system drops n components starting from the beginning of the file. If n is negative, the system drops (|n) components from the end of the file. If n is 0, the system does not drop any components.

Access:
The file must be tied, the passnumber must match the one in effect, and the user must have drop access. The access code for ⎕fdrop is 32.

Examples:

```
    ⎕fsize 27
1 10 7424 0 0
    ⎕fdrop 27 2 ⋄ ⎕fsize 27
3 10 7424 0 1200
    ⎕fdrop 27 ¯3 ⋄ ⎕fsize 27
3 7 2536 0 1200
```

⎕fdup Duplicate Component File

Purpose:
Creates an exact copy of a file with a new name and if possible, compacts it to occupy less disk space. (See also the extended file function ⎕xfdup, which supersedes this function.)

Syntax:
```
'fileid'                ⎕fdup tieno
'fileid size/compno'    ⎕fdup tieno [passno]
```

Arguments:
fileid is a file identifier;

size specifies the file size limit in bytes, expressed as numerals, up to 4294967295 (4 gigabytes);

compno is an integer that specifies the initial component number;

tieno is a positive integer file tie number;

passno is an integer file passnumber.

The left argument must be a character scalar or vector that designates the new file you want to create. It contains the file identifier (fileid) and, optionally, the file size limit (size) and starting component (compno). The file identifier must be different from any others in that directory or library.

The optional size specifies a limit on the amount of storage a file can occupy on disk. If you omit size, the default is the size limit of the file being duplicated. If you specify compno, it must follow a slash (/) in the argument. If you omit compno, the starting component number is the same as in the original file. If you omit size, you must include a space between fileid and the slash.

The right argument must be an integer scalar or one- or two-element, integer vector that defines the file you want to duplicate. It contains a valid positive integer tie number and optional valid passnumber. If you omit the passnumber, the system assumes it is 0.

Effect:
⎕fdup creates a new file with the specified name (fileid) and copies all the data from the file specified by tieno into it. In the process, it recovers unused space created by replacing records with a different sized component, which may allow the new file to occupy less disk space than the original file. The old file remains unchanged.

You can use the same name as the file being duplicated for the new file. Using the same name replaces the file with the compacted version. The file must be exclusively tied to duplicate it with the same name; otherwise, a NONCE ERROR occurs.

Access:
The file you want to duplicate must be tied, the passnumber must match the one in effect, and you must have both duplicate access and the authority to create files in the specified (or default) directory or library. The access code for ⎕fdup is 16384.

Example:

```
      ⎕flib ''
LISTINGS
      'LISTINGS' ⎕ftie 10
      ⎕fnames
C:\LISTINGS
      'LEANINGS' ⎕fdup 10
      ⎕fnames
C:\LISTINGS
      ⎕flib ''
LEANINGS
LISTINGS
```

Note: The system appends the default extension (.SF) to the fileid you provide. Using the traditional file functions, you are limited in names you can use. For new programming, the extended file functions provide more capability and flexibility.

⎕ferase Erase Component File

Purpose:
Erases a tied component file.
(See also the extended file function ⎕xferase, which supersedes this function.)

Syntax: 'fileid' ⎕ferase tieno
'fileid' ⎕ferase tieno [passno]

Arguments:
fileid is a file identifier;

tieno is a positive integer file tie number;

passno is an optional file passnumber.

The left and right arguments designate the same file. The left argument is a character scalar or vector that contains the file identifier (fileid). The right argument must be an integer scalar or one- or two-element, integer vector that defines the file you want to erase. It contains a valid positive tie number and optional valid passnumber. If you omit passnumber, the system assumes it is 0.

Effect:
Unties a file and erases it from the directory or library. All of the data in the file are destroyed.

Access:
The file must be tied; the system cannot erase the file if any other user also has it tied. The passnumber must match the one in effect and you must have erase access. The access code for ⎕ferase is 4.

Examples:

```
      'TEXTFILE' ⎕ftie 10
      'TEXTFILE' ⎕ferase 10

      'PRTFILE' ⎕fstie 33 707
      'PRTFILE' ⎕ferase 33 707
```

Note: The system assumes the default extension (.SF) for the fileid you provide. For new programming, the extended file functions provide more capability and flexibility.

⎕ff Find First Occurrence

Purpose:
Performs a left-to-right string search, locating the first occurrence or repeated occurrences of a character scalar or vector within another character vector or string.

Syntax: result ← target [startOffset [endOffset]] ⎕ff pattern [mode [repeat]]

Arguments:

target is a character vector or string to be searched in.

startOffset is an optional integer scalar in target at which the search is to begin (left end of the range).

endOffset is an optional integer scalar in target at which the search is to end (right end of the range). This will be the offset used when only one offset is specified. **Note**: endOffset is always 1 character past the end of the desired search range. Because the endOffset is specified as 1 past the end of the range, this means that the length of the range is always endOffset - startOffset for either ⎕io.

pattern is a character vector or scalar for the system to locate in target. If pattern is empty (''), result is (0 0) for the non-repeat mode and a shape (0 2) empty array for the repeat modes.

Mode is the search mode and indicates: 0=Literal (the default if omitted), 1= RegEx expression, or 2= Wildcard expression (may contains * and ? chars).

repeat is the repeat mode and indicates: 0=return only the first occurrence, 1=return all non-overlapping occurrences, 2=return all occurrences (including overlapping).

Effect:
The result is either a 2-element vector or a 2-column matrix representing the indices. Indexes are ⎕io dependent:

- It is a two-element vector if repeat=0 is specified or implied by omission (default). The 1st element is the offset where the match starts and the 2nd element is the offset where the match ends (this is the first character AFTER the last character of the match). If a match is not found the result depends upon ⎕io. For ⎕io=1 we return 0 0 but for ⎕io=0 we return ¯1 ¯1. The length of the match is given by indexes[2]-indexes[1].
- It is a two-column matrix if repeat≠0 is specified. In this case there is one row for each match that was found (zero rows if no matches were found). The 1st column is the offset where the match starts and the 2nd column is the offset where the match ends (this is the first character AFTER the last character of the match). The length of each match is given by indexes[row;2]-indexes[row;1].

Refer to ⎕fl for the counterpart to ⎕ff.
Examples (assuming ⎕io=1):

```
      ⍝ Set up target string that will be searched
      s←'abababababababababa'
      s,[.5]⍳⍴s
a b a b a b a b a  b  a  b  a  b  a  b  a
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

      ⍝ Find first occurrence of ‘bab’ in target
      s ⎕ff 'bab'
2 5

      ⍝ Find forward non-overlapping occurrences of ‘bab’ in target
      s ⎕ff 'bab' 0 1
  2  5
  6  9
```

```
 10 13
 14 17

      ⍝ Find forward overlapping occurrences of ‘bab’ in target
      s ⎕ff 'bab' 0 2
  2  5
  4  7
  6  9
  8 11
 10 13
 12 15
 14 17

      ⍝ Find first occurrence of regular expression in target
      s ⎕ff 'b.b' 1
2 5

      ⍝ Find last occurrence of wildcard expression in target
      s ⎕ff 'b?b' 2
2 5

      ⍝ Find non-overlapping regular expression occurrences in target
      s ⎕ff 'b.b' 1 1
  2  5
  6  9
 10 13
 14 17

      ⍝ Find overlapping regular expression occurrences in target
      s ⎕ff 'b.b' 1 2
  2  5
  4  7
  6  9
  8 11
 10 13
 12 15
 14 17

      ⍝ Begin searching forward from offset 5
      s 5 ⎕ff 'bab'
6 9

      ⍝ Begin searching forward from offset 6
      s 6 ⎕ff 'bab'
6 9

      ⍝ Begin searching forward from offset 7
      s 7 ⎕ff 'bab'
8 11

      ⍝ Find non-overlapping occurrences starting at 5
      s 5 ⎕ff 'bab' 0 1
  6  9
 10 13
 14 17

      ⍝ Find non-overlapping occurrences between offset 5 and 14
      s 5 14 ⎕ff 'bab' 0 1
  6  9
 10 13

      ⍝ Find overlapping occurrences between offset 5 and 14
      s 5 14 ⎕ff 'bab' 0 2
  6  9
  8 11
 10 13
```

⎕fflush Extended Flush Component File

Purpose:
Flush file buffers.

Syntax: ⎕fflush tieno

Arguments:
tieno is a positive integer file tie number

Effect:
Clears buffers and causes any buffered data to be written to the file.

Access:
The file must be exclusively tied, the passnumber must match the one in effect, and you must have flush access. The access code for ⎕fflush is 262144.

Examples:

```
⎕fflush 1
```

⎕fhist Component File History

Purpose:
Provides historical information about an APL component file.

Syntax: result ← ⎕fhist tieno passno

Arguments:
tieno is a positive integer file tie number;

passno is an integer file passnumber.

The argument must be an integer scalar or one- or two-element, integer vector that defines the file for which you want information. It contains a valid positive tie number and optional valid passnumber. If you omit the passnumber, the system assumes it is 0.

Result:
result is a three-row matrix containing information about the history of the file. Row 1 contains the user number of the file owner and the timestamp of the file's creation in packed form and ⎕ts form.

Row 2 contains the user number and timestamp associated with the most recent change to the file.

Row 3 contains the user number and timestamp associated with the most recent setting of the file access matrix.

Access:
The file must be tied and the passnumber must match the one in effect. In addition, the operating system must allow you to read the file. If not, a HOST ACCESS ERROR results.

Example:

```
    'TESTFILE' ⎕ftie 1 ⋄ ⎕fhist 1
103 2.783508749E15 1984 3 16 12 52 29 0      (Created)
199 2.784878374E15 1984 4  1  9 19 34 0    (Last change)
103 2.783699813E15 1984 3 18 17 56 53 0      (Access set)
```

⎕fhold File Hold

Purpose:
Synchronizes file operations in shared file systems.

Syntax: ⎕fhold tieno
⎕fhold tieno passno

Arguments:
The argument designates the file tie numbers and passnumbers of the files you wish to reserve for update. If you omit a passnumber, the system assumes it is 0. File tie numbers must be distinct, and they must designate tied files. The argument is an integer array constructed as follows:
tieno is scalar, vector, or one-row matrix of positive integer file tie numbers, or the first row of a two-row matrix containing one or more file tie numbers;
passno, if supplied, is the second row of a two-row matrix holding file passnumbers corresponding to the tie numbers in the first row.

Effect:
Provides an interlock scheme by which multiple users can synchronize file updates. Only one user can have the interlock at any one time. Each user who executes ⎕fhold waits in a queue for a turn to have the interlock. (**Note**: ⎕fhold does not lock files; it is a cooperative system. While an interlock is set, the system delays other users in turn from completing execution of their ⎕fhold operations but not from executing other file operations.)

⎕fhold first releases any current interlocks that you have, and then, when it is your turn, sets an interlock on each designated file. The system does not set any interlocks while another user has an interlock set on any of the designated files; ⎕fhold execution waits until all such other interlocks have been released.

The system releases all interlocks when the user who set them executes another ⎕fhold, exits APL64, enters immediate execution mode, or signals a strong interrupt. You can release the interlock on an individual file without affecting other interlocks by untying or retying the file. An empty vector or a one- or two-row, zero-column matrix releases all interlocks and does not set any. The system does not release file interlocks when a program stops for quad (⎕) or quote-quad (⍞) input. Stopping for input when files are held can impose long delays on other users and should be avoided except when necessary.

Note: Because of the limitations of the operating system, the order in which users with contending file holds are granted access is not necessarily the same as the order in which they requested the file holds.

Access:
The files must be tied, the passnumbers must match the ones in effect, and you must have hold access. The access code for ⎕fhold is 2048.

Examples:
Hold two files, one with and one without a passnumber.

```
    ⎕fhold 2 2⍴27 33 0 ¯317232 ⋄ ...
```

The following example holds a file while the system performs an update.

```
   ∇ FOO
.
.
.
[5] ⍝ UPDATE DIRECTORY
[6]  ⎕fhold TN
[7]  ENTRY←((⎕fread TN,1),[1] NEW)
[8]  ENTRY ⎕freplace TN,1
[9]  ⎕fhold ⍳0
. . .
   ∇
```

⎕fi Formatted Input Conversion

Purpose:
Converts a character string to numeric values.

Syntax: result ← ⎕fi 'data'

Argument:
data is a character singleton or vector to convert.

Result:
result is a numeric vector formed by taking a character argument and converting it to numbers. The conversion process interprets the character strings as numbers just as the interpreter recognizes characters you type from the keyboard in immediate execution mode as numbers. You can use numerals, the decimal point, the high minus, and the uppercase letter E for scientific notation. Additionally, ⎕fi converts the normal minus sign to a high minus when it precedes a number with no space between them. You cannot, however, use the plus sign. Multiple spaces between numbers are treated as a single space.

A group of characters that is invalid as a number appears as a single zero in result. This function is often used in conjunction with ⎕vi, which verifies the validity of data for formatting. ⎕vi returns a Boolean vector with 1s in the positions where groups of characters represent well-formed numbers, and 0s where they do not.

Examples:

```
    A←'666 ¯1.20  .1  314159E¯5  8,9,10'
    ⎕fi A
666 ¯1.2 0.1 3.14159 0
    ⎕fi '  2  '
2
    ρ⎕fi '  2  '
1
    ρ⎕fi ''
0
    ⎕fi 'ANSWER: 666'
0 666
    B←'ANSWER IS 666 LBS.'
    ⎕fi B
0 0 666 0
    (⎕vi B)/⎕fi B
666
```

⎕findinfiles Interface to .Net FindInFiles method

Purpose:
Performs a string search, locating all occurrences of a character scalar or vector or string scalar or vector, within files.

Syntax: ⎕FindInFiles targetFolder targetText ignoreCase ignoreSubdirectories [fileWildCards]

Arguments:
- targetFolder: The directory path of a file system folder containing files to be considered in the search, not a file path and not including wild cards
- targetText: Text to find in the contents of the files considered in the search
- ignoreCase: 0/1 or case-insensitive character vector or string scalar 'False'/'True'
- ignoreSubdirectories: 0/1 or case-insensitive character vector or string scalar 'False'/'True'

- fileWildCards: Optional file name wild card patterns:
    - Character scalar or vector
    - String scalar or vector
    - Vector of character vectors and string scalars

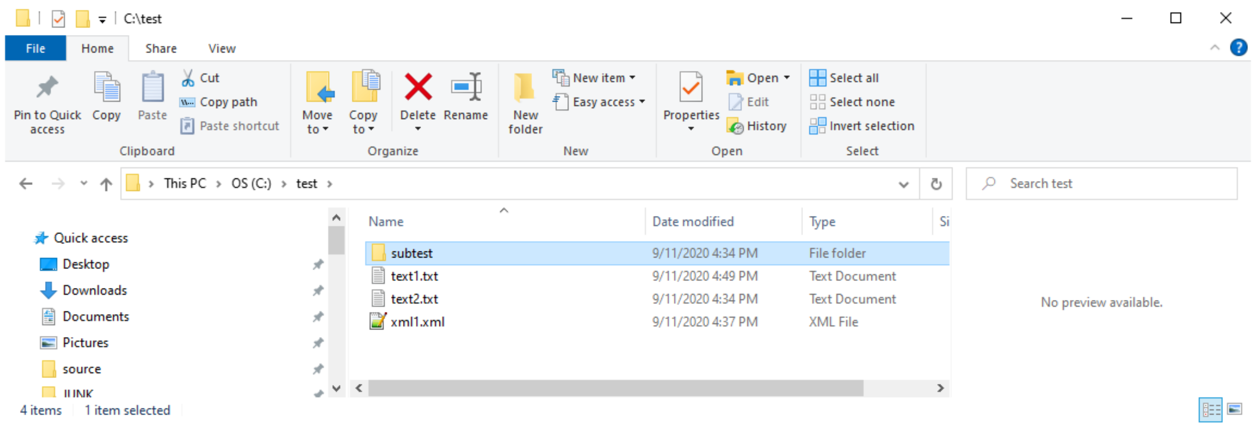
Result: Character matrix containing the file names, if any, of files containing the targetText. An exception will be thrown in the specified folder in the 1st argument does not exist or is not accessible.

Notes:

- The search is carried out using Unicode encoding
- File types: .txt, .csv, .xml, .json and other file types which contain explicit text are supported

Examples:

The c:\test\ folder on the target workstation contains:

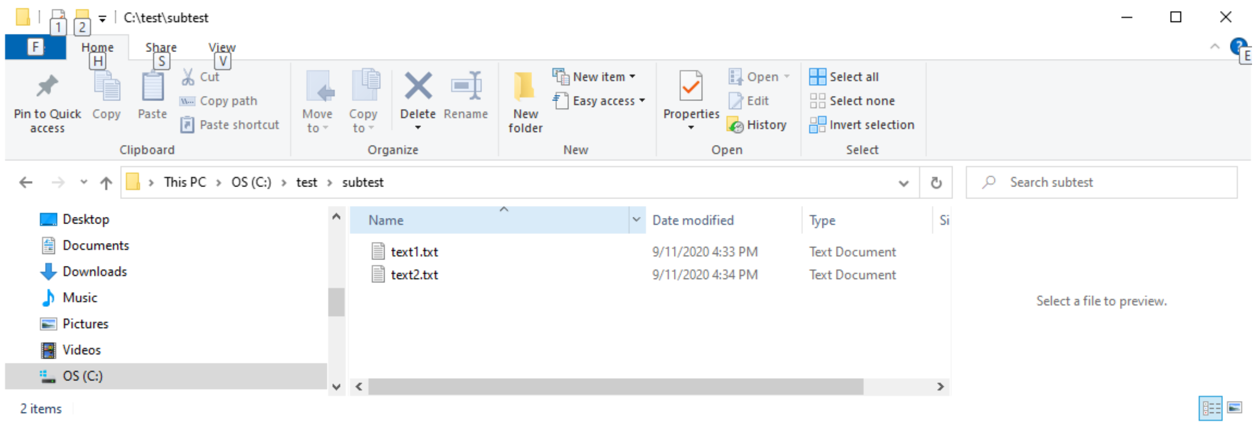

C:\test

This PC › OS (C:) › test ›

Name	Date modified	Type
subtest	9/11/2020 4:34 PM	File folder
text1.txt	9/11/2020 4:49 PM	Text Document
text2.txt	9/11/2020 4:34 PM	Text Document
xml1.xml	9/11/2020 4:37 PM	XML File

4 items 1 item selected

The c:\test\subtest\ folder on the target workstation contains:

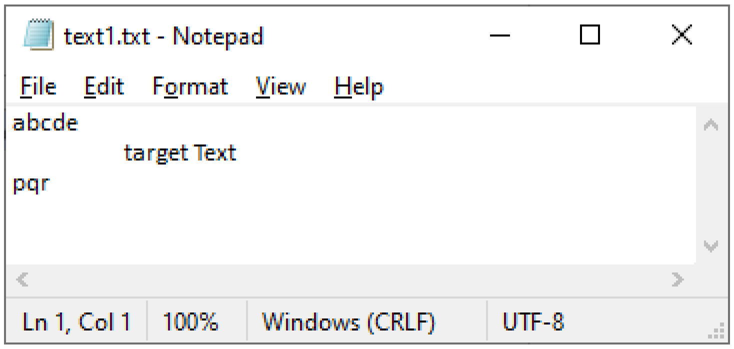

C:\test\subtest

This PC › OS (C:) › test › subtest

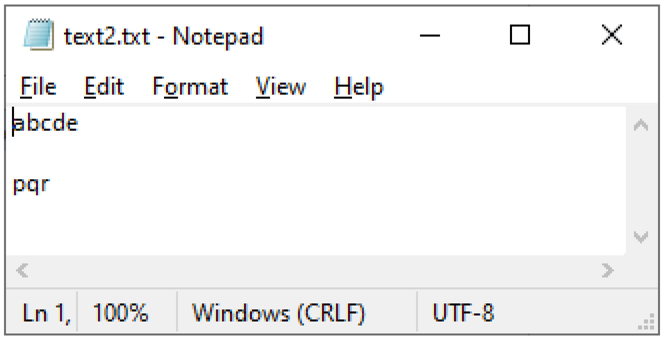
Name	Date modified	Type
text1.txt	9/11/2020 4:33 PM	Text Document
text2.txt	9/11/2020 4:34 PM	Text Document

2 items

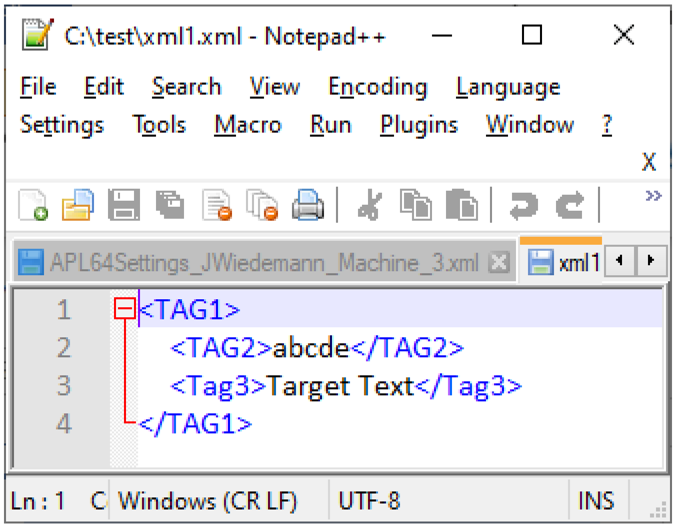
The content of the files:


text1.txt - Notepad
File Edit Format View Help
Ln 1, Col 1 100% Windows (CRLF) UTF-8

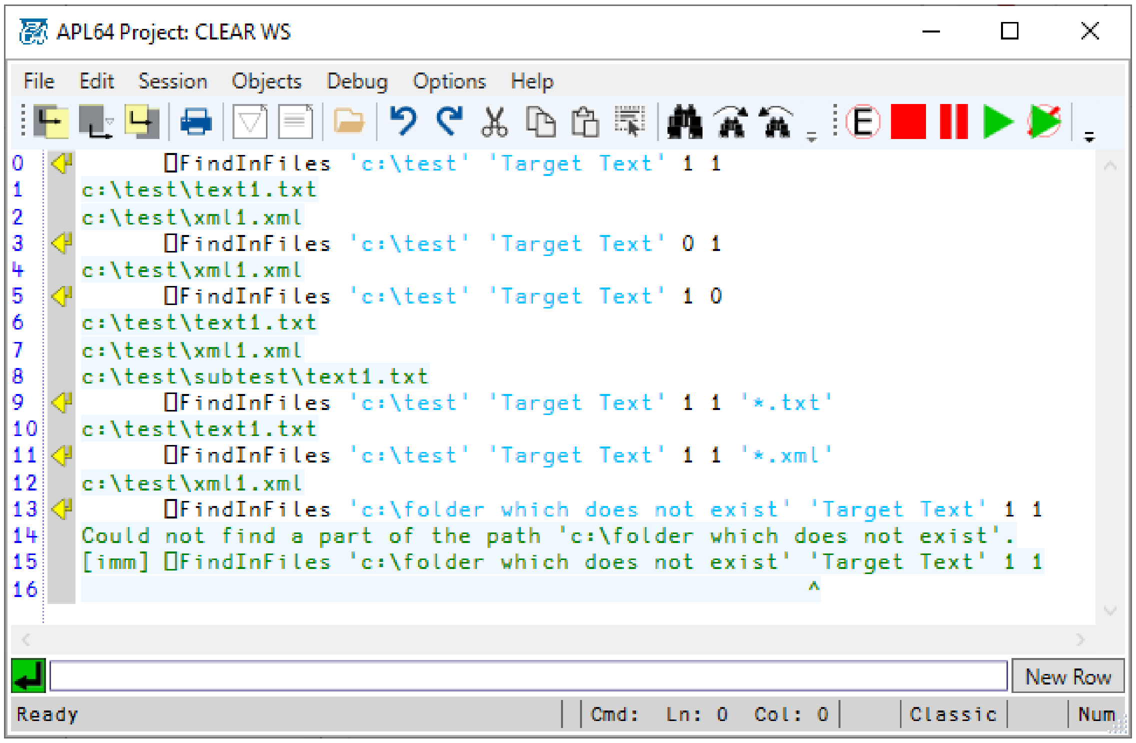
```
abcde
           target Text
pqr
```



```
abcde

pqr
```


C:\test\xml1.xml - Notepad++
File Edit Search View Encoding Language
Settings Tools Macro Run Plugins Window ?
APL64Settings_JWiedemann_Machine_3.xml
xml1
Ln : 1 C Windows (CR LF) UTF-8 INS

```
<TAG1>
   <TAG2>abcde</TAG2>
   <Tag3>Target Text</Tag3>
</TAG1>
```



```
      ⎕FindInFiles 'c:\test' 'Target Text' 1 1
c:\test\text1.txt
c:\test\xml1.xml
      ⎕FindInFiles 'c:\test' 'Target Text' 0 1
c:\test\xml1.xml
      ⎕FindInFiles 'c:\test' 'Target Text' 1 0
c:\test\text1.txt
c:\test\xml1.xml
c:\test\subtest\text1.txt
      ⎕FindInFiles 'c:\test' 'Target Text' 1 1 '*.txt'
c:\test\text1.txt
      ⎕FindInFiles 'c:\test' 'Target Text' 1 1 '*.xml'
c:\test\xml1.xml
      ⎕FindInFiles 'c:\folder which does not exist' 'Target Text' 1 1
Could not find a part of the path 'c:\folder which does not exist'.
[imm] ⎕FindInFiles 'c:\folder which does not exist' 'Target Text' 1 1
                                                             ^
```

⎕first Return First Item of Array

Purpose:
Returns the first item of an array.

Syntax: result ← ⎕first array

Argument:
array is any array.

Result:
The result is the first item of the argument. If the argument is not already a simple scalar, result is nested one level less deeply than the first element of the argument (array[1] or 1↑array).

Note: At Evolution Level 2, this function is associated with monadic up arrow (↑). At Evolution Level 1, this function is associated with disclose (⊃) used on arrays with more than one item. ⎕first returns the first item of an array regardless of Evolution Level.

Example:

```
   C←'ONE' (2 3 4 5)
   ⎕first C
ONE
   ρ ⎕first C
3
```

⎕fl Find Last Occurrence

Purpose:
Performs a right-to-left string search, locating the last occurrence or repeated occurrences of a character scalar or vector within another character vector or string.

Syntax: result ← target [[startOffset] endOffset] ⎕fl pattern [mode [repeat]]

Arguments:
target is a character vector or string to be searched in.

startOffset is an optional integer scalar in target at which the search is to begin (left end of the range).

endOffset is an optional integer scalar in target at which the search is to end (right end of the range). This will be the offset used when only one offset is specified. **Note**: endOffset is always 1 character past the end of the desired search range. Because the endOffset is specified as 1 past the end of the range, this means that the length of the range is always endOffset - startOffset for either ⎕io.

pattern is a character vector or scalar for the system to locate in target. If pattern is empty (''), result is (0 0) for the non-repeat mode and a shape (0 2) empty array for the repeat modes.

Mode is the search mode and indicates: 0=Literal (the default if omitted), 1= RegEx expression, or 2= Wildcard expression (may contains * and ? chars).

repeat is the repeat mode and indicates: 0=return only the first occurrence, 1=return all non-overlapping occurrences, 2=return all occurrences (including overlapping).

Effect:
The result is either a 2-element vector or a 2-column matrix representing the indices. Indexes are ⎕io dependent:

- It is a two-element vector if repeat=0 is specified or implied by omission (default). The 1st element is the offset where the match starts and the 2nd element is the offset where the match ends (this is the first character AFTER the last character of the match). If a match is not found the result depends upon ⎕io. For ⎕io=1 we return 0 0 but for ⎕io=0 we return ¯1 ¯1. The length of the match is given by indexes[2]-indexes[1].
- It is a two-column matrix if repeat≠0 is specified. In this case there is one row for each match that was found (zero rows if no matches were found). The 1st column is the offset where the match starts and the 2nd column is the offset where the match ends (this is the first character AFTER the last character of the match). The length of each match is given by indexes[row;2]-indexes[row;1].

Refer to ⎕ff for the counterpart to ⎕fl.

Examples (assuming **⎕io=1**):

```
      ⍝ Set up target string that will be searched
      s←'abababababababababa'
      s,[.5]⍳⍴s
a b a b a b a b a  b  a  b  a  b  a  b  a
```

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

      ⍝ Find last occurrence of ‘bab’ in target
      s ⎕fl 'bab'
14 17

      ⍝ Find reverse non-overlapping occurrences of ‘bab’ in target
      s ⎕fl 'bab' 0 1
 14 17
 10 13
  6  9
  2  5

      ⍝ Find reverse overlapping occurrences of ‘bab’ in target
      s ⎕fl 'bab' 0 2
 14 17
 12 15
 10 13
  8 11
  6  9
  4  7
  2  5
```

⎕flib Library List of Component Files

Purpose:
Produces a character matrix of component files with the .sf extension in a library or directory.

Syntax: result ← ⎕flib ''
result ← ⎕flib 'dir'
result ← ⎕flib lib

Arguments:
dir is a directory name;

lib is a library number.

The argument must be empty, a valid path/directory name, or a positive integer that has been associated with a directory by using ⎕libd or the Libraries choice on the Options menu. If the argument is empty, the system uses the current working directory.

Result:
The result of ⎕flib is an alphabetic (uppercase Z preceding lowercase a) listing of file names. If you supply a directory name (with or without the path), the result is a character matrix of file names, left-justified. The number of columns is the length of the longest file name in the list.

If the argument is a numeric library number, the result is a character matrix that contains one file identifier in each row. The first ten columns contain the right-justified library number with a space in the eleventh column; the remaining columns contain the left-justified file name.

)flib produces the same list of files formatted in multiple columns, without library numbers.

Examples:

```
     ⎕flib 'C:\APL\FILES'
CONVERT
DATEFILE
XFER
     ρ⎕flib 'C:\APL\FILES'
3 8
     ⎕libd '123 C:\APL\FILES'
     ⎕flib 123
      123 CONVERT
      123 DATEFILE
      123 XFER
     ρ⎕flib 123
3 22
```

Note: The system assumes the default extension (.SF) for component files. Thus, it returns only those files created with the traditional file functions and those that use the same naming convention. For new programming, the extended file functions, including ⎕xlib, provide more capability and flexibility.

⎕flock Extended Lock Component File

Purpose:
Lock or unlock a file.

Syntax: ⎕flock tieno operation position length passno

Arguments:
tieno is a positive integer file tie number; it must be an exclusive extended component tie;

operation is a Boolean value: 1 for locking the file, 0 for unlocking the file

position is a 64-bit integer specifying the beginning of the range to lock (the value of this argument must be equal to or greater than zero)

length is a 64-bit integer specifying the range to be locked (the value of this argument cannot be negative)
passno is an optional integer passnumber (If you omit the passnumber, the system assumes it is 0)

Effect:
Locking part of a file prevents other processes from reading from or writing to that part of the file. Unlocking a file allows access by other processes to all or part of a file that was previously locked.

Access:
The file must be exclusively tied, the passnumber must match the one in effect, and you must have lock access. The access code for ⎕flock is 131072.

Examples:

```
⎕flock 2 1 1024 1
⎕flock 2 0 1024 1
```

⎕fmt Format Output

Purpose:
Formats character and numeric data into a character matrix. For details on this system function, see the Formatting Output chapter in this manual.

Syntax:
```
result ← 'formatstring' ⎕fmt data
result ← 'formatstring' ⎕fmt (⊂data1),(⊂data2). . .⊂datan
result ← 'formatstring' ⎕fmt (data1;data2; . . .;datan)
```

Arguments:
data, data**i** are APL arrays; each one can be any non-nested numeric or character array. The right argument can be one array, a vector of enclosed arrays, or a string of arrays delimited by semicolons (see the caution below).

formatstring is a character vector that contains phrases to be applied to data, or data1, data2, and so on. These phrases, shown below, control how the system edits and displays the right argument.

Phrases:

rmAw	Character data field
rmEw.s	Exponential numeric data field
rmFw.d	Fixed point numeric data field
rmG<pattern>	Format data by supplied pattern
rmIw	Integer numeric data field
Tp or T	Absolute tab position for next field
rXp	Relative tab position for next field
r<text>	Insert supplied text
where	
d =	Number of positions after the decimal point (F)
m =	Optional format phrase modifier (see below)
p =	Position (column number) (T, X)
pattern =	Pattern of digit selectors and text characters (G)
r =	Optional repetition factor
s =	Number of significant digits (E)
w =	Field width (A, E, F, I)

You use commas to separate multiple format phrases for individual data columns within formatstring. You can repeat a group of format phrases by enclosing it in a pair of parentheses and preceding the left parenthesis with a repetition factor.

You can use one or a combination of the following modifiers with the phrases shown in parentheses.

Format Phrase Modifiers:

B	Blank if zero (F,G,I)
C	Comma insertion (F,I)

K **i**	Scale argument by 10***i** (E,F,G,I)
L	Left justify (F,I)
M<text>	Negative left decoration (F,G,I)
N<text>	Negative right decoration (F,G,I)
O<text>	Format zero as text (F,G,I)
O<value><text>	Format value as text (F,G,I)
P<text>	Positive or zero left decoration (F,G,I)
Q<text>	Positive or zero right decoration (F,G,I)
R<text>	Background fill (A,E,F,G,I)
S<symbolpairs>	Symbol substitution (F,G,I)
Z	Zero fill (F,I)

You can use any of the following pairs of symbols to delimit the text in the decorations, background fill, symbol substitution, and text insertion:

```
<   >
⊂   ⊃
¨   ¨
⎕   ⎕
⍞   ⍞
/   /
```

Result: The result is a character matrix of the data formatted as specified.

Caution:
The preferred format for the right argument is a vector of vectors. The semicolon-delimited list is included for historical compatibility. This format produces the same effect as the original list form except when all the arrays in the list are simple scalars of the same type. In this case the effect is a space -consuming row display rather than a column result. To achieve the column result, make one of the list items a vector.

Examples:

```
    'I3' ⎕fmt 1 2 3
  1
  2
  3
    'I3' ⎕fmt (1;2;,3)
  1  2  3

    'I5,2F8.1,E12.3' ⎕fmt 3 4ρι12
    1     2.0     3.0   4.00E0
    5     6.0     7.0   8.00E0
    9    10.0    11.0   1.20E0

    'G<(999) 999-9999>' ⎕fmt 3015645020
(301) 564-5020
```

```
    FSTR←'3A1,<+2000 >,6A1'
    FSTR ⎕fmt 1 9ρ'APLSYSTEM'
APL+2000 SYSTEM
```

⎕fn Function name

Purpose:
Returns the name of the currently executing function.

Syntax: result ← ⎕fn

Domain:
The result is a character vector containing the name of the function currently executing in a function or an empty character vector when called from immediate execution.

⎕fnames Names of Tied Component Files

Purpose:
Returns the file identifiers of component files tied with the traditional functions ⎕ftie or ⎕fstie. (See also the extended file function ⎕xfnames.)

Syntax: result ← ⎕fnames

Result:
The result is a character matrix of file identifiers. The rows of result have the same order as ⎕fnums. ⎕fnames formats result to be as wide as needed; it contains the full path and file names, unless you have defined libraries. With a library number, the first ten columns contain the right-justified library number with a space in the eleventh column; the remaining columns contain the left-justified file name.

Example:

```
    ⎕fnames
C:\APL\WSS\CHAPTER1
A:\TEMP
     101 TEMP2
```

Note: When using the traditional file functions, you are limited in names you can use. For new programming, the extended file functions provide more capability and flexibility.

⎕fnedhist Function Editing History

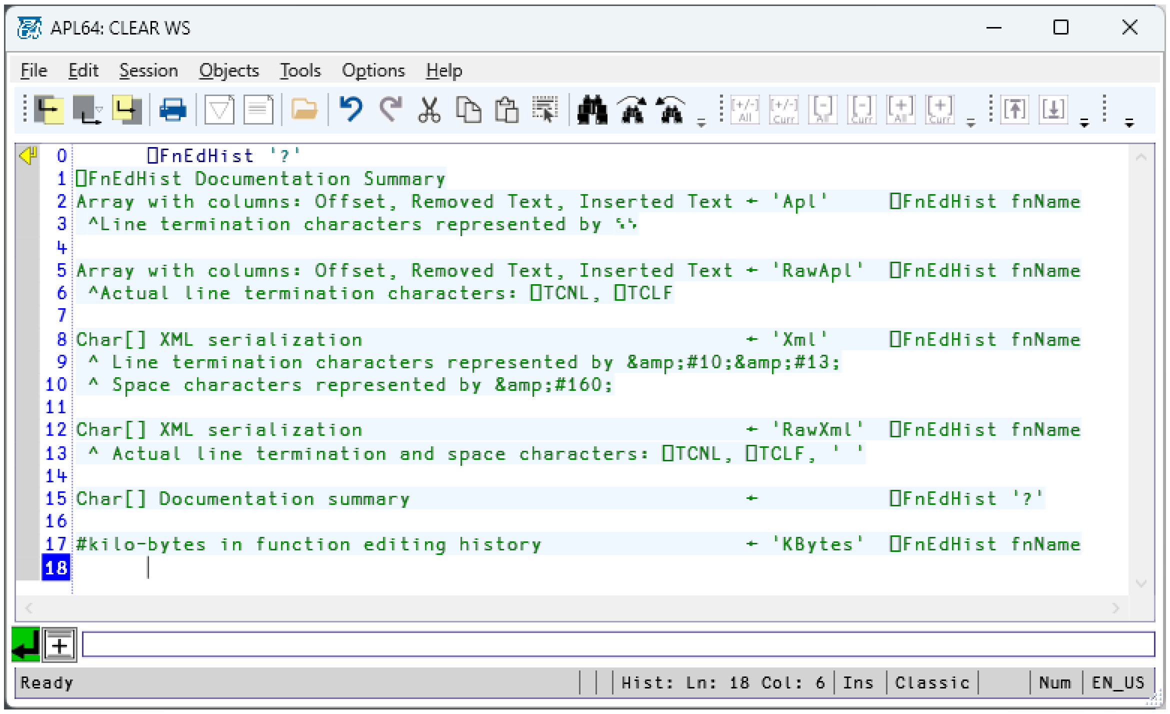
Purpose:
Displays the function editing history.

Syntax:

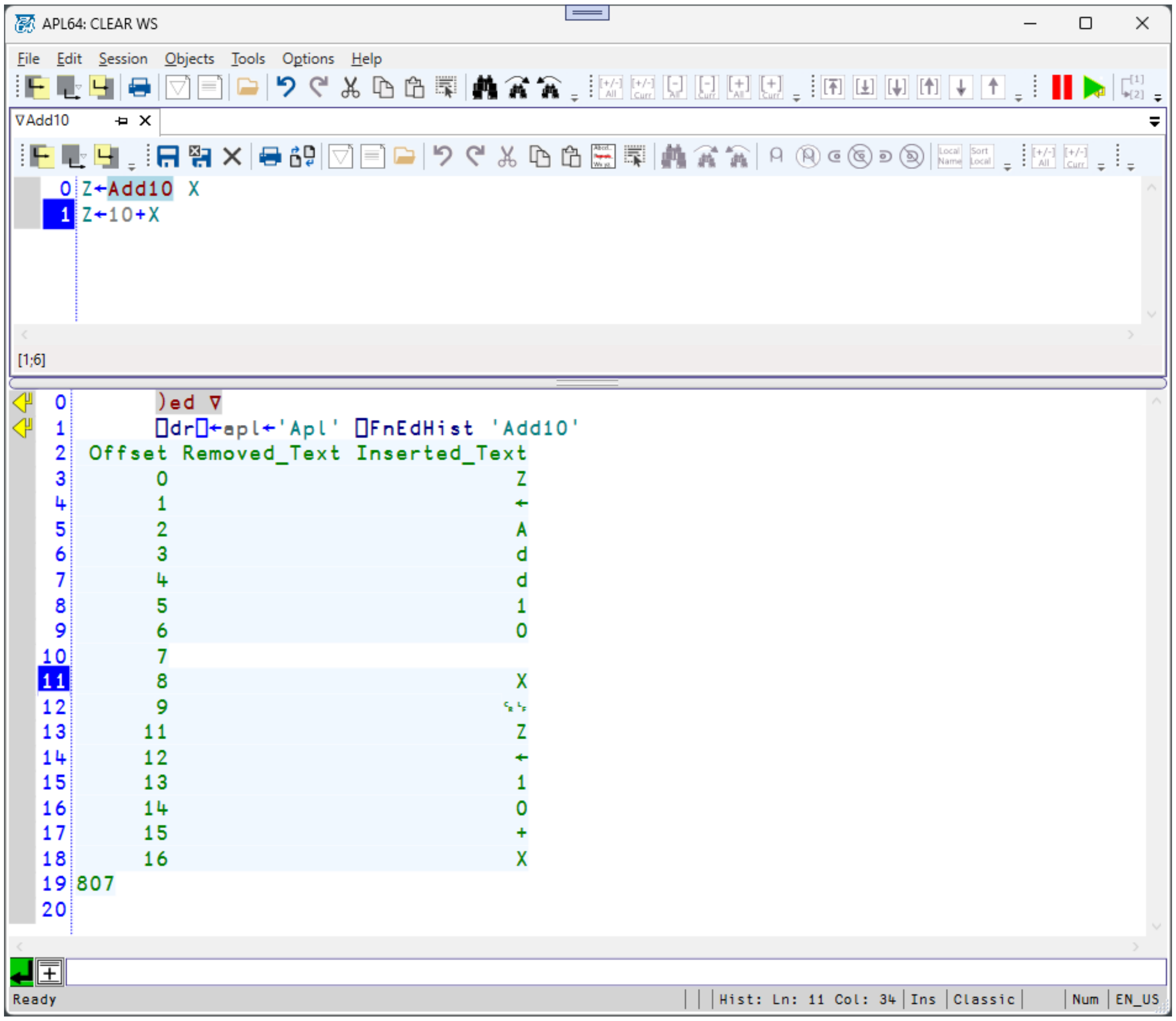
Array with columns: Offset, Removed Text, Inserted Text	← 'APL'	⎕FnEdHist fnName
Char[] XML serialization	← 'XML'	⎕FnEdHist fnName
Char[] Documentation summary	←	⎕FnEdHist '? '
Char[] Documentation summary	←	⎕FnEdHist 'Help'
Double #kilobytes in the function editing history	← 'KBytes'	⎕FnEdHist fnName

Effect:
Function editing history can be captured as an APL nested array or an XML document using the ⎕FnEdHist system function. Refer to Function Editor Compare Format & Editing History for additional information.

Examples:



```
      ⎕FnEdHist '?'
⎕FnEdHist Documentation Summary
Array with columns: Offset, Removed Text, Inserted Text ← 'Apl'     ⎕FnEdHist fnName
 ^Line termination characters represented by ␊␍

Array with columns: Offset, Removed Text, Inserted Text ← 'RawApl'  ⎕FnEdHist fnName
 ^Actual line termination characters: ⎕TCNL, ⎕TCLF

Char[] XML serialization                                ← 'Xml'     ⎕FnEdHist fnName
 ^ Line termination characters represented by &amp;#10;&amp;#13;
 ^ Space characters represented by &amp;#160;

Char[] XML serialization                                ← 'RawXml'  ⎕FnEdHist fnName
 ^ Actual line termination and space characters: ⎕TCNL, ⎕TCLF, ' '

Char[] Documentation summary                            ←           ⎕FnEdHist '?'

#kilo-bytes in function editing history                 ← 'KBytes'  ⎕FnEdHist fnName
```



APL64: CLEAR WS

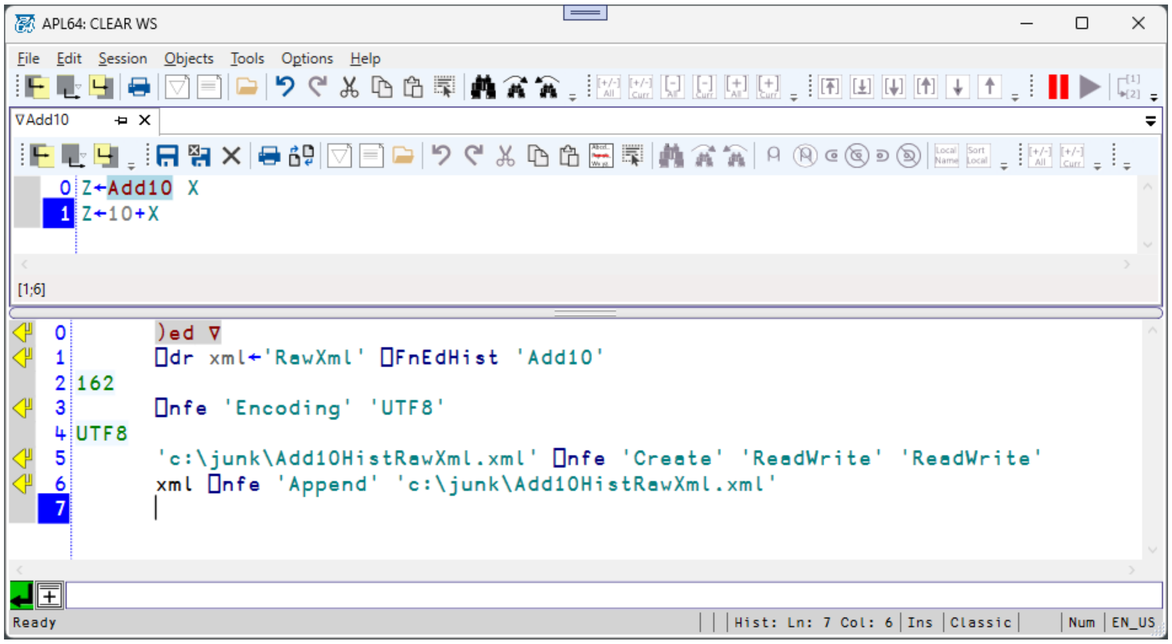
∇Add10

```
Z←Add10 X
Z←10+X
```

[1;6]

```
      )ed ∇
      ⎕dr⎕←apl←'Apl' ⎕FnEdHist 'Add10'
 Offset Removed_Text Inserted_Text
      0                          Z
      1                          ←
      2                          A
      3                          d
      4                          d
      5                          1
      6                          0
      7
      8                          X
      9                         ␍␊
     11                          Z
     12                          ←
     13                          1
     14                          0
     15                          +
     16                          X
807
```

Ready | Hist: Ln: 11 Col: 34 | Ins | Classic | Num | EN_US



APL64: CLEAR WS

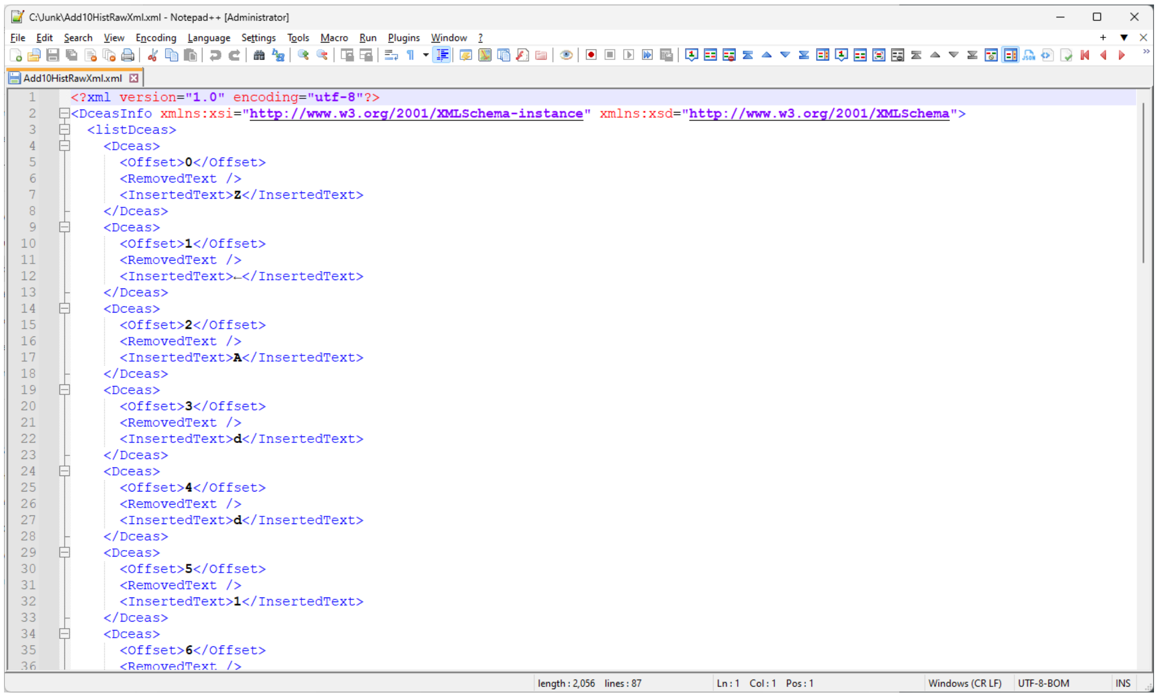
∇Add10

```
Z←Add10 X
Z←10+X
```

[1;6]

```
      )ed ∇
      ⎕dr xml←'RawXml' ⎕FnEdHist 'Add10'
162
      ⎕nfe 'Encoding' 'UTF8'
UTF8
      'c:\junk\Add10HistRawXml.xml' ⎕nfe 'Create' 'ReadWrite' 'ReadWrite'
      xml ⎕nfe 'Append' 'c:\junk\Add10HistRawXml.xml'
```

Ready | Hist: Ln: 7 Col: 6 | Ins | Classic | Num | EN_US


C:\Junk\Add10HistRawXml.xml - Notepad++ [Administrator]

```
<?xml version="1.0" encoding="utf-8"?>
<DceasInfo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <listDceas>
    <Dceas>
      <Offset>0</Offset>
      <RemovedText />
      <InsertedText>Z</InsertedText>
    </Dceas>
    <Dceas>
      <Offset>1</Offset>
      <RemovedText />
      <InsertedText>←</InsertedText>
    </Dceas>
    <Dceas>
      <Offset>2</Offset>
      <RemovedText />
      <InsertedText>A</InsertedText>
    </Dceas>
    <Dceas>
      <Offset>3</Offset>
      <RemovedText />
      <InsertedText>d</InsertedText>
    </Dceas>
    <Dceas>
      <Offset>4</Offset>
      <RemovedText />
      <InsertedText>d</InsertedText>
    </Dceas>
    <Dceas>
      <Offset>5</Offset>
      <RemovedText />
      <InsertedText>1</InsertedText>
    </Dceas>
    <Dceas>
      <Offset>6</Offset>
      <RemovedText />
```

length : 2,056 lines : 87 Ln : 1 Col : 1 Pos : 1 Windows (CR LF) UTF-8-BOM INS

⎕fnums Tie Numbers of Tied Component Files

Purpose:
Displays the tie numbers of component files tied with ⎕ftie or ⎕fstie.
(See also the extended file function ⎕xfnums.)

Syntax: result ← ⎕fnums

Result:
The result is a numeric vector of file tie numbers. The tie numbers are in the same order as the file names that ⎕fnames reports; that is, the order in which they were tied.

Examples:

```
      ⎕fnums
27 33 17

     'I5,⊂ - ⊃,22A1' ⎕fmt ⎕fnums ⎕fnames
   27 - C:\APL\WSS\CHAPTER1
   33 - A:\TEMP
   17 -       101 TEMP2
```

Note: When using the traditional file functions, you are limited in names you can use. For new programming, the extended file functions provide more capability and flexibility.

⎕frdac File Read of Access Matrix

Purpose:
Reports the current access matrix for an APL component file.

Syntax: result ← ⎕frdac tieno
result ← ⎕frdac tieno passno

Arguments:
tieno is a positive integer file tie number;

passno is an integer passnumber.

The argument must be an integer scalar or one- or two-element, integer vector that defines the file for which you want the access matrix. It contains a valid positive tie number and optional valid passnumber. If you omit the passnumber, the system assumes it is 0.

Result:
The result is a three-column numeric matrix that contains the file's access matrix. A newly created file has an access matrix with no rows. For an explanation of the access matrix and access codes, see the Using Files manual.

Access:
The file must be tied, the passnumber must match the one in effect, and you must have authority to read the access matrix. The access code for ⎕frdac is 4096.

Examples:

```
     ⎕frdac 33 7655
12304  16059  7566
23405  16063     0
A file with an empty access matrix.
     ρ⎕frdac 27
0 3
```

⎕frdci File Read of Component Information

Purpose:
Returns information about one component of a file.

Syntax: result ← ⎕frdci tieno compno
result ← ⎕frdci tieno compno [passno]

Arguments:
tieno is a positive integer file tie number;

compno is an integer component number;

passno is an integer passnumber.

The argument must be a two- or three-element, integer vector that identifies the component for which you want information. It contains a valid positive tie number, a valid positive component number, and optional valid passnumber. If you omit the passnumber, the system assumes it is 0.

Result:
The result is a 10-element numeric vector:

- The first element is the workspace storage needed to hold the component's value, in bytes.
- The second element is the user number of the person who last replaced or appended the component.
- The third element is the time that the component was last replaced or appended, in a packed-timestamp form given in microseconds since 00:00, 1 January 1900.
- The remainder is the component timestamp in a seven-element unpacked form (year, month, date, hour, minute, second, millisecond). The microsecond resolution is maintained for compatibility with other APL systems. The clock accuracy, however, is one millisecond.

Access:
The file must be tied, the passnumber must match the one in effect, and you must have the authority to read component information. The access code for ⎕frdci is 512.

Example:

```
      ⎕frdci 27 1
3712 1 2.783963338E15 1988 3 21 19 8 58 0
```

⎕fread Read from a Component File

Purpose:
Reads a component of a file and makes it available in the workspace as a variable.

Syntax: result ← ⎕fread tieno compno
result ← ⎕fread tieno compno passno

Arguments:
tieno is a positive integer file tie number;

compno is an integer component number;

passno is an integer passnumber.

The argument must be a two- or three-element, integer vector that identifies the component you want to make available. It contains a valid positive tie number, a valid positive component number, and optional valid passnumber. If you omit the passnumber, the system assumes it is 0.

Result:
The result is the actual value stored in the file component.

Access:
The file must be tied, the passnumber must match the one in effect, and compno must be a valid component number. The access code for ⎕fread is 1.

Examples:

```
    ⎕fread 27 1
THIS FILE CONTAINS DATA FOR 1987
CREATED 26 JANUARY 1988.
    ⎕fread 27 2
SMALLS, BARRY T. 4856739 6/30/85
    A←⎕fread 27 3
    A
SMITH, KAREN M. 3847384 3/01/86
```

⎕frename Rename Component File

Purpose:
Changes the name and sets the size limit of a component file.
(See also the extended file function ⎕xfrename, which supersedes the naming capability of this function.)

Syntax:
```
'fileid'        ⎕frename tieno
'fileid size'   ⎕frename tieno
'fileid'        ⎕frename tieno passno
'fileid size'   ⎕frename tieno passno
```

Arguments:
fileid is a file identifier

size is an integer that specifies the file size limit in bytes;

tieno is a positive integer file tie number;

passno is an integer passnumber.

The left argument, a character scalar or vector, designates the new file identifier and, optionally, a new size limit. The new file name must not already exist in the directory. You can specify the fileid in either path or library form. If you omit both the directory and library number, the system uses the default directory.

The right argument must be an integer scalar or one- or two-element, integer vector that defines the file you want to rename. It contains a valid positive tie number and optional valid passnumber. If you omit the passnumber, the system assumes it is 0.

Effect:
⎕frename changes the file name to the one specified in the left argument, potentially moving it to a different directory on the same disk drive. If the file name already exists, the system signals a FILE NAME ERROR. The result of ⎕fnames reflects the new file identifier. The user who renames the file becomes the new file owner.

Access:
The file must be exclusively tied, the passnumber must match the one in effect, and you must have rename access. The access code for ⎕frename is 128.

Examples:

```
      ⎕flib ''
PRIMES
      'PRIMES' ⎕ftie 10
      'PRIMENUM' ⎕frename 10
      ⎕flib ''
PRIMENUM
      'NEWNAME' ⎕frename 10
      ⎕libd '101 C:\APL\FILES'
'101 NEWNAME' ⎕frename 10
```

Note: The system appends the default extension (.SF) to the fileid you provide. Using the traditional file functions, you are limited in names you can use. For new programming, the extended file functions provide more capability and flexibility.

⎕freplace Replace Component

Purpose:
Changes the value of an existing component of a file.

Syntax: `value ⎕freplace tieno compno`
`value ⎕freplace tieno compno [passno]`

Arguments:
value is the value you want to store; it can be any rank, shape, or datatype;

tieno is a positive integer file tie number;

compno is an integer component number;

passno is an integer passnumber.

The right argument must be a two- or three-element, integer vector that identifies the component you want to replace. It contains a valid positive tie number, a valid positive component number, and optional valid passnumber. If you omit the passnumber, the system assumes it is 0.

Effect:
Replaces the designated component of the file with a new value. It also updates the component information (⎕frdci) and file history (⎕fhist).

Replacing a component may change the file size. If the new component is larger than the one it replaces, the new component may be written at the physical end of the file, leaving the previous size of the component as wasted space. This is known as an exploding ⎕freplace; the amount of wasted space is reflected in the last element of ⎕fsize. You can recover this space by using ⎕fdup.

Access:
The file must be tied, the passnumber must match the one in effect, and you must have append access. The access code for ⎕freplace is 16.

Example:

```
    SpeedDial←⎕fread 33 10
    SpeedDial←SpeedDial,[1] Name extension
    SpeedDial ⎕freplace 33 10
```

⎕fresize Resize Component File (Limit)

Purpose:
Resets the file size limit of a component file.

Syntax: size ⎕fresize tieno
              size ⎕fresize tieno [passno]

Arguments:
size specifies the file size limit in bytes; it must be less than 2*32 (4 gigabytes);

tieno is a positive integer file tie number;

passno is an integer passnumber.

The size is the new file size limit in bytes. It must be a positive integer-valued scalar or one-element vector greater than or equal to the current size of the file. It can also be zero, meaning that the file has no size limit.

The right argument must be an integer scalar or one- or two-element, integer vector that defines the file you want to resize. It contains a valid positive tie number and optional valid passnumber. If you omit the passnumber, the system assumes it is 0.

Effect:
Changes the file size limit to the specified value. If size is zero (the default for a new file), the file has no size limit; it can grow as large as needed, up to the maximum allowable size for component files.

Access:
The file must be tied, the passnumber must match the one in effect, and the user must have resize access. The access code for ⎕fresize is 1024.

Example:

```
    ⎕fsize 27
1 50 94560 100000 0
    2600000 ⎕fresize 27
    ⎕fsize 27
1 50 94560 2600000 0
```

⎕fsize Component File Size Information

Purpose:
Returns the size limits of a component file.

Syntax: result ← ⎕fsize tieno
              result ← ⎕fsize tieno [passno]

Arguments:
tieno is a positive integer file tie number;

passno is an integer passnumber.

The argument must be an integer scalar or one- or two-element, integer vector that defines the file for which you want size information. It contains a valid positive tie number and optional valid passnumber. If you omit the passnumber, the system assumes it is 0.

Result:
The result is a five-element numeric vector that contains the information shown below.

Element	**Description**
[1]	The number of the first component in the file.
[2]	The number of the next available component.
[3]	The physical storage (in bytes) that the file uses, including data, overhead, and access matrix
[4]	The size limit for the file as set by the user (a value of zero means no upper limit)
[5]	Estimated number of bytes that a file compaction can recover

Caution: ⎕fsize, as described here, is specific to this APL64 system; in particular, the elements dealing with file size may be floating point numbers, which in other APL systems were integers.

Examples:

```
      'PRIMES' ⎕fstie 37
      ⎕fsize 37
7 53 28672 100000 9648

      'NEWFILE' ⎕fcreate 13
      ⎕fsize 13
1 1 1056 0 0
```

⎕fstac Set Access Information of Component File

Purpose:
Sets the access matrix of a component file.

Syntax:
```
access ⎕fstac tieno
access ⎕fstac tieno [passno]
```

Arguments:
access is a three-column integer matrix or a three-element vector containing the values of an access matrix: user number, operations allowed; passnumber (see the Using Files manual for details).

tieno is a positive integer file tie number;

passno is an integer passnumber.

The right argument must be an integer scalar or one- or two-element, integer vector that defines the file for which you want to set the access matrix. It contains a valid positive tie number and optional valid passnumber. If you omit the passnumber, the system assumes it is 0.

Effect:
Replaces the access matrix for the file. For an explanation of the access matrix and access codes, see the Using Files manual. The system imposes the new access restrictions on any user the next time that user ties the file. ⎕fstac may increase the amount of disk storage that the file occupies.

Access:
The file must be tied, the passnumber must match the one in effect, and the user must have the authority to change the access matrix. The access code for ⎕fstac is 8192.

Example:

```
    MAT ←2 3ρ4772490 2 666 1000 ¯1 0
    MAT ⎕fstac 33
    ⎕frdac 33
4772490  2 666
  1000 ¯1   0
```

⎕fstie Share Tie a Component File

Purpose:
Ties a component file for shared use.
(See also the extended file function ⎕xfstie, which supersedes this function.)

Syntax: 'fileid' ⎕fstie tieno
'fileid' ⎕fstie tieno [passno]

Arguments:
fileid is a file identifier;

tieno is a positive integer file tie number;

passno is an integer passnumber.

The fileid must be a character vector or singleton that contains the file identifier of an existing file. If you do not specify the directory or library number, the system uses the default directory. You can tie a file that has a tilde character (~) in the traditional representation (8.3) of a longer file name.

The right argument must be an integer scalar or one- or two-element, integer vector that specifies the number by which you will define the file to other functions. It contains a valid positive tie number and optional valid passnumber. If you omit the passnumber, the system assumes it is 0.

Effect:
The file is share tied; more than one user can share-tie a file simultaneously. File ties are "slippery;" that is, if a file is already tied to one tie number, you can tie that file to the same number or to another unused tie number without untying the file.

Access:
The file must exist and must not be exclusively tied (⎕ftie or ⎕xftie) by anyone, although it can be share-tied by others. The user must have some form of access to the file, and the passnumber must match an appropriate one in the access matrix.

Examples:

```
    'PRIMES' ⎕fstie 37
    'C:\APLW\FILES\MYFILE' ⎕fstie 22
    ⎕libd '12345 C:\APLW\WSS'
    '12345 PRINTOUT' ⎕fstie 1 666
```

Note: The system assumes the default extension (.SF) for the fileid you provide. Using the traditional file functions, you are limited in names you can use. For new programming, the extended file functions provide more capability and flexibility. Purpose: Ties a component file for shared use.
(See also the extended file function ⎕xfstie, which supersedes this function.)

⎕ftie Tie a Component File (Exclusive)

Purpose:
Ties a component file for exclusive (non-shared) use.
(See also the extended file function ⎕xftie, which supersedes this function.)

Syntax:
```
'fileid' ⎕ftie tieno
'fileid' ⎕ftie tieno [passno]
         ⎕ftie tieno
```

Arguments:
fileid is a file identifier;

tieno is an available positive integer file tie number;

passno is an integer passnumber.

The fileid must be a character vector or singleton that contains the file identifier of an existing file. If you do not specify the directory name or library number, the system uses the default directory. You can tie a file that has a tilde character (~) in the traditional representation (8.3) of a longer file name.

The right argument must be an integer scalar or one- or two-element, integer vector that specifies the number by which you will define the file to other functions. It contains a valid positive tie number and optional valid passnumber. If you omit the passnumber, the system assumes it is 0.

The optional left argument must be a valid component file identifier.

Effect:
The file is exclusively tied. No other user can tie the file as long as it remains exclusively tied. File ties are "slippery;" that is, if a file is already tied to one tie number, you can tie that file to the same number or to another unused tie number without first untying the file.

Monadic ⎕ftie takes a tie number as the right argument and returns a two-element integer vector indicating the file tie state. The value of the first element is the same as the open mode arguments to the dyadic version of ⎕ntie and indicates what the current open mode of the file is. The open mode values are sums of the following:

Access Requested	**Access granted to other users**
0 = read access	16 = no access allowed (exclusive)
1 = write access	32 = read access allowed, write access denied
2 = read and write access	48 = write access allowed, read access denied
	64 = read and write access allowed

The values are dependent on whether the tie request was for a shared or exclusive tie and whether read or write access is granted.

The second element indicates if the tie was share tie or an exclusive tie. Share ties show a value of 0, exclusive ties show a value of 1.

Access:
The file must exist, it must not be tied by anyone else, the user must have the authority to exclusively tie the file, and the passnumber must match an appropriate one in the access matrix. The access code for ⎕ftie is 2.

Examples:

```
'PRIMES' ⎕ftie 37
'C:\APLW\FILES\MYFILE' ⎕ftie 2
⎕libd '12345 C:\APLW\FILES'
'12345 MYFILE' ⎕ftie 1
```

Note: The system assumes the default extension (.SF) for the fileid you provide. Using the traditional file functions, you are limited in names you can use. For new programming, the extended file functions provide more capability and flexibility.

⎕funtie Untie Component Files

Purpose:
Unties one or more component files.

Syntax: ⎕funtie tieno1 tieno2 tieno3 . . .tieno**n**

Argument:
tieno1 tieno2 tieno3 . . . tieno**n** are file tie numbers of files you want to untie.

The argument is an integer scalar or vector of possible file tie numbers. Elements of the argument need not be in use as file tie numbers. An empty vector is permitted as an argument and does not affect any file ties.

Effect:
The files tied to any of the tie numbers in the argument are untied. This frees the file tie number for reuse with another file. The system releases any file holds in effect for the referenced files. This function works on files tied with ⎕xftie and ⎕xfstie as well as ⎕ftie and ⎕fstie.

Examples:
The second example unties all tied component files.

```
     ⎕funtie 33

     ⎕funtie (⎕fnums,⎕xfnums)
     ρ⎕xfnums
 0
```

⎕fx Function Fix

Purpose:
Defines (fixes) a function from a character matrix (canonical) representation of the function (see the description of ⎕cr in this chapter).

Syntax: result ← ⎕fx fnrep

Argument:
fnrep is the character matrix or a nested vector of character vectors (or scalars) that contains the canonical representation of a function (the result of ⎕cr). The lines of the matrix should not contain bracketed line numbers, nor should they contain del (∇) or del-tilde (⍫) other than in comments or character constants. ⎕fx ignores trailing blanks in fnrep.

Result:
If the function definition is successful, result is a character vector that contains the name of the function defined. If the function definition is not successful because of a poorly formed argument, result is a numeric scalar that contains the row index of the argument matrix where the system found the first fault. This value depends on the index origin (⎕io).

Effect:
Defines the specified function in the active workspace unless an error condition occurs. The amount of available workspace and the number of entries in the symbol table may change.

If the name of the newly defined function corresponds to a local identifier in a currently executing, pendent, or suspended function, the newly defined function is local to that function and the system erases the newly defined function when the function in which it is localized completes execution.

If the name of the newly defined function corresponds to the name of an existing function, the system replaces the existing function and removes any ⎕stop or ⎕trace settings in the function.

Note: ⎕def and ⎕fx provide similar capabilities. ⎕def is a more powerful and general tool than ⎕fx. See the description of ⎕def in this chapter for the differences.

Example:

```
     ⎕fx (3 8ρ'res←testA←1+s10 res←2+A ')
test
     ⎕vr 'test'
   ∇ res←test
[1]   A←1+s10
[2]   res←2+A
   ∇
```

```
    test
4 5 6 7 8 9 10 11 12 13

    ⎕fx 'res←test' 'A←1+s10' 'res←2+A'
test
    ⎕vr 'test'
  ∇ res←test
[1]   A←1+s10
[2]   res←2+A
  ∇

    test
4 5 6 7 8 9 10 11 12 13
```

⎕guid Global Unique Identifier Generator

Purpose:
Returns a character vector containing a new Global Unique Identifier (GUID).

Syntax: result ← ⎕guid

Result:
result is a character vector, generated by the Windows operating system, which is enclosed in 'curly braces'. The representation is of a 128-bit integer in the format of 32 hexadecimal digits grouped according to convention.

Effect:
The ⎕guid system function is used to obtain a character vector containing a new Global Unique Identifier (GUID). The result may be used as a persistent unique identifier with a high degree of certainty. For more information see http://en.wikipedia.org/wiki/Globally_unique_identifier.

Example:

```
    )clear
CLEAR WS
    ⎕guid
{A3EE881B-7AB4-4B38-9237-A2846E43EF9A}
```

⎕gvte Get Value Tip

Purpose:
Returns a value tip for an object in the workspace.

Syntax: result ←   ⎕gvte objectName
              result ← [scope]   ⎕gvte objectName

Arguments:
scope is an optional numeric argument that specifies the scope at which names are interpreted, where 1 = local (default); and 0 = global.
objectName is the name of the function, character variable, or numeric variable for which the value tip

which will be returned. If no object with the specified name exists or object is an empty character vector, ⎕gvte returns an empty character vector.

Results:
result is a two-element vector

Element #	Description
1	Object type
2	Object name
3	Value tip text

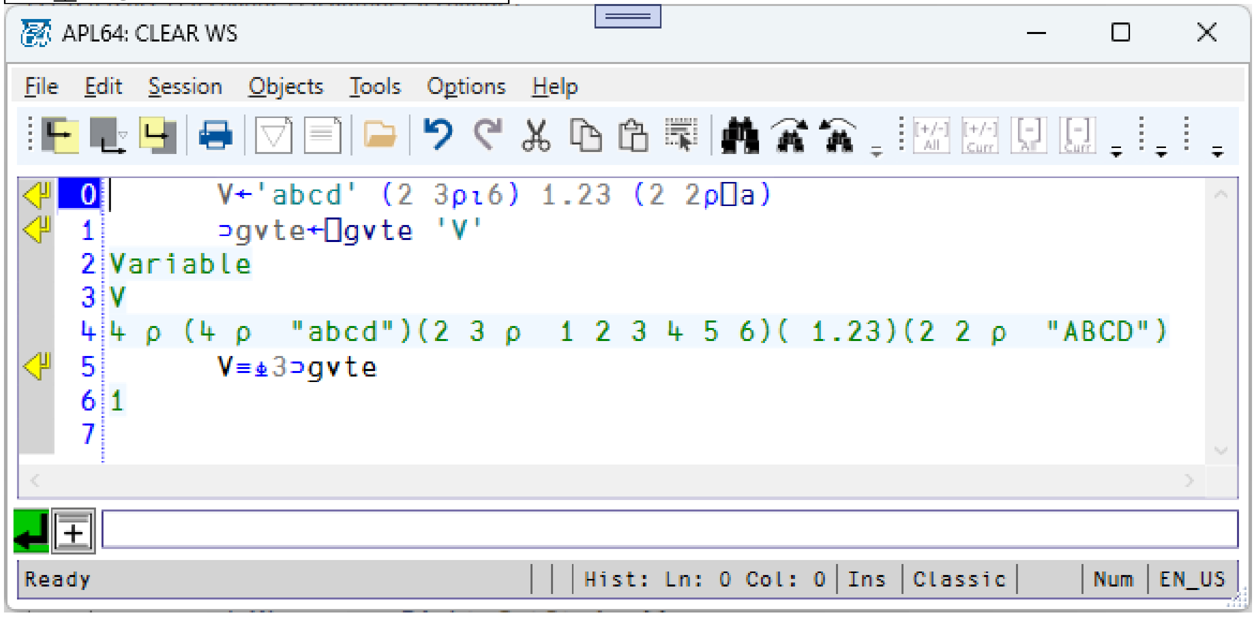
Effect:
The 'Value tip text' for an APL64 system variable or programmer-defined variable is a character vector which when executed will return the value of the variable.

The 'Value tip text' for all other APL64 object types is the value tip text produced by the APL64 Developer version when the mouse pointer is hovered over that object.

Examples:

Example 1:

```
V←'abcd' (2 3ρι6) 1.23 (2 2ρ⎕a)
⊃gvte←⎕gvte 'V'
V≡⍎3⊃gvte
```

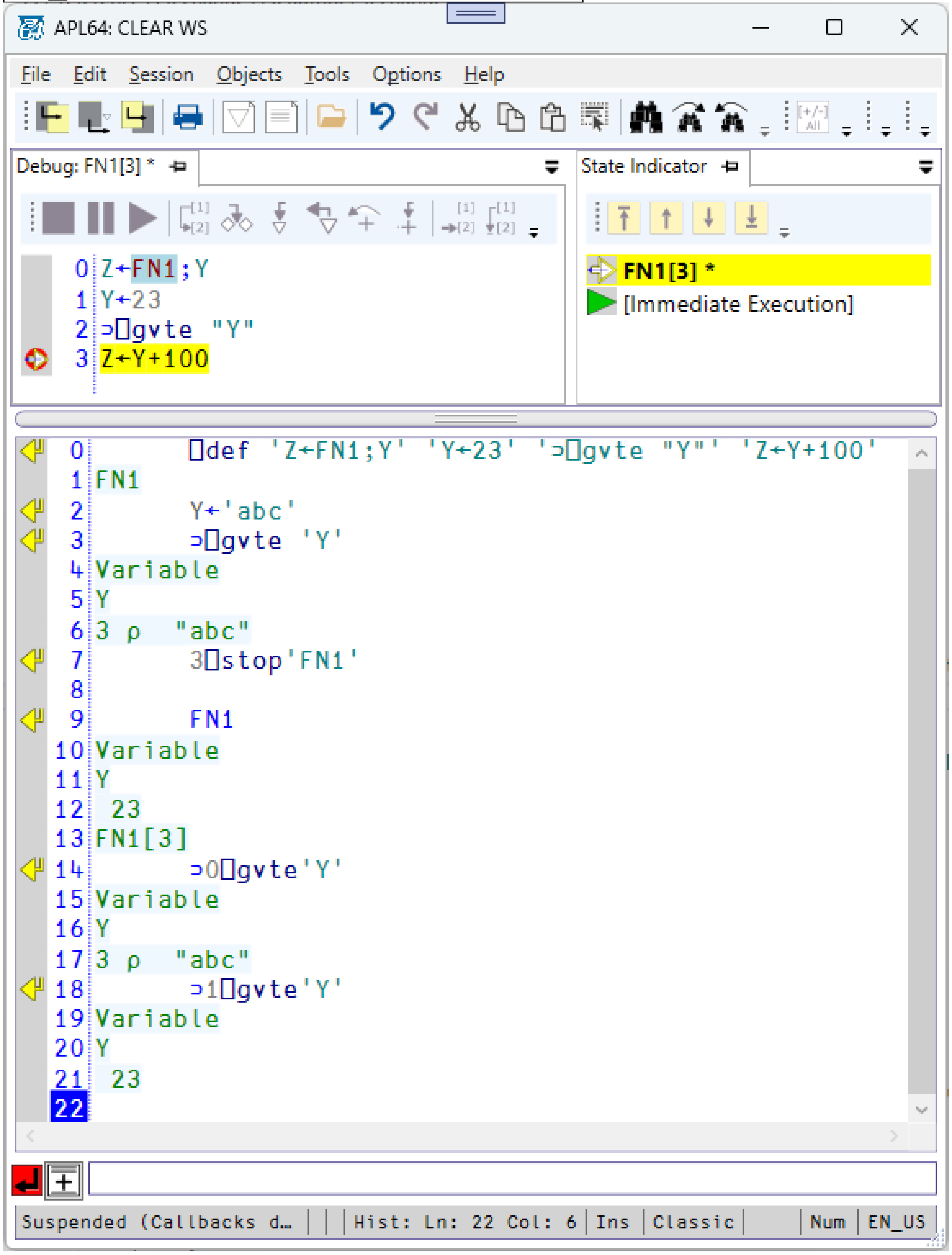


```
      V←'abcd' (2 3ρι6) 1.23 (2 2ρ⎕a)
      ⊃gvte←⎕gvte 'V'
Variable
V
4 ρ (4 ρ  "abcd")(2 3 ρ  1 2 3 4 5 6)( 1.23)(2 2 ρ  "ABCD")
      V≡⍎3⊃gvte
1
```

Example 2:

```
⎕def 'Z←FN1;Y' 'Y←23' '⊃⎕gvte "Y"' 'Z←Y+100'
Y←'abc'
⊃⎕gvte 'Y'
```

```
3⎕stop'FN1'
FN1
⊃0⎕gvte'Y'
⊃1⎕gvte'Y'
```



APL64: CLEAR WS
File Edit Session Objects Tools Options Help
Debug: FN1[3] *
State Indicator
FN1[3] *
[Immediate Execution]

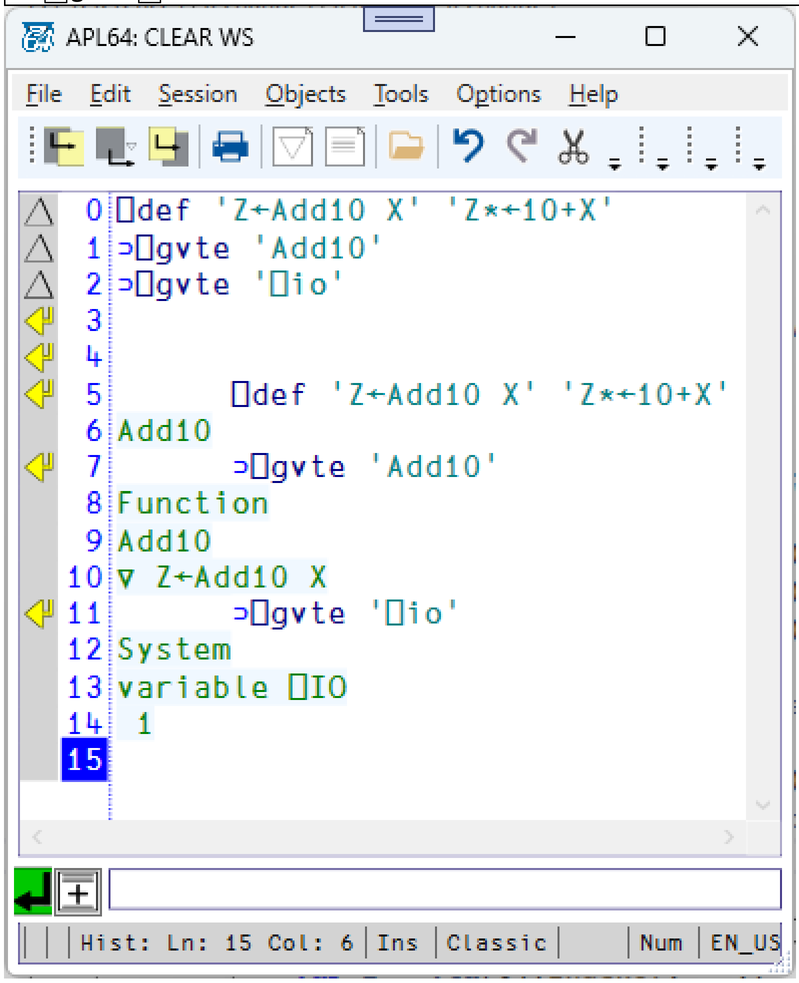
```
Z←FN1;Y
Y←23
⊃⎕gvte "Y"
Z←Y+100
```

```
      ⎕def 'Z←FN1;Y' 'Y←23' '⊃⎕gvte "Y"' 'Z←Y+100'
FN1
      Y←'abc'
      ⊃⎕gvte 'Y'
Variable
Y
3 ρ  "abc"
      3⎕stop'FN1'

      FN1
Variable
Y
 23
FN1[3]
      ⊃0⎕gvte'Y'
Variable
Y
3 ρ  "abc"
      ⊃1⎕gvte'Y'
Variable
Y
 23
```

Suspended (Callbacks d… | Hist: Ln: 22 Col: 6 | Ins | Classic | Num | EN_US

Example 3:

```
⎕def 'Z←Add10 X' 'Z*←10+X'
⊃⎕gvte 'Add10'
⊃⎕gvte '⎕io'
```

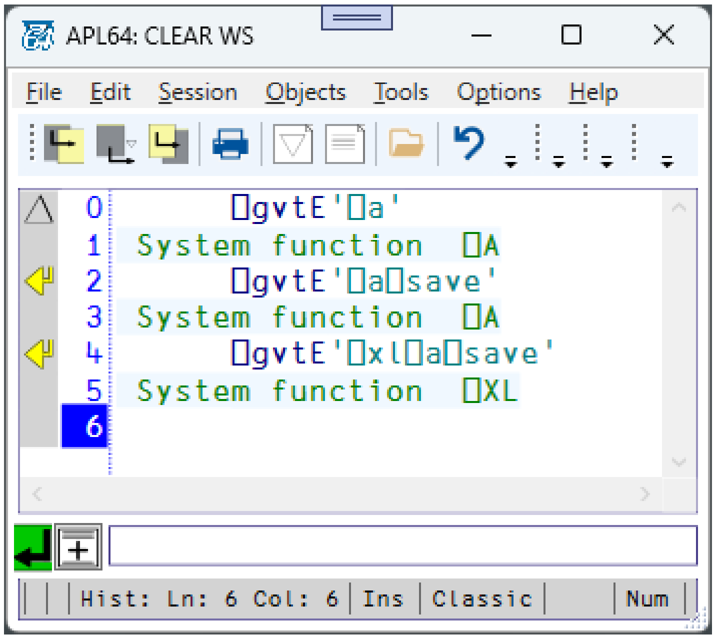

APL64: CLEAR WS
File Edit Session Objects Tools Options Help
Hist: Ln: 15 Col: 6 Ins Classic Num EN_US

```
⎕def 'Z←Add10 X' 'Z*←10+X'
⊃⎕gvte 'Add10'
⊃⎕gvte '⎕io'


      ⎕def 'Z←Add10 X' 'Z*←10+X'
Add10
      ⊃⎕gvte 'Add10'
Function
Add10
∇ Z←Add10 X
      ⊃⎕gvte '⎕io'
System
variable ⎕IO
 1
```

Example 4:

If multiple object names are included in the right argument of ⎕gvte, the APL64 interpreter will identify the left-most name as target object name.

```
⎕gvte'⎕a'
⎕gvte'⎕a⎕save'
⎕gvte'⎕xl⎕a⎕save'
```



```
      ⎕gvtE'⎕a'
System function  ⎕A
      ⎕gvtE'⎕a⎕save'
System function  ⎕A
      ⎕gvtE'⎕xl⎕a⎕save'
System function  ⎕XL
```

```
Hist: Ln: 6 Col: 6 | Ins | Classic | | Num
```

⎕idlist Identifier List

Purpose:
Returns a character matrix of the names of functions, variables and/or labels in the active workspace.

Syntax: result ← ⎕idlist class
result ← [letters] ⎕idlist class
result ← [scope ⎕idlist class

Arguments:
class is the type of identifier, where 1 = functions; 2 = variables; and 8 = labels. You can use the sum of the appropriate values to obtain a combination of types;
letters is an optional character scalar or vector that specifies the first letters of the names you want to select;
scope is an optional numeric argument that specifies the scope at which names are interpreted, where 1 = local (default); and 0 = global.

Result:
The result is a character matrix of names with the rows in alphabetical order. If you specify letters, the system returns only those names starting with the designated letters.

Note: ⎕idlist and ⎕nl provide similar capabilities, but they use different classification codes as arguments. See the description of ⎕idloc below for a listing of the differences.

Example:
List all functions that begin with T, U, or V.

```
    'TUV' ⎕idlist 1
TRI
VALIDATE
```

⎕idloc Identifier Localization

Purpose:
Returns the local and global classifications of functions, variables, and/or labels. This function is useful primarily for debugging while one or more functions are pendent on the stack. You can also use it under program control to determine the existence of an identifier.

Syntax: result ← ⎕idloc 'idlist'

Argument:
idlist is a list of the names of functions, variables and/or labels for which you want information. It can be a character vector that contains the names separated by one or more blanks or a character matrix with one name in each row.

Result:
The result is a numeric matrix with one row for each identifier named in idlist. The matrix has one column for each function in the state indicator, progressing from the most local to the most global in increasing column order. The last column contains the global classifications. The values in the last column are always non-negative.

Values that the system can return for ⎕idloc identifiers are shown below.

Value	**Classification**
¯1	Not localized at this level
0	Localized with no assigned value at this level or globally undefined.
1	System, user-defined, or associated function
2	System or user-defined variable with value
8	Label

Note: ⎕idloc and ⎕nc provide similar capabilities, but they use different classification codes. Other differences include the following.

- ⎕idloc returns all local and global classifications; ⎕nc returns only the locally active classifications of the identifier.
- ⎕idloc returns a more informative classification code than ⎕nc.
- ⎕idloc accepts either a character matrix or a character vector; ⎕nc accepts only a character matrix as an argument.
- ⎕idlist returns a result consistent with ⎕idloc; ⎕nl returns a result consistent with ⎕nc.
- ⎕nc accepts an ill-formed identifier name; ⎕idloc produces a DOMAIN ERROR.

For maximum portability to other APL systems, use ⎕nl and ⎕nc (which are part of standard APL); for more specific information, use ⎕idlist and ⎕idloc (which are APL64 standard features).

Example:
This example shows that A is undefined (0) in the most local environment (TRI), where it is localized, but it has not been defined by assigning it a value. In the environment of TEST, A is defined as a label (8). A has no global definition (0).

```
    )sinl
TRI[1]   *    N    A
TEST[1]       A

    ⎕idloc 'A N TRI'
 0  8  0
 2 ¯1  0
¯1 ¯1  1
```

⎕inbuf Set the Character Input Buffer

Purpose:
Specifies or empties the contents of the character input buffer.

Syntax: ⎕inbuf 'chars'
            style ⎕inbuf 'chars'

Arguments:
chars is a vector of characters or strings you want to insert into the buffer. You can include ⎕tcnl to represent an Enter keyboard action. If the vector contains more characters than the input buffer can hold, a LIMIT ERROR occurs.
style is an integer with a value between 1 (the default value) and ¯1. If you omit the left argument, or if the left argument is 1, the system inserts the keystrokes before the current contents of the buffer. If the left argument is zero, the right argument replaces the previous contents of the buffer; the system discards any previously typed or inserted keystrokes. If the left argument is ¯1, the system inserts the keystrokes after the current contents of the buffer.

Effect:
Puts the characters of the argument into the input buffer. The system uses these characters when it next looks for input, as if you had typed them on the keyboard.

The system function ⎕sys[33] controls the behavior of the input buffer when an error is signaled or evaluation errors in quote-quad or quad input.

Examples:
Upon an error, change the workspace name to prevent saving over the original copy with a)save command.

```
⎕elx←'0 ⎕inbuf ")WSID PROBLEM", ⎕tcnl ⋄ ⎕dm'
```

⎕it Interpreter Tuning

Purpose:
Used for various unrelated experiments, measurements, and queries in the 32-bit APLNow32 interpreter. ⎕it in APL64 is provided for compatibility with APL+Win but that it is largely deprecated and most of the operations don't pertain to or affect the APL64 computational environment in any way.

Syntax:

```
result ← ⎕it name
name     ⎕it value
name     ⎕it ''
```

Arguments:
value is any appropriate value, where "appropriate" depends on the name chosen. If you prefix name with a question mark (?), a description of it will be returned.

name is any valid argument to the command (others give a Domain Error). name can be one of the following values:

Value	**Description**
AuditRefcounts	Audit object reference counts (System Failure for problems)
AuditRefcountsC	Audit object reference counts (cleans up problems)
AuditRefcountsS	Audit object reference counts (report problems without cleaning)
Dump	Generate system debug dump.
IR	Returns Internal Representation (IR) of function named by right
MaxAlloc	Returns two-element vector indicating the maximum size object that can be allocated
SymbolTable	Returns a matrix of information about the symbol table
TraceClear	Clears and/or resize TraceLog buffer.
TraceLog	Display current TraceLog buffer.
TraceMark	Place a #MARK in the TraceLog buffer.
WsFrisk	Force immediate workspace frisk - detects inconsistent internal state
WsMerge	Force immediate workspace merge - partial freespace consolidation
WsPack	Force immediate workspace pack - complete freespace consolidation
WsResize	Force immediate workspace resize - commits memory
wsAllocMode	Set or query the memory allocation mode.
wsAllocsPerMerge	Controls how often memory is merged (allocs per merge)
wsAllocsPerPack	Controls how often memory is packed (allocs per pack)
wsAllocsPerResize	Controls how often memory is resized (allocs per resize)
wsMergeMode	Merge mode
wsResizeExcessFree	Memory resize factor (percent of free-space to commit in excess)
wsResizeExcessMax	Maximum excess memory commit size (bytes)
wsResizeExcessMin	Minimum excess memory commit size (bytes)
wsResizeExcessUsed	Memory resize factor (percent of used-space to commit in excess)
wsResizeMax	Maximum memory commit size (bytes)
wsResizeMin	Minimum memory commit size (bytes)
wsSlackMode	Slack mode
wsSlackSize	Initial size of catenation slack space
wsSplitsPerPack	Controls how often memory is packed (splits per pack)
wsStatistics	Returns a vector of workspace memory management statistics
wsUsage	Returns a matrix of workspace memory usage statistics

Result:
The result is dependent of the value of name.

Remarks:
The features of ⎕it operate on the associated APLNow32 workspace (owned by the 32-bit process) but this workspace doesn't contain any functions or variables.

Example:

```
CLEAR WS
    ⎕it 'wsUsage'
 2145648640 12288 7548 2145641060      0    0 2147483648 12288    0 12288
  964755456     0    0  963706848 1048576 4096  587530240     0 4096  4096
```

⎕it64 APL64 Interpreter Tuning

Purpose:
Used for various unrelated experiments, measurements, and queries in the interpreter.

Syntax:
```
result ← ⎕it64 name
name     ⎕it64 value
name     ⎕it64 ''
```

Arguments:
value is any appropriate value, where "appropriate" depends on the name chosen. If you prefix name with a question mark (?), a description of it will be returned.

name is any valid argument to the command (others give a Domain Error). name can be one of the following values:

Value	Description
Crash	Intentionally crash system
DamageMismatch	Intentionally damage an Aval by storing Value that doesn't match
DamageNelm	Type
DamageNulls	Intentionally damage an Aval Nelm
DamageShape	Intentionally damage an Aval by planting null elements
DamageType	Intentionally damage an Aval Shape
DamageValue	Intentionally damage an Aval Type
ExecuteCache	Intentionally damage an Aval Value
GCLatency	Query or set the code block cache size limit (see Note 1)
MemPool	Query or set GCLatency mode
NS	Query or set memory pooling (see Note 2)
RefCount	Query Native State
TuneDS	Returns RefCount info about an array
	Tune Dyadic Scalar Primitive Functions (see Note 3)

Note 1: This setting controls how many recent code blocks are cached. The default is 100 with a max of 100000. Set the limit to 0 to suppress execute caching.

Note 2: The default setting is 1, enabled. This setting controls when memory pooling is in effect to increase performance in situations where similar sized arrays are created and reused. But resulting in significant memory usage. Set this to 0 to suppress memory pooling.

Note 3: This setting controls when to apply the native-code and standard algorithms for optimizing system performance for the arithmetic functions (+ - × ÷). The default threshold value is 100 elements with a max of 1000000000.

Result:
The result is dependent of the value of name.

Example:

```
     ⎕it64 '?NS'
Query Native State
     res ← ⎕IT64 'NS'
     ⎕it64 'NS'
Loaded: C:\Program Files\APLNowLLC\APL64\APLNow.X64.dll
```

⎕lc Line Counter

Purpose:
Returns the current value of the execution line counter

Syntax: result ← ⎕lc

Result:
The result is a numeric vector of line numbers from the state indicator, beginning with the most local. It does not include any values that correspond to callouts, callbacks, or the execute (⍎), quad (⎕), or user command processor (]) symbols that appear in ⎕si or)si.

Example:
Although ⎕lc returns only line numbers, you can use it in the expression →⎕lc to resume a stopped or interrupted execution.

```
     ⎕si
TRI[2]*
⍎
EXAMPLE[3]

     ⎕lc
2 3
     →⎕lc
```

⎕l List of Characters in an APL identifier

Purpose: Returns all characters that can be part of an APL identifier.

Syntax: chars ← ⎕l

Result:
chars is a character vector.

Example:

```
    ⎕l
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789∆⍙_
```

⎕lcase Convert Uppercase to Lowercase

Purpose: Converts uppercase characters to lowercase characters.

Syntax: res ← ⎕lcase rArg

Argument:
arg is any APL64 value. The character or string elements of arg will be converted to lower case. Non-text elements of arg will not be modified in res.

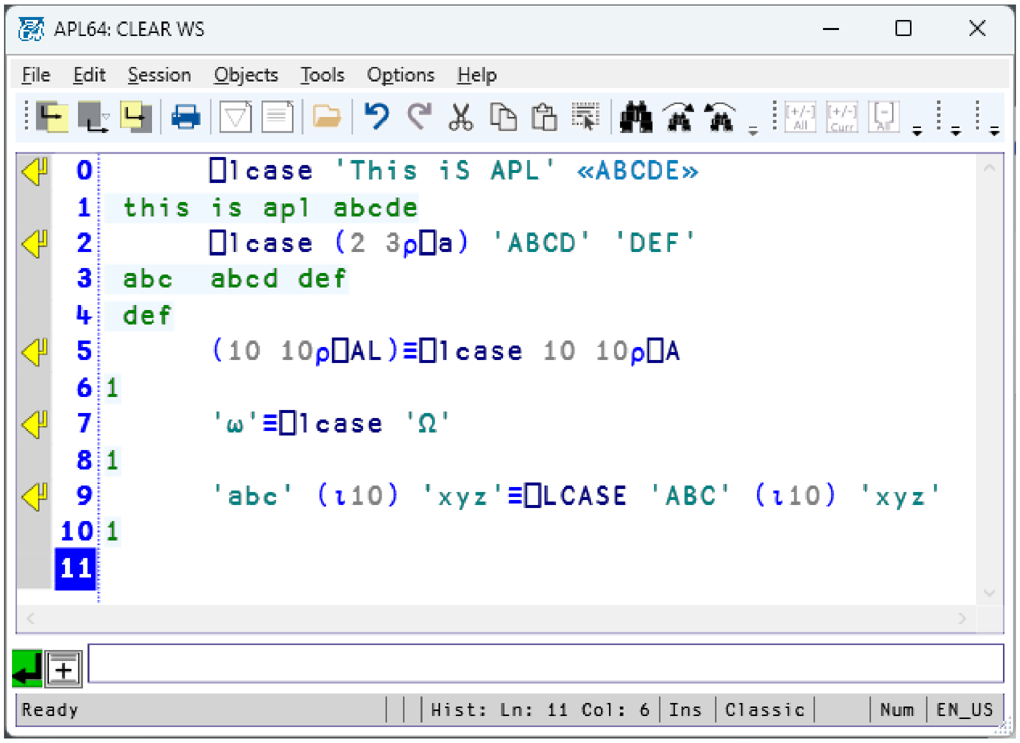
Result:
res has the same data type, rank and shape as rArg

Remarks:
⎕lcase replaces the assembler function LOWERCASE in the ASMFNS.w3 workspace in APL+Win.

⎕lcase uses the .Net String.ToUpperInvariant method.

Example:

```
⎕lcase 'This iS APL' «ABCDE»
⎕lcase (2 3ρ⎕a) 'ABCD' 'DEF'
(10 10ρ⎕AL)≡⎕lcase 10 10ρ⎕A
'ω'≡⎕lcase 'Ω'
'abc' (ι10) 'xyz'≡⎕LCASE 'ABC' (ι10) 'xyz'
```


APL64: CLEAR WS

```
      ⎕lcase 'This iS APL' «ABCDE»
 this is apl abcde
      ⎕lcase (2 3ρ⎕a) 'ABCD' 'DEF'
 abc  abcd def
 def
      (10 10ρ⎕AL)≡⎕lcase 10 10ρ⎕A
1
      'ω'≡⎕lcase 'Ω'
1
      'abc' (ι10) 'xyz'≡⎕LCASE 'ABC' (ι10) 'xyz'
1
```

Ready | Hist: Ln: 11 Col: 6 | Ins | Classic | Num | EN_US

⎕lib Library List

Purpose:
Returns a character matrix of file and directory names in the specified library.

Syntax:
```
result ← ⎕lib ''
result ← ⎕lib 'dir'
result ← ⎕lib lib
```

Arguments:
dir is a fully qualified path name;
lib is a library number.

The argument must be empty, a valid path, or a positive integer that has been associated with a directory by using ⎕libd or the Libraries choice on the Options menu. If the argument is empty, the system uses the current working directory.

Result:
The result is a character matrix that contains an alphabetic (uppercase Z preceding lowercase a) listing of file names or subdirectory identifiers contained in the specified directory. Subdirectories are marked by a backslash (\) character following the name. The number of columns in result depends on the longest name in the list. If the argument is the empty vector, the system uses the current working directory.

Examples:

```
      ⎕lib 'C:\WSS'
DATES.WS
SERXFER.WS
UTILITY.WS

      ⎕libd '12 C:\WSS'
      ⎕lib 12
DATES.WS
SERXFER.WS
UTILITY.WS
```

⎕libd Define Library

Purpose:
Associates a library number with a directory.

Syntax:
```
           ⎕libd 'libdefn'
result ←⎕libd libno
           ⎕libd 'libno'
```

Arguments:

libdefn must be a character vector that contains both a library number and the path, separated by at least one space. The library number should be an integer numeral (in character form) and the path should be a directory identifier.
libno can be either an integer scalar or a numeral character string, with strikingly different results.

Result:
When the argument is numeric, ⎕libd returns a character vector that contains the path name associated with the number. When you provide a character argument, ⎕libd does not return an explicit result.

Effect:
When you provide the library number and the directory designation as a character argument, the function equates the library number with the directory specified by the path. The path does not need to exist at the time the system executes ⎕libd, but it must exist when the library number is used. If the name is ill-formed or the library does not exist, the system displays a LIBRARY NOT FOUND message when you attempt to use the library definition.

The result of ⎕libs changes when you establish an association; you can use the number in workspace and file names, and you can use the number to query the contents of the directory. If the library number was defined previously, the new definition replaces the previous one.

If you enclose the library number in quotes or otherwise provide it as a character argument, the function deletes the existing association between that number and the directory.

Examples:

```
    ⎕libs
1 C:\APL

    ⎕libd '11 C:\APL\WSS' ⋄ ⎕libs
 1 C:\APL
11 C:\APL\WSS

    ⎕libd 1
C:\APL

    ⎕libd '1'
    ⎕libs
11 C:\APL\WSS
```

⎕libs Library to Directory Correspondence

Purpose:
Lists the defined APL libraries and the directories to which they correspond. You can use a library number in workspace and file identifiers, and you can use the number to query the contents of the directory.

Syntax: result ← ⎕libs

Result:
The result is a character matrix with one row for each defined APL library. Each row shows a library number and the directory to which it corresponds. If you have not defined libraries, the result is a zero-row matrix.

You can associate libraries and paths by using ⎕libd or by selecting the Libraries choice on the Options menu. The libraries listed in ⎕libs are not guaranteed to exist. If you attempt to access or create a file or workspace in a library that corresponds to a directory that the system cannot locate, the system returns a LIBRARY NOT FOUND error message.

Examples:

```
      ⎕libs
1 C:\APL\
11 C:\APL\OLD WORKSPACES
```

⎕ljust Left Justify

Purpose:
Returns the character elements in an array or string left justified.

Syntax: result ← ⎕ljust array

Arguments:
array is character array or scalar or string.

Results:
The result is the character elements in the array or string left justified.

Effects:
The ⎕ljust system function will not throw a DOMAIN ERROR exception for a right argument that is not appropriate for ⎕ljust and instead will return the right argument modified if possible.

Examples:

```
      ⎕←heading←2 15⍴'     First
Name       Last Name'
     First Name
      Last Name
      ⎕ljust heading
First Name
Last Name
      ⎕ljust (1 2 ' XY') 2.34 ' AB'
  1 2 XY   2.34 AB
```

⎕load Load a Workspace

Purpose:
Replaces the active workspace by loading a designated workspace.

Syntax: ⎕load 'wsid'

Argument:
wsid is a character scalar or vector that specifies the workspace to be loaded. The system assumes the default extension (.ws64). (You can use the file extension .w3 to load an APL+Win workspace.) If you omit the directory name or library number, the system uses the default directory.

Effect:.
The specified workspace becomes the new active workspace, ⎕wsid changes, and the system executes the latent expression (⎕lx). ⎕qload and ⎕xload provide a similar capability. ⎕qload does not display the SAVED message; ⎕xload does not execute ⎕lx.

Example:

```
      ⎕load 'TESTWS'
“2 TESTWS.ws64” LAST SAVED 01/22/2021 12:27:39 PM
```

⎕lock Lock Defined Functions

Purpose:
Locks one or more functions; the system allows no one to edit a locked function.

Syntax: result ← ⎕lock 'fnlist'

Argument:
fnlist is a list of function names, either a character matrix with one name in each row or a character vector that contains function names separated by blanks.

Result:
The result is a character matrix of requested function names whose definitions cannot be locked. If all the requested names are locked, result is an empty matrix with shape 0 0.

The system locks only the most local definition of a function. The system does not lock functions that are shadowed by more local use of the same name. Locking a function also removes any stop or trace settings it may have (see the descriptions of ⎕stop and ⎕trace in this chapter).

If you attempt to open a locked function, the system returns a **LOCKED FUNCTION ERROR: Cannot edit locked function *function_name*** error message.

Warning:
Locking a function is not reversible. Save an unlocked source version of any function that you lock.

Examples:
The last example locks all the functions in the active workspace.

```
     ρ⎕vr 'TRI'
72

     ρ⎕cr 'TRI'
3 32

     ⎕lock 'TRI'
     ρ⎕vr 'TRI'
0
     ρ⎕cr 'TRI'
0 0

 ⎕lock ⎕nl 3
```

⎕log Write to OS-dependent Event & Programmer-specified Log Files

Purpose:

Writes programmer-specified information to the operating system-specific Event Log or a log file.

Syntax:

result ← '?' ⎕log anyRightArgument

Returns a character vector containing the full path to the programmer-specified log file, if any. If no programmer-specified information has yet been written to the programmer-specified log file in the current APL64 instance, "" is returned.

result ← ‘Config’ ⎕log propName

Returns a character vector containing the value of the ⎕log property specified by propName

result ← ‘Config’ ⎕log propName propValue

Sets the value of the ⎕log property specified by propName with the value specified by propValue and returns a character vector containing the value of the ⎕log property specified by propName prior to its update.

result ← (LogFileOptions EventLogOptions) ⎕log data

Based on the LogFileOptions element of the left argument of ⎕log, the right argument is formatted and written to the programmer specified log file.

Based on the EventLogOptions element of the right argument of ⎕log, the right argument is formatted and written to the operating system-specific event log file.

⎕log Configuration Properties:

The ⎕log configuration properties are specific to an instance of APL64 and are not retained in the APL64 xml-format configuration file.

CurrentLogFileName: APL64 character vector

This is a read-only property which is constructed when ⎕log is used to write data to the programmer-specified log file. The components of the CurrentLogFileName property value are the LogFilePath, Application Name and LogFileName property values.

LogFilePath: APL64 character vector

This property is the base path for the programmer-specified log file which may be prefixed to the programmer-specified LogFileName property value when the LogFileName property value begins with ".".

The default value is operating system specific, e.g. 'C:\ProgramData\' for Windows. This property value may be specified by the APL64 programmer provided that the current workstation user has read/write access to the resulting programmer-specified log file.

ApplicationName: APL64 character vector

- This property value may be appended to the LogFilePath property value and included in the programmer-specified log file name. The default value for the APL64 Developer version is ⎕DEVCAP, e.g. 'APL64'. The default value for an APL64 runtime version is the name of the runtime executable. This property value may be specified by the APL64 programmer provided that the current workstation user has read/write access to the resulting programmer-specified log file.
- Used as the 'event source name' when ⎕log is used to write programmer-specified data to the operating system-specific log file.

LogFileName

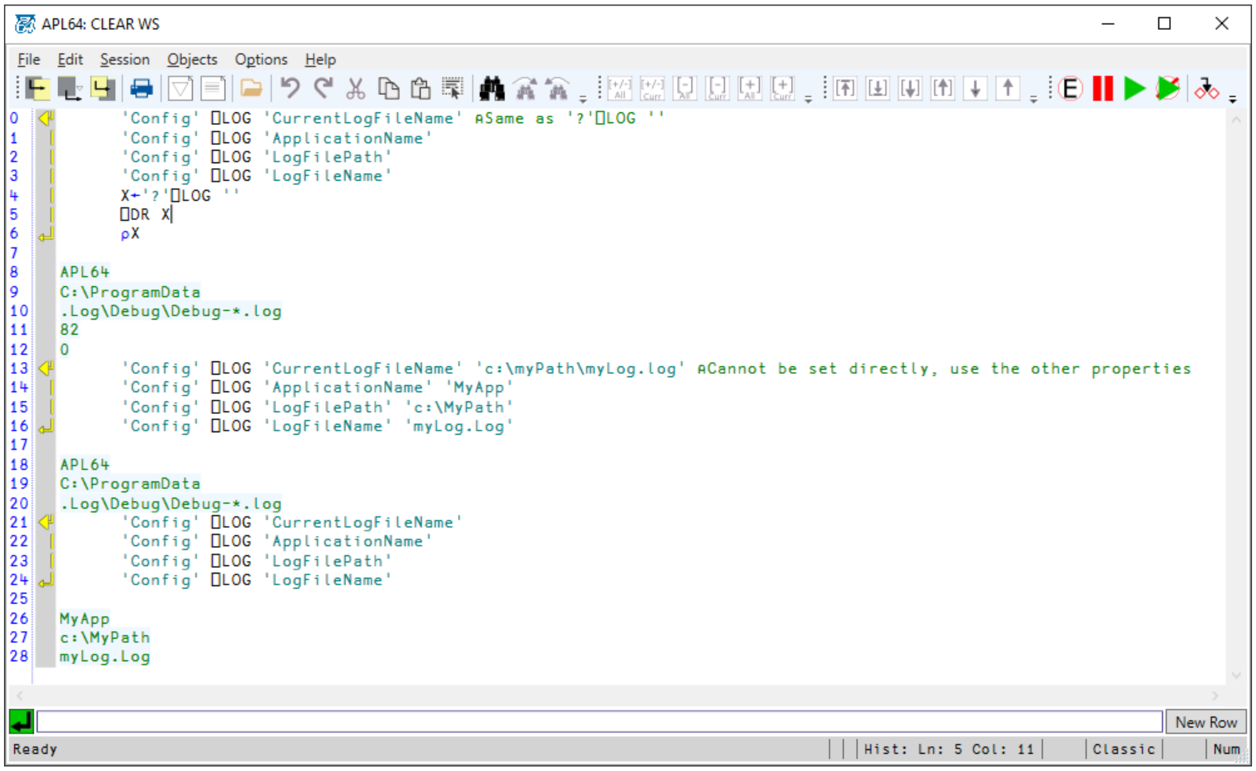
The filename of the programmer-specified log file. The default value is '.Log\Debug\Debug-*.log'. This property value may be specified by the APL64 programmer provided that the current workstation user has read/write access to the resulting programmer-specified log file.

When the programmer-specified log file name is used by the ⎕log system function, an '*' in the LogFileName property value will be substituted with the current UTC date and time in the format: 'YYYYMMDDhhmmssttt'.

If the LogFileName property value begins with '.', the programmer-specified log file path will be the concatenation of the LogFilePath property value, the ApplicationName property value and the LogFileName property value. For example: 'c:\Program Data\APL64\Log\Debug\Debug-20210401051033000.log'.

This property value is operating system-dependent and it is up to the APL64 programmer to provide the appropriate value according to the operating system of the target workstation. The default value is appropriate for the Windows operating system.

Examples:



```
      'Config' ⎕LOG 'CurrentLogFileName' ⍝Same as '?'⎕LOG ''
      'Config' ⎕LOG 'ApplicationName'
      'Config' ⎕LOG 'LogFilePath'
      'Config' ⎕LOG 'LogFileName'
      X←'?'⎕LOG ''
      ⎕DR X
      ⍴X

APL64
C:\ProgramData
.Log\Debug\Debug-*.log
82
0
      'Config' ⎕LOG 'CurrentLogFileName' 'c:\myPath\myLog.log' ⍝Cannot be set directly, use the other properties
      'Config' ⎕LOG 'ApplicationName' 'MyApp'
      'Config' ⎕LOG 'LogFilePath' 'c:\MyPath'
      'Config' ⎕LOG 'LogFileName' 'myLog.Log'

APL64
C:\ProgramData
.Log\Debug\Debug-*.log
      'Config' ⎕LOG 'CurrentLogFileName'
      'Config' ⎕LOG 'ApplicationName'
      'Config' ⎕LOG 'LogFilePath'
      'Config' ⎕LOG 'LogFileName'

MyApp
c:\MyPath
myLog.Log
```

LogFileOptions:

LogFileOptions is a three-element vector:

- LogEntryAction:
  - 'Append': A new log entry will be appended to the file with name CurrentLogfileName
  - 'Update': The data in the right argument of ⎕log will be added to the most recent log entry in the file with name CurrentLogFileName
  - 'NA': No data will be written to the programmer-specified log file.
- ReturnCurrentLogFileName
  - 0: The CurrentLogFileName will not be returned by the ⎕log system function
  - 1: The CurrentLogFileName will be returned by the ⎕log system function
- CloseCurrentLogFile

  Boolean value indicating if the current programmer-defined log file will be closed after writing the current data.

  - 0: The file with the name CurrentLogfileName not be closed and will continue to be used by future uses of the ⎕log system function.
  - 1: The file with the name CurrentLogfileName will be closed and future uses of the ⎕log system function will use a file defined by the current values of the LogFilePath, ApplicationName and LogFileName.

If LogFileOptions is APL64 programmer-provided as ⍬, these default values apply:

- LogEntryAction: Append

- ReturnCurrentLogFileName:
- CloseCurrentLogFile: 0

LogFileOptions Examples:

- ((‘NA’ 0 0) EventLogOptions) ⎕log data
  - Do not write a log file entry
  - Do not return the CurrentLogFileName
  - Do not close the current log file
- ((‘NA’ 1 0) EventLogOptions) ⎕log data
  - Do not write a log file entry
  - Return the CurrentLogFileName
  - Do not close the current log file
- ((‘NA’ 1 1) EventLogOptions) ⎕log data
  - Do not write a log file entry
  - Return the CurrentLogFileName
  - Close the current log file
- ((‘Append’ 0 0) EventLogOptions) ⎕log data
  - Append a new log file entry to the current log file
  - Do not return the CurrentLogFileName
  - Do not close the current log file
- ((‘Update’ 1 1) EventLogOptions) ⎕log data
  - Add the formatted data to the most recent entry in the current log file
  - Return the CurrentLogFileName
  - Close the current log file
- ((‘Append’ 0 1) EventLogOptions) ⎕log data
  - Add the formatted data to the most recent entry in the current log file
  - Do not return the CurrentLogFileName
  - Close the current log file
- (⍬ EventLogOptions) ⎕log data
  - Use default values of LogFileOptions

EventLogOptions

EventLogOptions: is a four-element vector:

- EventLogEntryAction:
  - ‘Append’: A new log entry will be appended to the OS-specfic event log file
  - ‘NA’: No data will be added to the OS-specfic Event log file
- `EventSeverityCode:`
  - `‘Info’: The` OS-specfic event log file entry will be marked as information
  - `‘Warn’: The` OS-specfic event log file entry will be marked as warning
  - `‘Error’: The` OS-specfic event log file entry will be marked as error
    - `‘Fatal’: The` OS-specfic event log file entry will be marked as fatal error
- EventLogCategoryId: Integer, default 0
- EventId: Integer, default 0

If EventLogOptions is APL64 programmer-provided as ϑ, these default values apply:

- EventLogEntryAction: NA
- `EventSeverityCode: Info`
- EventLogCategoryId: 0
- EventId: 0

EventLogOptions Examples:

- (LogFileOptions (‘Append’ ‘Info’ 1 1000)) ⎕log data
  - Append a new event to the event log
  - The new event has:
    - `EventSeverityCode: Info`
    - EventLogCategoryId: 1
    - EventId: 1000
- (LogFileOptions (‘Append’ ‘Warn’ 2 2000)) ⎕log data
  - Append a new event to the event log
  - The new event has:
    - `EventSeverityCode: Warn`
    - EventLogCategoryId: 2
    - EventId: 2000
- (LogFileOptions (‘Append’ ‘Error’ 3 3000)) ⎕log data
  - Append a new event to the event log
  - The new event has:
    - `EventSeverityCode: Error`
    - EventLogCategoryId: 3
    - EventId: 3000
- (LogFileOptions (‘Append’ ‘Fatal’ 3 3000)) ⎕log data
  - Append a new event to the event log
  - The new event has:
    - `EventSeverityCode: Fatal`
    - EventLogCategoryId: 3
    - EventId: 3000
- (LogFileOptions ϑ) ⎕log data
  - No event added to the event log
- (LogFileOptions (‘NA’ ‘Info’ 0 0)) ⎕log data
  - No event added to the event log

Formatting of the Right Argument of ⎕log:

Each new line of formatted data from the right argument is output into the file as a CR+LF pair. This happens for output data that is delimited as the row of a matrix right argument or by ⎕tcnl, ⎕tclf, or ⎕tcnl+⎕tclf pair.

The data is in Unicode format.

If a custom event source, other than one built into APL64, it should define one field (via %1 notation) into which the formatted content of the right argument is written.

Each new line of formatted data from the right argument is output into the file as a CR+LF pair. This occurs for output data that is delimited as the row of a matrix right argument or by ⎕tcnl, ⎕tclf, or ⎕tcnl+⎕tclf pair.

⎕av[⎕io] is translated to '§' (null characters in the event message mark end of string).

Microsoft Windows places a restriction on including %n such as %1, %2, in the string value (this refers to the right argument of ⎕log. The APL programmer must avoid using these character sequences when logging message to the Windows Event Log.

> The APL executable statement ('E' ⎕log 'Bad%1idea') causes the (Bad Bad Bad Bad %1 idea idea idea idea) message text in the event log.
>
> The string recursively references itself (because the message string from APL is passed as %1 to Windows). Fortunately, Windows stops recursing after 4 iterations! There is no way around this unfortunate Windows-imposed restriction (you cannot even use double-% such as %% to work around it). Be careful when formatting IPv6 address strings since they can contain %n codes.

The maximum length message that Windows allows to be logged is 32K (according to documentation). However, in practice the true limit is around 31900 bytes. If too long a string is passed to Windows, it simply fails to write anything to the event log. APL64 truncates any output in excess of length greater than 31900 to avoid complete failure if longer strings are passed to Windows. If this restriction is smaller on other operating systems, the built-in 31900 limit imposed by APL64 might be too large and the function will fail. It is up to the APL programmer to limit excessively so they can be logged to the operating system-specific event log.

Action of ⎕log

Each entry written to the debug log file has a timestamp line included in the file before the data. This timestamp line is prefixed and suffixed by CR+LF pairs.

The timestamps are UTC which does not vary as a function of the local time zone.

All data from the right argument written to the Windows Event Log or to the debug log file is "normalized" to make it readable by standard Windows editor applications. This involves translating all ⎕av characters to their Unicode representation. In addition, newline characters are normalized to CR+LF pairs. This means that when your output contains a ⎕tcnl, ⎕tclf, or ⎕tcnl+⎕tclf pair, these are all transformed into a CR+LF pair in the output.

Any ⎕tcnul characters that exist in the right argument of ⎕log are converted to the character "§" before writing to the Windows Event Log. This is necessary because nulls indicate end of string to Windows and any characters in the message following them would not be written to the event log. This transformation does not occur when writing to the debug log file.

⎕mathnet Interface to MathNet toolkit

Purpose:
An interface to the MathNet.Numerics.LinearAlgebra toolkit. To learn more about ⎕mathnet in the APL64 Developer version go to **Help | APL Language | Using ⎕MATHNET**.

⎕matrify Convert Delimited Segmented String to Matrix

Purpose:
Converts a delimited segmented string into a character matrix.

Syntax: charmat ← ⎕matrify 'segstring'
charmat ← [delimiter] ⎕matrify 'segstring'

Arguments:
The optional left argument, enclosed in quotes, specifies the delimiter for the segmented string. The delimiter can also be a vector. However, adjacent delimiters are treated as a single delimiter. The default value is a blank space.
The right argument is the segmented string you want to segment.

Effects:
The ⎕matrify system function will not throw a DOMAIN ERROR exception for a right argument that is not appropriate for ⎕matrify and instead will return the right argument unmodified.

Example:

```
      ⎕matrify '/   A/  BBB// CCCCC/DDDDDD/'
/
A/
BBB//
CCCCC/DDDDDD/
      '/' ⎕matrify '/   A/  BBB//
CCCCC/DDDDDD/'
   A
  BBB
 CCCCC
DDDDDD
```

```
      ⎕matrify ⊂,'X'
 X
      ((⍳10) (3 1⍴'abc') (2 3⍴⎕a)) ≡ ','
⎕matrify (⍳10) 'a,b,c' (2 3⍴⎕a)
1
```

⎕mattoss Convert Matrix to Segmented String

Purpose:
Converts a character matrix to a segmented string.

Syntax: segstring ← ⎕mattoss 'charmat'
segstring ← [delimiter] ⎕mattoss 'charmat'

Arguments:
The optional left argument, enclosed in quotes, specifies the delimiter for the segmented string. The default value is ⎕tcnl.
The right argument is the character matrix you want to segment. ⎕mattoss removes trailing blanks as it converts each row to a segment.

Remarks:
⎕mattoss is the replacement for the assembler function MATtoSS in the ASMFNS.w3 workspace in APL+Win.

Example:

```
      CM2
   AA
  BBBB
 CCCCCC
DDDDDDDD

      ⍴CM2
 4 8

      '*' ⎕mattoss CM2
*   AA*  BBBB* CCCCCC*DDDDDDDD
```

⎕mf Monitor Function

Purpose:
Sets and unsets the monitoring of the time consumed during execution of selected functions and returns data about the relative amount of time consumed by each line of those functions. You cannot set or unset monitoring on functions that are locked, suspended, pendent, or executing.
Syntax: result ← flag ⎕mf 'fnlist'
result ← ⎕mf calibrate
result ← ⎕mf 'fnname'

Arguments:
flag is a Boolean scalar that controls the monitoring setting. A 1 sets monitoring on, and a zero turns it off.

fnlist is a list of function names, either a character matrix with one name in each row or a character vector that contains function names separated by blanks.
calibrate is a numeric singleton that allows you to control certain aspects of the timing.

fnname is a character scalar or vector that contains the name of one function.

Result:
The result depends on the arguments you supply and the capability of your processor. To use the monitoring capability, you supply the name of a function or a list of functions (fnlist) as the right argument and turn monitoring on or off for those functions by using 1 or 0 (flag) as the left argument. The result is a Boolean vector with one element for each function name in fnlist. A 1 indicates that ⎕mf successfully set or unset monitoring for the corresponding function. A zero indicates that ⎕mf was unable to set or unset monitoring.

After you turn monitoring on, whenever you run one of the named functions, the system collects timing data for each line of the function. When you turn monitoring off or reset monitoring on for a function, the system discards any accumulated data for that function. Hence, you must turn monitoring on, run the function one or more times, and retrieve the data before resetting the status. You retrieve the timing data by supplying the name of the timed function monadically (see below).

Calibration Arguments for ⎕mf
You can specify which timer to use by supplying a scalar or one-element vector right argument in the range of ¯2 to 3 inclusive. There are three possibilities for the timer: 1) the millisecond timer; 2) the high resolution timer, which is based on the Win32 QueryPerformanceCounter() API; or 3) the CPU clock rate, which is based on the Pentium Read Time Stamp Counter Instruction. The system defaults to the highest resolution counter available on the host machine.

You can reset the time source by using the source number as a monadic right argument to ⎕mf. However, you cannot set the source to a higher number than the default. Resetting the time source does not reinitialize the data accumulated for the output matrix; however, data collected using different counters cannot be mixed meaningfully, so it is best to set the timer before turning on the timing for a function. You can query the current resolution timer by executing: ⎕mf ¯2

If you want to try to compare figures collected using different timer settings, you can determine the value of the tick interval in milliseconds for the current resolution by executing: ⎕mf ¯1. However, use this capability with caution. Some processors adjust their clock rate depending on the temperature of the CPU.

Whenever you set the timer with a nonnegative numeric, monadic argument to ⎕mf, the system performs a compaction on the workspace, which reduces the likelihood of an extraneous system action distorting the monitoring data. You can force a compaction without changing the timer setting by using ⎕mf 0.

Summary of Calibration Arguments to ⎕mf

⎕mf ¯2	return timer currently in use	⎕mf 2	use high resolution timer
⎕mf ¯1	current tick interval in milliseconds	⎕mf 3	use CPU clock rate timer
⎕mf 0	force compaction	⎕mf 4	use CPU clock rate timer
⎕mf 1	use millisecond timer		

Timing Data
Being that the default ⎕mf mode is 4 in APL64, if you supply only fnname, the result is a five-column integer matrix with one row for each function line, including the header. In mode 4, the times for execution (columns 1 and 2) and compilation (column 4) are separated into different columns whereas in modes 1, 2, and 3, columns 1 and 2 contain execution and compile times added together (like was traditionally done in APL+Win with ⎕mf 3).

The first row of the result contains information about the execution of the entire function. The second and subsequent rows of the result contain information about the corresponding function line. The table below shows the elements of the result matrix.

Result Matrix Elements for ⎕mf

Element	Description
[1;1]	Total units of time for entire function
[1;2]	[Reserved for future use]
[1;3]	Number of times the function was called
[1;4]	Execution time
[1;5]	Count for compiling each line of function
[k;1]	Accumulated units of time for line [K-1] of the function.
[k;2]	Units of time for line [K-1] minus the time the system used to execute subfunctions that were called on the same line
[k;3]	Number of times line [K-1] was executed

Note: K ranges from 2 to n+1, where n is the number of lines in the function.

Caution:
The values for time for any function that is called recursively may be misleading, since the first call will include subsequent ones.

Effects:
1 ⎕mf fnlist sets monitoring on for a list of functions and causes them to expand internally to include space for accumulated monitor data. If a function is already being monitored, 1 ⎕mf fnlist resets the monitor data. A monitored function that is subsequently locked continues to accumulate data while it is executing.

0 ⎕mf fnlist unsets monitoring. When monitoring is unset, the function contracts to its normal size.

Using the name of a timed function as the monadic right argument displays accumulated data for that function.

Using an appropriately valued numeric monadic right argument specifies which timer to use, forces a compaction, or returns information about the timer.

Example:
Monitor all functions whose names start with C in the workspace. Display the monitor data for the function named COMPLEX. The first line shows the time for the entire function; the next three lines show the data for lines 1, 2, and 3.

```
      ρF←'C' ⎕idlist 3
24 15
      ρA←1 ⎕mf F
24
      ⎕mf 'COMPLEX'
15 0 3
 8 8 3
 4 4 3
 3 3 3
```

⎕mix Reduce Level of Nesting

Purpose:
Rearrange an array by reducing one level of nesting and increasing the rank. This function produces the Evolution Level 1 behavior of monadic up arrow (↑) as an aid to converting applications.

Syntax: result ← ⎕mix array
              result ← ⎕mix[i] array

Arguments:
array is any array whose items have the same shape; ⎕mix does not pad items to match shapes.
i is a fractional scalar that indicates where to place the new dimensions in the shape vector.

Result:
The result is a new array, of higher rank, formed from the items of array. ⎕mix positions the items of its argument by inserting one or more dimensions in the consecutive positions starting with the ceiling of the fraction that you specify. The number of dimensions inserted is equal to the rank of the items of array.

If you do not specify the fraction, ⎕mix catenates the additional coordinates to the right end of the shape vector. To specify the dimensions, use a fractional value enclosed in brackets directly after the function. For example, to insert the coordinates after dimension 1 of the argument, you use the value 1.5.

Note: The Disclose function (monadic ⊃) provides a more flexible function to reduce nesting. Disclose allows you to specify non-adjacent axes for the new dimensions in the result, and it pads nested arrays that are not the same shape along the required axis with fill items to produce conforming items of the result.

Examples:

```
      A←(1 2 3 4) (5 6 7 8)
      ρA
```

```
2
     ρ¨A
 4  4
     ρ ⎕mix A
2 4

     ⎕mix A
1 2 3 4
5 6 7 8

     ⎕mix [.5]A
1 5
2 6
3 7
4 8

M←⊂[3 4]2 3 4 5ρι120
     M
  1  2  3  4  5   21 22 23 24 25       41 42 43 44 45
  6  7  8  9 10   26 27 28 29 30       46 47 48 49 50
 11 12 13 14 15   31 32 33 34 35       51 52 53 54 55
 16 17 18 19 20   36 37 38 39 40       56 57 58 59 60

 61 62 63 64 65   81 82 83 84  85   101 102 103 104 105
 66 67 68 69 70   86 87 88 89  90   106 107 108 109 110
 71 72 73 74 75   91 92 93 94  95   111 112 113 114 115
 76 77 78 79 80   96 97 98 99 100   116 117 118 119 120
     ρM
2 3
     (⎕mix M)≡ ⊃[3 4] M
1
     (⎕mix[.5] M)≡ ⊃[1 2]M
1
     (⎕mix[1.5] M)≡ ⊃[2 3]M
1
```

⎕mkdir Make a Directory

Purpose:
Creates a new disk directory.

Syntax: ⎕mkdir 'dir'

Argument:
dir is a character vector that contains a valid operating system directory identifier.

Effect:
The system creates a new directory on the specified or default disk drive, provided that a directory with this name does not already exist. ⎕mkdir is equivalent to the DOS command MKDIR or MD.

Examples:
Use a full path name starting from the root.

```
    ⎕mkdir 'C:\APLWIN\TESTWSS'
```

Create a subdirectory of the current default directory.

```
    ⎕mkdir 'TEST'
```

⎕mom MOM Object System

⎕mom has been deprecated in APL64.

⎕monadic Monadic Function

Purpose:
Returns the valence for the current function.

Syntax: result ← ⎕monadic

Result:
The result of ⎕monadic is 1 if the function is being called monadically or 0 otherwise such as when called from immediate execution.

⎕mself Current MOM Object

⎕mself has been deprecated in APL64.

⎕na External Interface using Name Association

Purpose:
Creates a function associated with a compiled language routine in a Dynamic Link Library (DLL) and a .Net assembly; verifies that such a function exists or returns its definition. For more details on executing a function in a .Net assembly, see EXT interface of ⎕NA to Access a .Net Assembly below.

Syntax: status ← 'defn'   ⎕na 'name'
              status ← ''   ⎕na 'name'
              defn   ←   ⎕na 'name'

Arguments:
defn is a character vector that describes the syntax and parameters of the non-APL routine. (See below for more detail on the structure of the left argument.)
name is a character vector or scalar that is the name of the associated function in the APL workspace.

Result:
Used dyadically, ⎕na returns a Boolean scalar. A status of 1 indicates that the function has been defined; zero means the function has not been defined. If the left argument is empty, ⎕na merely checks to see if the named function is an associated function.

If the left argument is not empty, ⎕na creates a locked, monadic function in the APL workspace that is associated with an external routine. Using this locked function causes APL to call the routine specified by defn, passing the pointers (or actual values) of the arguments supplied to name. The number of

items in the right argument of name must match the number of parameters specified in defn.
Used monadically, ⎕na returns the definition of the associated function.

Structure of the Left Argument:
The left argument defines the content of the function that ⎕na creates. It defines the type of routine, the name of the routine, and the form of the routine's parameters. The definition can have embedded ⎕tcnl characters between fields. Trailing comments that begin with a lamp (⍝) are accepted at the end of each line, allowing the definition to be self-documenting. The system ignores superfluous blanks between fields. The argument takes the form:
'language result ← library.routine (parm1, parm2 ...)'

- ***language***
  Defines the APL linkage scheme that matches the calling convention of the language and compiler in which the DLL-based subroutine was written. 'DLL' is the only choice currently supported; it specifies that you want to call a Windows subroutine in a 32-bit DLL. The subroutine can use a linkage with either a fixed parameter list or a variable number of arguments. The subroutine must use either the _ _cdecl or _ _stdcall Microsoft-specific keyword; _ _fastcall is not supported.
- ***result←***
  (Optional) The result (if any) the external routine returns. The result uses the same datatype notation as parm. If result has exactly one item, you can use a disclose symbol (⊃) as a prefix to have ⎕na automatically disclose the result.
- ***library***
  The name of a Win32 dynamic link library with extension .DLL. You do not supply the path or extension. APL64 searches for the DLL first in the directory where APL is installed, then in \WINDOWS\SYSTEM, and \WINDOWS, and finally in each directory in the DOS PATH. The order of search may vary by platform.
- ***routine***
  The name of the routine in the library. The distinction between uppercase and lowercase is important. When specifying the routine portion of defn, the spelling of the routine name must match the spelling as it is exported by the DLL. Some compilers may modify the name as given in the original C source code. Note: Some Windows libraries that have character arguments have two forms: they use 'A' and 'W' suffixes for 1-byte and 2-byte characters. You must supply the 'A' or 'W' suffix in the name of routine.
- ***parm1, parm2, …***
  The form of arguments to be passed to the routine. Enclose the list of argument specifications in parentheses and separate them with commas. If the routine does not require parameters, you must still supply an empty list enclosed in parentheses. Each parm describes the datatype of an argument, how it is passed, and whether the routine modifies it.
  If a parm is marked with a left arrow (←) as being modifiable, it is an output array that the routine returns as part of its explicit result, regardless of whether the routine actually modified it. If the argument consists of exactly one item, you can precede the specification with an enclose symbol (⊂) to have ⎕na automatically enclose the item.

APL64 recognizes the following datatypes for ⎕na

Datatype	Description
B1	Boolean (1 bit per element in byte-wise order; 8 bits per byte, most significant bit first).
C1	Character (1 byte per element) or C parameter declared as `char` (`unsigned` or `signed`).
F4	Four-byte double precision IEEE floating point or C parameter declared as `float`.
F8	Eight-byte double precision IEEE floating point or C parameter declared as `double`.
G0	General object. A variable in the internal form used by APL. Passed by reference with the address of the variable in the workspace.
I2	Two-byte signed integer or C parameter declared as `short`.
I4	Four-byte signed integer or C parameter declared as `long`.
U2	Two-byte unsigned integer or C parameter declared as `unsigned short`.
U4	Four-byte unsigned integer or C parameter declared as `unsigned long`.

Note: APL64 does not use two-byte integers in ordinary variables in the workspace. The system creates such values temporarily to provide them to the associated function, but it converts any returned results back to four-byte integers.

⎕na allows you to pass scalars by value. You must pass arrays of more than one element by reference. To indicate how you want APL to pass the arguments, add a prefix to the datatype. The possible prefixes are:

Prefixes for Parameter Passing

Prefix	Description
=	Default for most datatypes. Passed by value. The actual value is passed to the external routine. Only one-element, simple arrays can be passed by value.
⋆	Passed by reference. The machine address of the value is passed to the external routine. (Default for datatype G0, which cannot be passed by value.)
⍉	Passed by reference in column-major order as expected by FORTRAN; must be used instead of ⋆. For arrays with ranks greater than 2, only the last two coordinates are transposed.
⍟	Forced coercion. Forces a modifiable copy of the argument item to be made. The modified argument is not returned in the result

Example:

```
    'DLL I4←DLLEXP32.isum(*I4,I4)'⎕na 'ISUM'
1
    )FNS
ISUM
    '' ⎕na 'ISUM'
1
    ⎕na 'ISUM'
DLL I4←DLLEXP32.isum(*I4,I4)
    ISUM (ι100)(100)
5050
```

Errors:

The first two errors occur at the time you specify the ⎕na function. The other errors can occur at the time you call the DLL routine.

- DOMAIN ERROR
  The right argument is not character or does not contain a valid APL variable name, or the left argument is not character.
- LENGTH ERROR
  The right argument is empty.
- MODULE NOT FOUND
  The DLL does not exist.
- ENTRY POINT NOT FOUND
  The DLL does not contain the entry point indicated by the associated function.
- LANGUAGE INTERFACE NOT INSTALLED
  The language parameter to the ⎕na function does not match what the system expects.

Using the EXT interface of ⎕NA to Access a .Net Assembly

Using the EXT interface of ⎕NA, .Net-based methods may be used in an APL64 application. The desired .Net methods can utilize .Net features and the features of the APL64 interpreter exposed in the EXT interface. The .Net assembly (container) for the methods can use any .Net programming language, e.g., C#, VB.Net, Managed C++.

The EXT interface of ⎕NA can map any .Net Core- or .Net Standard Framework-based method to an APL64 user-defined function. These cross-platform frameworks support Windows, Linux, macOS and Docker.

APL64 Syntax

Mapping the .Net Assembly Method to an APL64 User-defined Function

In APL64 the mapping of a .Net method in an .Net assembly to an APL function is performed using the ⎕NA system function.

```
bool ← 'EXT .NetMethodName @.NetAssemblyPath' ⎕NA 'AplFunctionName'
```

The value of the scalar, Boolean result of the APL64 ⎕NA mapping APL64 statement, 'bool', is 1 if the mapping is successful. In that case the .Net assembly is loaded into the memory space of the APL64 interpreter and an APL64 user-defined function called 'AplFunctionName' is created in the APL64 workspace.

The value of the scalar, Boolean result of the APL64 ⎕NA mapping APL64 statement, 'bool', is 0 if the mapping is not successful.

The 'EXT' prefix indicates to APL64 that the .Net interface will be used to map the .Net method to the user-defined APL64 function.

The '.NetMethodName' is the fully-qualified .Net method name in the .Net assembly which will be mapped to the user-defined APL64 function.

'.NetAssemblyPath' is the fully-qualified path to the file containing the .Net assembly which will be mapped to the user-defined APL64 function. This file must be available while the APL64 application using the mapped function is active.

'⎕NA' is the APL64 system function which supports the .Net interface mapping feature.

Using the APL64 User-defined Function Mapped to a .Net Assembly Method

The 'mapped' APL64 user-defined function created by the ⎕NA mapping is a locked function which can be used in the APL64 application. The APL64 user-defined function is monadic or dyadic depending on the argument validation defined in the .Net method mapped to the APL function.

.Net Assembly Method Syntax

The method in the .Net assembly to be mapped to an APL64 user-defined function must have a specific syntax so that the APL64 interpreter can utilize it.

```
public static Aval MethodName(IExtContext context, Aval left, Aval right)
```

The method must have the 'public' attribute so that it can be accessed from outside the .Net assembly containing it.

The method must have the 'static' attribute so that only one memory-based copy of the method is made, no matter how many times it is used in the APL64 application.

The result of the method must be an APL64 'Aval' data type.

The 'MethodName' is up to the designer of the .Net method and should follow the .Net method naming guidelines.

.Net Method Argument	**.Net Method Argument Description**
1	'IExtContext' contains the APL64 interpreter context interface so that the .Net method can access APL64 variables and methods
2	'Aval left' contains the APL64 mapped function left argument
3	'Aval right' contains the APL64 mapped function right argument

The .Net data types 'APLNow.Data.IExtContext' and 'APLNow.Data.Aval' are provided by the APL64 APLNow.Data assembly which must be referenced by the .Net assembly containing the .Net method to be mapped to an APL64 user-defined function.

Performance

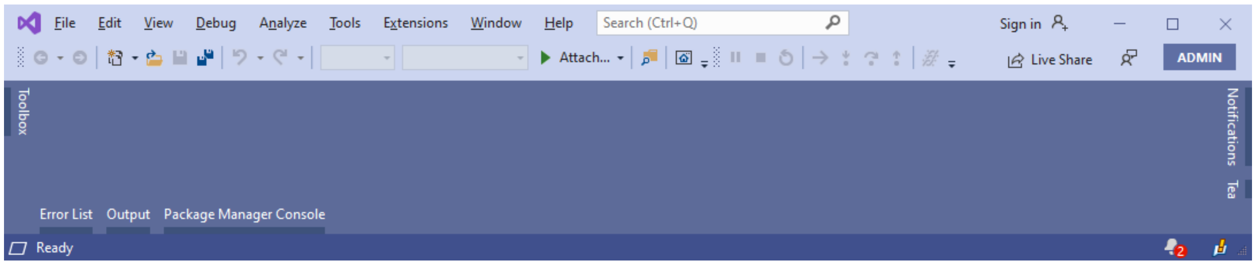
Since the .Net assembly containing the method to be mapped to an APL64 function is loaded into the memory space of the APL64 interpreter, performance of the ⎕NA EXT interface to the mapped APL64 function is excellent. Overall performance of the mapped APL64 function will depend on:

- The amount of data passed as the arguments to the mapped APL64 function
- The skill of the .Net programmer of the assembly which is mapped to that APL64 function by the ⎕NA EXT interface.

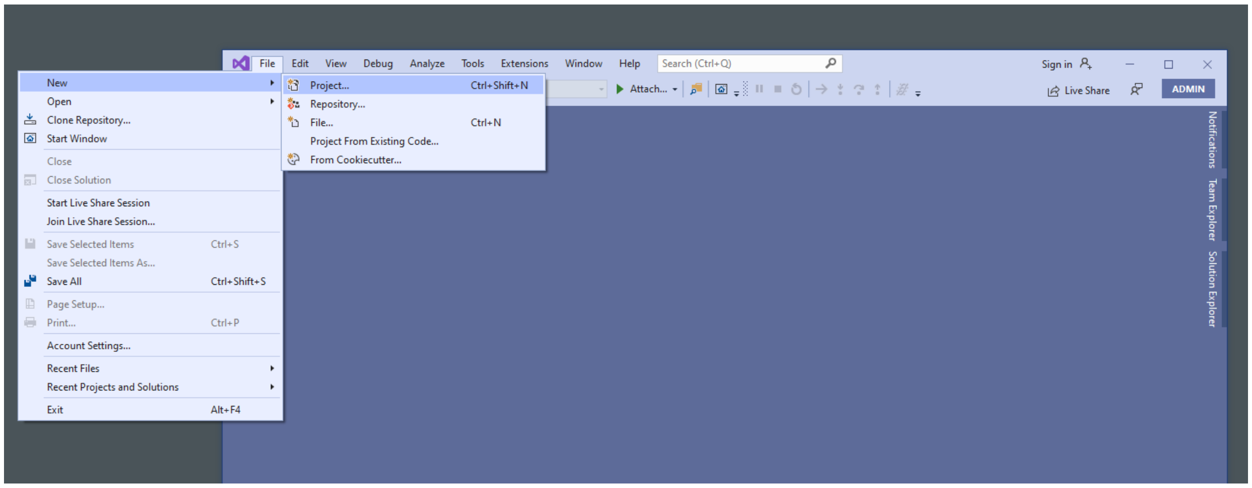
Example

Creating .Net Methods in an Assembly using Visual Studio 2019

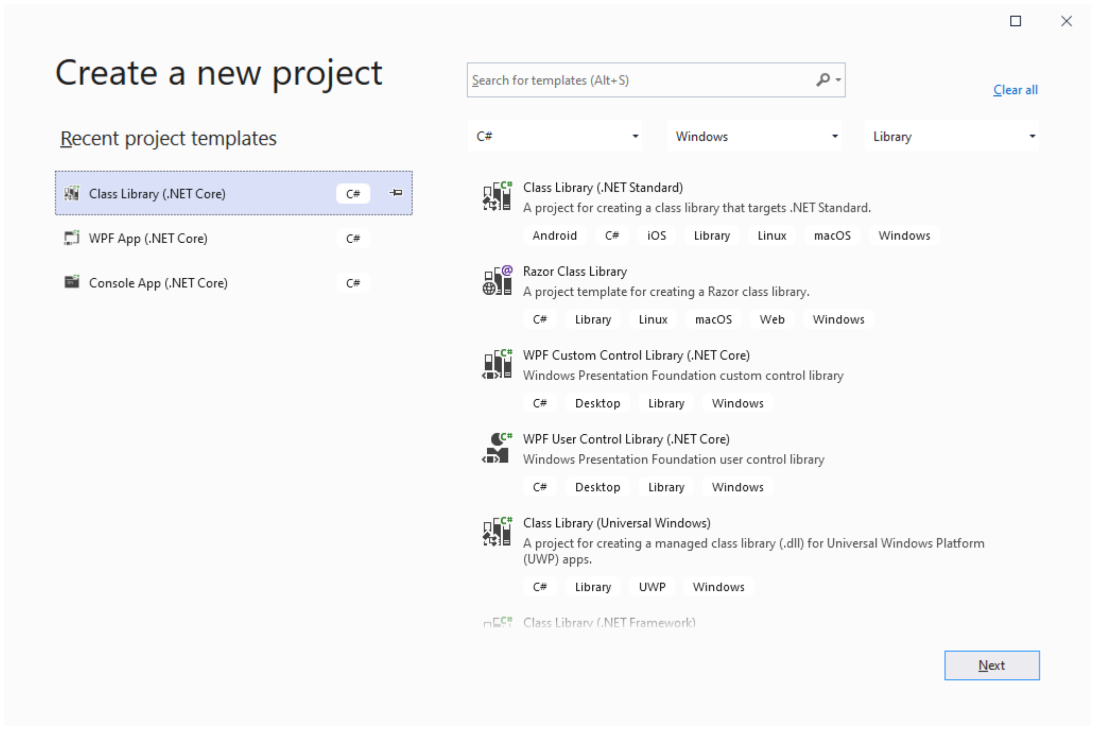
Start and instance of Microsoft Visual Studio:



Use File | New | Project to create a new project:



Select the project type as ‘Class Library (.Net Core)’ or ‘Class Library (.Net Standard)’ and click the Next button:



Class Library (.NET Standard)
A project for creating a class library that targets .NET Standard.
Android C# iOS Library Linux macOS Windows

Razor Class Library
A project template for creating a Razor class library.
C# Library Linux macOS Web Windows

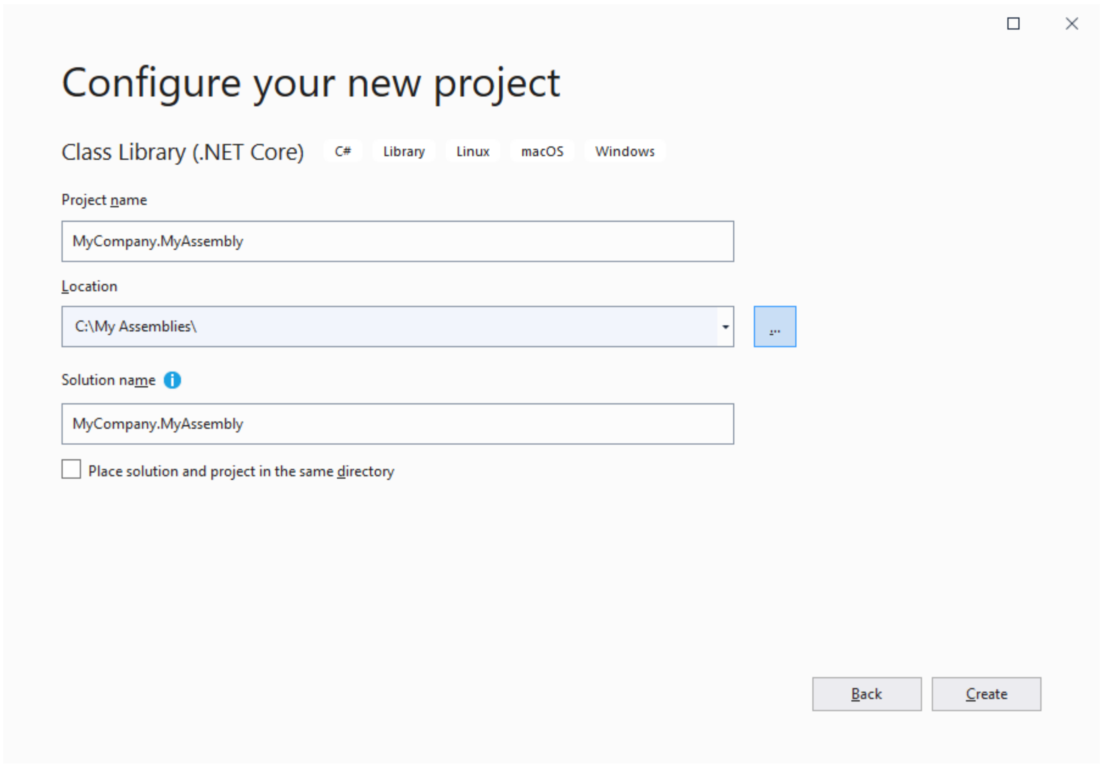
WPF Custom Control Library (.NET Core)
Windows Presentation Foundation custom control library
C# Desktop Library Windows

WPF User Control Library (.NET Core)
Windows Presentation Foundation user control library
C# Desktop Library Windows

Class Library (Universal Windows)
A project for creating a managed class library (.dll) for Universal Windows Platform (UWP) apps.
C# Library UWP Windows

Class Library (.NET Framework)

Complete the Configuration dialogue and click the Create button:



Configure your new project

Class Library (.NET Core) C# Library Linux macOS Windows

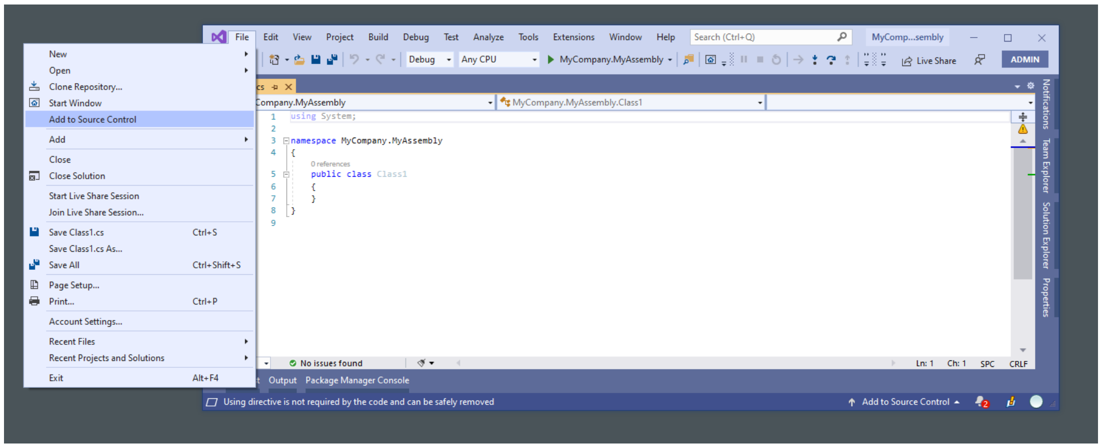
Project name: MyCompany.MyAssembly

Location: C:\My Assemblies\

Solution name: MyCompany.MyAssembly

☐ Place solution and project in the same directory

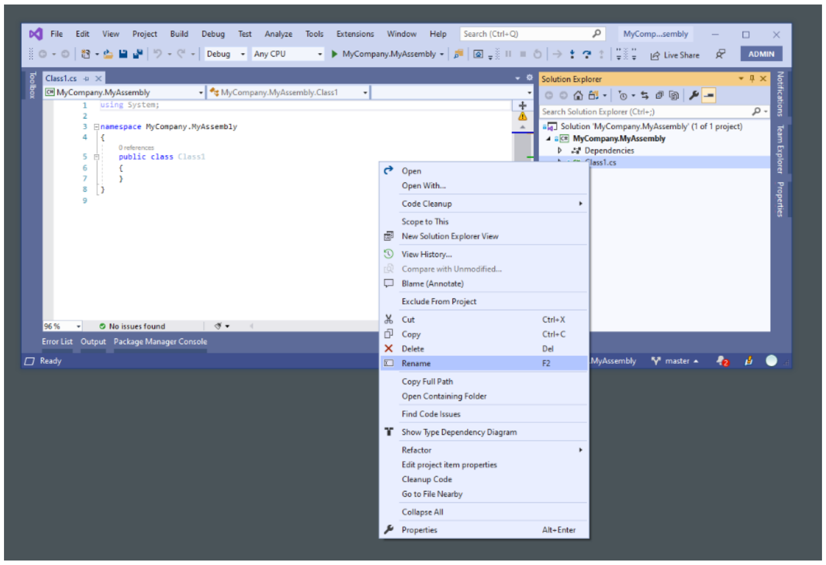
Use **File | Add to Source Control** to store the new project in a local GitHub repository:



```
using System;

namespace MyCompany.MyAssembly
{
    0 references
    public class Class1
    {
    }
}
```

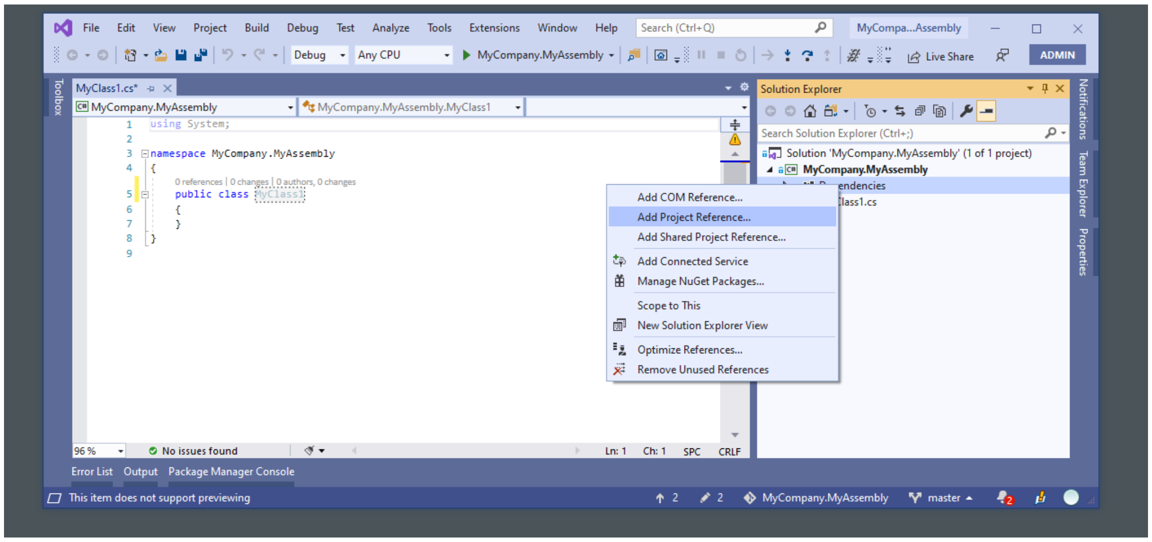
Using directive is not required by the code and can be safely removed

In the Solution Explorer use the Rename option of the context menu of the class1.cs file to rename the file and class it contains as desired (MyClass1):



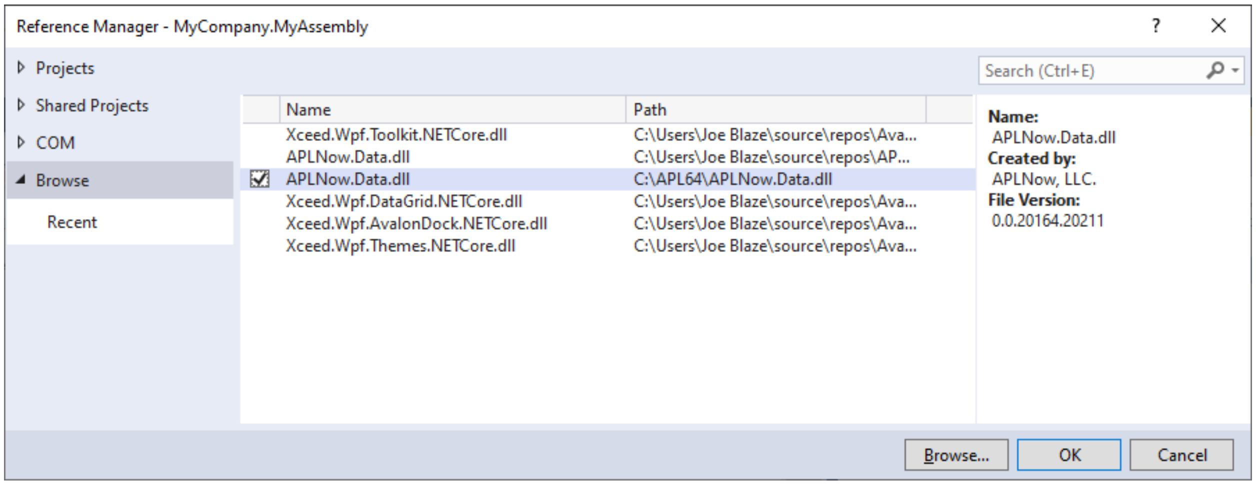
```
using System;

namespace MyCompany.MyAssembly
{
    public class Class1
    {
    }
}
```

In the Solution Explorer use the Add Project Reference of the context menu of the Dependencies item to reference the APLNow.Data assembly which is included in the installation of APLNow APL64:



```
using System;

namespace MyCompany.MyAssembly
{
    public class MyClass1
    {
    }
}
```

Browse to the APL64 installation folder and select the APLNow.Data.dll file and click the OK button of the Reference Manager dialogue:


Reference Manager - MyCompany.MyAssembly

	Name	Path
	Xceed.Wpf.Toolkit.NETCore.dll	C:\Users\Joe Blaze\source\repos\Ava...
	APLNow.Data.dll	C:\Users\Joe Blaze\source\repos\AP...
✓	APLNow.Data.dll	C:\APL64\APLNow.Data.dll
	Xceed.Wpf.DataGrid.NETCore.dll	C:\Users\Joe Blaze\source\repos\Ava...
	Xceed.Wpf.AvalonDock.NETCore.dll	C:\Users\Joe Blaze\source\repos\Ava...
	Xceed.Wpf.Themes.NETCore.dll	C:\Users\Joe Blaze\source\repos\Ava...

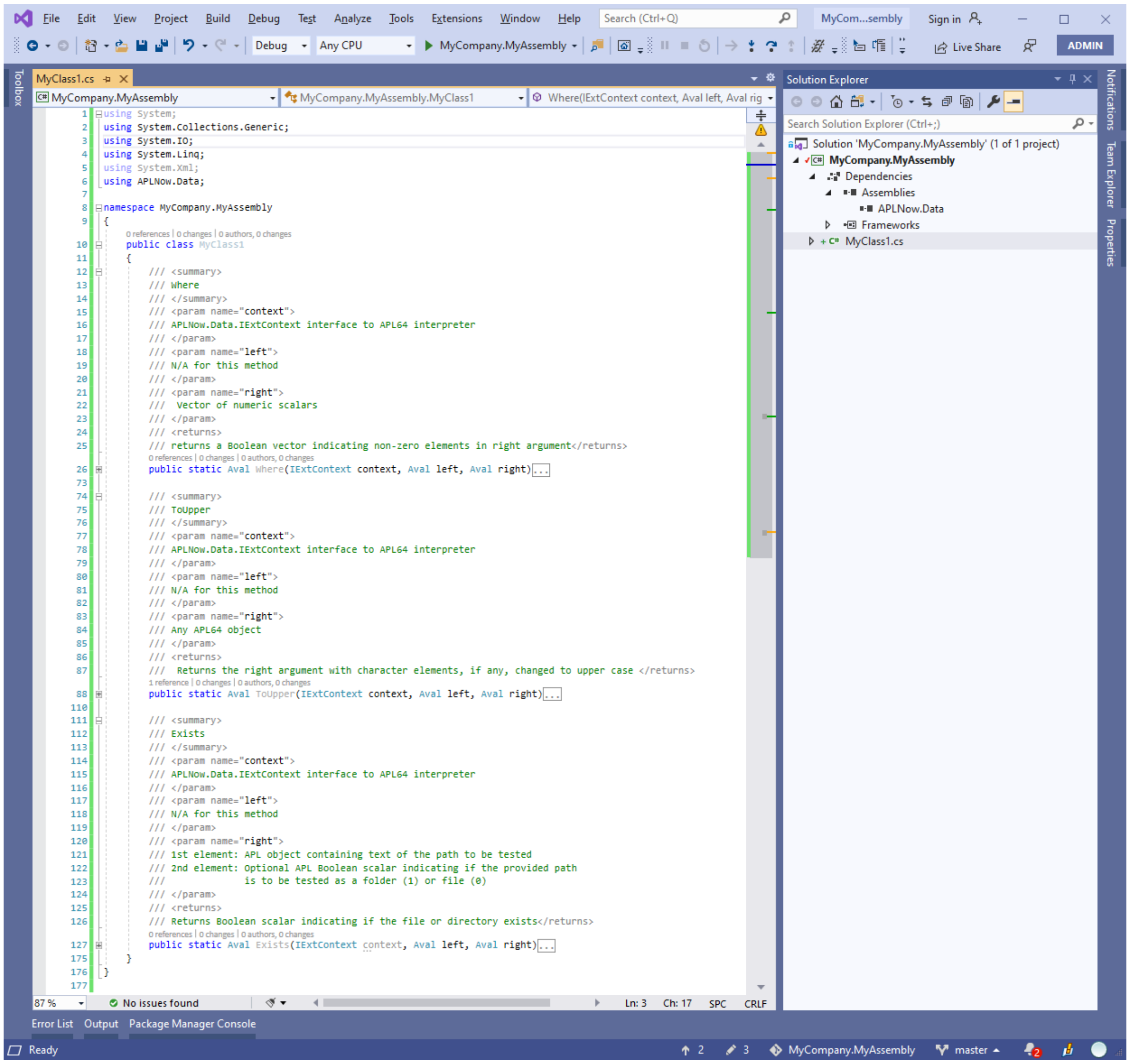
Name: APLNow.Data.dll
Created by: APLNow, LLC.
File Version: 0.0.20164.20211

For purposes of this workflow illustration, replace the text of the MyClass1 file with the following source code:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Xml;
using APLNow.Data;

namespace MyCompany.MyAssembly
{
    public class MyClass1
    {
        /// <summary>
        /// Where
        /// </summary>
        /// <param name="context">
        /// APLNow.Data.IExtContext interface to APL64 interpreter
        /// </param>
        /// <param name="left">
        /// N/A for this method
        /// </param>
        /// <param name="right">
        ///  Vector of numeric scalars
        /// </param>
        /// <returns>
        /// returns a Boolean vector indicating non-zero elements in right argument</returns>
        public static Aval Where(IExtContext context, Aval left, Aval right)
        {
            // Monadic only. Throw VALENCE ERROR if called dyadically
            if (left != null) throw new ValenceError();

            // Rank must be <= 1
            if (right.Rank > 1) throw new RankError();

            // Tailor results to current ⎕io setting
            var io = context.QuadIO;
            // For numeric types, count number of non-zero elements in argument,
            // then allocate result with counted number of elements,
            // then fill in result elements with indexes of each non-zero element.
            var z = new List<int>();
            switch (right.Type)
            {
                case DT.Bool:
                    var bv = right.BoolValues;
                    for (var i = 0; i < bv.Length; ++i)
                        if (bv[i])
                            z.Add(i + io);
                    break;

                case DT.Int:
                    var iv = right.IntValues;
                    for (var i = 0; i < iv.Length; ++i)
                        if (iv[i] != 0)
                            z.Add(i + io);
                    break;

                case DT.Double:
                    var ct = context.QuadCT;
                    var dv = right.DoubleValues;
                    for (var i = 0; i < dv.Length; ++i)
                        if (Num.qctcomp(dv[i], 0.0, ct) != 0)
                            z.Add(i + io);
                    break;

                default:
                    // Method does not support non-numeric types
                    throw new DomainError();
            }

            // Return integer vector result containing z values added above
            return new Aval(z);
        }

        /// <summary>
        /// ToUpper
        /// </summary>
        /// <param name="context">
        /// APLNow.Data.IExtContext interface to APL64 interpreter
        /// </param>
        /// <param name="left">
        /// N/A for this method
        /// </param>
        /// <param name="right">
        /// Any APL64 object
        /// </param>
        /// <returns>
        ///  Returns the right argument with character elements, if any, changed to upper case </returns>
        public static Aval ToUpper(IExtContext context, Aval left, Aval right)
        {
            // Monadic only. Throw VALENCE ERROR if called dyadically
            if (left != null)
```

```
            throw new ValenceError();
        switch (right.Type)
        {
            case DT.Zc:
                return new Aval(right.ZcValues.Select(Zc.ToUpper), right.Shape);
            case DT.Char:
                return new Aval(right.CharValues.Select(char.ToUpper), right.Shape);
            case DT.Ucs:
                return new Aval(right.UcsValues.Select(Ucs.ToUpper), right.Shape);
            case DT.String:
                return new Aval(right.StringValues.Select(c => c.ToUpper()), right.Shape);
            case DT.Aval:
                return new Aval(right.AvalValues.Select(c => ToUpper(context, null, c)), right.Shape);
            default:
                //Return right argument if ToUpper has no action upon it
                return right;
        }
    }

    /// <summary>
    /// Exists
    /// </summary>
    /// <param name="context">
    /// APLNow.Data.IExtContext interface to APL64 interpreter
    /// </param>
    /// <param name="left">
    /// N/A for this method
    /// </param>
    /// <param name="right">
    /// 1st element: APL object containing text of the path to be tested
    /// 2nd element: Optional APL Boolean scalar indicating if the provided path
    ///             is to be tested as a folder (1) or file (0/Default value)
    /// </param>
    /// <returns>
    /// Returns Boolean scalar indicating if the file or directory exists</returns>
    public static Aval Exists(IExtContext context, Aval left, Aval right)
    {
        // Monadic only. Throw VALENCE ERROR if called dyadically
        if (left != null)
            throw new ValenceError();
        var path = string.Empty;
        bool isPathFolder = false;
        switch (right.Type)
        {
            case DT.Zc:
            case DT.Char:
                if (right.Rank != 1)
                    throw new RankError();
                path = right.GetString();
                break;
            case DT.String:
                if (right.Rank != 0)
                    throw new RankError();
                path = right.GetString();
                break;
            case DT.Aval:
                switch (right.Rank)
                {
                    case 0:
                        path = right.GetString();
                        break;
                    case 1:
                        path = right.Pick(0).GetString();
                        isPathFolder = right.PickBool(1);
                        break;
                    default:
                        throw new RankError();
                }

                break;
            default:
                throw new DomainError();
        }

        if (isPathFolder)
        {
            //Employ appropriate .Net method
            return new Aval(Directory.Exists(path));
        }
        else
        {
            //Employ appropriate .Net method
            return new Aval(File.Exists(path));
        }
    }
  }
}
```

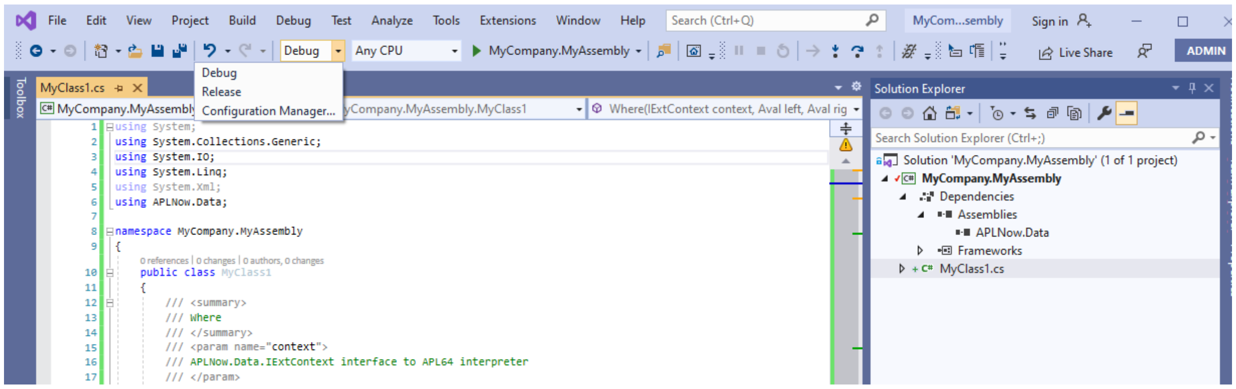
As a result MyClass1 now contains three methods: ‘Where’, ‘ToUpper’ and ‘Exists’. Review the source code of these methods to observe the techniques involved in their creation.



```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Xml;
using APLNow.Data;

namespace MyCompany.MyAssembly
{
    0 references | 0 changes | 0 authors, 0 changes
    public class MyClass1
    {
        /// <summary>
        /// Where
        /// </summary>
        /// <param name="context">
        /// APLNow.Data.IExtContext interface to APL64 interpreter
        /// </param>
        /// <param name="left">
        /// N/A for this method
        /// </param>
        /// <param name="right">
        ///  Vector of numeric scalars
        /// </param>
        /// <returns>
        /// returns a Boolean vector indicating non-zero elements in right argument</returns>
        0 references | 0 changes | 0 authors, 0 changes
        public static Aval Where(IExtContext context, Aval left, Aval right){...}

        /// <summary>
        /// ToUpper
        /// </summary>
        /// <param name="context">
        /// APLNow.Data.IExtContext interface to APL64 interpreter
        /// </param>
        /// <param name="left">
        /// N/A for this method
        /// </param>
        /// <param name="right">
        /// Any APL64 object
        /// </param>
        /// <returns>
        ///  Returns the right argument with character elements, if any, changed to upper case </returns>
        1 reference | 0 changes | 0 authors, 0 changes
        public static Aval ToUpper(IExtContext context, Aval left, Aval right){...}

        /// <summary>
        /// Exists
        /// </summary>
        /// <param name="context">
        /// APLNow.Data.IExtContext interface to APL64 interpreter
        /// </param>
        /// <param name="left">
        /// N/A for this method
        /// </param>
        /// <param name="right">
        /// 1st element: APL object containing text of the path to be tested
        /// 2nd element: Optional APL Boolean scalar indicating if the provided path
        ///              is to be tested as a folder (1) or file (0)
        /// </param>
        /// <returns>
        /// Returns Boolean scalar indicating if the file or directory exists</returns>
        0 references | 0 changes | 0 authors, 0 changes
        public static Aval Exists(IExtContext context, Aval left, Aval right){...}
    }
}
```

Use the **File | Save All** menu item to save the project.

During the development phase of the .Net assembly, select the Debug version from the Visual Studio toolbar. After the .Net assembly has been perfected, the Release version can be used.

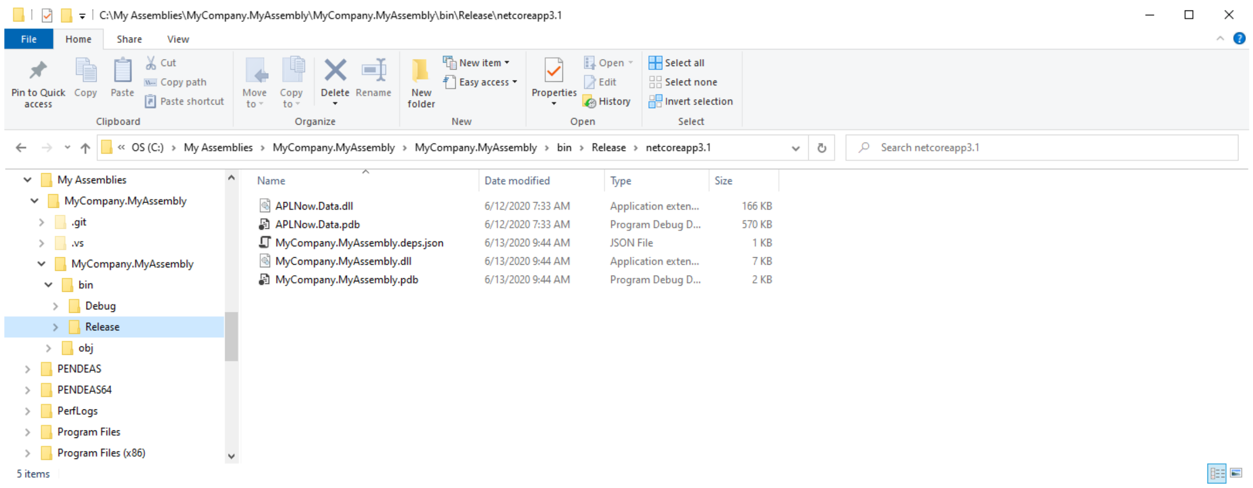


```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Xml;
using APLNow.Data;

namespace MyCompany.MyAssembly
{
    0 references | 0 changes | 0 authors, 0 changes
    public class MyClass1
    {
        /// <summary>
        /// Where
        /// </summary>
        /// <param name="context">
        /// APLNow.Data.IExtContext interface to APL64 interpreter
        /// </param>
```

Use the **Build | Build Solution** menu item to build (compile to .Net Common Intermediate Language (CIL)).

Once the .Net assembly has been tested, the resulting .dll-format file can be copied to the production environment so that it can be associated with the APL64-based application system which will use the ⎕NA EXT interface to map the .Net methods of the .Net assembly to APL64 user-defined functions.

The applicable files can be found in the .Net assembly's solution folder:



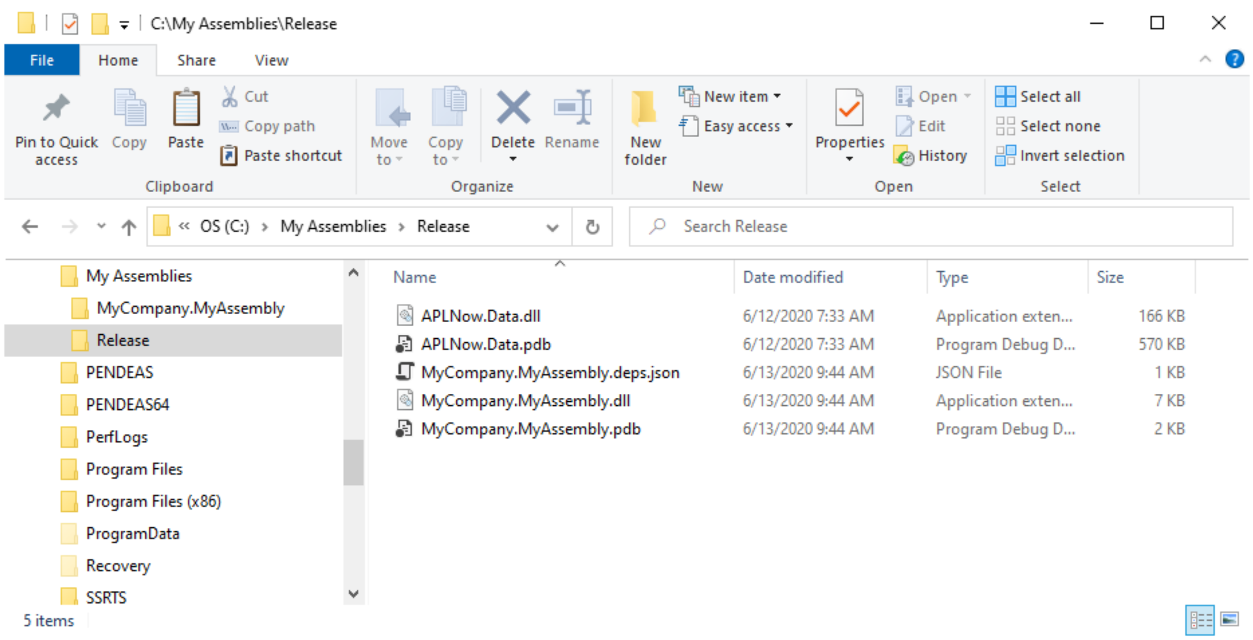
Name	Date modified	Type	Size
APLNow.Data.dll	6/12/2020 7:33 AM	Application exten...	166 KB
APLNow.Data.pdb	6/12/2020 7:33 AM	Program Debug D...	570 KB
MyCompany.MyAssembly.deps.json	6/13/2020 9:44 AM	JSON File	1 KB
MyCompany.MyAssembly.dll	6/13/2020 9:44 AM	Application exten...	7 KB
MyCompany.MyAssembly.pdb	6/13/2020 9:44 AM	Program Debug D...	2 KB

If the Visual Studio Team Explorer and local GitHub repository features are used, that repository will also contain the .Net assembly's required components.

Application deployment tools are available to package the components of the .Net assembly into an installation file, such as [Microsoft Visual Studio Installer Projects]() and [Microsoft Windows 10 Packaging Project]()

Mapping .Net Methods to an APL64 User-defined Function

Copy the required components of the MyCompany.MyAssembly assembly to the c:\MyAssemblies\ folder from the applicable sub-folder of the Visual Studio solution …\bin\release folder:

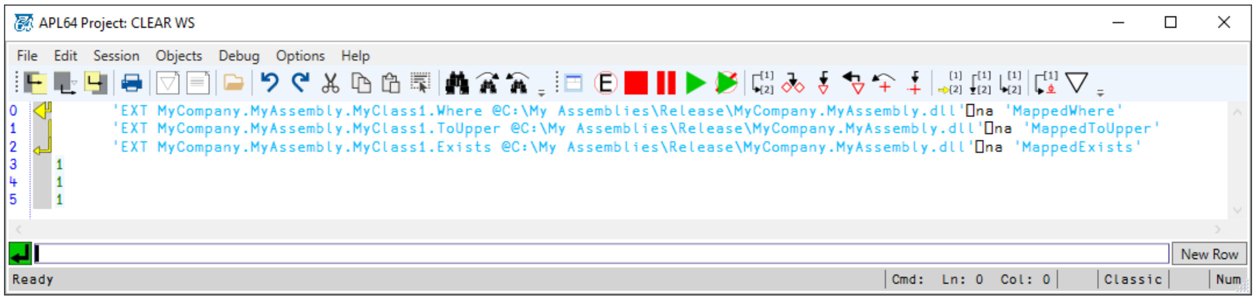


C:\My Assemblies\Release

Name	Date modified	Type	Size
APLNow.Data.dll	6/12/2020 7:33 AM	Application exten...	166 KB
APLNow.Data.pdb	6/12/2020 7:33 AM	Program Debug D...	570 KB
MyCompany.MyAssembly.deps.json	6/13/2020 9:44 AM	JSON File	1 KB
MyCompany.MyAssembly.dll	6/13/2020 9:44 AM	Application exten...	7 KB
MyCompany.MyAssembly.pdb	6/13/2020 9:44 AM	Program Debug D...	2 KB

5 items

Create an instance of APL64 and execute the ⎕NA EXT mapping APL statements:



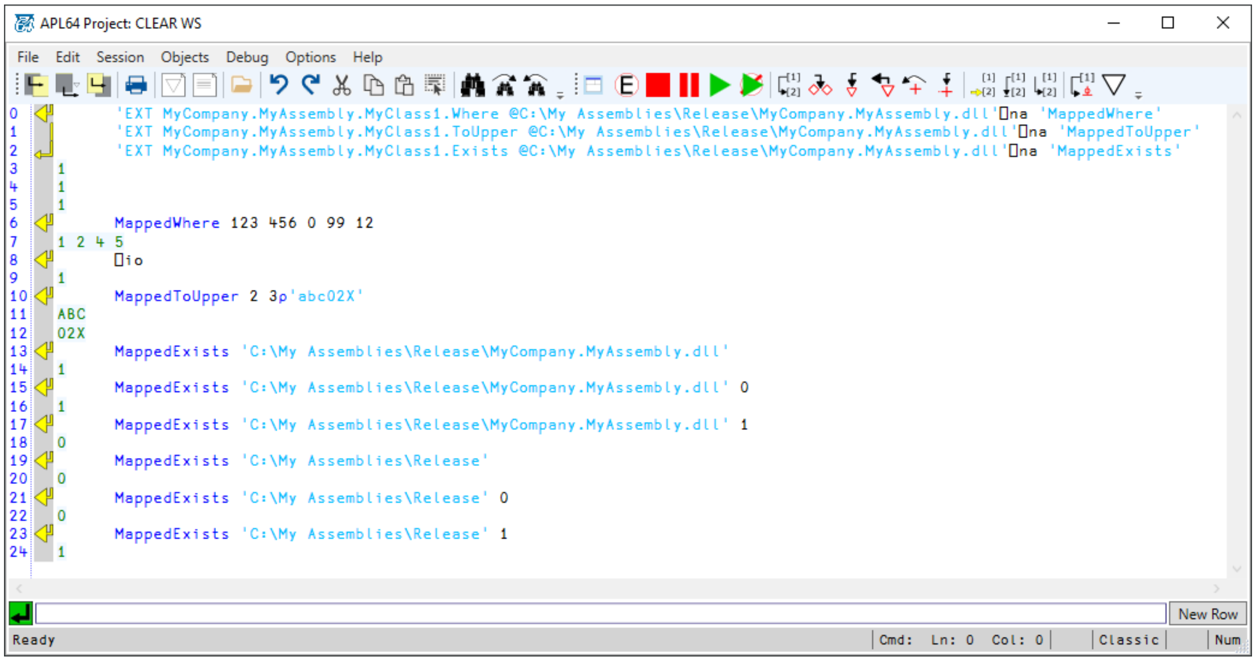
```
      'EXT MyCompany.MyAssembly.MyClass1.Where @C:\My Assemblies\Release\MyCompany.MyAssembly.dll'⎕na 'MappedWhere'
      'EXT MyCompany.MyAssembly.MyClass1.ToUpper @C:\My Assemblies\Release\MyCompany.MyAssembly.dll'⎕na 'MappedToUpper'
      'EXT MyCompany.MyAssembly.MyClass1.Exists @C:\My Assemblies\Release\MyCompany.MyAssembly.dll'⎕na 'MappedExists'
1
1
1
```

The APL64 mapped function names may be selected according to the APL64 user-defined function name rules.

.Net Method Name	.Net Assembly Path	APL64 Mapped Function Name
MyCompany.MyAssembly.MyClass1.Where	C:\My Assemblies\Release\MyCompany.MyAssembly.dll	MappedWhere
MyCompany.MyAssembly.MyClass1.ToUpper	C:\My Assemblies\Release\MyCompany.MyAssembly.dll	MappedToUpper
MyCompany.MyAssembly.MyClass1.Exists	C:\My Assemblies\Release\MyCompany.MyAssembly.dll	MappedExists

Using .Net Methods in APL64

In the same instance of APL64 where the ⎕NA mapping of the .Net methods to the APL64 functions was done, try the following APL64 executable statements:



```
      'EXT MyCompany.MyAssembly.MyClass1.Where @C:\My Assemblies\Release\MyCompany.MyAssembly.dll'⎕na 'MappedWhere'
      'EXT MyCompany.MyAssembly.MyClass1.ToUpper @C:\My Assemblies\Release\MyCompany.MyAssembly.dll'⎕na 'MappedToUpper'
      'EXT MyCompany.MyAssembly.MyClass1.Exists @C:\My Assemblies\Release\MyCompany.MyAssembly.dll'⎕na 'MappedExists'
1
1
1
      MappedWhere 123 456 0 99 12
1 2 4 5
      ⎕io
1
      MappedToUpper 2 3⍴'abc02X'
ABC
02X
      MappedExists 'C:\My Assemblies\Release\MyCompany.MyAssembly.dll'
1
      MappedExists 'C:\My Assemblies\Release\MyCompany.MyAssembly.dll' 0
1
      MappedExists 'C:\My Assemblies\Release\MyCompany.MyAssembly.dll' 1
0
      MappedExists 'C:\My Assemblies\Release'
0
      MappedExists 'C:\My Assemblies\Release' 0
0
      MappedExists 'C:\My Assemblies\Release' 1
1
```

Additional Information

The APL64 Development Team members are available on a consulting basis to implement .Net Assemblies containing .Net methods suitable for any APL64-based application system.

⎕nappend Append to a Native File

Purpose:
Appends a value to the end of a designated native file. The value is identical to the ravel of the value in the left argument.

Syntax: value ⎕nappend tieno

Arguments:
value is any simple, homogeneous APL array;
tieno is a negative integer file tie number.

The left argument is the simple, homogeneous value that you want to append to the file. It can be of any rank, shape, or datatype, but the system ravels it. The right argument must be a negative integer that designates the file to which you want to append data.

Effect:
The system appends new data to the end of the native file. The system writes the bit pattern of the data to the file, but it does not record the datatype of the argument. You must remember or record the information somewhere so that you can supply it when you retrieve the data later.
In particular, note that an APL numeric array, even if it currently contains all integers, might be represented internally as floating-point data. You can use Floor (⌊) or Ceiling (-) before filing data to ensure not only integer values but integer storage format (four bytes each) if all values are between ¯2*31 and ¯1+2*31.

Warning:
⎕nappend may not immediately place data on the disk. It may wait until there are more data or until the file is untied using ⎕nuntie or)off.

Examples:

```
(⎕vr 'TRI') ⎕nappend ¯27
LIBRARY ⎕nappend ¯33
```

⎕nblength Non-Blank Length

Purpose:
Returns length through the last non-blank character.

Syntax: result ←   ⎕nblength array
              result ← [char]   ⎕nblength array

Argument:
array is a character scalar, a character vector or string.

char is the optional left argument to specify the selected character; the default is ' '. This can be a character scalar, a non-empty character vector, or a string scalar.

Result:
The result is the length of the text up to and including the last character in the text which is not the selected character.

Examples:

```
    ⎕NBLENGTH '    abcd     '
8
    ⎕NBLENGTH '  '
0
    ⎕NBLENGTH ''
0
    'c' ⎕NBLENGTH '    abcd'
8
    «c» ⎕NBLENGTH '    abcd'
8
    «c» ⎕NBLENGTH «    abcd»
8
    'c' ⎕NBLENGTH «    abcd»
8
    'c' ⎕NBLENGTH 'cccc'
0
    'c' ⎕NBLENGTH ''
0
```

⎕nc Name Classification

Purpose:
Returns the classifications of a list of names.

Syntax: result ← ⎕nc 'namelist'
result ← [scope] ⎕nc 'namelist'

Argument:
namelist is a list of function, variable, or label names. It can be a character vector with the names separated by blanks or a character matrix with one name per row;
scope is an optional numeric argument that specifies the scope at which names are interpreted, where 1 = local (default); and 0 = global.

Result:
The result is a numeric vector of classification codes, one for each name in the argument. Values that the system can return are:

Value	Classification
0	Not defined
1	Label
2	Variable
3	Defined function
4	Other: the name is invalid or it is the name of a system function or variable (that is, it begins with a ⎕).

Note: ⎕idloc and ⎕nc provide similar capabilities, but they use different classification codes. See the description of ⎕idloc for a listing of other differences.

Examples:

```
    ⎕nc 'A TRI'
2 3
    ⎕nc 2 3 ρ'A  TRI'
2 3
    ⎕nc '⎕wa'
4
```

⎕ncreate Create Native File

Purpose:
Creates a new native file and ties the file.
(See also the extended file function ⎕xncreate, which supersedes this function.)

Syntax: 'fileid.ext' ⎕ncreate tieno

Arguments:
fileid is a valid native file name;
ext is the file extension;
tieno is a negative integer file tie number.

The left argument must be a valid native file identifier. Note that APL64 does not use library numbers for native file operations. The extension is an optional, user-chosen mnemonic classification; it can consist of any characters allowed by the operating system.

The right argument must be a negative integer that designates an available file tie number. You cannot have another file currently tied with this number.

Effect:
The system creates and ties a new file. Microsoft operating systems create files in openmode 66. To change this type of file open, you can use a slippery tie with another mode (such as fileid ⎕ntie tn, 18) right after you create the file.

Examples:

```
    'MEMO.TXT' ⎕ncreate ¯27
    'E:\CIRCUL\MEMO.TXT' ⎕ncreate ¯27
    'B:SCRATCH.GIG' ⎕ncreate ¯25
```

Note: Using the traditional file functions, you are limited in names you can use. For new programming, the extended file functions provide more capability and flexibility.

⎕nerase Erase Native File

Purpose:
Erases a native file and deletes the file name from the disk directory.
(See also the extended file function ⎕xnerase, which supersedes this function.)

Syntax: 'fileid.ext' ⎕nerase tieno

Arguments:
fileid is a valid native file name;
ext is the file extension;
tieno is a negative integer file tie number.

The left and right arguments designate the same file. The left argument must be a valid native file identifier. Note that APL64 does not allow library numbers for native file operations. The extension is an optional, user-chosen mnemonic classification; it can consist of any characters allowed by the operating system. The right argument must be a negative integer that defines the file you want to erase.

Effect:
The system unties and erases the file; if the erasure is unsuccessful, the system attempts to retie the file.

Examples:

```
    'B:MEMO.TXT' ⎕ntie ¯27
    'B:MEMO.TXT' ⎕nerase ¯27
    'SCRATCH.GIG' ⎕ntie ¯33
    'SCRATCH.GIG' ⎕nerase ¯33
```

Note: Using the traditional file functions, you are limited in names you can use. For new programming, the extended file functions provide more capability and flexibility.

⎕nexto Catenate Arrays Side by Side

Purpose:
Laminates one array next to another array.

Syntax: result ← array2 ⎕over array1

Arguments:
array1 and array2 are any array of character arrays or scalars or strings.

Results:
The result is array2 laminated next to array1.

Examples:

```
    «1 2 3» ⎕nexto «4 5 6»
 1 2 3 4 5 6
    (4 4ρ'abcd') ⎕nexto 3 3ρ'zzz'
abcdzzz
abcdzzz
abcdzzz
abcd
```

⎕nfe Native Files with Encoding

Purpose:
The ⎕nfe system functions supports reading and writing of text to files using several encoding types.

Syntax: result ← [larg] ⎕nfe action [rarg]

Arguments:
larg are the optional arguments to the action.
action is a text vector that determines how the system function will operate on the left and right argument.

rarg are the optional arguments to the action.

Result:
The result, if any, depends on the action called.

Effect:
⎕Nfe supports reading and writing of text to files using standard encodings including ASCII, UTF7 (see Note), UTF8, UTF16 and UTF32. Note: UTF7 encoding has been deprecated by Microsoft for security reasons.

A character encoding is a '1 to 1' mapping of abstract glyphs (characters) to values that represent those glyphs. The values resulting from the encoding of glyphs can be persisted and transmitted without ambiguity.

Remark:
Refer to the detailed documentation for ⎕NFE in **Help | APL Language | Using ⎕NFE**.

Example:

```
      'Document1.txt' ⎕nfe 'Create' 'ReadWrite' 'Read'
      ⎕nfe 'encoding' 'ASCII'
      'abc' ⎕nfe 'Write' 'Document1.txt' 0
      'Document1.txt' ⎕nfe 'Read' 0 3
abc
      ⎕nfe 'Delete' 'Document1.txt'
      ⎕nfe 'Close'
```

⎕nflush Extended Flush Native File

Purpose:
Flush file buffers.

Syntax: ⎕nflush tieno

Arguments:
tieno is a negative integer file tie number

Effect:
Clears buffers and causes any buffered data to be written to the file.

Examples:

```
⎕nflush ¯1
```

⎕ni Network Interface

Purpose:
Creates, controls, and runs a network communications interface using sockets.

Syntax:
```
result ←          ⎕ni 'action_name' [arguments]
result ←socket    ⎕ni 'action_name' [arguments]
```

Arguments:
socket is the number of the socket;
action_name is the action to be performed on object;
arguments are the optional arguments to the action.

Result:
For most actions, ⎕ni returns a multi-element result. The first element is a status code. A status code of 0 indicates a successful operation. A status code of ¯1 indicates an error in the underlying sockets action. A status code of 1 indicates a non-error condition that prevents completion of the operation. For a 0 status code, the subsequent elements provide in-depth information about the completion of the operation, about the socket state, or returned data. For a ¯1 status code, the next element is an integer error value; you can look up the meaning using the Error method of ⎕ni.

For a 1 status code, the next element is one of a small number of integer values that further define the condition preventing successful completion. If a receive is pending on a socket when another operation is attempted on that same socket, this element has a value of 1. The operation will not succeed until the Recv is completed. If the partner socket has been closed, this element has a value of 0. A close should be done on the local socket.

Not every method follows this format; the socket methods that always return information and have no normal error return conditions do not follow the status code format.

Effect:
⎕ni is the central feature of the interface between your application and communications over a network. See the TCP/IP and Sockets in APL64 chapter in this manual for more information.

⎕nl Name List

Purpose:
Returns a character matrix of function, variable, and/or label names in the active workspace.

Syntax:
```
result ←            ⎕nl class
result ← [letters]  ⎕nl class
result ←  [scope]   ⎕nl class
```

Arguments:
class is the type of identifier, where 1 = labels; 2 = variables; and 3 = functions;
letters is an optional character scalar or vector that specifies the first letters of the names you want to select;
scope is an optional numeric argument that specifies the scope at which names are interpreted, where 1 = local (default); and 0 = global.

Result:
The result is a character matrix of names with the rows in alphabetical order. If you specify letters, the system returns only those names starting with the designated letters. If you designate more than one class, the system returns names that belong to any of those classes. For example, ⎕nl 2 3 produces a matrix of the names of all of the variables and functions in the workspace. The system uses the most local definitions.

Note: ⎕idlist and ⎕nl provide similar capabilities, but they use different classification codes as arguments. See the description of ⎕idloc for a listing of the differences.

Examples:

```
      )fns
TRI    UPDATE  VOID    WITH    WITHOUT  XMIT

      'TX' ⎕nl 3
TRI
XMIT

      )vars
```

```
ARC    TERM   XRAY

      'TX' ⎕nl 3 2
TERM
TRI
XMIT
XRAY
```

⎕nlock Extended Lock Native File

Purpose:
Lock or unlock a file.

Syntax: ⎕nlock tieno operation position length

Arguments:
tieno is a negative integer file tie number

operation is a Boolean value: 1 for locking the file, 0 for unlocking the file

position is a 64-bit integer specifying the beginning of the range to lock (the value of this argument must be equal to or greater than zero)

length is a 64-bit integer specifying the range to be locked (the value of this argument cannot be negative)

Effect:
Locking part of a file prevents other processes from reading from or writing to that part of the file. Unlocking a file allows access by other processes to all or part of a file that was previously locked.

Examples:

```
⎕nlock ¯1 1 1024 1
⎕nlock ¯1 0 1024 1
```

⎕nnames Names of Tied Native Files

Purpose:
Returns the file identifiers of all the files currently tied with ⎕ntie.
(See also the extended file function ⎕xnnames.)

Syntax: result ← ⎕nnames

Result:
The explicit result is a character matrix that contains one file identifier per row. The rows of the result have the same ordering as the result of ⎕nnums. The system displays the names as they were defined when they were tied.

Example:

```
      ⎕nnames
B:\PERSONS.ASF
PRIMES.INT
C:\MAIN\LETTER.TXT
..\W2F.DOC
```

Note: Using the traditional file functions, you are limited in names you can use. For new programming, the extended file functions provide more capability and flexibility.

⎕nnums Tie Numbers of Tied Native Files

Purpose:
Returns the file tie numbers of all the files currently tied with ⎕ntie.
(See also the extended file function ⎕xnnums.)

Syntax: result ← ⎕nnums

Result:
The explicit result is a numeric vector of file tie numbers, in the same order as the result of ⎕nnames.

Examples:

```
      ⎕nnums
¯27 ¯52 ¯3 ¯37 ¯4

      ⎕nuntie ⎕nnums
      ρ⎕nnums
0
```

Note: Using the traditional file functions, you are limited in names you can use. For new programming, the extended file functions provide more capability and flexibility.

⎕novalue No Value

Purpose:
This session-related system function is provided to explicitly return a no-value result in the :RETURN statement and the :RES clause of the :RETURNIF statement.

Syntax: :RETURN ⎕novalue
:RETURNIF condition :RES ⎕novalue

Effect:
It can only be used as an argument to the :RETURN statement or the :RES clause of :RETURNIF statement, overriding whatever value might have been previously set for the declared result variable. For example:

```
       :RETURN
       :RETURN result
       :RETURN ⎕NOVALUE
```

```
    :RETURNIF condition
    :RETURNIF condition :RES result
    :RETURNIF condition :RES ⎕NOVALUE
```

You cannot :RETURN with value from a function that does not declare a result variable (that gives you a VALUE ERROR). It also gives a VALUE ERROR when used in all other contexts. In other words, it is just like referencing a undefined name, in all contexts other than as return argument.

⎕nread Read from a Native File

Purpose:
Reads data from a native file and returns values of the data as the result.

Syntax: result ← ⎕nread tieno datatype count startbyte

Arguments:
tieno is a negative integer file tie number;
datatype is an integer that designates how the system interprets the bit patterns of the data it reads;
count is a positive, integer-valued number less than or equal to 2147483647 bytes that designates how many elements of datatype the system should read;
startbyte is a positive number that represents the starting byte at which the system should begin reading, in origin 0.

The argument must be a numeric vector of three or four elements, of which only the first element is negative. The argument designates the file by its tie number, the datatype, the element count of the result, and the byte at which the system should begin reading (optional). If you omit the starting location, the system assumes it to be the current pointer position (where the last native file operation on this file finished or at the beginning of the file if there has been no ⎕nread, ⎕nreplace, or ⎕nappend since the file was last tied). The datatypes for ⎕nread are:

Value	Meaning
11	Read one bit per element; result Boolean; in this case, *count* is the number of bytes to read, not the number of bits
82	Read one byte per element; result character
163	Read two bytes per element; result returned as 32-bit integer
323	Read four bytes per element; result integer
645	Read eight bytes per element; result floating-point

Result:
The explicit result is an APL vector containing the values of the data from the file. The system reads the bit patterns of the file data and returns them as the specified type of APL data.

Example:

```
      ⎕nread ¯12 82 57 0
THIS FILE CONTAINS SALES DATA FOR 2001. CREATED 1/26/01.
```

⎕nrename Rename Native File

Purpose:
Changes the name of a native file, optionally moving it to a different directory on the same disk drive. (See also the extended file function ⎕xnrename, which supersedes this function.)

Syntax: 'fileid.ext' ⎕nrename tieno

Arguments:
fileid is a valid native file name;
ext is the file extension;
tieno is a negative integer file tie number.
The left argument specifies the new identifier for the file. It must be a vector that contains a valid file name, optionally preceded by a disk drive letter and path name. The right argument must be a negative integer that designates the file you want to rename.

Effect:
The system renames the file tied to the tie number in the right argument to the name specified in the left argument. If a file with the specified name already exists, the system signals FILE NAME ERROR. ⎕nrename is an extension to the DOS RENAME command. It changes the name of a file, but cannot move the file to a different disk drive. However, it can move the file to another directory on the same disk drive.

Example:

```
      'C:\APL\TOOLS\TEST.AWS' ⎕ntie ¯1
      'C:\APL\NEW\TEST2.AWS' ⎕nrename ¯1
      'DATA' ⎕ntie ¯66
      'OLDDATA' ⎕nrename ¯66
```

Note: Using the traditional file functions, you are limited in names you can use. For new programming, the extended file functions provide more capability and flexibility.

⎕nreplace Replace Native File Data

Purpose:
Stores a new value in an existing native file storage space, replacing the data already there.

Syntax: value ⎕nreplace tieno startbyte

Arguments:
value is any simple, homogeneous APL array;
tieno is a negative integer file tie number;
startbyte is a positive integer-valued number less than 4 petabytes (up to 4503599627370495) that represents the starting byte where the system is to place the new data.

The left argument is the simple, homogeneous value you want to store in the file. It can be of any rank, shape, or datatype, but ⎕nreplace ravels the value before storing it.

The right argument must be a two-element, numeric vector with a negative first element and a non-negative second element. The first element designates the file in which you want to replace data. The second element is the location where you want to start replacing data.

Effect:
The system changes the value of the designated storage space in the file. If the storage from the specified startbyte to the end of the file is insufficient for the specified value, the system extends the file to accommodate it.

Warning:
⎕nreplace may not immediately place data on the disk. It may wait until there are more data or until the file is untied using ⎕nuntie or)off.

Example:

```
    BLOCK←⎕nread ¯33 323 10 1048520
    BLOCK
23 7 1984 ¯22 79 22 48 41 68 82
    BLOCK[3]←1982
    BLOCK ⎕nreplace ¯33 1048520
```

⎕nresize Resize Native File

Purpose:
Alters the size of a native file on the disk. ⎕nresize is useful for preallocating space to reduce fragmentation or for discarding material at the end of the file.

Syntax: newsize ⎕nresize tieno

Arguments:
newsize is the number of bytes of disk storage to be allocated to this file, up to 4503599627370495 (less than 4 petabytes);
tieno is a negative integer file tie number.
The left argument to ⎕nresize is the new file size in bytes. It should be a non-negative integer-valued number that is less than the available space on disk.
The right argument must be a negative integer that designates the file you want to resize.

Effect:
⎕nresize changes the amount of disk space that is allocated for the file. If the new size is smaller than current size, any material beyond the new end point in the file is lost. Any subsequent use of ⎕nappend adds material at the new end of the file.

If the new size is larger than the current size, the contents of additional storage added to the end of the file is undefined.

Caution:
⎕nresize, as described here, is specific to this APL system; in particular size values in other APL+ systems were limited to integers.

Examples:
Reserve 32 kilobytes on the disk for use in the file tied to ¯312.

```
   32768 ⎕nresize ¯312
```

Discard all data in the file tied to ¯17, but preserve its directory entry.

```
   0 ⎕nresize ¯17
```

⎕nsize Native File Size Information

Purpose:
Returns a scalar that shows the amount of disk storage occupied by a file.

Syntax: result ← ⎕nsize tieno

Argument:
tieno is a negative integer file tie number that designates the file for which you want the size.

Result:
The explicit result of ⎕nsize is a numeric scalar that indicates the total disk storage (in bytes) that the file uses.

Caution:
⎕nsize, as described here, is specific to this APL system; in particular, the result may be a floating point number, which in other APL systems was an integer.

Example:

```
      'A:PRIMES.SF' ⎕ntie ¯37
      ⎕nsize ¯37
```

⎕ntie Tie Native File

Purpose:
Establishes a file tie for a native file.
(See also the extended file function ⎕xntie, which supersedes this function.)

Syntax:

```
'fileid.ext' ⎕ntie tieno
'fileid.ext' ⎕ntie tieno [openmode]
             ⎕ntie tieno
```

Arguments:
fileid is a valid native file name;
ext is the file extension;
tieno is a negative integer file tie number;
openmode is an optional valid integer DOS open mode.

The optional left argument must be a valid native file identifier (or a device that the operating system recognizes as a native file). Note that APL64 does not use library numbers for native file operations. The extension is an optional, user-chosen mnemonic classification; it can consist of any characters allowed by the operating system.

The right argument is a negative integer scalar or a two-element, integer vector, of which the first element is negative. The argument designates the file tie number and, optionally, a DOS open mode. The open mode allows access to native files by multiple users or processes. The open mode values consist of the sum of one code that indicates the type of access you want and a second code that indicates the type of access you grant to others while you have the file tied. If another user has opened the file first, the access you want must be compatible with the access granted by the other user, and the access you grant must be compatible with the access that user requested.

If you specify open mode explicitly, the domain for this function is a value 0 through 255 inclusive; however, as of DOS 3.3, the only reasonable values are the sum of one value from the Access Requested column plus one non-zero value from the Access granted column in the table below. Thus, if you want to be able to read and write to a file but allow other users read-only access while you have the file tied, you would specify 34 for open mode.

```
    'C:\DRAFT\PROPOSAL.TXT' ⎕ntie ¯1 34
```

Note that the word "reasonable" above, has limited scope. If you specify open mode 49, any user could write to the file, but no user could read it.

Access Requested	Access granted to other users
0 = read access	16 = no access allowed (exclusive)
1 = write access	32 = read access allowed, write access denied
2 = read and write access	48 = write access allowed, read access denied
	64 = read and write access allowed

If a second user attempts to tie the file with an open mode that is incompatible with the open mode specified by a first user, the system signals a FILE ACCESS ERROR. The question of compatibility involves both the access requested value and the access granted value. If the first user includes 16 in open mode, then no second user can tie the file. If the first user specifies 32 through 34, then the second user can request only read access (zero); any value that includes 1 or 2 results in the error. If the first user specifies 48, 49 or 50, then the second user can request only write access (1). If the first user specifies 64 through 66, the second user can request read and/or write access.

The second user must also specify an access granted value that is compatible with the first user's requested access. If the first user has read-only access, the second user cannot include 48 in his open mode value. If the first user has write access, the second user cannot include 32. If the first user requested both read and write access, the second user must include 64. A second user cannot successfully specify exclusive access.

If you specify one of the access requested values, but do not specify an access granted value (that is, open mode of zero, 1 or 2), the effect for APL is the same as specifying exclusive access; another instance of APL cannot tie the file. If you do not specify open mode at all, the default value is 66. APL64 tries open mode 64 if an ACCESS DENIED error occurs. (Except see the historical note below.)

Note that you can also trigger an error for access reasons unconnected with another user, for example, requesting write access (including 1 or 2) on a file which the operating system considers to be read-only.

Monadic ⎕ntie takes a tie number as the right argument and returns a two-element integer vector indicating the file tie state. The value is the same as the open mode arguments to the dyadic version of ⎕ntie and indicates what the current open mode of the file is. If an open mode argument was specified when the file was tied, the result of monadic ⎕ntie is that value. If no open mode argument was specified the return value is dependent on whether read or write access is granted and the value of "Network" specified in the APL configuration file. The value of the second element is applicable only to component file ties. Therefore, the second element value is meaningless for native file ties.
Historical Baggage:

In the days of PC DOS, there was an option called "Network" that you could (can) specify in the APLW.INI file. By default, when not specified, this is "on." There is no known situation where it is currently useful to specify this option, but it remains in the system to protect old code. If "Network" is off, and you do not specify openmode, the system attempts read/write access with an exclusive tie, which blocks anyone else from tying the file and fails if another has the file tied in any mode. If ⎕ntie fails in this situation, the system does not retry.

Effect:
The system ties the native file. You can retie a file that you already have tied without first using ⎕nuntie. The tie number can be the same number or a different number. The only restrictions are that no other file can already be tied with the new tie number. You can use this "slippery" tie to ensure that a file is tied without looking up its name in ⎕nnames.

If the needed access is not available or if the open mode is not acceptable to the operating system, a FILE ACCESS ERROR results. A slippery tie closes the file and reopens it, possibly with a different specified open mode. Another user could conceivably open the file at that precise moment and block the retie.

Caution: By manipulating directories with ⎕chdir and ⎕libd, you conceivably can tie the same file to two different tie numbers without the system detecting it. The result of this condition can be unpredictable; you should avoid creating it.

Examples:

```
      'A:PRIMES.INT' ⎕ntie ¯37
      'C:\SPREAD\PRINT.OUT' ⎕ntie ¯1 34
```

Note: Using the traditional file functions, you are limited in names you can use. For new programming, the extended file functions provide more capability and flexibility. The system function ⎕xntie operates similarly, but not identically to this function, with long file names. The differences include the domain, the error messages, and the default behavior when open mode is not specified.

⎕nuntie Untie Native Files

Purpose:
Unties designated files that were tied using ⎕ntie or ⎕xntie.

Syntax: ⎕nuntie tieno1 tieno2 tieno3 . . . tienon

Argument:
tienoi are negative integer file tie numbers.
The argument designates the file tie numbers of the files tied with ⎕ntie to be untied. The argument can be a scalar or a vector of negative integer file tie numbers. The system does not check if elements of the argument are duplicates or if they designate tied files. An empty vector is permitted as an argument and does not affect any file ties.

Effect:
The system unties any files tied to any of the tie numbers in the argument, and updates their directories on disk. Untying native files ensures that the system writes all data to the files and places the updated file directories on the disk.

Warning:
If you writing to files on a floppy disk, to avoid damaging your files, always make sure that the disk that was in the disk drive when you tied a file is in that disk drive when you untie that file or perform any action that forces a write, such as)cmd or ⎕cmd.

Examples:

```
      ⎕nuntie ¯33
      ⎕nuntie ⎕nnums
```

⎕odbc Interface to the Open Database Connectivity (ODBC) Interface

Purpose:
Provides a structured query language (ODBC)-based interface to a relational database. To learn more about ⎕odbc in the APL64 Developer version go to Help | APL Language | SQL System Functions | Using ⎕ODBC.

⎕os Operating System

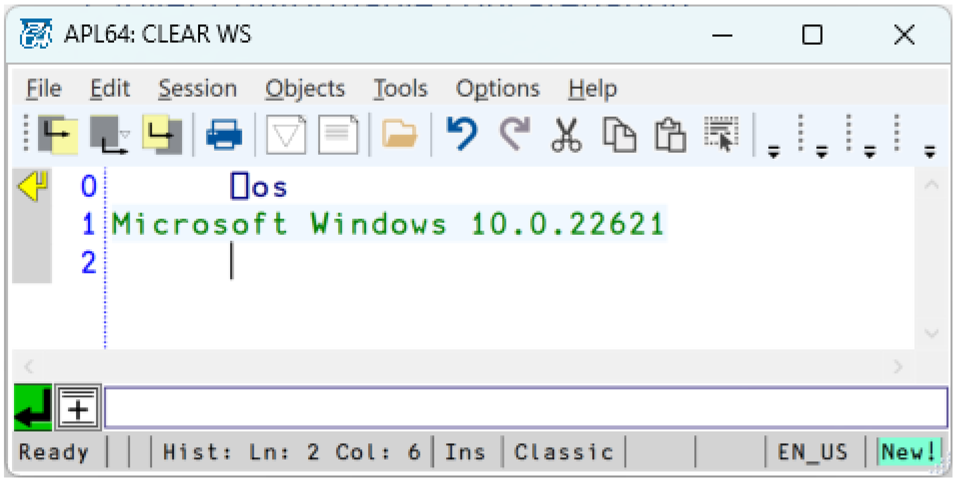
Purpose:
Returns the operating system.

Syntax: result ← ⎕os

Domain:
The result is a character vector containing the operating system on which APL64 is running.

Result:
The result will depend on the specific operating system.

Example:


APL64: CLEAR WS
File Edit Session Objects Tools Options Help

```
      ⎕os
Microsoft Windows 10.0.22621
```

Ready | | Hist: Ln: 2 Col: 6 | Ins | Classic | | EN_US | New!

⎕over Conformable concatenation

Purpose:

Laminates one array over another array.

Syntax: result ← ⎕over array1

result ← array2 ⎕over array1

Arguments:

array1 and array2 are any array of character arrays or scalars or strings.

Results:

The result is array1 when used monadically.

The result is array2 laminated over array1 when used dyadically.

Remarks:

⎕over R: Returns R

L ⎕over R: Laminates R under L and returns result

Examples:

```
      ⎕over 'abcd' (⍳10)
 abcd  1 2 3 4 5 6 7 8 9 10
      100 ⎕over 'abcd' (⍳10)
  100              0
 abcd  1 2 3 4 5 6 7 8 9 10
      100 ⎕over «abcd»
 100
abcd
```

⎕overv Vector Conformable Concatenation

Purpose:
Consecutive laminations of one array over another array.

Syntax: result ← ⎕overv R
result ← L ⎕overv R

Arguments:
R and L are any array of character arrays or scalars or strings.

Results:
For monadic case, the result is:

- 0=ρρR: Returns R
- 1<ρρR: RANK ERROR
- 1=ρρR: Consecutive lamination of elements of R:
  R[1] ⎕over R[2] ⎕over R[2] …

For dyadic case, the result returns L ⎕over ⎕overv R.

Examples:

```
      1 2⎕overv 'ABCD' 'DE' 'PQRST'
1 2 0 0 0
A B C D
D E
P Q R S T
      (3 1ρ'abc') ≡ ⎕←⎕overv 'abc'
a
b
c
1
      (4 3ρ'abc',,3 3↑3 1ρ1 2 3)≡⎕←'abc'⎕overv 1 2 3
a b c
1 0 0
2 0 0
3 0 0
1
      100 ⎕overv 'abcd'
100
  a
  b
  c
  d
      ⎕overv 'ABCD' 'DE' 'PQRST'
ABCD
DE
PQRST
      X←2 3ρι6⋄⎕overv X
RANK ERROR: ⎕overV: Right arg: Rank not 0 or 1
```

```
[imm] ⋄ ⎕overV X
         ^
         X←2 3⍴⍳6⋄⎕overv ⊂X
   1 2 3
   4 5 6
         1 2⎕overv 'ABCD' 'DE' 'PQRST'
1 2 0 0 0
A B C D
D E
P Q R S T
```

⎕path Interface to .NET Path class

Purpose:
Performs operations on String instances that contain file or directory path information.

Syntax: result ← ⎕path '?'
result ← [larg] ⎕path action

Arguments:
larg are the optional arguments to the action.

action is a text vector that determines how the system function will operate on the left and right argument.

Result:
The result, if any, depends on the action called and is operating system dependent.

Remarks:
Refer to the detailed documentation for ⎕path in **Help | APL Language | Using ⎕PATH**.

⎕pcopy Protected Copy from Saved Workspace

Purpose:
Copies APL functions and variables from a saved workspace into the active workspace provided the object does not already exist. This function is available in a runtime application, but the copied workspace must be a runtime workspace.

Syntax: result ← ⎕pcopy 'wsid'
result ← 'namelist' ⎕pcopy 'wsid'

Arguments:
wsid is a workspace name. The system assumes the default extension (.ws64). You must use the file extension .w3 to load an APL+Win workspace. When copying an APL+Win workspace file name that includes one or more spaces, you must append a semi-colon to the long file name.

namelist is a list of functions and variables to copy either a character matrix with one name per row or a character vector with names separated by one or more blanks.

Result:
The result is an integer vector of codes that represents the success or failure of ⎕pcopy. If you specify

namelist, result contains a response code for each object in namelist. The table below shows the values used in the result vector.

Response Codes for the Result of ⎕pcopy

Code	Explanation
2	A variable was copied successfully.
1	A function was copied successfully.
0	No object was copied; none was found with the supplied name.
¯1	An object with this name already exists in the workspace.
¯2	The object was too large to copy given the available free workspace.
¯4	There is insufficient space in the symbol table to copy this object and the workspace is too full to expand the symbol table.
¯6	The amount of workspace available is too small to perform the copy.
¯7	The object was not copied due to an INTERFACE CONVERSION ERROR.

If you use ⎕pcopy without specifying namelist, then result is empty if the system successfully copies everything from wsid. If not, everything is not copied, the system may return a single error code. If an unanticipated error occurs, the system does not return a result.

Effect:
Copies functions and variables from the local environment of the specified workspace into the local environment of the active workspace unless they would replace any objects by the same name. The system copies a label as a variable only if the function in the source workspace is pendent or suspended and the name does not exist in the active workspace. The system recognizes a label in the active workspace only when its function is pendent or suspended.
Copying a function copies only its source form; the system discards all precompiled internal code and clears ⎕stop and ⎕trace settings in the function.

Note: You cannot copy a workspace that was saved in APL+DOS.

Examples:
The example below shows that two of the five requested objects were successfully copied, but the value of MTRX did not change. Compare this function to ⎕copy, which would change the value of MTRX.

```
    MTRX
1 2
3 4
    'MTRX FUN X DAT SPNDLAB' ⎕pcopy 'WS3'
¯1 1 0 2 ¯1
    )vars
DAT MTRX
     MTRX
1 2
3 4
```

⎕pdist Probability Distribution

Purpose:
Provides the Samples action to obtain sample values for the supported probability distributions using an APL64 programmer-selected pseudo-random number generators.

For detailed documentation use the **Help | APL Language | Using ⎕PDIST** menu item in the APL64 developer version.

⎕penclose Partitioned Enclose

Purpose:
Creates a nested vector from portions of an array. This function is a cover function for the Evolution Level 1 behavior of dyadic Enclose (⊂). You can use this system function at any Evolution Level. At Evolution Level 2, dyadic Enclose is the Partition function.

Syntax: result ← boolvec ⎕penclose array
result ← boolvec ⎕penclose[i] array

Arguments:
array is an array that contains the data you want to rearrange; it can have any type, rank, and shape;
i is a non-negative, integer-valued scalar that indicates the axis along which you want the function to enclose the subarrays; the default is along the last dimension.

boolvec is a Boolean vector that is used to partition the right argument into items of the result; it must be the same length as the selected dimension of the right argument. Each 1 represents the beginning of an element in result; each zero following the first 1 represents a corresponding subarray of array that is nested with its predecessor.

Effect:
The system starts partitioning with the subarray of array that corresponds to the first 1 in the left argument. It groups that subarray with as many subsequent subarrays as there are zeroes following the 1. With each 1 in the left argument, the system creates another item in the result; it encloses the subarrays represented by each 1 and the subsequent zeroes to form each element of the result.

Result:
The result is a nested vector with items selected and enclosed in the pattern specified by boolvec. Those subarrays of array corresponding to leading zeroes are omitted. The length of result is equal to +/boolvec.

Examples:

```
      ]display 0 0 1 0 1 1 0 0 ⎕penclose ⍳8
.→-----------------.
|.→--. .→. .→----.|
||3 4| |5| |6 7 8||
|'~--' '~' '~----'|
'∊-----------------'

      ]display 1 ⎕penclose 1 2 3
.→-----------.
|.→. .→. .→.|
||1| |2| |3||
|'~' '~' '~'|
'∊-----------'
```

```
      ]display 1 0 1 ⎕penclose[1] 3 4⍴⍳12
.→---------------------.
|.→------.  .→---------.|
|↓1 2 3 4|  ↓9 10 11 12||
||5 6 7 8|  '~---------'|
|'~------'              |
'∊---------------------'
```

⎕pf Key Bindings

Purpose:

The ⎕PF system function may be used to associate a keyboard shortcut with a user-defined APL executable statement for the convenience of the APL64 programmer.

Syntax: result ← ⎕pf 0
result ← ⎕pf keyCode
⎕pf keyCode actionText flag

Arguments:

keycode is an integer scalar which is the sum of these values:

- Virtual key code: Keys Enum (System.Windows.Forms) | Microsoft Docs
- Shift code: 100000
- Control code: 200000
- Alt code: 400000

actionText is a character vector containing the associated user-defined APL executable statement or the name of a user-defined function

flag is a Boolean scalar indicating the disposition of the user-defined APL executable statement when the keyboard shortcut is used:

- False(0):

The actionText is executed when the keyboard shortcut is used. The history will be updated to indicate that the ⎕PF-defined keyboard shortcut was used and the result, if any, of the execution of the actionText will be used to update the history. The current content, if any, of the command line is not considered.

- True(1):

If the actionText contains ⎕TCNL, the actionText will be inserted into the command line at the current caret offset in the command line and the updated content of the command line will be executed. The history will be updated to indicate that the ⎕PF-defined keyboard shortcut was used and the result, if any, of executing the updated content of the command line, will be used to update the history.

If the actionText does not contain ⎕TCNL, the actionText will not be executed. The action text will be inserted into the applicable APL64 Developer version GUI control at the current caret offset in that GUI control:

- If the keyboard focus is on a function editor, a text editor, the editable classic history pane or the command line, the actionText will be inserted into that GUI control.
- If the keyboard focus is on the command line, non-editable history pane, debugger pane or state indicator pane, the actionText will be inserted into the command line.

Effect:

- The ⎕PF system function is available only in the APL64 Developer version.
- ⎕PF keyboard shortcut definitions are transient, so they are not preserved between APL64 Developer version instances.
- ⎕PF keyboard shortcuts take precedence over user tool keyboard shortcuts.
- A ⎕PF keyboard shortcut does not interact with a programmer-developed ⎕WI application system interface.
- The Windows operating system, language, keyboard definition and installed software may affect available shortcut key combinations.

Result:
The result depends on the arguments supplied. The history will be updated when ⎕PF is used.

When ⎕PF 0:

- If no ⎕PF keyboard shortcut definitions have been created the result is a zero-row, three-column matrix.
- If ⎕PF keyboard shortcut definitions have been created the result is a three-column matrix with one row for each existing ⎕PF keyboard shortcut definition

When ⎕PF intScalar:

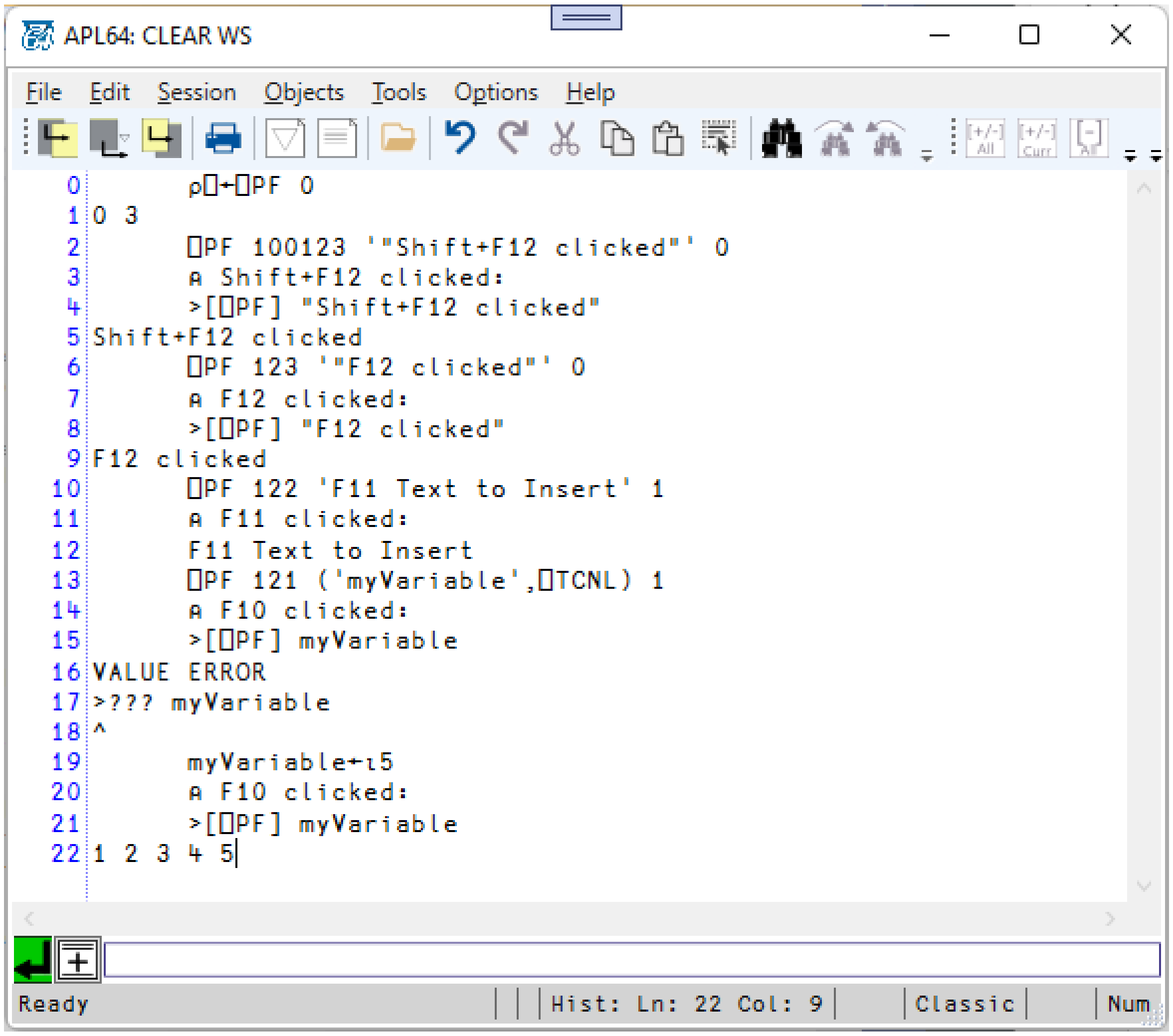
- If intScalar is positive and is the Win32 keyboard shortcut integer code of an existing ⎕PF keyboard shortcut definition, the result is a one row, three column matrix containing the definition
- If intScalar is positive and not a Win32 keyboard shortcut integer code of an existing ⎕PF keyboard shortcut definition, the result is a zero row, three column matrix
- If intScalar is negative and its absolute value is the Win32 keyboard shortcut integer code of an existing ⎕PF keyboard shortcut definition, that ⎕PF keyboard shortcut definition will be deleted and the result is 0 0⍴0

When ⎕PF intScalar charVec boolScalar:

- The result is 0 0⍴0 if successful.
- If a previously-defined, ⎕PF keyboard shortcut definition exists, that definition will be replaced by the updated definition.

Examples:

A new instance of the APL64 Developer version is started and the editable classic history is selected.

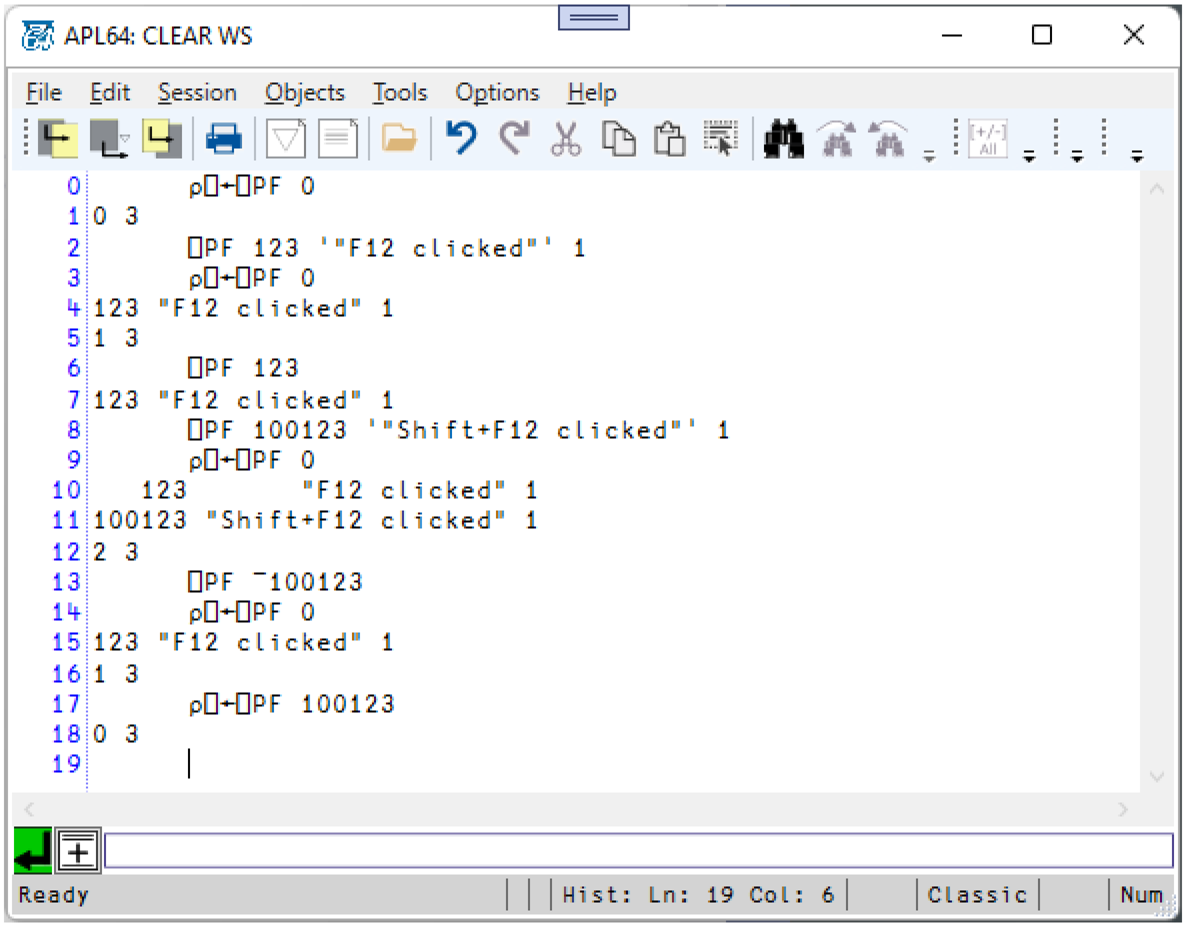


```
      ρ⎕←⎕PF 0
0 3
      ⎕PF 100123 '"Shift+F12 clicked"' 0
      ⍝ Shift+F12 clicked:
      >[⎕PF] "Shift+F12 clicked"
Shift+F12 clicked
      ⎕PF 123 '"F12 clicked"' 0
      ⍝ F12 clicked:
      >[⎕PF] "F12 clicked"
F12 clicked
      ⎕PF 122 'F11 Text to Insert' 1
      ⍝ F11 clicked:
      F11 Text to Insert
      ⎕PF 121 ('myVariable',⎕TCNL) 1
      ⍝ F10 clicked:
      >[⎕PF] myVariable
VALUE ERROR
>??? myVariable
^
      myVariable←ι5
      ⍝ F10 clicked:
      >[⎕PF] myVariable
1 2 3 4 5
```

```
Ready    Hist: Ln: 22 Col: 9    Classic    Num
```

Example 2:

A new instance of the APL64 Developer version is started and the editable classic history is selected.

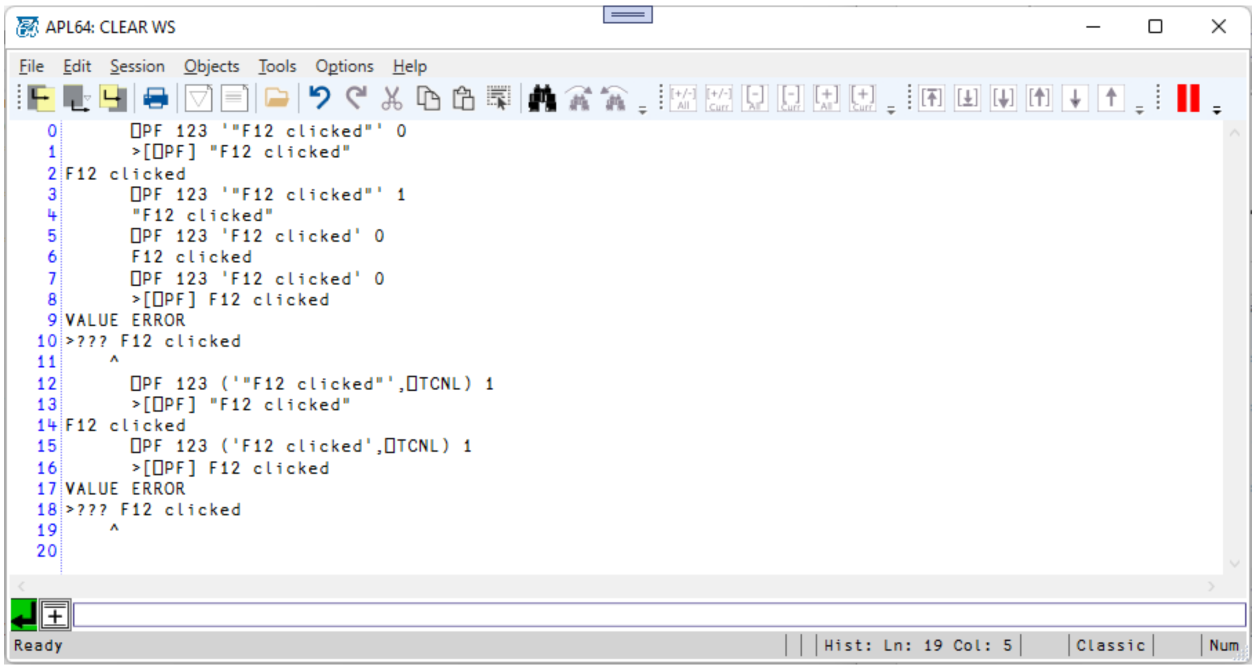


```
      ⍴⎕←⎕PF 0
0 3
      ⎕PF 123 '"F12 clicked"' 1
      ⍴⎕←⎕PF 0
123 "F12 clicked" 1
1 3
      ⎕PF 123
123 "F12 clicked" 1
      ⎕PF 100123 '"Shift+F12 clicked"' 1
      ⍴⎕←⎕PF 0
   123       "F12 clicked" 1
100123 "Shift+F12 clicked" 1
2 3
      ⎕PF ¯100123
      ⍴⎕←⎕PF 0
123 "F12 clicked" 1
1 3
      ⍴⎕←⎕PF 100123
0 3
      
```

```
Ready      Hist: Ln: 19 Col: 6      Classic      Num
```

Example 3:

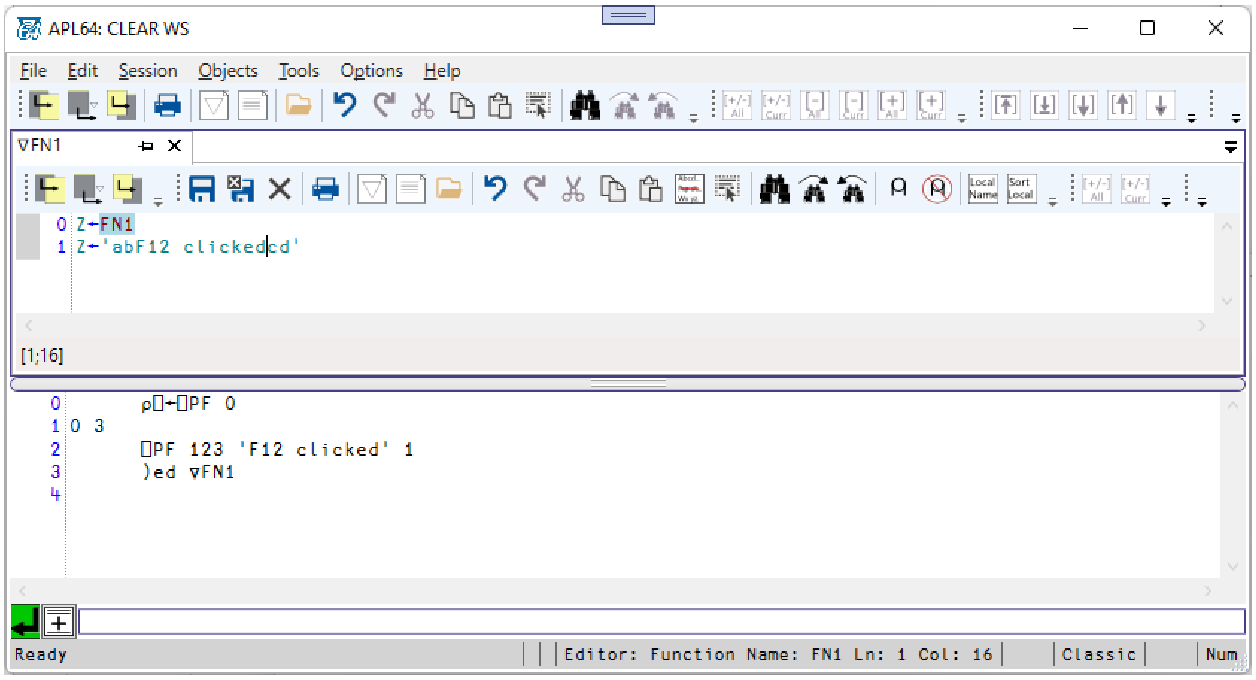
A new instance of the APL64 Developer version is started and the editable classic history is selected. When the third argument of ⎕PF(flag) is zero, execution of the second argument of ⎕PF (actionText) will be attempted. When the third argument of ⎕PF(flag) is one and the second argument of ⎕PF(actionText) contains ⎕TCNL , execution of the second argument of ⎕PF (actionText) will be attempted.



```
      ⎕PF 123 '"F12 clicked"' 0
      >[⎕PF] "F12 clicked"
F12 clicked
      ⎕PF 123 '"F12 clicked"' 1
      "F12 clicked"
      ⎕PF 123 'F12 clicked' 0
      F12 clicked
      ⎕PF 123 'F12 clicked' 0
      >[⎕PF] F12 clicked
VALUE ERROR
>??? F12 clicked
    ^
      ⎕PF 123 ('"F12 clicked"',⎕TCNL) 1
      >[⎕PF] "F12 clicked"
F12 clicked
      ⎕PF 123 ('F12 clicked',⎕TCNL) 1
      >[⎕PF] F12 clicked
VALUE ERROR
>??? F12 clicked
    ^
```

Example 4:

A new instance of the APL64 Developer version is started. Keyboard focus is on the function editor, with caret offset after 'b' in the function text. F11 is clicked, so the text 'F12 clicked' is inserted into the function text at the caret offset:

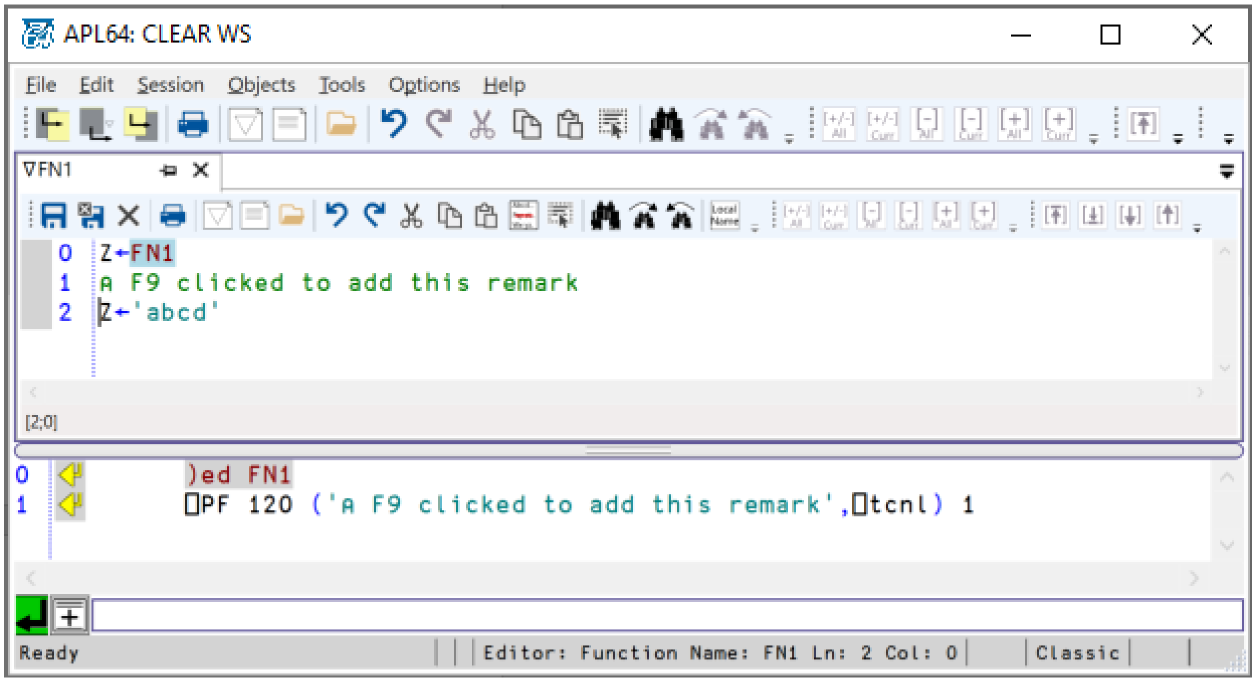


```
Z←FN1
Z←'abF12 clickedcd'
```

```
      ρ⎕←⎕PF 0
0 3
      ⎕PF 123 'F12 clicked' 1
      )ed ∇FN1
```

Example 5:

A new ⎕PF short-cut definition is created for the F9 key, keyboard focus is on the function editor, caret offset is at the beginning of line [1] of the function and F9 is clicked, so the remark is added to the function text:



```
Z←FN1
⍝ F9 clicked to add this remark
Z←'abcd'
```

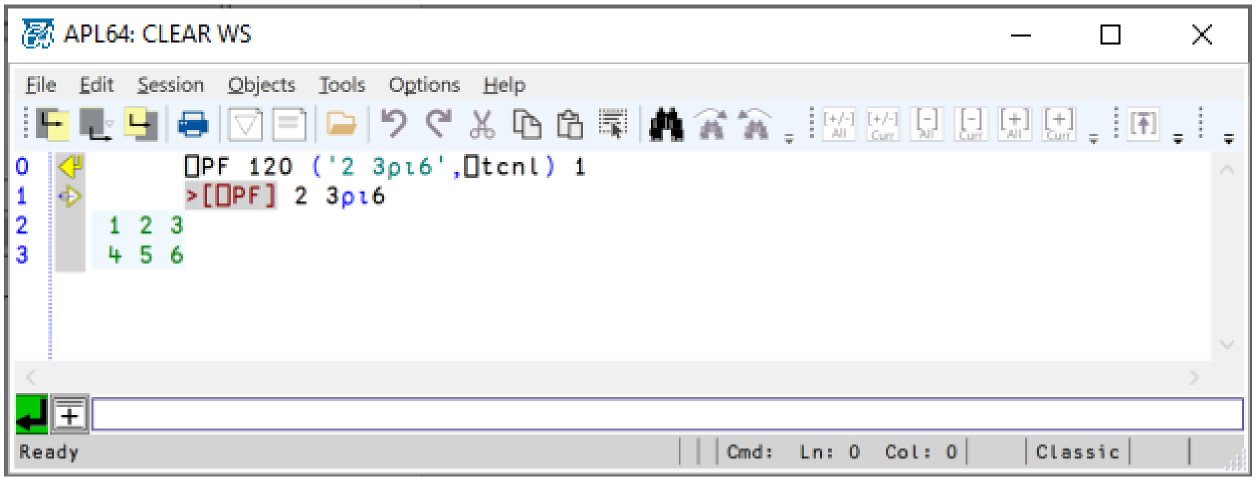
[2;0]

```
      )ed FN1
      ⎕PF 120 ('⍝ F9 clicked to add this remark',⎕tcnl) 1
```

Ready | Editor: Function Name: FN1 Ln: 2 Col: 0 | Classic

Example 6:

The ⎕PF short-cut definition for the F9 key is replaced, keyboard focus is on the empty command line and F9 is clicked, so the APL statement, 2 3⍴⍳6, is executed:



```
      ⎕PF 120 ('2 3⍴⍳6',⎕tcnl) 1
      >[⎕PF] 2 3⍴⍳6
 1 2 3
 4 5 6
```

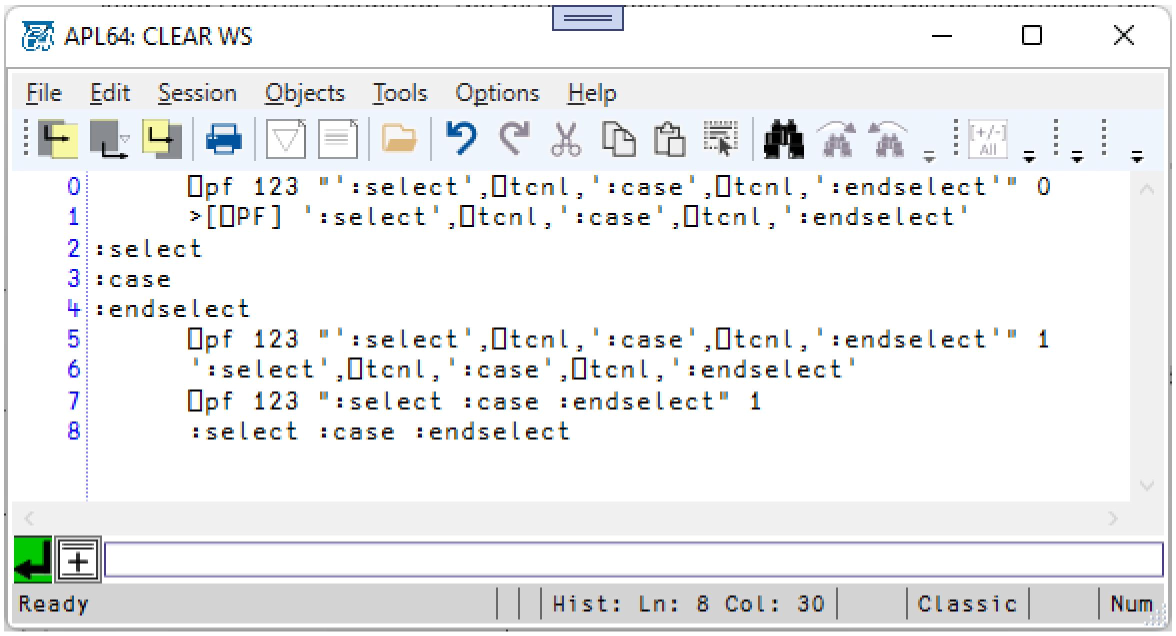
Ready | Cmd: Ln: 0 Col: 0 | Classic

Example 7:

Editable classic history is user-selected.

The ⎕PF definition for keycode 123 is defined so that execution of the actionText will be executed when the keyboard shortcut is used.

The ⎕PF definition for keycode 123 is changed so that insertion of the actionText instead of execution of the action text will occur when the keyboard shortcut is used. ⎕TCNL within the text of the actionText will not cause execution of the actionText.



```
      ⎕pf 123 "':select',⎕tcnl,':case',⎕tcnl,':endselect'" 0
      >[⎕PF] ':select',⎕tcnl,':case',⎕tcnl,':endselect'
:select
:case
:endselect
      ⎕pf 123 "':select',⎕tcnl,':case',⎕tcnl,':endselect'" 1
      ':select',⎕tcnl,':case',⎕tcnl,':endselect'
      ⎕pf 123 ":select :case :endselect" 1
      :select :case :endselect
```

```
Ready | | Hist: Ln: 8 Col: 30 | Classic | Num
```

⎕profile Profile Performance

Purpose:
Used to obtain a fine-grained performance profile of APL64 programmer-selected actions.

For detailed documentation use the **Help | APL Language | Using ⎕Profile** menu item in the APL64 developer version.

⎕psave Save Workspace without Replacement

Purpose:
Saves the active workspace under program control without halting execution but after checking that saving the workspace will not replace an existing workspace with the same name.

Syntax: [RESET] ⎕psave 'wsid'
⎕psave 'wsid'

Arguments:
wsid is a character scalar, vector, or one-row matrix that specifies the name to identify the saved workspace; the system appends the default extension (.ws64).

The optional left argument, a character vector that contains the value 'RESET', indicates that the system should save the workspace with a clear state indicator.

Effect:
Creates a copy of the active workspace as a file on disk with the name wsid.ws64, unless a file with that name already exists, without halting execution of the APL statement in which the command appears. Monadic ⎕psave produces a saved workspace with execution suspended at the start of the function line at the top of the state indicator at the time it is called. Dyadic ⎕psave saves the workspace with a clear state indicator.

If the argument you supply, wsid, is different from the workspace name, ⎕wsid, the system changes the system variables ⎕wsid, ⎕wsts, and ⎕wsowner for the active workspace as a side effect of ⎕psave. If a workspace already exists on disk with the supplied name, the system produces a WS NAME ERROR. The table below shows the differences among the system commands)save and)psave and the system commands ⎕save and ⎕psave.

Results of Save Operations

⎕wsid	C:\MYWS	C:\MYWS	C:\OTHER
Condition on disk	C:\MYWS.ws64 exists	C:\MYWS.ws64 does not exist	C:\MYWS.ws64 exists
Operation	**Result**	**Result**	**Result**
)save myws	SAVED	SAVED	SAVED
)psave myws	WS NAME ERROR: Workspace already exists	SAVED	WS NAME ERROR: Workspace already exists
⎕save 'myws'	SAVED	SAVED	SAVED
⎕psave 'myws'	WS NAME ERROR: Workspace already exists	SAVED	WS NAME ERROR: Workspace already exists

Notes: If you specify one of the system commands without an argument, the results are the same as the first two columns. If ⎕wsid is empty (CLEAR WS), the results are the same as the third column.

Examples:
This example uses monadic ⎕psave to checkpoint a running application. Localizing ⎕wsid allows ⎕psave to change the workspace identifier while the function is running but leaves the global value of ⎕wsid unchanged. Localizing ⎕lx to exit the function preserves the state of the workspace at the time it was saved.

```
    ∇ GRINDATA MyApp
      . . .
[m]    :for i :in iteration
      . . .
[n]    CHECKPOINT iteration
[n+1]   :endfor
 ∇

   ∇ CHECKPOINT n;⎕wsid;⎕lx
[1]   MyAppTemp←⎕wsid,on
[2]   ⎕lx←'→0' ⋄ ⎕psave MyAppTemp
   ∇
```

⎕qqdefault Quote-Quad input default user response text

Purpose:
Sets the default user response text in the quote-quad query.

Syntax:
```
                         ⎕qqdefault buffer
result ← [LeftArg] ⎕qqdefault prompt
```

Argument:
buffer is a character vector that specifies the default response text displayed in the quote-quad query. The value of this ‘buffer’, up to the first ⎕tcnl, or the entire ‘buffer’ value if it does not contain ⎕tcnl, is the default response to the next Quad or Quote-Quad query. The remaining value of the ‘buffer’, if any, is retained for subsequent Quad or Quote-Quad queries.

LeftArg is the optional scalar identifying the action target. The possible values are

value	Action on Internal ‘buffer’
1	Append right argument to ‘buffer’
0	Replace ‘buffer’ with right argument
¯1	Prepend right argument to ‘buffer’
2	Return the current value of ‘buffer’ (Right argument, e.g. '', ignored)

Result:
The result, if any, depends on the action called.

Effect:

⎕QQDefault renders no output to the APL64 developer version history pane when a left argument of ¯1, 0 or 1 is used.

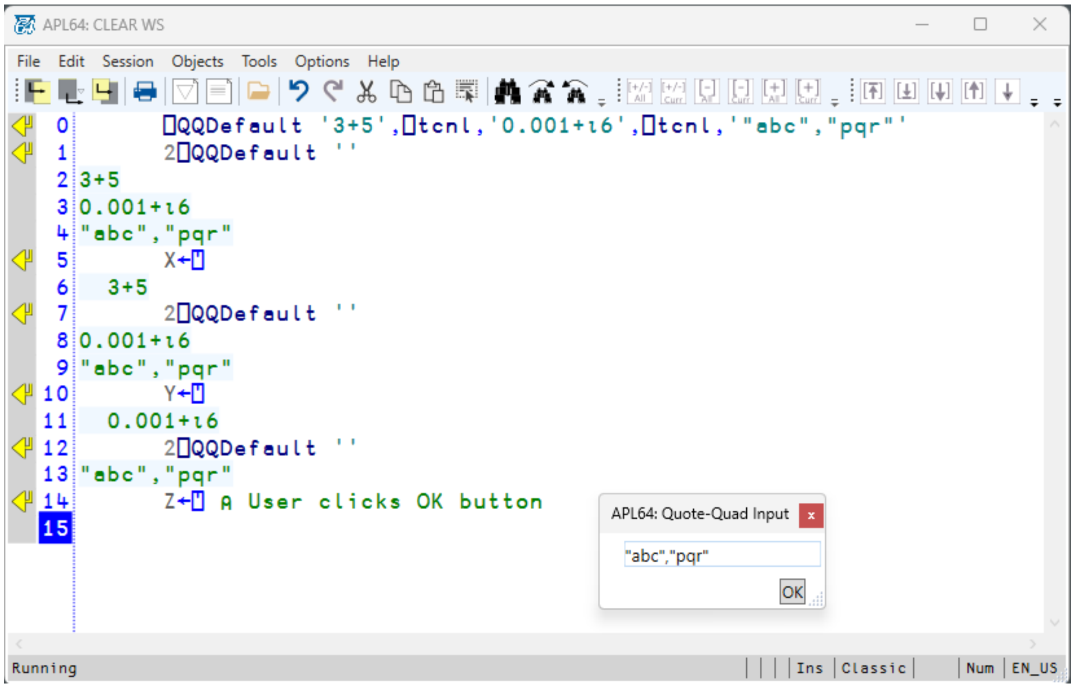
⎕QQDefault is available only in the APL64 developer version. The value of the ‘buffer’ does not persist when the current APL64 developer version instance ends.

Note: Refer to the menu Help | Developer Version GUI | Quad and Quote-Quad for additional information on this function.

Example:

```
⎕QQDefault '3+5',⎕tcnl,'0.001+⍳6',⎕tcnl,'"abc","pqr"'
2⎕QQDefault ''
X←⍞
2⎕QQDefault ''
Y←⍞
2⎕QQDefault ''
Z←⍞ ⍝ User clicks OK button
''≡2⎕QQDefault ''
```

Before user closes the Quote-Quad dialog:

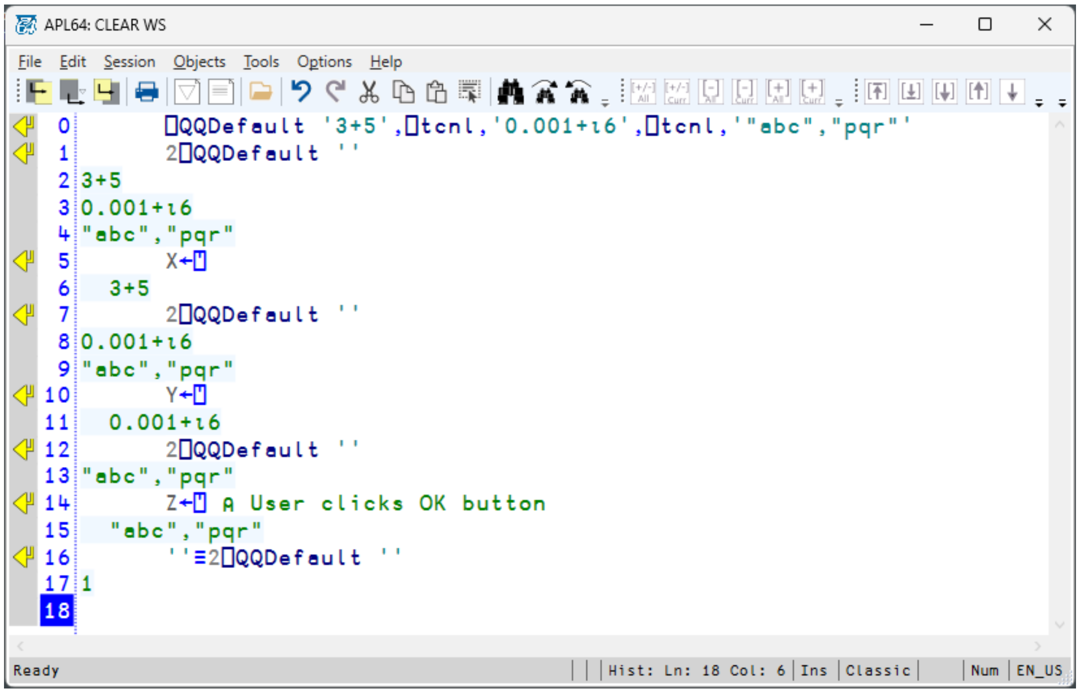


```
      ⎕QQDefault '3+5',⎕tcnl,'0.001+⍳6',⎕tcnl,'"abc","pqr"'
      2⎕QQDefault ''
3+5
0.001+⍳6
"abc","pqr"
      X←⍞
   3+5
      2⎕QQDefault ''
0.001+⍳6
"abc","pqr"
      Y←⍞
   0.001+⍳6
      2⎕QQDefault ''
"abc","pqr"
      Z←⍞ ⍝ User clicks OK button
```

```
"abc","pqr"
```

```
Running | Ins | Classic | Num | EN_US
```

After the user closes the Quote-Quad dialog:



```
      ⎕QQDefault '3+5',⎕tcnl,'0.001+⍳6',⎕tcnl,'"abc","pqr"'
      2⎕QQDefault ''
3+5
0.001+⍳6
"abc","pqr"
      X←⍞
   3+5
      2⎕QQDefault ''
0.001+⍳6
"abc","pqr"
      Y←⍞
   0.001+⍳6
      2⎕QQDefault ''
"abc","pqr"
      Z←⍞ ⍝ User clicks OK button
   "abc","pqr"
      ''≡2⎕QQDefault ''
1
```

```
Ready | Hist: Ln: 18 Col: 6 | Ins | Classic | Num | EN_US
```

⎕qload Quietly Load a Workspace

Purpose:

Loads a workspace under program control without displaying the saved message.

Syntax: ⎕qload 'wsid'

Argument:
wsid is a character scalar or vector that specifies the workspace to be loaded; the system assumes the default extension (.ws64). (You can use the file extension .w3 to load an APL+Win workspace.) If you omit the directory name or library number, the system uses the default directory.

Effect:
Replaces the active workspace with a copy of the contents of the designated workspace. The system does not display a SAVED message.

When you use ⎕qload, the new active workspace begins execution automatically if ⎕lx is set appropriately, giving the effect of continuing a multistep program through two or more workspaces. You can exchange information between the two workspaces by storing data in a file while in one workspace and then reading the data back while in another workspace.

Example:
In the example below, the point is that no SAVED message displays when you load a workspace with this function. The last two lines show that the system did load the specified workspace.

```
      )clear
CLEAR WS...
      ⎕qload 'STAGE2'
      ⎕wsid
C:\APL\STAGE2
```

⎕repl Replicate Items in an Array

Purpose:
Replicates selected items in a array. This function provides the same behavior that is derived by the Replicate operator (/) with a variable as the left argument. You can use this system function at any Evolution Level.

Syntax:
```
result ← repl ⎕repl    array
result ← repl ⎕repl[i]  array
```

Arguments:
repl is an integer scalar, or integer vector where the number of non-negative elements equals the length of the chosen dimension of array;

i is a non-negative, integer-valued scalar that indicates the dimension along which you want to replicate; array is any APL array; if the chosen dimension has length 1, arg is extended along that axis to the number of non-negative elements in the operand (+/repl≥0). Similar to ⎕expand, you can generate only fill elements with a non-positive operand and an argument that is a scalar or has length 1 along the chosen dimension.

Result:
The result is an array that contains each item of array replicated the number of times specified by the corresponding non-negative value in repl; for negative values in repl, the array is extended along the chosen axis by inserting the number of fill items equal to the absolute value of the negative element. ⎕repl replicates the last dimension of the argument array by default.

Examples:

```
     0 1 2 ⎕repl 'JMO'
MOO

     A←2 3ρ'ABCDEF'
     1 ¯1 2 ¯2 3⎕repl A
A BB  CCC
D EE  FFF
     0 1 ⎕repl[1] A
DEF
     1 ¯1 2 ⎕repl[1] A
ABC

DEF
DEF

    0 ¯2 0 ⎕repl 5
0 0
    0 ¯2 0 ⎕repl[1] 2 3ρι3
 0 0 0
 0 0 0
```

Note: At Evolution Level 1, there was a difference in the result for appropriate arguments between the expressions larg /¨ rarg and larg (/)¨rarg. The latter took a nested left argument, each item being an integer vector appropriate for the corresponding item of rarg. At Evolution Level 2, the syntax(/)¨ generates an EVOLUTION ERROR. Use ⎕repl¨ when you want to use Replicate with different arguments for each item of a nested array.

```
     nest←(1 2) (3 4)
     1 2 ⎕repl nest
 1 2  3 4  3 4
     1 2 / nest
 1 2  3 4  3 4
     1 2 /¨ nest
 1 2 2  3 4 4
     1 2 ⎕repl¨ nest
 1 2  3 3 4 4
     (1 2) (1 2) ⎕repl¨ nest
 1 2 2  3 4 4
     (1 2) (3 4) ⎕repl¨ nest
 1 2 2  3 3 3 4 4 4 4
     (1 2) (2 3 4) ⎕repl¨ (5 6) (7 8 9)
 5 6 6  7 7 8 8 8 9 9 9 9
```

⎕rjust Right Justify

Purpose:
Returns the character elements in an array or string right justified.

Syntax: result ← ⎕rjust array

Arguments:
array is character array or scalar or string.

Results:
The result is the character elements in the array or string right justified.

Effects:
The ⎕rjust system function will not throw a DOMAIN ERROR exception for a right argument that is not appropriate for ⎕rjust and instead will return the right argument unmodified if possible.

Examples:

```
      ⎕←heading←2 15⍴'First Name     Last Name      '
First Name
Last Name
      ⎕rjust heading
     First Name
      Last Name
      string←«ljust       »
      string
 ljust
      ⎕rjust string
        ljust
      ⎕rjust (1 2 'XY ') 2.34 'AB '
  1 2  XY  2.34  AB
```

⎕rmdir Remove a Directory

Purpose:
Removes a disk directory.

Syntax: ⎕rmdir 'dir'

Argument:
dir is a character vector that contains the identifier of the disk directory you want to remove.

Effect:
The system erases the directory from the specified or default disk drive. You can remove a directory only if it is empty, and if it is not the current default directory. ⎕rmdir is equivalent to the DOS command RMDIR.

Example:

```
   ⎕rmdir 'C:\TEMP'
```

⎕rnet R Statistical & Graphics Tools

Purpose:
⎕RNET provides a convenient interface between APL64 and the R statistical and graphics toolkit.

To learn more about ⎕RNET in the APL64 Developer version go to **Help | APL Language | Using ⎕RNET**.

⎕round Round Numbers

Purpose:
Provides options for rounding numbers to the nearest integer or to the specified number of fractional digits.

Syntax: result ← [nDigits rndOpt] ⎕round array

Argument:
array is a numeric array or scalar; all other datatypes will not be modified.
nDigits is the number of fractional digits for rounding: [0, 15], e.g. 0 indicates rounding to an integer; the default value is 2.
rndOpt is the rounding method to apply; the default value is 1.

rndOpt	Rounding Method
0	MidpointRounding.ToEven
1	MidpointRounding.AwayFromZero
2	MidpointRounding.ToZero
3	MidpointRounding.ToNegativeInfinity
4	MidpointRounding.ToPositiveInfinity

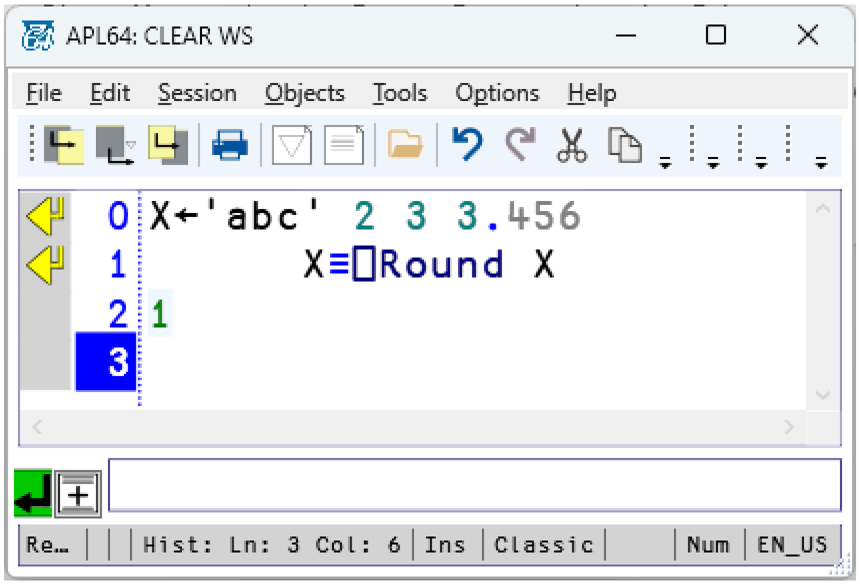
Results:
result is the same type, rank and shape as the argument array.

Examples:

Example #1

If the data type of arg is not Double, the right argument is returned as the result:

```
X←'abc' 2 3 3.456
X≡⎕Round X
```


APL64: CLEAR WS
File Edit Session Objects Tools Options Help

```
X←'abc' 2 3 3.456
      X≡⎕Round X
1
```

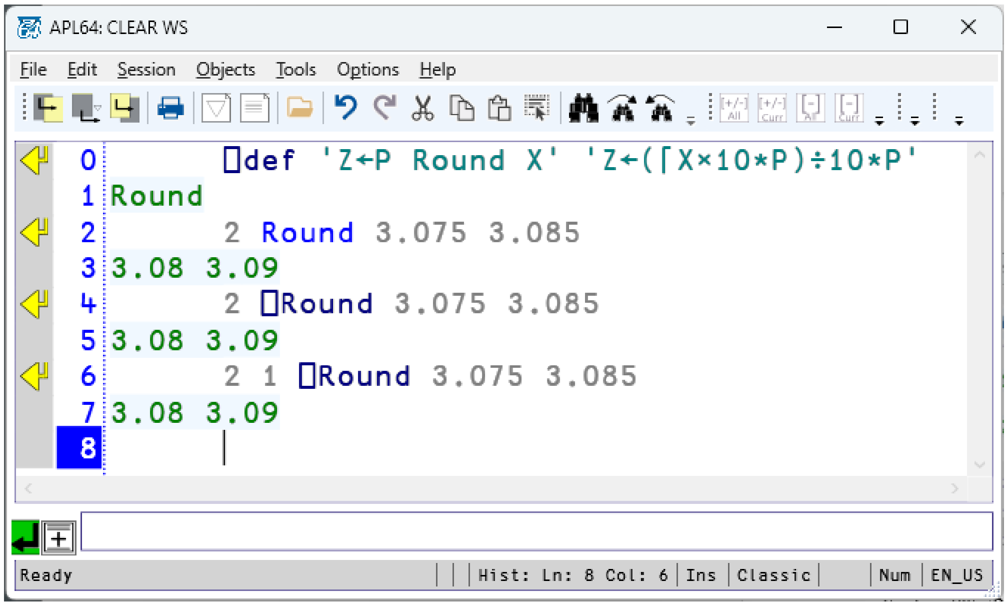
Re… Hist: Ln: 3 Col: 6 Ins Classic Num EN_US

Example #2

Frequently used rounding: res ← (nDigits 1) ⎕Round doubleArg

This rounding is equivalent to the ‘Round’ APL64 programmer-developed function for nDigits ≥ 0 and to res ← nDigits ⎕Round doubleArg

⎕def 'Z←P Round X' 'Z←(⌈X×10*P)÷10*P'
2 Round 3.075 3.085
2 ⎕Round 3.075 3.085
2 1 ⎕Round 3.075 3.085


APL64: CLEAR WS
File Edit Session Objects Tools Options Help

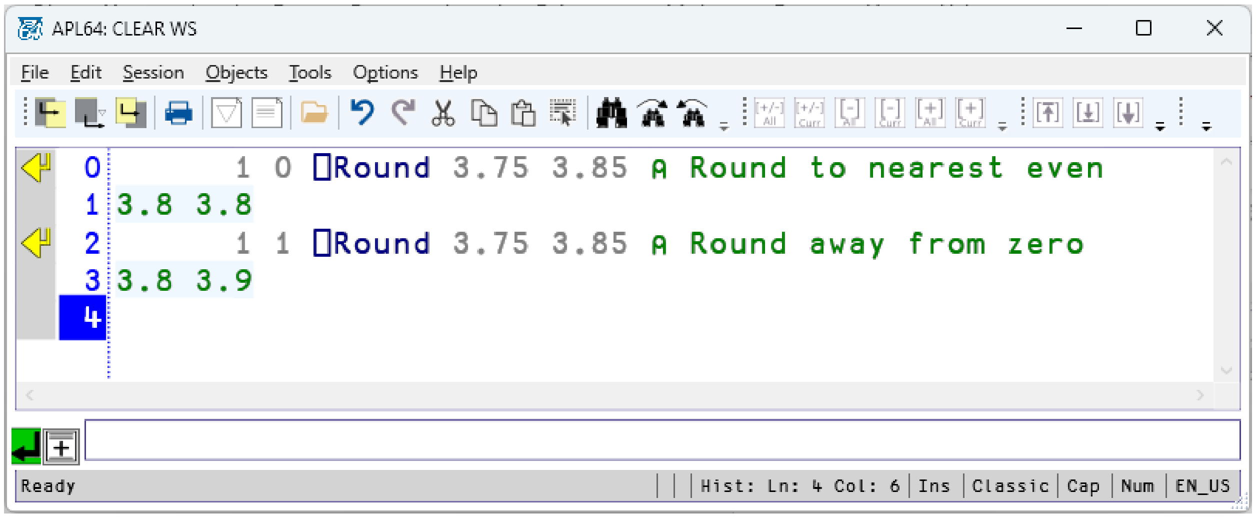
```
      ⎕def 'Z←P Round X' 'Z←(⌈X×10*P)÷10*P'
Round
      2 Round 3.075 3.085
3.08 3.09
      2 ⎕Round 3.075 3.085
3.08 3.09
      2 1 ⎕Round 3.075 3.085
3.08 3.09
```

Ready Hist: Ln: 8 Col: 6 Ins Classic Num EN_US

Example #3

The rndOp argument determines the rounding algorithm used:

```
1 0 ⎕Round 3.75 3.85 ⍝ Round to nearest even
1 1 ⎕Round 3.75 3.85 ⍝ Round away from zero
```

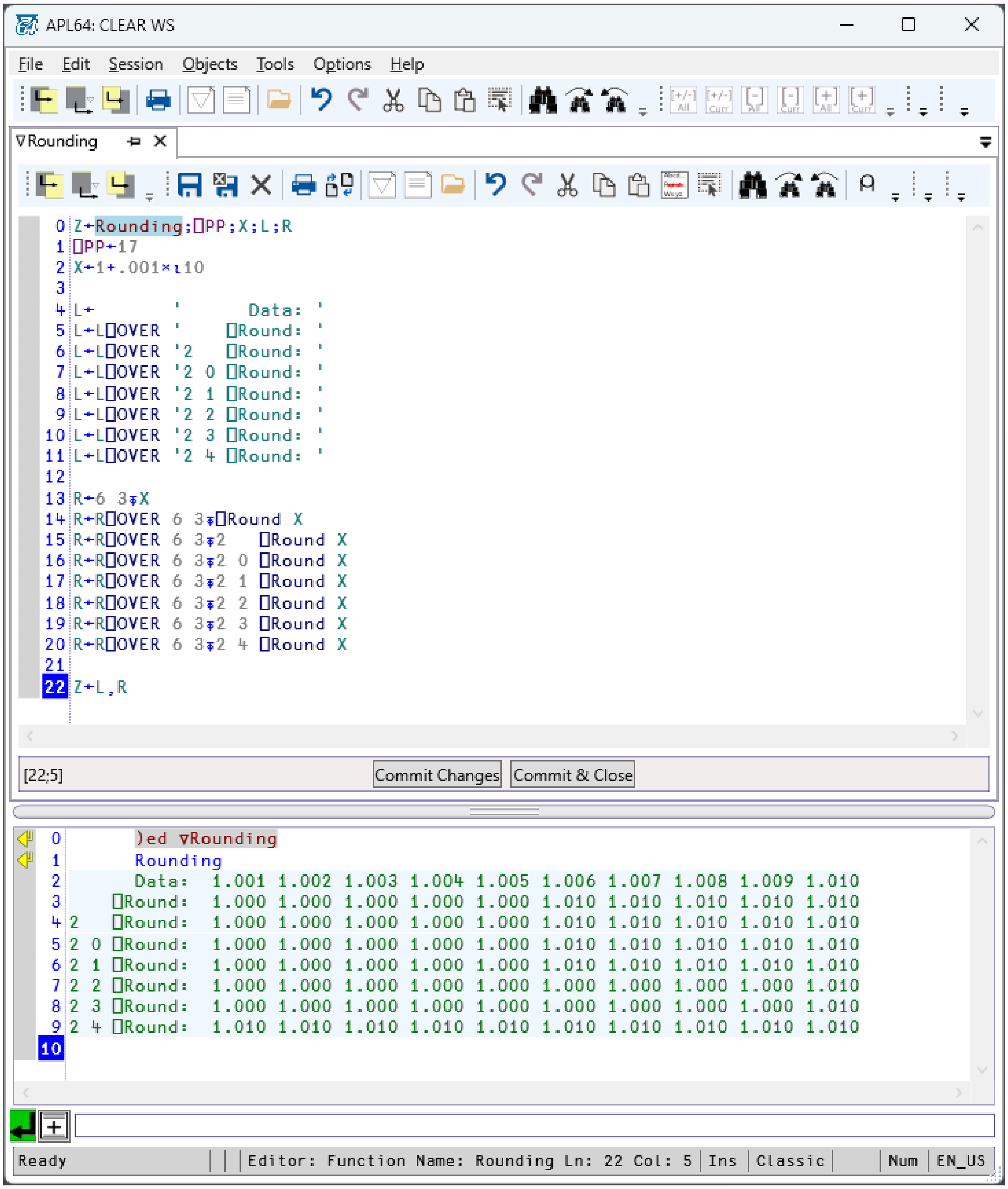


```
      1 0 ⎕Round 3.75 3.85 ⍝ Round to nearest even
3.8 3.8
      1 1 ⎕Round 3.75 3.85 ⍝ Round away from zero
3.8 3.9
```

Example #4

Create the Rounding function and run it.

```
Z←Rounding;⎕PP;X;L;R
⎕PP←17
L←     '      Data: '
L←L⎕OVER '    ⎕Round: '
L←L⎕OVER '2   ⎕Round:
L←L⎕OVER '2 0 ⎕Round: '
L←L⎕OVER '2 1 ⎕Round: '
L←L⎕OVER '2 2 ⎕Round: '
L←L⎕OVER '2 3 ⎕Round: '
L←L⎕OVER '2 4 ⎕Round: '
R←6 3⍕X←1+.001×⍳10
R←R⎕OVER 6 3⍕⎕Round X
R←R⎕OVER 6 3⍕2   ⎕Round X
R←R⎕OVER 6 3⍕2 0 ⎕Round X
R←R⎕OVER 6 3⍕2 1 ⎕Round X
R←R⎕OVER 6 3⍕2 2 ⎕Round X
R←R⎕OVER 6 3⍕2 3 ⎕Round X
R←R⎕OVER 6 3⍕2 4 ⎕Round X
Z←L,R
```



APL64: CLEAR WS

∇Rounding

```
Z←Rounding;⎕PP;X;L;R
⎕PP←17
X←1+.001×⍳10

L←          '      Data: '
L←L⎕OVER '     ⎕Round: '
L←L⎕OVER '2    ⎕Round: '
L←L⎕OVER '2 0 ⎕Round: '
L←L⎕OVER '2 1 ⎕Round: '
L←L⎕OVER '2 2 ⎕Round: '
L←L⎕OVER '2 3 ⎕Round: '
L←L⎕OVER '2 4 ⎕Round: '

R←6 3⍕X
R←R⎕OVER 6 3⍕⎕Round X
R←R⎕OVER 6 3⍕2   ⎕Round X
R←R⎕OVER 6 3⍕2 0 ⎕Round X
R←R⎕OVER 6 3⍕2 1 ⎕Round X
R←R⎕OVER 6 3⍕2 2 ⎕Round X
R←R⎕OVER 6 3⍕2 3 ⎕Round X
R←R⎕OVER 6 3⍕2 4 ⎕Round X

Z←L,R
```

[22;5]

```
      )ed ∇Rounding
      Rounding
      Data:  1.001 1.002 1.003 1.004 1.005 1.006 1.007 1.008 1.009 1.010
    ⎕Round:  1.000 1.000 1.000 1.000 1.000 1.010 1.010 1.010 1.010 1.010
2   ⎕Round:  1.000 1.000 1.000 1.000 1.000 1.010 1.010 1.010 1.010 1.010
2 0 ⎕Round:  1.000 1.000 1.000 1.000 1.000 1.010 1.010 1.010 1.010 1.010
2 1 ⎕Round:  1.000 1.000 1.000 1.000 1.000 1.010 1.010 1.010 1.010 1.010
2 2 ⎕Round:  1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.010
2 3 ⎕Round:  1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.010
2 4 ⎕Round:  1.010 1.010 1.010 1.010 1.010 1.010 1.010 1.010 1.010 1.010
```

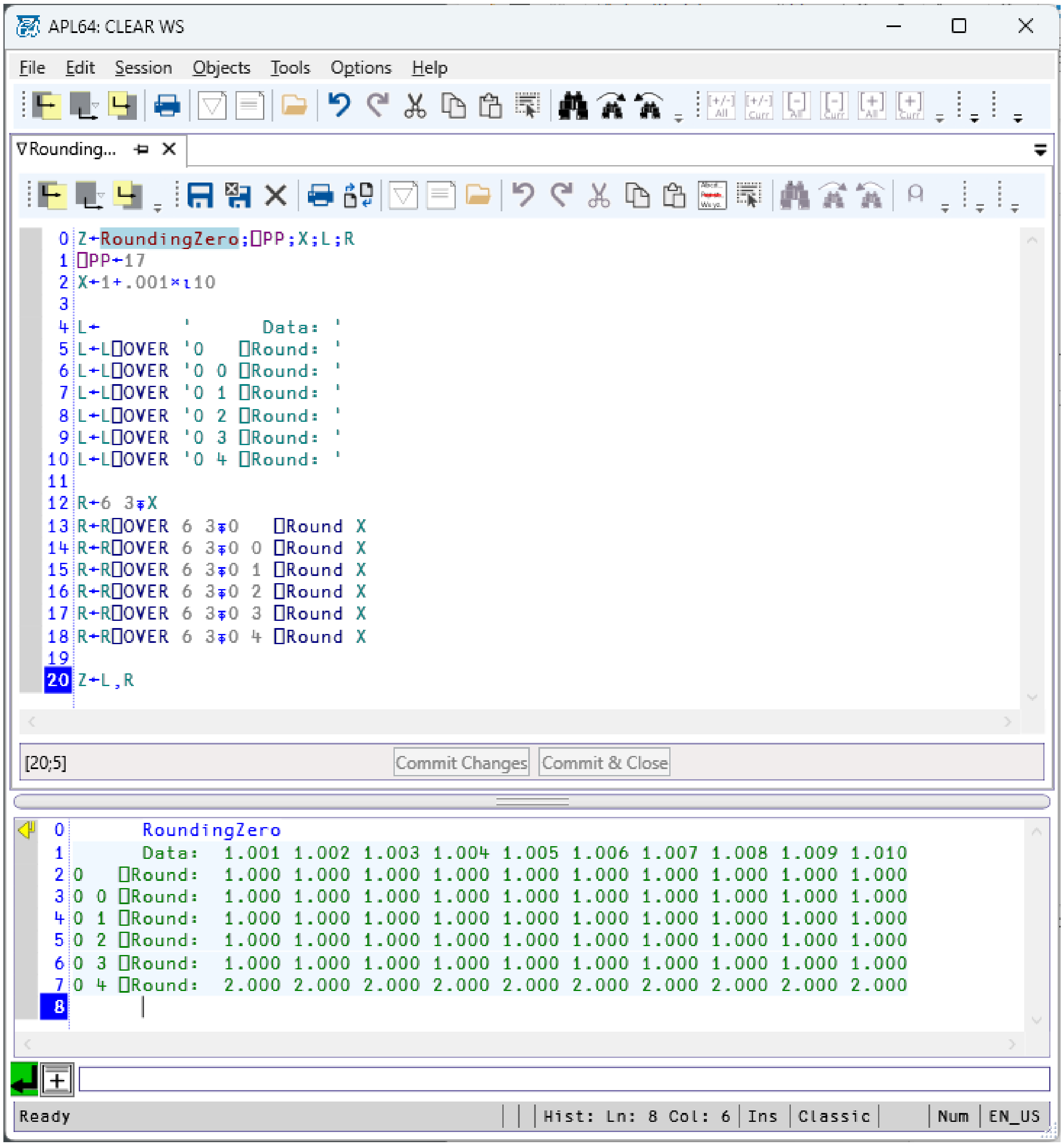
Ready | Editor: Function Name: Rounding Ln: 22 Col: 5 | Ins | Classic | Num | EN_US

Example #5

If nDigits is zero, ⎕Round rounds the right argument to the integers. Create the RoundingZero function and run it.

```
Z←RoundingZero;⎕PP;X;L;R
```

```
⎕PP←17
X←1+.001×⍳10
L←      '     Data: '
L←L⎕OVER '0   ⎕Round: '
L←L⎕OVER '0 0 ⎕Round: '
L←L⎕OVER '0 1 ⎕Round: '
L←L⎕OVER '0 2 ⎕Round: '
L←L⎕OVER '0 3 ⎕Round: '
L←L⎕OVER '0 4 ⎕Round: '
R←6 3⍕X
R←R⎕OVER 6 3⍕0   ⎕Round X
R←R⎕OVER 6 3⍕0 0 ⎕Round X
R←R⎕OVER 6 3⍕0 1 ⎕Round X
R←R⎕OVER 6 3⍕0 2 ⎕Round X
R←R⎕OVER 6 3⍕0 3 ⎕Round X
R←R⎕OVER 6 3⍕0 4 ⎕Round X
Z←L,R
```



```
Z←RoundingZero;⎕PP;X;L;R
⎕PP←17
X←1+.001×⍳10

L←        '      Data: '
L←L⎕OVER '0    ⎕Round: '
L←L⎕OVER '0  0 ⎕Round: '
L←L⎕OVER '0  1 ⎕Round: '
L←L⎕OVER '0  2 ⎕Round: '
L←L⎕OVER '0  3 ⎕Round: '
L←L⎕OVER '0  4 ⎕Round: '

R←6 3⍕X
R←R⎕OVER 6 3⍕0    ⎕Round X
R←R⎕OVER 6 3⍕0  0 ⎕Round X
R←R⎕OVER 6 3⍕0  1 ⎕Round X
R←R⎕OVER 6 3⍕0  2 ⎕Round X
R←R⎕OVER 6 3⍕0  3 ⎕Round X
R←R⎕OVER 6 3⍕0  4 ⎕Round X

Z←L,R
```

```
      RoundingZero
      Data:  1.001 1.002 1.003 1.004 1.005 1.006 1.007 1.008 1.009 1.010
0    ⎕Round:  1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
0  0 ⎕Round:  1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
0  1 ⎕Round:  1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
0  2 ⎕Round:  1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
0  3 ⎕Round:  1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
0  4 ⎕Round:  2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000 2.000
```

⎕rowfind Index of in a Rank-2 Array

Purpose:

Returns the indices of the character array in a character matrix.

Syntax: result ← array ⎕rowfind array

Arguments:

array is any array of character arrays or scalars or strings.

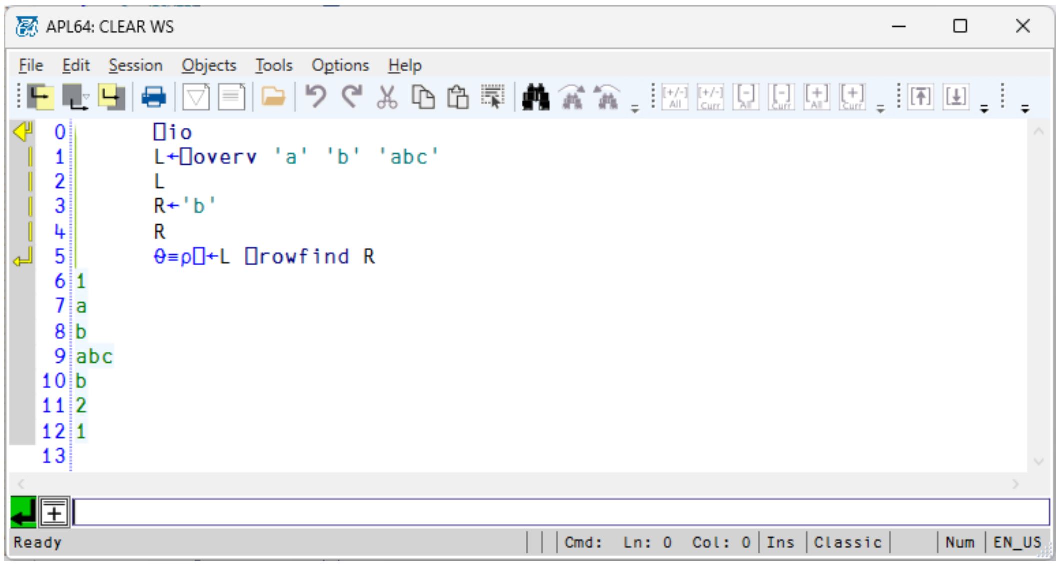
Results:
Result is the index (index origin 1) of the elements of the right argument in the elements of the left argument. An APL-based algorithm is used. For rank two left arguments the array's rows are the elements. Result is a 0 for each right argument element not found among the left argument elements.

Remarks:
⎕rowfind is the replacement for the assembler function ROWFIND in the ASMFNS.w3 workspace in APL+Win.

Examples:

Try It Code:

```
⎕io
L←⎕overv 'a' 'b' 'abc'
L
R←'b'
R
θ≡ρ⎕←L ⎕rowfind R
```

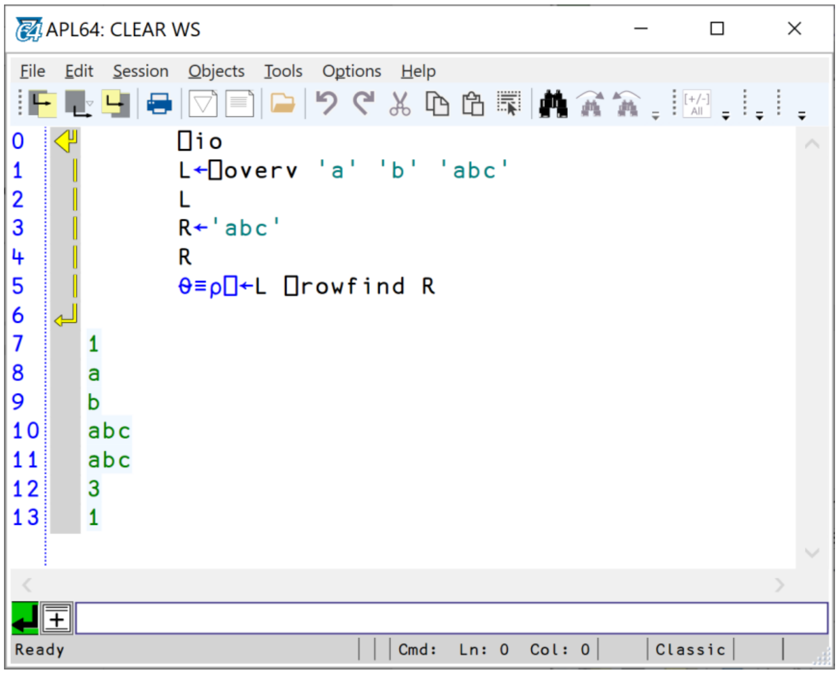


```
      ⎕io
      L←⎕overv 'a' 'b' 'abc'
      L
      R←'b'
      R
      θ≡ρ⎕←L ⎕rowfind R
1
a
b
abc
b
2
1
```

Ready | Cmd: Ln: 0 Col: 0 | Ins | Classic | Num | EN_US

Try It Code:

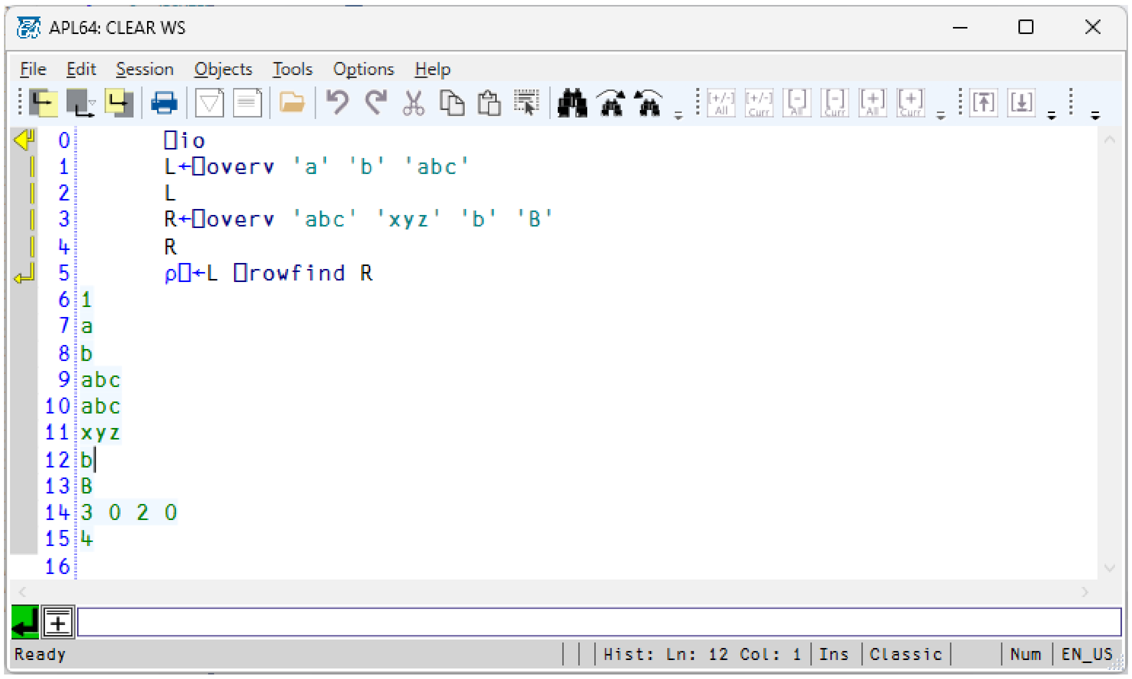
```
⎕io
L←⎕overv 'a' 'b' 'abc'
L
R←'abc'
R
θ≡ρ⎕←L ⎕rowfind R
```



```
      ⎕io
      L←⎕overv 'a' 'b' 'abc'
      L
      R←'abc'
      R
      θ≡ρ⎕←L ⎕rowfind R
1
a
b
abc
abc
3
1
```

Try It Code:

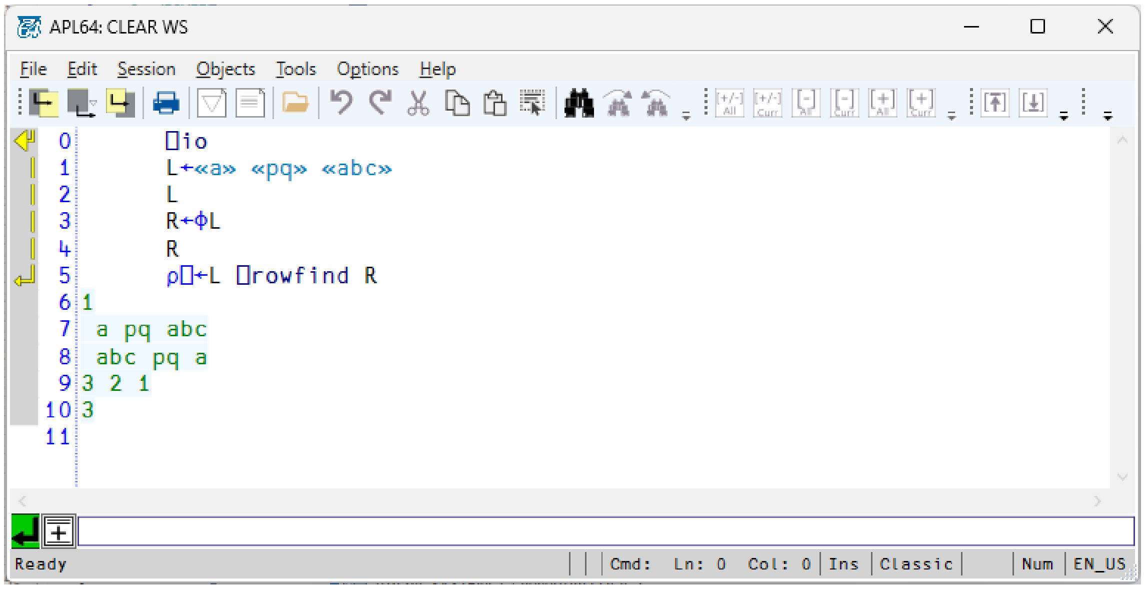
```
⎕io
L←⎕overv   'a'   'b'   'abc'
L
R←⎕overv  'abc'  'xyz'  'b'  'B'
R
ρ⎕←L ⎕rowfind R
```



```
      ⎕io
      L←⎕overv 'a' 'b' 'abc'
      L
      R←⎕overv 'abc' 'xyz' 'b' 'B'
      R
      ρ⎕←L ⎕rowfind R
1
a
b
abc
abc
xyz
b
B
3 0 2 0
4
```

Try It Code:

```
⎕io
L←«a»   «pq»   «abc»
L
R←⌽L
R
ρ⎕←L ⎕rowfind R
```


APL64: CLEAR WS
File Edit Session Objects Tools Options Help
Ready
Cmd: Ln: 0 Col: 0 Ins Classic Num EN_US

```
        ⎕io
        L←«a» «pq» «abc»
        L
        R←⌽L
        R
        ρ⎕←L ⎕rowfind R
1
  a pq abc
  abc pq a
3 2 1
3
```

⎕save Save Workspace with Replacement

Purpose:
Saves the active workspace under program control without halting execution.

Syntax: [RESET]   ⎕save 'wsid'
                        ⎕save 'wsid'

Arguments:
wsid is a character scalar, vector, or one-row matrix that specifies the name to identify the saved workspace; the system appends the default extension (.ws64).
The optional left argument, a character vector that contains the value 'RESET', indicates that the system should save the workspace with a clear state indicator.

Effect:
Saves a copy of the active workspace as a file on disk with the name wsid.ws64 without halting execution of the APL statement in which it appears. Monadic ⎕save produces a saved workspace with execution suspended at the start of the function line at the top of the state indicator at the time it is called. Dyadic ⎕save saves the workspace with a clear state indicator.

If wsid is different from the workspace name, the system updates the system variables ⎕wsid, ⎕wsts, and ⎕wsowner for the current workspace as a side-effect of ⎕save. Note that the effect of ⎕save is different from the effect of)save in that ⎕save does no checking to see if a workspace of the same name exists on disk. See the description of ⎕psave for a way to prevent overwriting an existing workspace and for the differences among the system commands)save and)psave and the system functions ⎕save and ⎕psave.

Example:
This example shows the use of dyadic ⎕save to save a workspace with a clear state indicator.

```
   ∇ INSTALL MyApp
  [1]  '⎕elx' ''RESET' ⎕save
⎕wsid''
    . . .
   ∇
```

⎕scom Serial Communication Function

Purpose:
Provides enhanced RS-232 serial port communication in APL64.

Syntax: result ← '?' ⎕scom ''
result ← 'action' ⎕scom 'portName' [rarg]

Arguments:
action is a text character that determines how the system function will operate on the portName.

portName is a character vector that is the name of the serial port.

rarg are the optional arguments to the action.

Results:
The result, if any, depends on the action called.

Remarks:

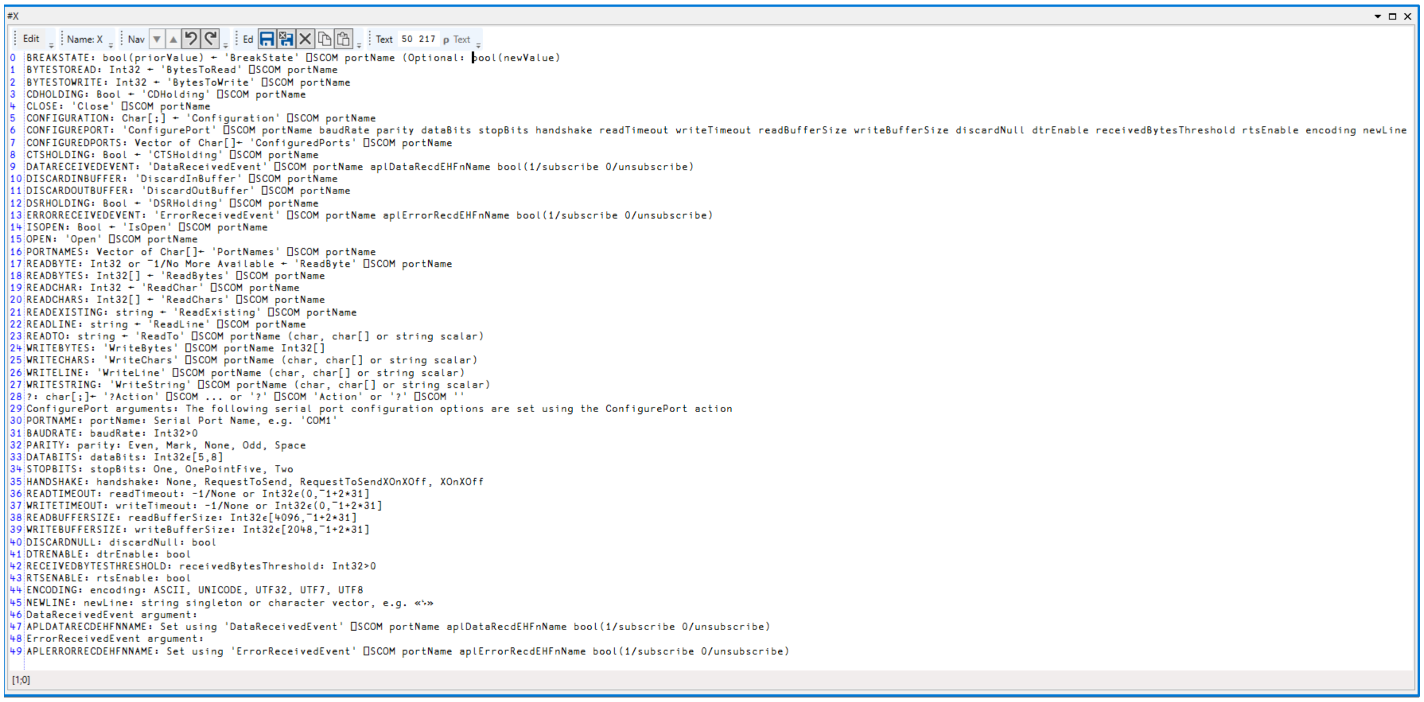
⎕scom Replaces ⎕arbin

The functionality of ⎕scom replaces the RS-232 serial port functionality of ⎕arbin, which is not supported in APL64. This section provides a comparison of typical RS-232 port communication settings in APL64 using ⎕scom and in APL+Win using ⎕arbin. For RS232-ports ⎕scom provides significant additional functionality compared to ⎕arbin.

- In APL64 all right arguments of ⎕scom are required. In APL+Win some arguments of ⎕arbin are optionally defaulted.
- In APL64 the RS-232 ports for reception or transmission are specified by their text names, e.g. 'COM1', 'COM2'. In APL+Win the ⎕arbin 'output' argument is zero for no port or a negative native tie# associated with the selected RS-232 port.
- In APL64 the ⎕scom 'baudRate' is an integer configuration argument. In APL+Win the ⎕arbin baud rate is selected using the DOS MODE command.

- In APL64 using ⎕scom it is possible to wait for reception using the ‘DATARECEIVED’ event and the ‘readTimeout’ and ‘writeTimeout’ arguments. In APL+Win using ⎕arbin there are no events supported and the ‘wait’ argument is ineffective in current Windows versions.
- In APL64 the ⎕scom RS-232 protocol is set explicitly in using the ‘ConfigurePort’ method, including ‘handshake’ for flow control [None, Rts, RtsXonXoff, XonXoff] . In APL+Win the ⎕arbin protocol is set using the DOS MODE command.
- In APL64 the ⎕scom character limits are set explicitly using the ‘readBufferSize’ and ‘writeBufferSize’ arguments. In APL+Win the ⎕arbin ‘charlimit’ parameter is available the reception port.
- In APL64 ending communication with a specific port is done using the ⎕scom ‘Close’ method possibly coordinated with application-specific checking of the received information. In APL+Win the ⎕arbin ‘terminator’ characters may be specified.
- In APL64 the ⎕scom ‘encoding’ argument and the ⎕scom methods for reading and writing data determine how the data will be transmitted and received. In APL+Win the ⎕arbin ‘translation’ argument is provided. .Net does not provide the APL-ASCII translations specific to APL+Win, however such translations are not necessary in APL64:
    - Using the ‘Ascii’ encoding option of ⎕scom, ASCII characters can be transmitted and received
    - Using the ‘Unicode’ encoding option of ⎕scom, APL64 characters can be transmitted successfully
    - Using the ⎕scom ‘ReadByte’, ‘ReadBytes’ and ‘WriteBytes’ method ‘raw untranslated numeric codes’ can be transmitted successfully
- In APL+Win ⎕arbin has additional functionality related to native file access and printers, but the use of this functionality is not recommended in APL+Win. In APL64 and APL+Win the ⎕XN… system functions, e.g. ⎕xntie, ⎕Xnappend, etc., should be used for native file access and the ⎕WI ‘Printer’ class should be used for printing purposes.

Execute '?' ⎕scom '' for a summary of ⎕scom actions and action-specific syntax:


```
BREAKSTATE: bool(priorValue) ← 'BreakState' ⎕SCOM portName (Optional: bool(newValue)
BYTESTOREAD: Int32 ← 'BytesToRead' ⎕SCOM portName
BYTESTOWRITE: Int32 ← 'BytesToWrite' ⎕SCOM portName
CDHOLDING: Bool ← 'CDHolding' ⎕SCOM portName
CLOSE: 'Close' ⎕SCOM portName
CONFIGURATION: Char[;] ← 'Configuration' ⎕SCOM portName
CONFIGUREPORT: 'ConfigurePort' ⎕SCOM portName baudRate parity dataBits stopBits handshake readTimeout writeTimeout readBufferSize writeBufferSize discardNull dtrEnable receivedBytesThreshold rtsEnable encoding newLine
CONFIGUREDPORTS: Vector of Char[]← 'ConfiguredPorts' ⎕SCOM portName
CTSHOLDING: Bool ← 'CTSHolding' ⎕SCOM portName
DATARECEIVEDEVENT: 'DataReceivedEvent' ⎕SCOM portName aplDataRecdEHFnName bool(1/subscribe 0/unsubscribe)
DISCARDINBUFFER: 'DiscardInBuffer' ⎕SCOM portName
DISCARDOUTBUFFER: 'DiscardOutBuffer' ⎕SCOM portName
DSRHOLDING: Bool ← 'DSRHolding' ⎕SCOM portName
ERRORRECEIVEDEVENT: 'ErrorReceivedEvent' ⎕SCOM portName aplErrorRecdEHFnName bool(1/subscribe 0/unsubscribe)
ISOPEN: Bool ← 'IsOpen' ⎕SCOM portName
OPEN: 'Open' ⎕SCOM portName
PORTNAMES: Vector of Char[]← 'PortNames' ⎕SCOM portName
READBYTE: Int32 or ¯1/No More Available ← 'ReadByte' ⎕SCOM portName
READBYTES: Int32[] ← 'ReadBytes' ⎕SCOM portName
READCHAR: Int32 ← 'ReadChar' ⎕SCOM portName
READCHARS: Int32[] ← 'ReadChars' ⎕SCOM portName
READEXISTING: string ← 'ReadExisting' ⎕SCOM portName
READLINE: string ← 'ReadLine' ⎕SCOM portName
READTO: string ← 'ReadTo' ⎕SCOM portName (char, char[] or string scalar)
WRITEBYTES: 'WriteBytes' ⎕SCOM portName Int32[]
WRITECHARS: 'WriteChars' ⎕SCOM portName (char, char[] or string scalar)
WRITELINE: 'WriteLine' ⎕SCOM portName (char, char[] or string scalar)
WRITESTRING: 'WriteString' ⎕SCOM portName (char, char[] or string scalar)
?: char[;]← '?Action' ⎕SCOM ... or '?' ⎕SCOM 'Action' or '?' ⎕SCOM ''
ConfigurePort arguments: The following serial port configuration options are set using the ConfigurePort action
PORTNAME: portName: Serial Port Name, e.g. 'COM1'
BAUDRATE: baudRate: Int32>0
PARITY: parity: Even, Mark, None, Odd, Space
DATABITS: dataBits: Int32∊[5,8]
STOPBITS: stopBits: One, OnePointFive, Two
HANDSHAKE: handshake: None, RequestToSend, RequestToSendXOnXOff, XOnXOff
READTIMEOUT: readTimeout: -1/None or Int32∊(0,¯1+2*31]
WRITETIMEOUT: writeTimeout: -1/None or Int32∊(0,¯1+2*31]
READBUFFERSIZE: readBufferSize: Int32∊[4096,¯1+2*31]
WRITEBUFFERSIZE: writeBufferSize: Int32∊[2048,¯1+2*31]
DISCARDNULL: discardNull: bool
DTRENABLE: dtrEnable: bool
RECEIVEDBYTESTHRESHOLD: receivedBytesThreshold: Int32>0
RTSENABLE: rtsEnable: bool
ENCODING: encoding: ASCII, UNICODE, UTF32, UTF7, UTF8
NEWLINE: newLine: string singleton or character vector, e.g. «␍»
DataReceivedEvent argument:
APLDATARECDEHFNNAME: Set using 'DataReceivedEvent' ⎕SCOM portName aplDataRecdEHFnName bool(1/subscribe 0/unsubscribe)
ErrorReceivedEvent argument:
APLERRORRECDEHFNNAME: Set using 'ErrorReceivedEvent' ⎕SCOM portName aplErrorRecdEHFnName bool(1/subscribe 0/unsubscribe)
```

For more details of ⎕scom method, properties and events see .Net Serial Port class.

Serial Port Default Values

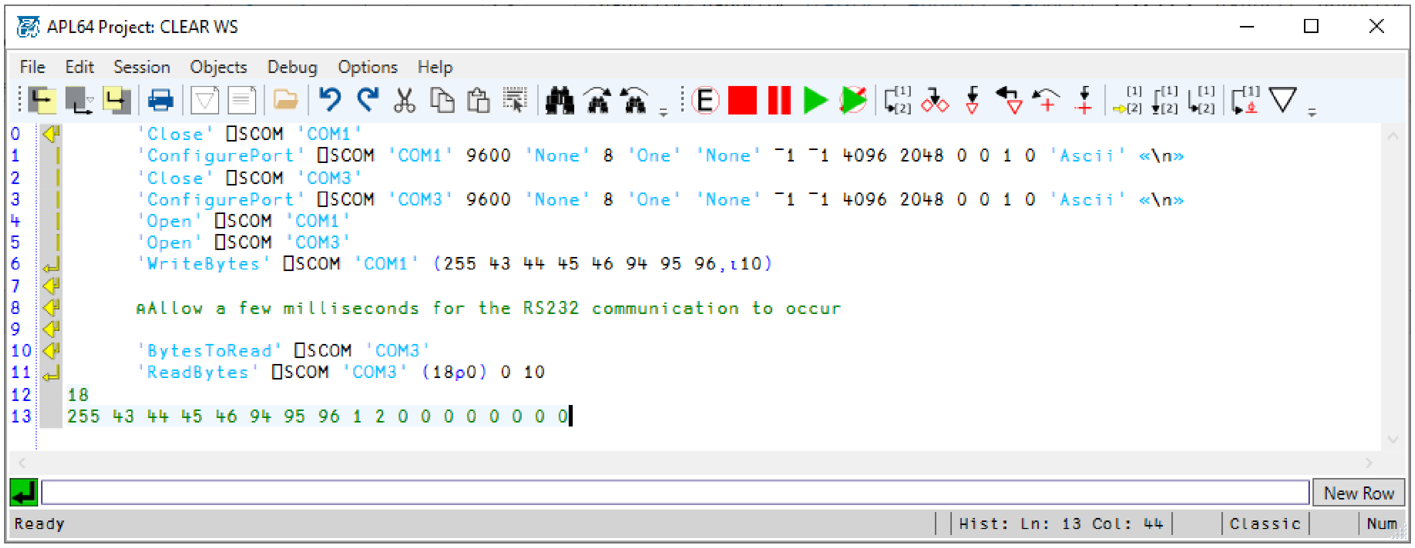
Review of the Microsoft .Net Serial Port class documentation provides the default values for the supported properties of this class. Of note are the infinite default values for the ReadTimeout and WriteTimeOut properties.

⎕scom Examples

In these examples APL64 is installed on a Windows workstation. The workstation's 'COM3' serial port is connected to the workstation's 'COM1' serial port by a 'null modem cable'. This is a convenient way to develop and debug RS232 communication applications.

The workstation's 'COM1' port will be used to receive data and the workstation's 'COM3' port will be used to transmit information to the workstation's 'COM1' port.

Writing & Reading Byte Codes Example

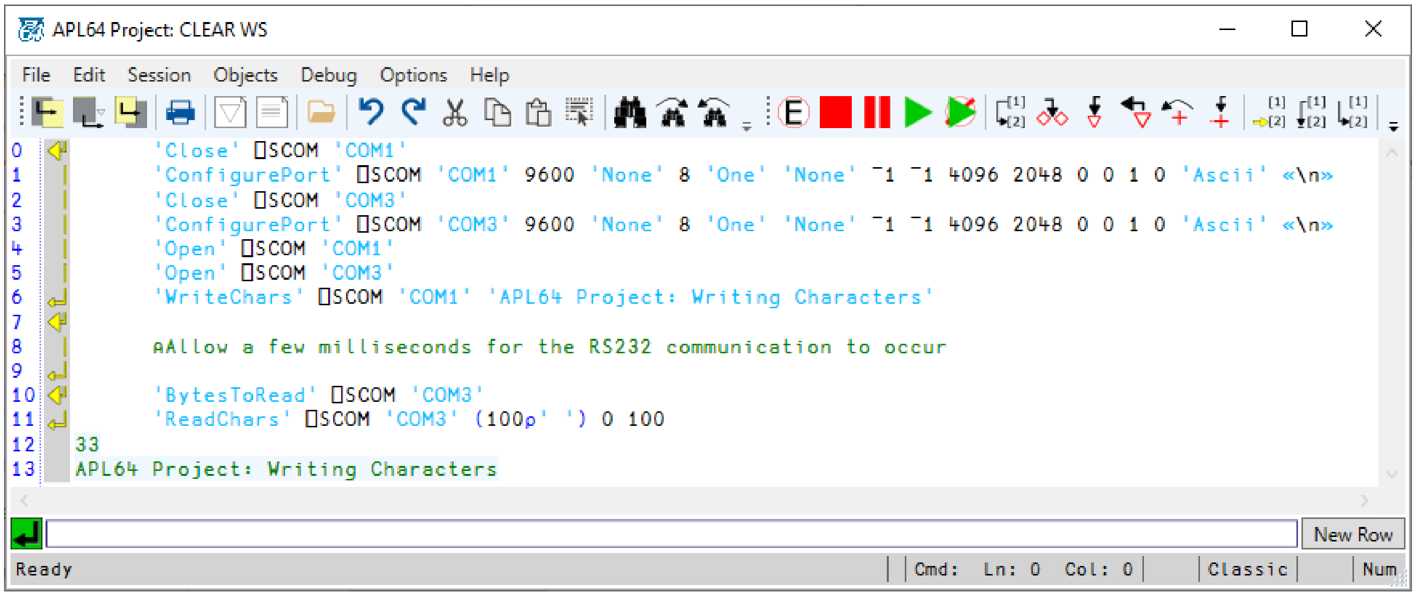


```
        'Close' ⎕SCOM 'COM1'
        'ConfigurePort' ⎕SCOM 'COM1' 9600 'None' 8 'One' 'None' ¯1 ¯1 4096 2048 0 0 1 0 'Ascii' «\n»
        'Close' ⎕SCOM 'COM3'
        'ConfigurePort' ⎕SCOM 'COM3' 9600 'None' 8 'One' 'None' ¯1 ¯1 4096 2048 0 0 1 0 'Ascii' «\n»
        'Open' ⎕SCOM 'COM1'
        'Open' ⎕SCOM 'COM3'
        'WriteBytes' ⎕SCOM 'COM1' (255 43 44 45 46 94 95 96,⍳10)

        ⍝Allow a few milliseconds for the RS232 communication to occur

        'BytesToRead' ⎕SCOM 'COM3'
        'ReadBytes' ⎕SCOM 'COM3' (18⍴0) 0 10
18
255 43 44 45 46 94 95 96 1 2 0 0 0 0 0 0 0 0
```

Try it code:

```
    'Close' ⎕scom 'COM1'
    'ConfigurePort' ⎕scom 'COM1' 9600 'None' 8 'One' 'None' ¯1 ¯1 4096 2048 0 0 1 0 'Ascii' «\n»
    'Close' ⎕scom 'COM3'
    'ConfigurePort' ⎕scom 'COM3' 9600 'None' 8 'One' 'None' ¯1 ¯1 4096 2048 0 0 1 0 'Ascii' «\n»
    'Open' ⎕scom 'COM1'
    'Open' ⎕scom 'COM3'
    'WriteBytes' ⎕scom 'COM1' (255 43 44 45 46 94 95 96,⍳10)

    ⍝Allow a few milliseconds for the RS232 communication to occur

    'BytesToRead' ⎕scom 'COM3'
    'ReadBytes' ⎕scom 'COM3' (18⍴0) 0 10
```

Writing & Reading Character Vectors Example

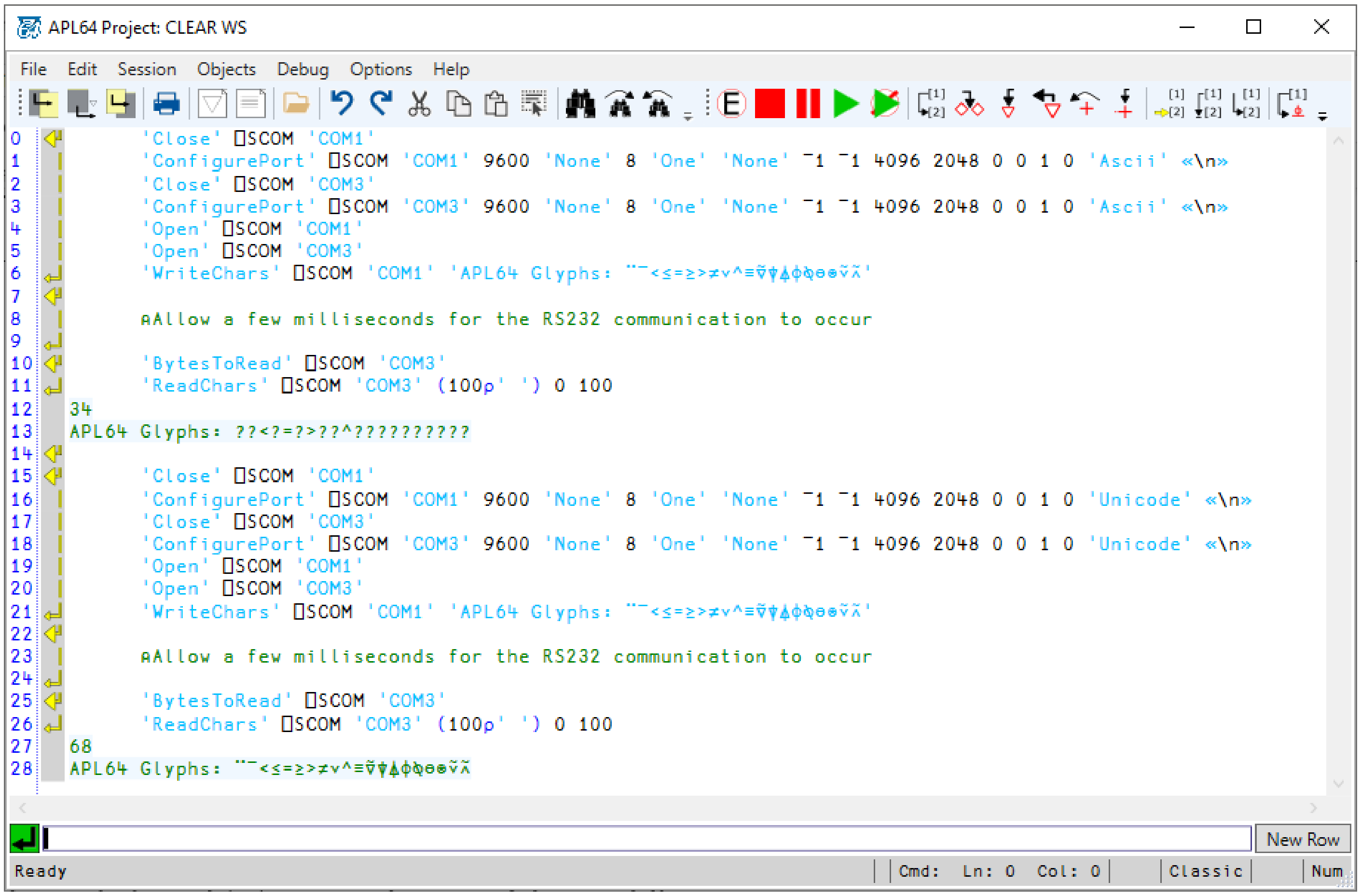


```
      'Close' ⎕SCOM 'COM1'
      'ConfigurePort' ⎕SCOM 'COM1' 9600 'None' 8 'One' 'None' ¯1 ¯1 4096 2048 0 0 1 0 'Ascii' «\n»
      'Close' ⎕SCOM 'COM3'
      'ConfigurePort' ⎕SCOM 'COM3' 9600 'None' 8 'One' 'None' ¯1 ¯1 4096 2048 0 0 1 0 'Ascii' «\n»
      'Open' ⎕SCOM 'COM1'
      'Open' ⎕SCOM 'COM3'
      'WriteChars' ⎕SCOM 'COM1' 'APL64 Project: Writing Characters'

      ⍝Allow a few milliseconds for the RS232 communication to occur

      'BytesToRead' ⎕SCOM 'COM3'
      'ReadChars' ⎕SCOM 'COM3' (100⍴' ') 0 100
33
APL64 Project: Writing Characters
```

Try it code:

```
'Close' ⎕scom 'COM1'
'ConfigurePort' ⎕scom 'COM1' 9600 'None' 8 'One' 'None' ¯1 ¯1 4096 2048 0 0 1 0 'Ascii' «\n»
'Close' ⎕scom 'COM3'
'ConfigurePort' ⎕scom 'COM3' 9600 'None' 8 'One' 'None' ¯1 ¯1 4096 2048 0 0 1 0 'Ascii' «\n»
'Open' ⎕scom 'COM1'
'Open' ⎕scom 'COM3'
'WriteChars' ⎕scom 'COM1' 'APL64 Project: Writing Characters'

⍝Allow a few milliseconds for the RS232 communication to occur

'BytesToRead' ⎕scom 'COM3'
'ReadChars' ⎕scom 'COM3' (100⍴' ') 0 100
```

When writing and reading characters, the selected encoding for the RS-232 ports is significant:



```
      'Close' ⎕SCOM 'COM1'
      'ConfigurePort' ⎕SCOM 'COM1' 9600 'None' 8 'One' 'None' ¯1 ¯1 4096 2048 0 0 1 0 'Ascii' «\n»
      'Close' ⎕SCOM 'COM3'
      'ConfigurePort' ⎕SCOM 'COM3' 9600 'None' 8 'One' 'None' ¯1 ¯1 4096 2048 0 0 1 0 'Ascii' «\n»
      'Open' ⎕SCOM 'COM1'
      'Open' ⎕SCOM 'COM3'
      'WriteChars' ⎕SCOM 'COM1' 'APL64 Glyphs: ¨¯<≤=≥>≠∨^≡⍫⍒⍋⌽⍉⊖⍟⍱⍲'

      ⍝Allow a few milliseconds for the RS232 communication to occur

      'BytesToRead' ⎕SCOM 'COM3'
      'ReadChars' ⎕SCOM 'COM3' (100⍴' ') 0 100
34
APL64 Glyphs: ??<?=?>??^??????????

      'Close' ⎕SCOM 'COM1'
      'ConfigurePort' ⎕SCOM 'COM1' 9600 'None' 8 'One' 'None' ¯1 ¯1 4096 2048 0 0 1 0 'Unicode' «\n»
      'Close' ⎕SCOM 'COM3'
      'ConfigurePort' ⎕SCOM 'COM3' 9600 'None' 8 'One' 'None' ¯1 ¯1 4096 2048 0 0 1 0 'Unicode' «\n»
      'Open' ⎕SCOM 'COM1'
      'Open' ⎕SCOM 'COM3'
      'WriteChars' ⎕SCOM 'COM1' 'APL64 Glyphs: ¨¯<≤=≥>≠∨^≡⍫⍒⍋⌽⍉⊖⍟⍱⍲'

      ⍝Allow a few milliseconds for the RS232 communication to occur

      'BytesToRead' ⎕SCOM 'COM3'
      'ReadChars' ⎕SCOM 'COM3' (100⍴' ') 0 100
68
APL64 Glyphs: ¨¯<≤=≥>≠∨^≡⍫⍒⍋⌽⍉⊖⍟⍱⍲
```

⎕setnewlines Replace line terminator characters

Purpose:
Replaces line terminator characters in a string with programmer-selected characters.

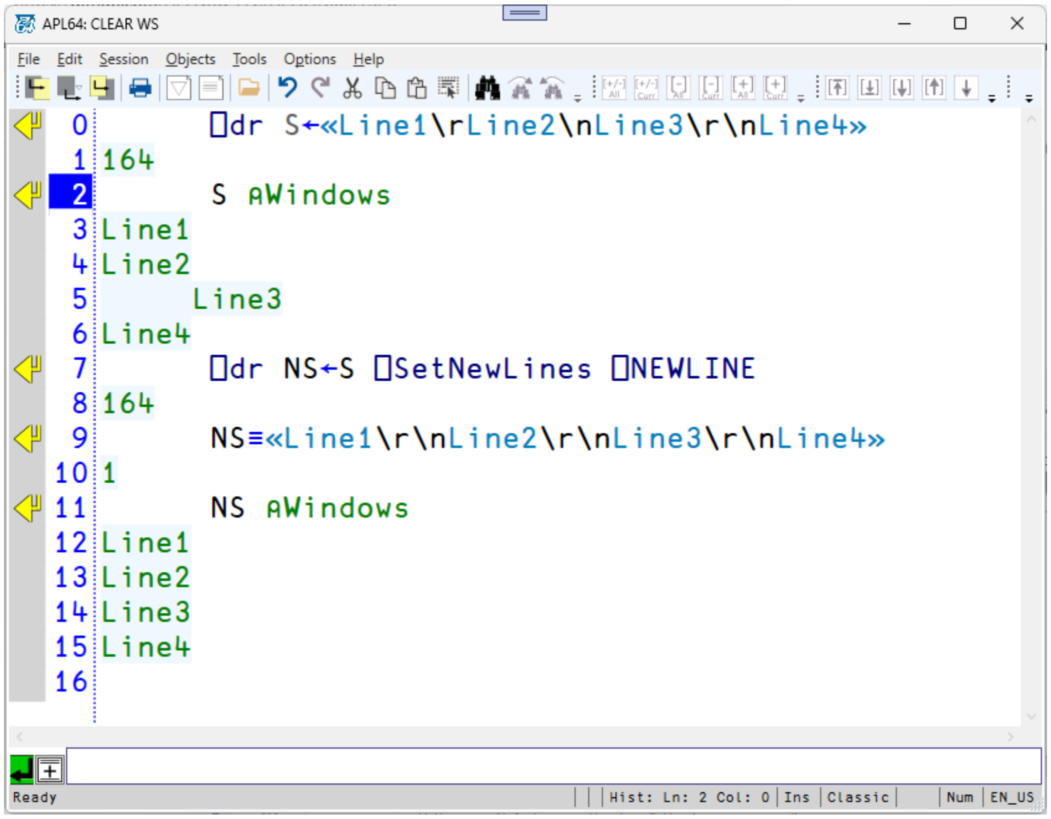
Syntax: result ← array ⎕setnewlines chars

Arguments:
array is any array of character arrays or strings
chars is generally the programmer-selected replacement characters are those obtained from the APL64 ⎕newline system function.

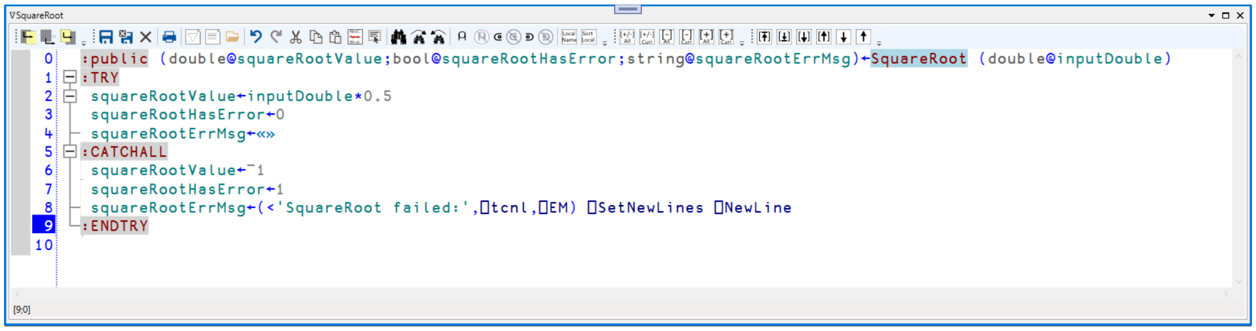
Results:
result is the same array with the line terminator characters replaced with the programmer-selected characters.

Remarks:
String text, returned as a result of an APL64 programmer-defined, public function in an APL64 cross-platform component, should use the OS-appropriate newline characters.

Examples:


APL64: CLEAR WS
File Edit Session Objects Tools Options Help
Ready
Hist: Ln: 2 Col: 0 Ins Classic Num EN_US

```
      ⎕dr S←«Line1\rLine2\nLine3\r\nLine4»
164
      S ⍝Windows
Line1
Line2
          Line3
Line4
      ⎕dr NS←S ⎕SetNewLines ⎕NEWLINE
164
      NS≡«Line1\r\nLine2\r\nLine3\r\nLine4»
1
      NS ⍝Windows
Line1
Line2
Line3
Line4
```



```
:public (double@squareRootValue;bool@squareRootHasError;string@squareRootErrMsg)←SquareRoot (double@inputDouble)
:TRY
 squareRootValue←inputDouble*0.5
 squareRootHasError←0
 squareRootErrMsg←«»
:CATCHALL
 squareRootValue←¯1
 squareRootHasError←1
 squareRootErrMsg←(<'SquareRoot failed:',⎕tcnl,⎕EM) ⎕SetNewLines ⎕NewLine
:ENDTRY
```

⎕si State Indicator

Purpose:
Returns a character matrix representation of the state indicator, showing at each row the names of functions or variables localized at that level.

Syntax: result ← ⎕si

Result:
result is a character matrix that displays the same information as ⎕si with the addition of localized names at each level of the stack. If the state indicator is empty, the result is an empty matrix of shape 0 0.

A line number of ¯1 indicates SI DAMAGE caused by modification of a suspended or pendent function in such a way that the system cannot resume executing it.

Example:

```
    1 ⎕stop 'TRIAL' ⋄ -'TRIAL 5'
1
TRIAL[1]
    ρ⎕←⎕sinl
TRIAL[1]*  A  N  ⎕io
-
2 17
```

⎕size Space Used by Identifiers

Purpose:
Returns the amount of space that a list of functions, variables, or labels uses.

Syntax: result ← [scope] ⎕size 'idlist'

Argument:
idlist is a list of function, variable, or label names represented as a character matrix with one name in each row or a character vector with names separated by blanks.
scope is an optional numeric argument that specifies the scope at which names are interpreted, where 1 = local (default); and 0 = global.

Result:
The result is a numeric vector. Each element of result is the amount of space (in bytes) required for the internal representation of the object named in the corresponding position of the argument. (Note: Symbol table space is not included.) The system returns 0 for undefined identifiers and ill-formed names. ⎕size references the most local definition of each name.

Caution:
You cannot reliably use the value of ⎕size to estimate the increase in workspace from erasing an object in the workspace. It is possible that multiple variable names refer to the same variable in the workspace. A nested array can also contain multiple items that have the same value and occupy the same storage in the workspace.

Note also that functions can change in size. In particular, a function grows larger when the system executes a line in the function for the first time and generates precompiled internal code for that line. Function monitoring (⎕mf) also changes the size of a function.

Examples:
The last line of the example below shows that the workspace available did not increase as the result of erasing two of the three copies of the variable.

```
    A←B←C←2 400 ⋄ ⎕size 'A B C'
112 112 112
    ⎕erase 'A C' ⋄ ⎕size 'A B C'
0 112 0
```

```
      ⎕size 'B _ILLFORMEDNAME'
112 0
```

⎕skd String Key Dictionary

Purpose:
Supports creating and using .Net dictionary instances in APL64.

For detailed documentation use the **Help | APL Language | Using ⎕SKD** menu item in the APL64 developer version.

⎕splash Splash Image

Purpose:
Controls the display of the user-provided splash image (BMP, GIF, PNG, or JPG) at the start of APL64 in the Windows operating system only.

Syntax: ⎕splash flag

Arguments:
flag is a Boolean scalar that controls the monitoring setting. A one shows the splash image, and a zero hides the splash image.

Effect:
Shows or hides the user-provided splash image displayed at the start of APL64 with one of these command line arguments:

Command Line Argument (Abbrev.)	Data Type	Action
SplashImagePath (SIP)	Text	Path to splash image to display
SplashImageCenter (SIC)	Bool	Splash image centered on monitor ensemble
SplashImageTop (SIT)	Bool	Splash image is the topmost window

See APL64 Command Line Arguments for more information.

⎕split Segment Array into Nested Array

Purpose:
Segments an array into a nested array by enclosing subarrays along one dimension, thereby reducing the rank. This is a cover function for the Evolution Level 1 behavior of monadic down arrow (↓); you can use this system function at any Evolution Level. At Evolution Level 2, use of monadic down arrow produces a SYNTAX ERROR.

Syntax: result ← ⎕split array
               result ← ⎕split[i] array

Arguments:
array is any APL array;
i is a non-negative, integer-valued scalar that indicates the axis along which you want the function to enclose the subarrays; the default is along the last dimension.

Result:
The result is a nested array. ⎕split encloses each item of array once; however, it does not change a simple scalar. When you enclose an array with ⎕split, the rank of the result is one less than the rank of the original array. Each item of the result is a vector with the length of the enclosed dimension.
By default, ⎕split encloses the last dimension of array. To enclose a different dimension, specify the axis in brackets after ⎕split. If you specify the axis, it is equivalent to using the primitive function Enclose (⊂) with axis.

Example:

```
      ]display ⎕split 3 4⍴⍳12
.→-------------------------------.
|.→------. .→------. .→---------.|
||1 2 3 4| |5 6 7 8| |9 10 11 12||
|'~------' '~------' '~---------'|
|ϵ-------------------------------'

      ]display ⎕split[1] 3 4⍴⍳12
.→----------------------------------.
|.→----. .→-----. .→-----. .→-----.|
||1 5 9| |2 6 10| |3 7 11| |4 8 12||
|'~----' '~-----' '~-----' '~-----'|
'ϵ----------------------------------'
```

⎕sql Interface to Microsoft SQL Server Database

Purpose:
Provides a structured query language (SQL)-based interface to a Microsoft SQL Server database. To learn more about ⎕sql in the APL64 Developer version go to **Help | APL Language | SQL System Functions | Using ⎕SQL**.

⎕sqldb2 Interface to IBM DB2 Database

Purpose:
Provides a structured query language (SQL)-based interface to a IBM DB2 database. To learn more about ⎕sqldb2 in the APL64 Developer version go to **Help | APL Language | SQL System Functions | Using ⎕SQLDB2**.

⎕sqlite Interface to SQLite Database

Purpose:
Provides a structured query language (SQL)-based interface to a SQLite database. To learn more about ⎕sqllite in the APL64 Developer version go to **Help | APL Language | SQL System Functions | Using ⎕SQLite**.

⎕sqlmy Interface to MySQL Database

Purpose:
Provides a structured query language (SQL)-based interface to a MySQL database. To learn more about

⎕sqlmy in the APL64 Developer version go to **Help | APL Language | SQL System Functions | Using ⎕SQLMy**.

⎕ss String Search

Purpose:
Performs a string search, locating all occurrences of a character scalar or vector within another character vector. This function is a special case of the primitive function Find (⍷) with the arguments reversed.

Syntax: result ← 'data' ⎕ss 'pattern'

Arguments:
data is a character vector for the system to search;
pattern is a character vector or scalar for the system to locate in data.

Result:
The result is a Boolean vector of the same length as the left argument, which shows the location of all occurrences of pattern within data. A 1 in the result signifies a match beginning at that position within data. The system shows all matches, including those that overlap. If pattern is empty (''), result is all 1s.

Examples:

```
    'MISSISSIPPI' ⎕ss 'ISSI'
0 1 0 0 1 0 0 0 0 0 0
    'EMPTY MATCHES ALL' ⎕ss ''
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
    CV←'THIS   IS   TOO   SPACED.'
    (~CV ⎕ss '  ')/CV
THIS IS TOO SPACED.
```

⎕ssassign Index Assignment into a Segmented String

Purpose:
Substitute segments in a segmented string at specified indices.

Syntax: newsegstr ← 'segstr1[indices]' ⎕ssassign 'segstr2'

Arguments:
segstr1 is the character vector to be index assigned. The first character is the segment delimiter.
segstr2 is the character vector with the substitute segments. The first character is the segment delimiter.
indicies must be a numeric scalar or vector of index numbers. (Sensitive to ⎕io.)

Remarks:
⎕ssassign may be used as replacement for the function SSASSIGN in the ASMFNS.w3 workspace in APL+Win.

Example:

```
    x←' One Dos 3 cuattro Five Six' ⋄ 'x[3 2 4]' ⎕ssassign ' Three Two Four'
    x
```

```
One Two Three Four Five Six
```

⎕sscat Catenate Segment Strings

Purpose:
Catenates two segmented strings to a new segmented string.

Syntax: newsegstr ← ‘segstr' ⎕sscat 'segstr'

Arguments:
The left and right arguments must be a character vector. The first character is the segment delimiter.

Remarks:
⎕sscat may be used as replacement for the function SSCAT in the ASMFNS.w3 workspace in APL+Win.

Examples:

```
     'Hello ' ⎕sscat 'World!'
Hello World!
     ' One Two Three' ⎕sscat ' Four Five Six'
One Two Three Four Five Six
```

⎕ssdeb Delete Extraneous Blanks from a Segmented String

Purpose:
Deletes leading, trailing, and multiple blanks or other characters from segments in a segmented string.

Syntax: newsegstr ←   ⎕ssdeb 'segstr'
              newsegstr ← [chars]   ⎕ssdeb 'segstr'

Arguments:
The right argument must be a character vector. The first character is the segment delimiter. Note the differences in the examples below when a character other than the slash is the delimiter (2↓SS) and when a space itself is the delimiter (1↓SS).
The optional left argument is a substitute list of extraneous characters (in addition to space).

Remarks:
⎕ssdeb may be used as replacement for the function SSDEB in the ASMFNS.w3 workspace in APL+Win.

Example:

```
     SS
/ A/B / C /44 / 5  55   555  55 5/6666/  /I
     SSDEB SS
/A/B/C/44/5 55 555 55 5/6666//I
     SSDEB 2↓SS
A/B / C /44 / 5 55 555 55 5/6666/ /I
     SSDEB 1↓SS
 A/B / C /44 / 5  55   555  55 5/6666/  /I
     '45' SSDEB SSDEB SS
/A/B/C// 5 5 5 /6666//I
```

⎕ssindex Return Index of Substring in Segmented String

Purpose:
Returns the segment substring in a segmented string that corresponds to specified indices.

Syntax: segsubstr ← 'segstr' ⎕ssindex ind

Arguments:
The left argument must be a character vector. The first character is the segment delimiter.
The right argument must be a numeric scalar or vector of index numbers. (Sensitive to ⎕io.)

Remarks:
⎕ssindex may be used as replacement for the function SSINDEX in the ASMFNS.W3 workspace in APL+Win.

Example:

```
    SS
/ A/B / C /44 / 5  55   555  55 5/6666/  /I
    SS ⎕ssindex 1 3 1 4
/ A/ C / A/44
```

⎕sstomat Convert Segmented String to Matrix

Purpose:
Converts a segmented string to a matrix.

Syntax: charmat ← ⎕sstomat 'segstring'

Arguments:
The argument is the character vector. The first character is the segment delimiter.

Remarks:
⎕sstomat is the replacement for the assembler function SStoMAT in the ASMFNS.W3 workspace in APL+Win.

Effects:
The ⎕sstomat system function will not throw a DOMAIN ERROR exception for a right argument that is not appropriate for ⎕sstomat and instead will return the right argument unmodified.

Example:

```
      ⎕sstomat '/   A/  BBB// CCCCC/DDDDDD/'
   A
  BBB

 CCCCC
DDDDDD
      ⎕sstomat (⍳10) ',a,b,c' (2 3⍴⎕a)
 1 2 3 4 5 6 7 8 9 10  a  ABC
                       b  DEF
                       c
```

⎕stop Stop Function Execution

Purpose:
Sets or reports stop flags for a function.

Syntax:
```
result ← linenums   ⎕stop 'fnname'
result ← linenums   ⎕stop '!fnname'
result ←            ⎕stop 'fnname'
result ←            ⎕stop '!'
```

Arguments:
fnname is a character vector containing the name of an unlocked function. If you prefix fnname with the optional exclamation mark (!), a latent stop is specified on the function, so that whenever it is defined in the future (such as when reading it from file and doing ⎕def or ⎕defl or ⎕fx) then the system will assert any latent stop settings in the specified function. You can also query all pending latent stops by using a right argument of exclamation mark ('!') without following it by a function name.

linenums is an integer vector or scalar that contains the line numbers of the function on which the system should set stop flags. The system ignores 0 and integers that are not line numbers in the specified function. You can clear all the stop flags for a function with (⍳0) or ⍬.

Result:
The result is an integer vector of the lines of fnname for which the system set prior stops. If you use ⎕stop monadically, the function reports the settings. If you use it dyadically, it reports prior settings and replaces them with stops on the lines specified in linenums

Effect:
Executing ⎕stop has no immediate effect. However, on subsequent executions of the function, each time the function reaches a line that is flagged, the system stops and enters immediate execution mode, preserving the state indicator and all local values and definitions. You can then explore and even alter the local environment before branching (→) back into or out of the suspended function. The ability to observe and alter the local environment at chosen points during execution of the function is a valuable aid for debugging.

Latent stops do persist across workspace boundaries. But, unlike normal stops, latent stops are not saved with the workspace. The system saves and reloads normal stop settings with a workspace, but does not copy them along with the function to which they apply when you use ⎕copy,)copy, ⎕pcopy, or)pcopy. Redefining a function with either ⎕def or ⎕defl or ⎕fx removes all regular stop settings from that function; latent stops are not affected. However, clearing latent stops will clear both latent and normal stops, if the function already exists in the workspace. Also setting regular stops on a function that's already defined in the workspace sets those stops without affecting latent stops. But if you do ⎕def or ⎕defl or ⎕fx of the function again, the latent stops will be applied and the normal stops that were in workspace before will be ignored.

Editing a function line with ⎕defl removes any setting associated with that line of code. Locking a function with ⎕lock removes all stop settings in the function.

Examples:
The example sets stop flags on two lines of the specified function; the empty explicit result means no lines were previously set. When you run the function, it stops on each flagged line, where you can check the value of the variable R.

```
   ∇ R←FIBONA N
[1]   R←1 1
[2]  BACK: R←R,+/¯2↑R
[3]   →BACK×N>ρR
   ∇

     2 3 ⎕stop 'FIBONA'
     FIBONA 5
FIBONA[2]
     R
1 1
     →2
FIBONA[3]
     R
1 1 2
     →3
FIBONA[2]
     →2
FIBONA[3]
     R
1 1 2 3

     →3
FIBONA[2]
     →2
FIBONA[3]
     R
1 1 2 3 5
     →3
1 1 2 3 5
```

⎕stptr Symbol Table Pointer

Purpose:
Returns indirect pointers called symbolic handles to the most local objects associated with a list of names of functions, variables, and/or labels in the active workspace. Beware that these handles are no longer used in conjunction with ⎕call, which has been deprecated in APL64.

Syntax: result ← ⎕stptr 'idlist'

Argument:
idlist is a list of names of functions, variables and/or labels for which you want symbolic handles. It can

be a character vector that contains the names separated by one or more blanks or a character matrix with one name in each row.

Result:
The explicit result of ⎕stptr is a numeric vector with one element for each identifier in the argument. Element values are non-negative. A 0 element indicates that the corresponding name is ill-formed. You can use positive elements, referred to as symbolic handles, to access the object currently associated with the name.

Effect:
The result of ⎕stptr allows you to address the data stored in an APL object and provides access to other information such as the type, rank, and shape of a variable.

Caution:
The interpreter may at any time compact the workspace, causing many or all of the objects contained in it to move to different locations in storage. These compactions do not affect a result from ⎕stptr. However, results may be affected by activities that impact the interpreter's symbol table, such as the creation of an object. For maximum compatibility with future developments, recalculate ⎕stptr values immediately before each use.

In processing names, ⎕stptr adds them to the interpreter's symbol table if they are not already present. The symbol table is automatically enlarged as needed, which may also trigger a compaction of the workspace.

Example:

```
      ⎕stptr 'XDATA'
1180
```

⎕string Interface to .Net String Class

Purpose:
⎕string exposes many of the methods and properties of the .Net String class.

Syntax: result ← [larg] ⎕string action [rarg]

Arguments:
larg are the optional arguments to the action.

action is a text vector that determines how the system function will operate on the left and right argument.

rarg are the optional arguments to the action.

Result:
The result, if any, depends on the action called.

For more information about ⎕string, use the **Help | APL Language | Using ⎕STRING** menu item.

⎕symb Workspace Symbols

Purpose:
Returns the current number of symbol table entries in the active workspace. The symbol table contains entries for all functions, variables, and labels referenced in defined functions and executing statements.

Syntax: result ← ⎕symb

Result:
The result is a two-element numeric vector. The first element is the number of entries for which the symbol table of the active workspace is sized. The system automatically increases the symbol table size as needed; you can change the size with the system command)symbols. The second element is the number of entries already used in the symbol table.

Examples:

```
      )clear
CLEAR WS
      ⎕symb
200 0
      A←1
      ⎕symb
200 1
      )erase A
      ⎕symb
200 1
```

⎕sysconfigfile Path to APL64 xml-format configuration file

Purpose:
This system function returns the path to the APL64 xml-format system configuration file.

Syntax: result ← ⎕sysconfigfile

Domain:
The result is a character vector containing the path of the APL64 xml-format configuration file is current use by an APL64 developer or run-time version.

The result is the same value which is copied to the Clipboard with the menu
Session | Configuration Settings | Settings File Path.

Example:

```
      ⎕sysconfigfile
C:\Users\Guest\AppData\Roaming\APLNowLLC\APL64\APL64.xml
```

⎕sysid System Identifier

Purpose: Returns the identification of the APL64 system currently in use.

Syntax: result ← ⎕sysid

Result:
The result is a character vector that contains the identification of the APL64 system currently in use. ⎕sysid uses all characters to identify a system.

Example:

```
 ⎕sysid
APLNow/64
```

⎕sysinit .Net Information about the Current APL64 Instance

Purpose:
This session-related system function returns information about the current APL64 process.

Syntax: result ← ⎕sysinit

Result:
The result is a nested vector containing information about the current APL64 process.

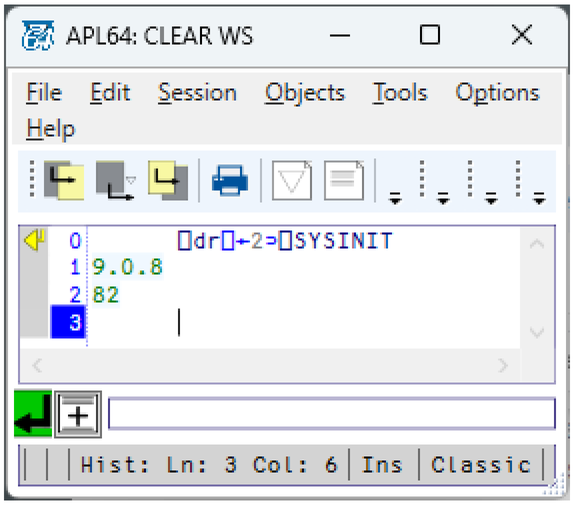
Web links to the .Net documentation of each of the elements are provided in the table below. Review these links for cross-platform information about the available values. Availability of values depends on the operating system environment, and the developer or runtime version of APL64 in use.

The number of elements in ⎕sysinit may be modified in a future version of APL64.

Index (⎕io 1)	Type	Index ⊃⎕SYSINIT
1	Int32	Process.GetCurrentProcess().Id
2	Char[]	.Net Version (major.minor.build.rev)
3	Char[]	Environment.CommandLine
4	Char[;]	EnvironmentVariables
5	Char[]	Process.GetCurrentProcess().ProcessName
6	Char[]	Environment.MachineName
7	Int32	Logical Processor Count
8	Char[]	Main Window Title
9	Int32	Main Window Handle
10	Char[][]	Environment.GetCommandLineArgs()

Example:

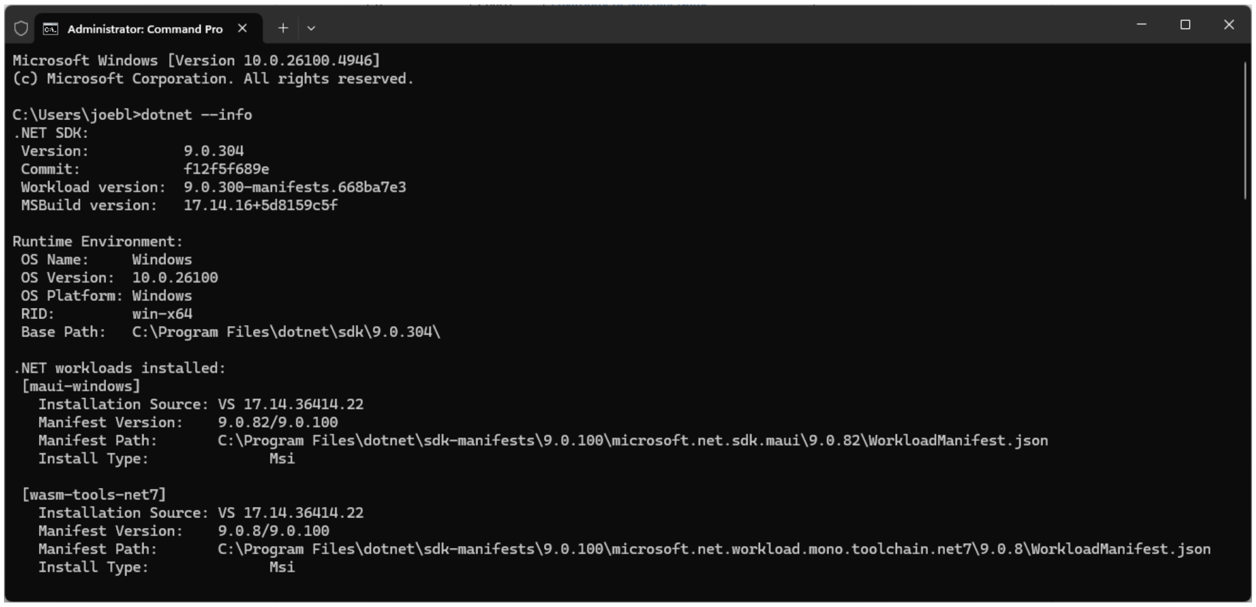
```
2⊃⎕SYSINIT
```


APL64: CLEAR WS
File Edit Session Objects Tools Options Help
Hist: Ln: 3 Col: 6 Ins Classic

```
        ⎕dr⎕←2⊃⎕SYSINIT
9.0.8
82
```

2⊃⎕SYSINIT returns a character vector indicating the .Net runtime on which APL64 is running on the target workstation.

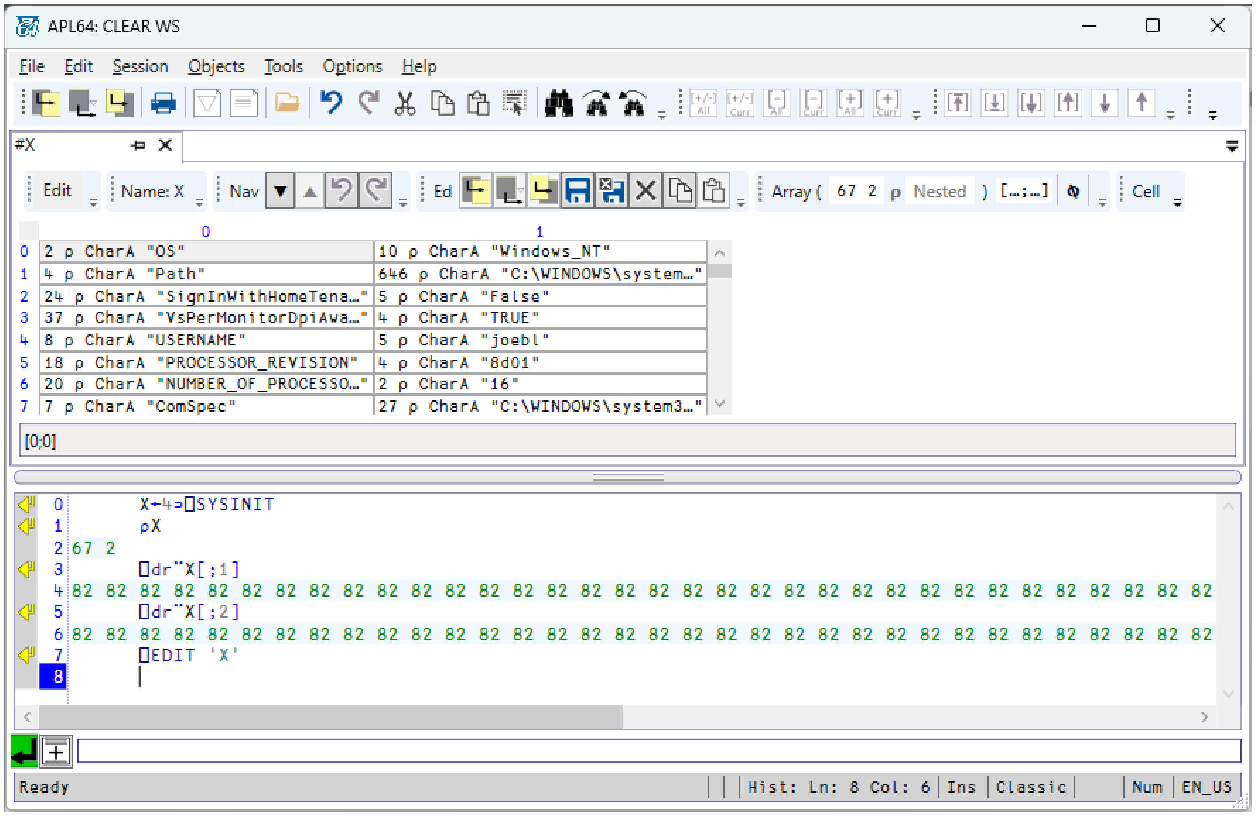
Execute dotnet –info in a command prompt to display more information about the .Net components installed on the target workstation. Administrator credentials may be required to execute this command.



```
Microsoft Windows [Version 10.0.26100.4946]
(c) Microsoft Corporation. All rights reserved.

C:\Users\joebl>dotnet --info
.NET SDK:
 Version:           9.0.304
 Commit:            f12f5f689e
 Workload version:  9.0.300-manifests.668ba7e3
 MSBuild version:   17.14.16+5d8159c5f

Runtime Environment:
 OS Name:     Windows
 OS Version:  10.0.26100
 OS Platform: Windows
 RID:         win-x64
 Base Path:   C:\Program Files\dotnet\sdk\9.0.304\

.NET workloads installed:
 [maui-windows]
   Installation Source: VS 17.14.36414.22
   Manifest Version:    9.0.82/9.0.100
   Manifest Path:       C:\Program Files\dotnet\sdk-manifests\9.0.100\microsoft.net.sdk.maui\9.0.82\WorkloadManifest.json
   Install Type:              Msi

 [wasm-tools-net7]
   Installation Source: VS 17.14.36414.22
   Manifest Version:    9.0.8/9.0.100
   Manifest Path:       C:\Program Files\dotnet\sdk-manifests\9.0.100\microsoft.net.workload.mono.toolchain.net7\9.0.8\WorkloadManifest.json
   Install Type:              Msi
```

Example:

```
X←4⊃⎕SYSINIT
⍴X
⎕dr¨X[;1]
⎕dr¨X[;2]
⎕EDIT 'X'
```


APL64: CLEAR WS
File Edit Session Objects Tools Options Help
#X
Edit
Name: X
Nav
Ed
Array (67 2 ρ Nested) [...;...]
Cell

	0	1
0	2 ρ CharA "OS"	10 ρ CharA "Windows_NT"
1	4 ρ CharA "Path"	646 ρ CharA "C:\WINDOWS\system…"
2	24 ρ CharA "SignInWithHomeTena…"	5 ρ CharA "False"
3	37 ρ CharA "VsPerMonitorDpiAwa…"	4 ρ CharA "TRUE"
4	8 ρ CharA "USERNAME"	5 ρ CharA "joebl"
5	18 ρ CharA "PROCESSOR_REVISION"	4 ρ CharA "8d01"
6	20 ρ CharA "NUMBER_OF_PROCESSO…"	2 ρ CharA "16"
7	7 ρ CharA "ComSpec"	27 ρ CharA "C:\WINDOWS\system3…"

[0;0]

```
      X←4⊃⎕SYSINIT
      ⍴X
67 2
      ⎕dr¨X[;1]
82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82
      ⎕dr¨X[;2]
82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82 82
      ⎕EDIT 'X'
```

Ready   Hist: Ln: 8 Col: 6 | Ins | Classic | Num | EN_US

⎕sysver System Version

Purpose:
Returns the version of the APL64 system currently in use.

Syntax: result ← ⎕sysver

Result:
The result is a character vector that contains information on the version of APL64 currently in use.

Example:

```
      ⎕sysver
2023.0.13.31374  12/12/2023 7:35:54 PM APL64
```

⎕sysvern System Version Number

Purpose:
Returns the version of the APL64 system currently in use.

Syntax: result ← ⎕sysvern

Result:
The result is a 4-element numeric vector that contains the following information:
[1] major version number
[2] minor version number

[3] build number
[4] revision number

Example:

```
    ⎕sysvern
 2020 0 0 11531
```

⎕tc Terminal Control

Purpose:
Returns a backspace, a newline, and a linefeed as a three-character vector.

Syntax: result ← ⎕tc

Result:
The explicit result is a character vector that contains three special characters: backspace, newline, and linefeed. The system transmits these characters to remote devices as ASCII BS, CR, and LF (decimal 8, 13 and 10).

Result:
Each element of the explicit result of ⎕tc produces its own effect. The effect of ⎕tc varies, depending on the device to which you transmit it. The first element of the explicit result is identical to ⎕tcbs and displays ␈ on the screen. The second element of the explicit result of ⎕tc is identical to ⎕tcnl and has the same effect as moving the cursor to the leftmost column of the following line. The third element of the explicit result of ⎕tc is identical to ⎕tclf; on this system it moves the cursor down one line in the same column on the screen.

Example:

```
    'NOW',⎕tc[1],'DOWN',⎕tc[2],'WE', ⎕tc[3],'GO.'
NOW␈DOWN
WE
  GO.
```

⎕tcxx Terminal Control Constants

Purpose:
These constants represent terminal control codes used by the operating system. While some do produce visible output on the screen, some do not; rather, they may have an effect, such as changing the position of subsequent characters. Each constant that you specify is treated as a character by the interpreter, for example, when measuring the length of a vector.

The terminal control constants are shown in the table below. The Value column shows the location n of the character in the atomic vector, (⎕av[⎕io+n]). The Visible Output column shows the output to the screen, when applicable.

Terminal Control Constants

Name	Description	Value (n)	Visible Output
⎕tcbel	Bell	7	
⎕tcbs	Backspace	8	␈
⎕tcdel	Delete	127	≢
⎕tcesc	ASCII escape code	27	␛
⎕tcff	Newline	12	
⎕tcht	Horizontal tab	9	
⎕tclf	Line feed	10	
⎕tcnl	Newline	13	
⎕tcnul	Null	0	

Effects:

⎕tcbel generates a "beep" or bell sound.

⎕tcbs displays the Unicode glyph ␈ in the session.

⎕tcdel displays the glyph ≢ in the session.

⎕tcesc displays the Unicode glyph ␛ in the session.

⎕tcff moves the cursor to the first position of the next line. See Note below.

⎕tcht is transmitted to remote devices as an ASCII HT (decimal 9); its effect will depend on the device receiving it. When included in a string that displays in the session, it moves the cursor to the right to the next horizontal tab position before the next character is processed.

⎕tclf varies with the device receiving it. On some terminals and printers, the screen or paper advances one line, but the cursor stays in the same column position as on the previous line. Other terminals and printers may treat it as ⎕tcnl or ignore it.

⎕tcnl moves the cursor to the first position of the next line.

⎕tcnul contains an ASCII NUL code (0). It does not move the cursor. This character is often used as a terminating character for strings. Using it for unintended purposes may have unwanted side effects.

Note: ⎕TCFF generates output similar to ⎕tcnl in the APL64 Developer version. To simulate the APL+Win action of executing ⎕tcff, use the Session | Scroll Up Session History menu item. To clear the current session history use the menu Session | Clear Session History menu item. These actions are also available in the context menu of the session history or session command line.

⎕telprint Reshape Character Matrix into Multiple Columns

Purpose:
Reshape a character matrix or string vector into a multi-column character matrix.

Syntax: Result ← [Left] ⎕telprint Right

Arguments:
Right argument is a rank-2 character matrix, or a rank-1 string vector

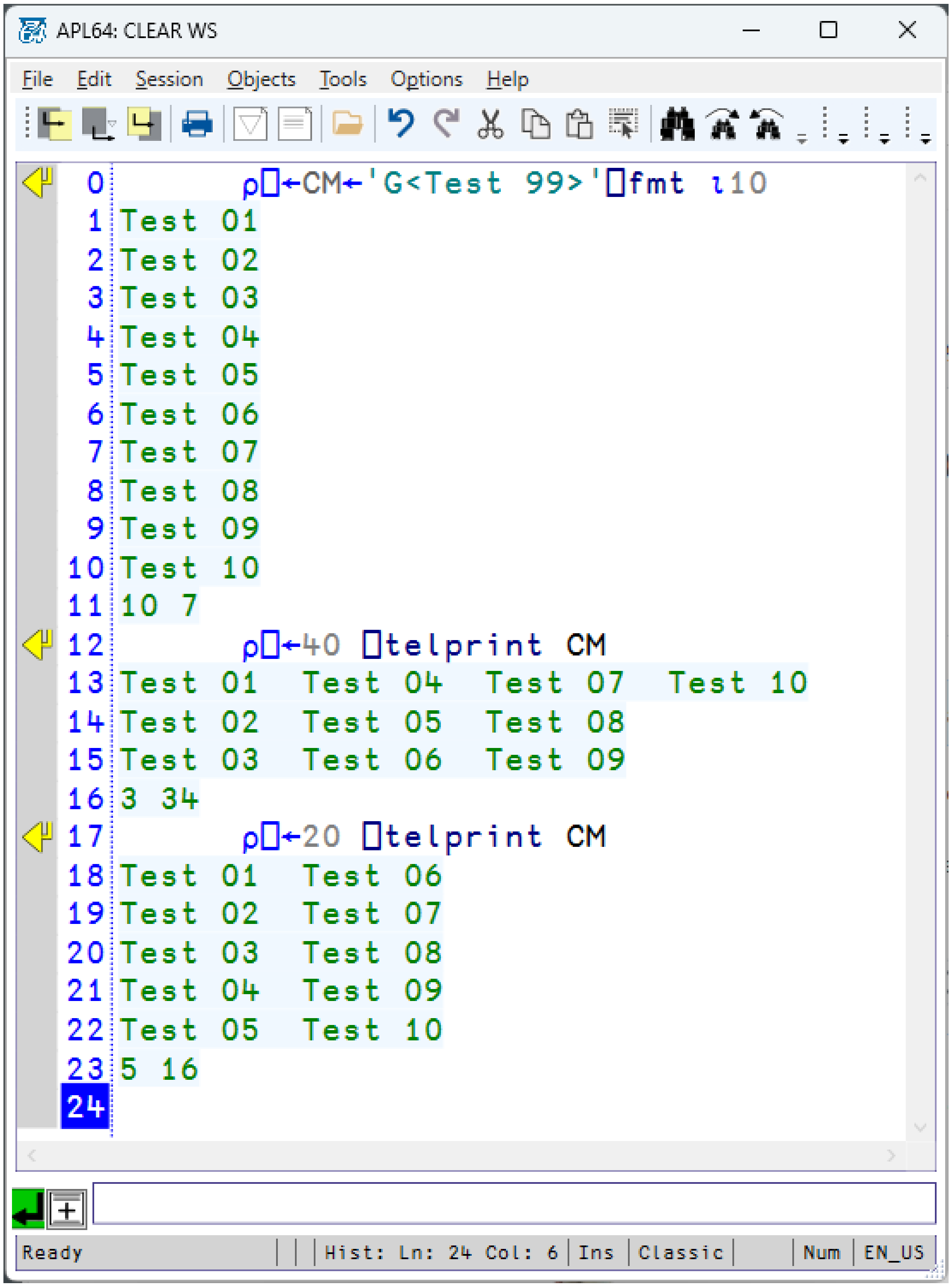
Left argument is an optional integer limit on the width of the result, with default value ⎕pw.

Result:
Result is a rank-2, character matrix. Spaces the right argument are preserved in the result. The left argument will be increased if the length of any right argument element exceeds the left argument value. The left argument will be decreased to minimize the resulting width of the result.

Remarks:
⎕telprint may be used as replacement for the function TELPRINT in the APL+Win ASMFNS.w3 workspace.

Examples:

```
ρ⎕←CM←'G<Test 99>'⎕fmt ι10
ρ⎕←40 ⎕telprint CM
ρ⎕←20 ⎕telprint CM
```

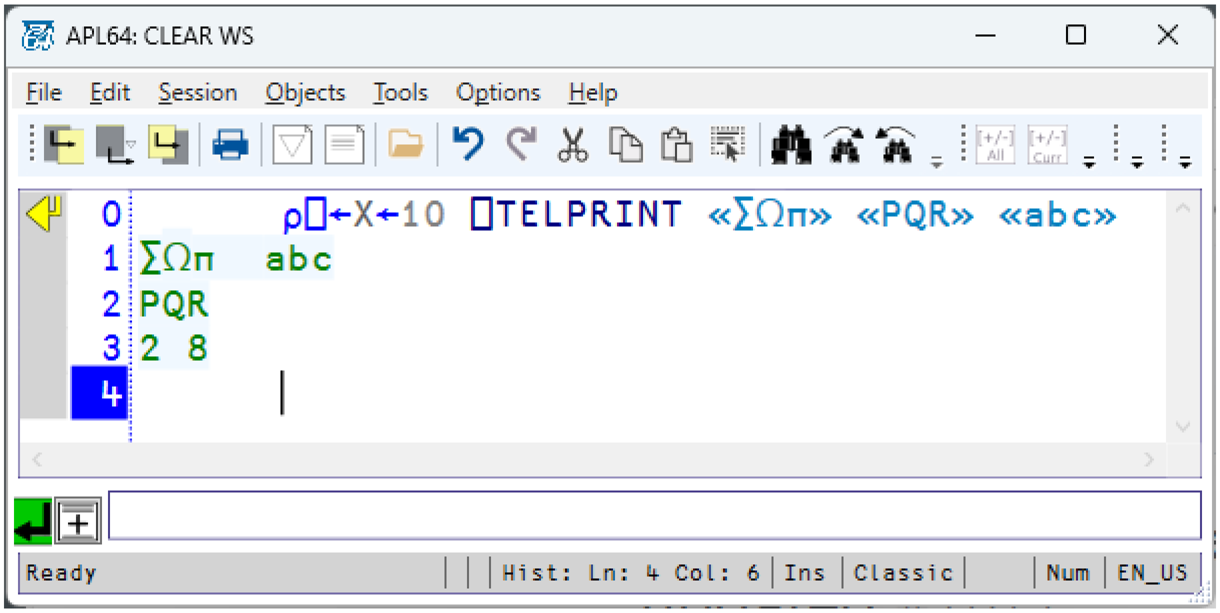


APL64: CLEAR WS

```
      ρ⎕←CM←'G<Test 99>'⎕fmt ⍳10
Test 01
Test 02
Test 03
Test 04
Test 05
Test 06
Test 07
Test 08
Test 09
Test 10
10 7
      ρ⎕←40 ⎕telprint CM
Test 01   Test 04   Test 07   Test 10
Test 02   Test 05   Test 08
Test 03   Test 06   Test 09
3 34
      ρ⎕←20 ⎕telprint CM
Test 01   Test 06
Test 02   Test 07
Test 03   Test 08
Test 04   Test 09
Test 05   Test 10
5 16
```

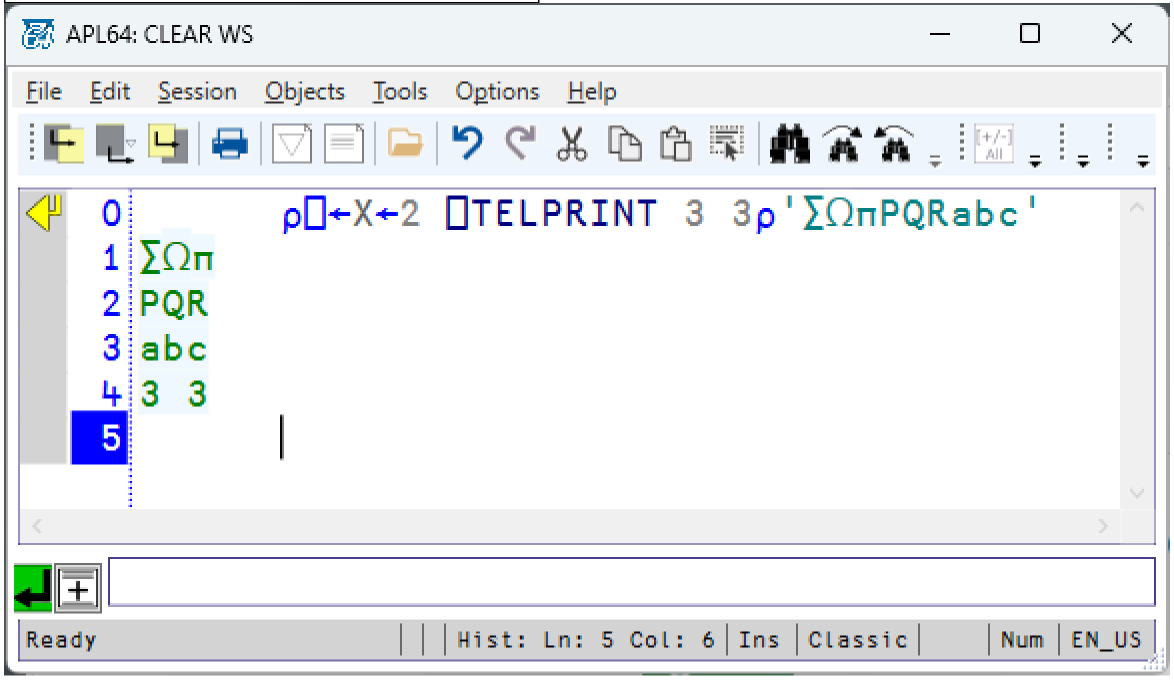
Ready | Hist: Ln: 24 Col: 6 | Ins | Classic | Num | EN_US

```
      ρ⎕←X←10 ⎕TELPRINT «∑Ωπ» «PQR» «abc»
```



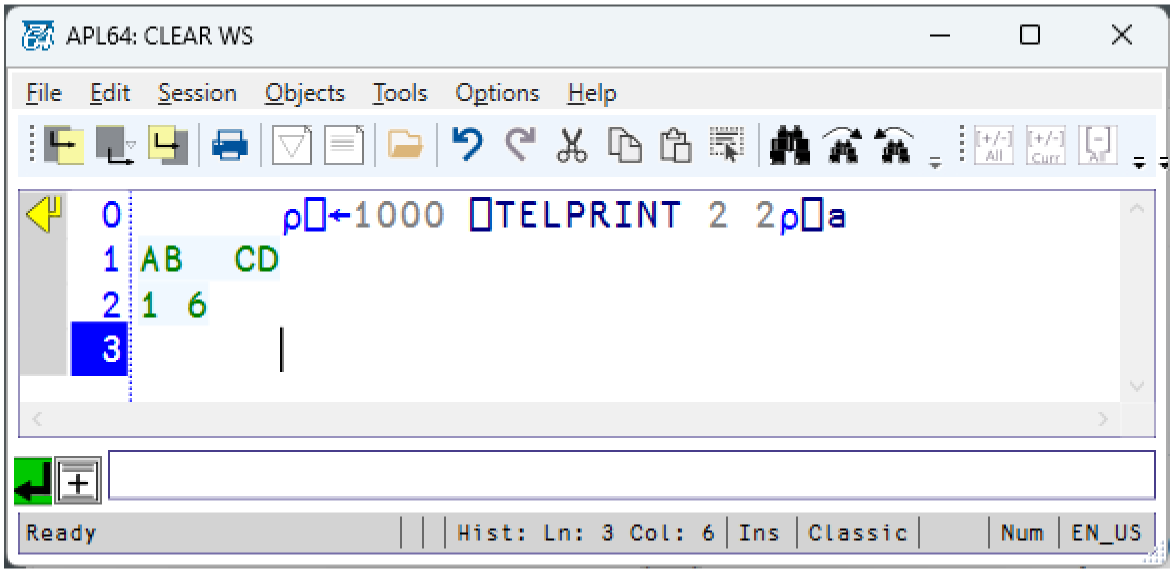
```
      ρ⎕←X←10 ⎕TELPRINT «∑Ωπ» «PQR» «abc»
∑Ωπ   abc
PQR
2 8
```

ρ⎕←X←2 ⎕TELPRINT 3 3ρ'∑ΩπPQRabc'



```
      ρ⎕←X←2 ⎕TELPRINT 3 3ρ'∑ΩπPQRabc'
∑Ωπ
PQR
abc
3 3
```

ρ⎕←1000 ⎕TELPRINT 2 2ρ⎕a



```
        ρ⎕←1000 ⎕TELPRINT 2 2ρ⎕a
AB   CD
1 6
```

```
Ready | | Hist: Ln: 3 Col: 6 | Ins | Classic | | Num | EN_US
```

⎕test Test Functions in Workspace

⎕test is not available in APL64. It is under review and may be implemented in a future release.

⎕textrepl Text Replacement

Purpose:
Replaces existing characters with new characters.

Syntax: newvec ← SegText ⎕textrepl sourceText [permitOverlap(bool)]

Arguments:
The left argument is segmented text containing pairs of text to replace and replacement text. The first element of the left argument is the segmentation delimiter.

The first element of the right argument is the source text which will be processed.

The optional second element of the right argument is a Boolean scalar indicating if 'over-lapping' replacement are permitted. The default value is 0 (false).

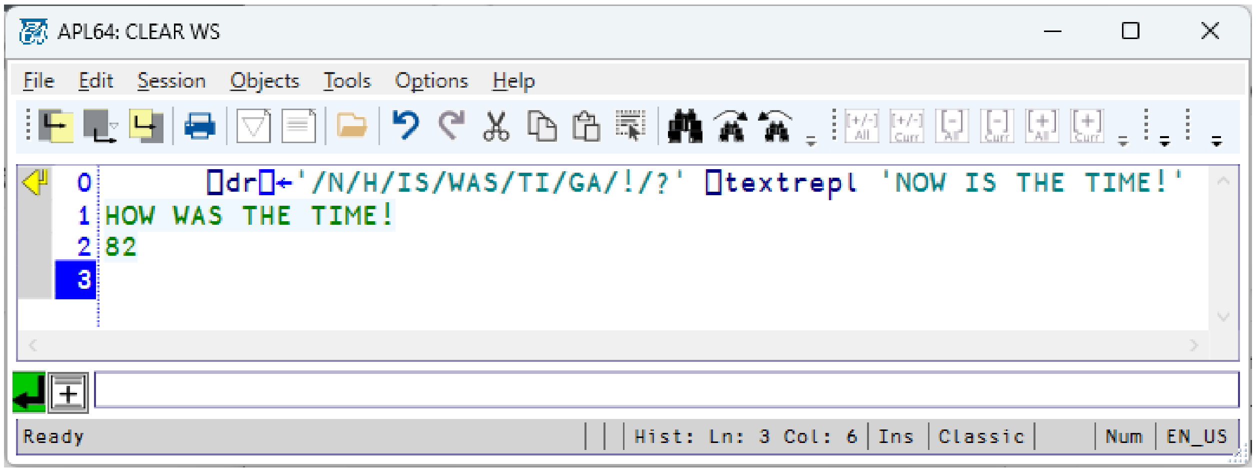
Result: The result is the source text with the replacements specified by the left argument. If the first element of the right argument is an APL64 string scalar, the result will be an APL64 string scalar, otherwise the result will be an APL64 character vector.

Remarks:
⎕textrepl with 'overlap' prevented can be used instead of the assembler function TEXTREPL in the APL+Win ASMFNS.W3 workspace.

⎕textrepl with 'overlap' permitted performs successive replacements like in Microsoft Word, Microsoft Excel, Notepad and Notepad++.

Examples:

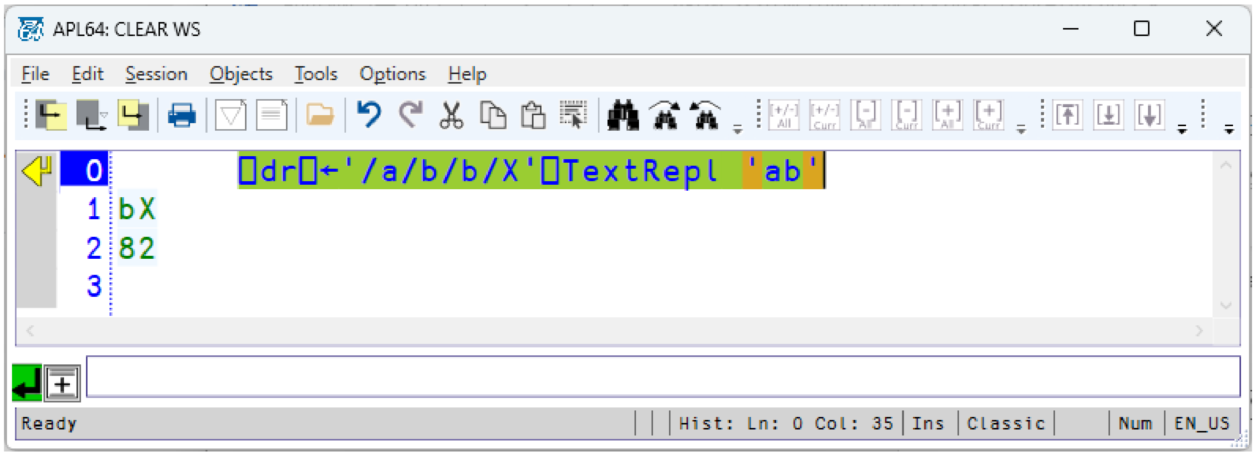
```
⎕dr⎕←'/N/H/IS/WAS/TI/GA/!/?' ⎕textrepl 'NOW IS THE TIME!'
```



```
0        ⎕dr⎕←'/N/H/IS/WAS/TI/GA/!/?' ⎕textrepl 'NOW IS THE TIME!'
1 HOW WAS THE TIME!
2 82
3
```

Ready | Hist: Ln: 3 Col: 6 | Ins | Classic | Num | EN_US

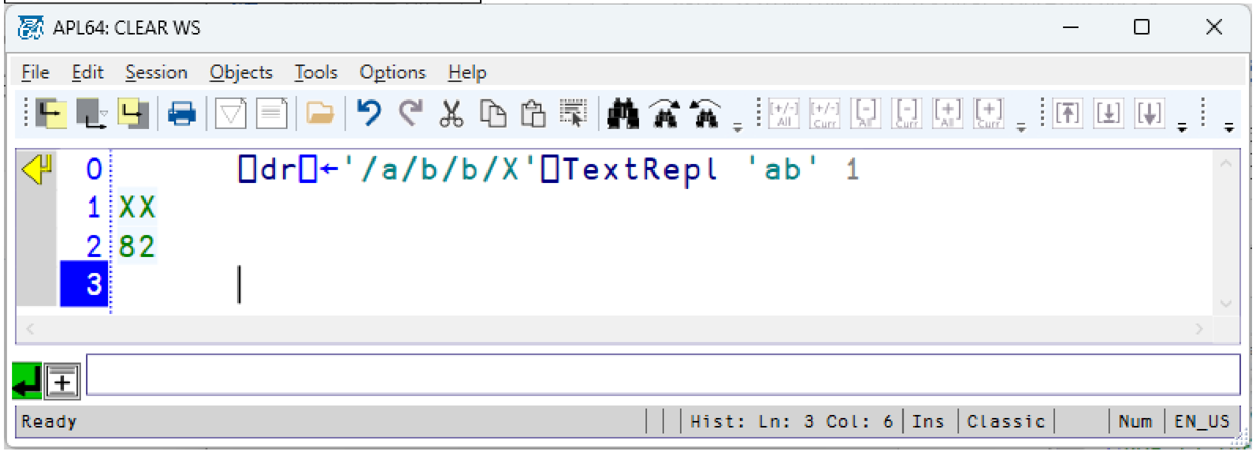
```
⎕dr⎕←'/a/b/b/X'⎕TextRepl 'ab'
```



```
0        ⎕dr⎕←'/a/b/b/X'⎕TextRepl 'ab'
1 bX
2 82
3
```

Ready | Hist: Ln: 0 Col: 35 | Ins | Classic | Num | EN_US

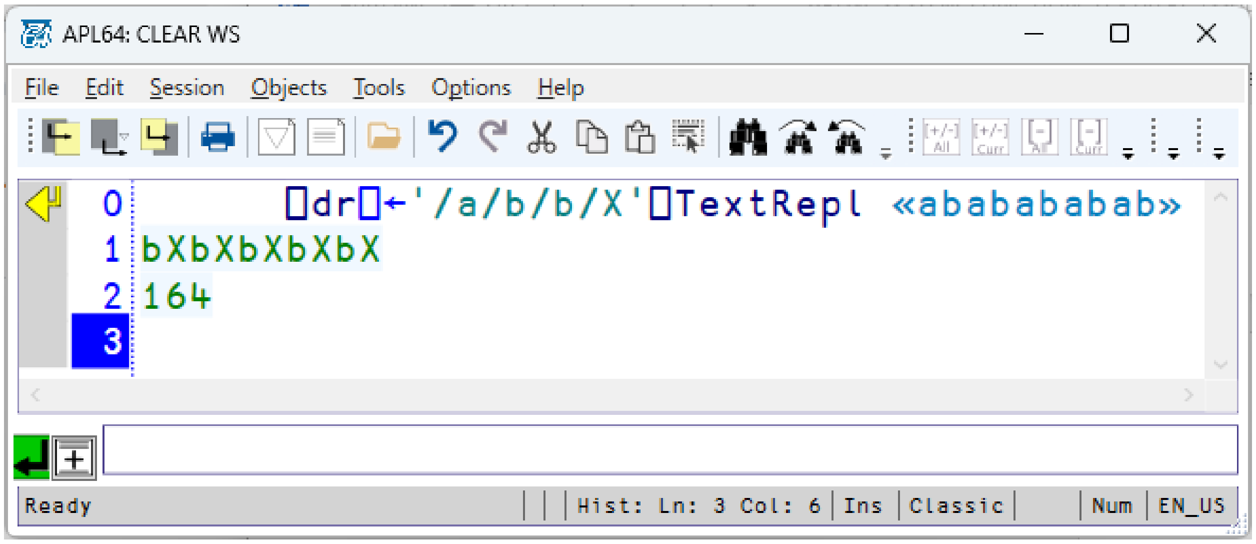
```
⎕dr⎕←'/a/b/b/X'⎕TextRepl 'ab' 1
```



```
0        ⎕dr⎕←'/a/b/b/X'⎕TextRepl 'ab' 1
1 XX
2 82
3
```

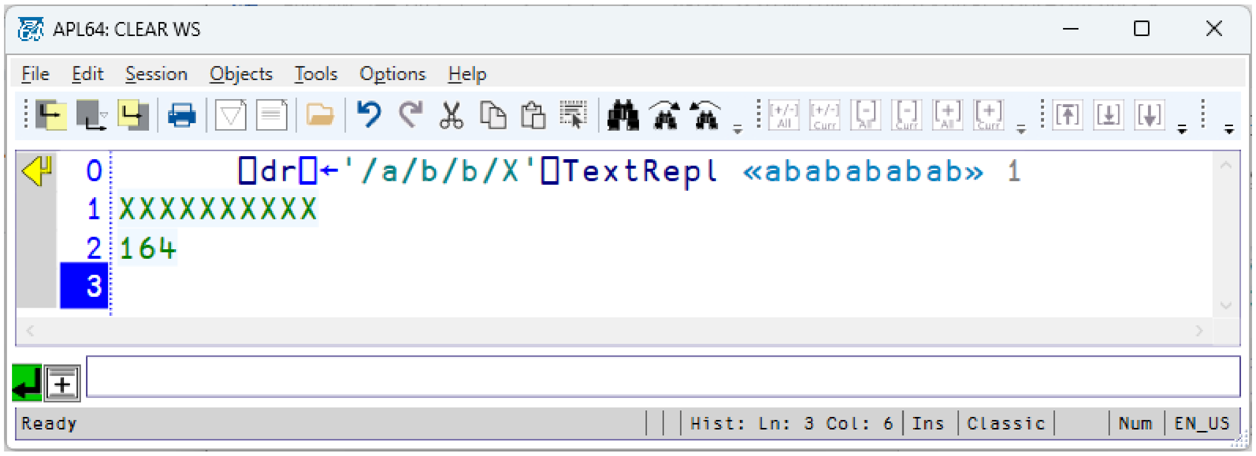
Ready | Hist: Ln: 3 Col: 6 | Ins | Classic | Num | EN_US

```
⎕dr⎕←'/a/b/b/X'⎕TextRepl «ababababab»
```



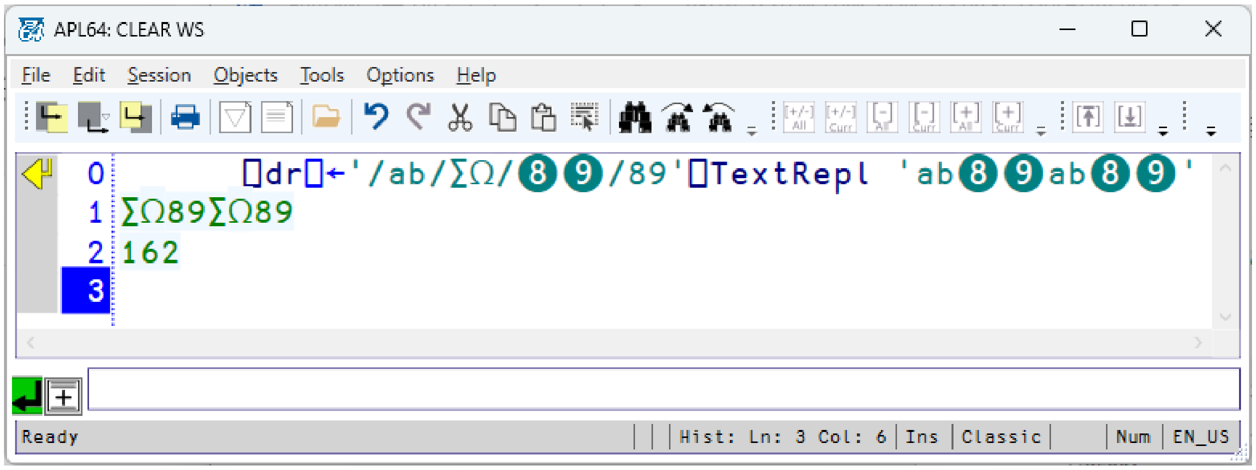
```
      ⎕dr⎕←'/a/b/b/X'⎕TextRepl «ababababab»
bXbXbXbXbX
164
```

⎕dr⎕←'/a/b/b/X'⎕TextRepl «ababababab» 1



```
      ⎕dr⎕←'/a/b/b/X'⎕TextRepl «ababababab» 1
XXXXXXXXXX
164
```

⎕dr⎕←'/ab/∑Ω/❽❾/89'⎕TextRepl 'ab❽❾ab❽❾'



```
      ⎕dr⎕←'/ab/∑Ω/❽❾/89'⎕TextRepl 'ab❽❾ab❽❾'
∑Ω89∑Ω89
162
```

⎕tf Performance Counter Frequency

Purpose:
Returns the frequency of the performance counter in terms of number of timer ticks per second.

Syntax: result ← ⎕tf

Result:
If the function succeeds, the return value is non-zero.
The frequency is fixed at system boot and needs only be queried at the beginning of a new session, and the result can be cached.

Caution:
If the installed hardware doesn't support a high-resolution counter, the return value is zero.

Example:
The following expression provides the frequency of the performance counter:

```
     ⎕tf
 2648436
```

⎕throw Throw Exception

Purpose:
Generates a user-defined error exception.

Syntax: ⎕throw 'message'

Argument:
message is a diagnostic message; that is, a character singleton or vector that contains the first line of the diagnostic message associated with the resulting error exception.

Effect:
⎕throw is like ⎕error except it throws the error in the context of the calling function rather than exiting from the function before throwing it. This allows errors thrown in a :Try block to be handled locally rather than exiting from the function.

⎕throw provides facilities:

- the ability of a function to signal an exception natively rather than popping the calling function of the ⎕si stack before signaling the error
- eliminates the need for ERROR helper functions (because natively signaled)
- the ability to signal an error from the :catch clause of a nested inner :tr clause to the :catch clause of an outer :try clause
- behaves like ⎕error when called by an ⎕elx handler.

If message is empty (''), ⎕throw does not signal an exception; this permits conditional signaling of error exceptions with a statement of the form ⎕throw condition /'message'.

Example:
The example creates an error that emulates a native error.

```
    ∇ THROW
[1]   :try
[2]       ⎕throw 'DOMAIN ERROR'
[3]   :catchall
[4]       'caught error'
[5]   :endtry
    ∇
     THROW
caught error
```

⎕trace Trace Function Execution

Purpose:
Aids in debugging a program by allowing you to flag lines of functions for diagnostic output when the system next executes them.

Syntax: result ← linenums ⎕trace 'fnname'

result ← ⎕trace 'fnname'

Argument:
The result is an integer vector of the lines of fnname for which tracing was in effect until this execution of ⎕trace. If you use ⎕trace monadically, the function reports the settings. If you use it dyadically, it reports the prior settings and replaces them with the lines specified in linenums. You can clear all trace flags for a function with: ⍬ ⎕trace fnname

Effect:
⎕trace does not trace output as its direct result. Instead, it flags lines in a function so that, in future execution, the system produces diagnostic output.

During execution of a function that is being traced, the system displays the final value that is calculated in each statement on each traced line. The value appears on the next line after the function's name and the bracketed line number or bracketed line number.part in the case of secondary statements on diamondized lines. This is true even for values that would not display in normal (untraced) execution. In the case of a branch in the function, the system displays the value to which the function branched after the function’s name and the bracketed line number.

The resulting ability to observe the sequence in which the system executes the lines and the internal values of statements (those not normally displayed) is a valuable aid in debugging a program.

The system saves and reloads trace settings with a workspace, but does not copy them along with the function to which they apply when you use ⎕copy,)copy, ⎕pcopy, or)pcopy. Redefining a function with either ⎕def or ⎕fx removes all trace setting from that function.

Editing a function line with ⎕defl removes any setting associated with that line of code. Locking a function with ⎕lock removes all trace settings in the function.

Remark:
There are several key notable differences in the output in APL64 from APL+Win:

1. Output from any traced statement will always be displayed on the next line following the *function[line#]* prefix
2. Branch, scalar, and vector outputs will not be indented
3. Rather than displaying diamonds as prefixes for secondary statements on diamondized lines, the output will instead display as *function[line#.part]* such as GO[3.1]

Example:
The example sets trace flags on all the lines of the function GO; the empty explicit result means no traces were previously set. The last line is the explicit result of the function.

```
    ∇ RESULT←GO
[1]   'LINE OF OUTPUT!'
[2]   RESULT←1
[3]  L1:RESULT←(0,RESULT)+RESULT,0 ⋄ →L1×4>⍴RESULT
    ∇

      (⍳5) ⎕trace 'GO'

      GO
GO[1]
LINE OF OUTPUT1
GO[2]
1
GO[3.1]
1 1
GO[3.2]
→3
GO[3.1]
1 2 1
GO[3.2]
→3
GO[3.1]
1 3 3 1
GO[3.2]
→0
1 3 3 1
```

⎕translate Translate Character Array

Purpose:
Translates characters of a character array to other arbitrary characters.

Syntax: newchars ← 'transtab' ⎕translate 'chars'

Arguments:
The left argument specifies the translate table that you want to use.
The nth character is the replacement for ⎕av[n]. The right argument specifies the text that you want to modify.

Remark:
⎕translate is the replacement for the assembler function TRANSLATE in the ASMFNS.w3 workspace in APL+Win.

Example:
Replace ⎕tcbel and ⎕tcff characters with question marks.

```
     TT←⎕av
     TT[⎕io+7 12]←'?'
     NEW←TT ⎕translate 'A',⎕TCBEL,'B',⎕TCFF,'C'
     NEW
A?B?C
```

⎕ts Current Timestamp

Purpose:
Returns the current date and UTC time of day as represented by the system clock.

Syntax: result ← ⎕ts

Result:
The result is a seven-element numeric vector that contains the following information:
[1] year
[2] month
[3] day
[4] hour
[5] minute
[6] second
[7] millisecond

⎕ts relies on the system clock that the operating system maintains for its time measurement. The seventh element of the result is included for consistency with other APL+ systems. However, the computer system's clock precision determines if this element provides useful information.
The first three elements in the result of ⎕ts always indicate a date, and the last four elements always indicate a UTC time of less than 24 hours.

Example:

```
     ⎕ts
2021 2 2 12 36 12 379
```

⎕tt High-Resolution Timer Tick Count

Purpose:
Returns the number of ticks since the start of the APL64 session.

Syntax: result ← ⎕tt

Result:
If the function succeeds, the return value indicates the tick interval between now and the start of APL64 session.

The measurement is high resolution and is based on a hardware counter that is independent of any external time-of-day reference.

When you divide ⎕tt by ⎕tf, the result will be the elapsed time in seconds since the APL64 session started; basically the same value as ⎕ai[2] but without the added overhead of computing the other elements of ⎕ai and without the need to extract the second element of ⎕ai to access the time information.

Caution:
The resolution, precision, accuracy and stability of the counter are affected by the characteristics of the low-level hardware clock.

Example:
The following expression provides the number of ticks since starting the APL64 session:

```
      ⎕tt
 7514246
```

⎕tt32 High-Resolution Timer Tick Count for APLNow32 operations

Purpose:
Returns the number of ticks for all APLNow32 operations (⎕wi and ⎕wcall) since the start of the APL64 session to the time the value of ⎕tt32 was obtained.

Syntax: result ← ⎕tt32

Result:
If the function succeeds, the return value indicates the tick interval between now and the start of APL64 session for all APLNow32 operations. APLNow32 operations include application activities performed using ⎕wi and ⎕wcall.

The measurement is high resolution and is based on a hardware counter that is independent of any external time-of-day reference.

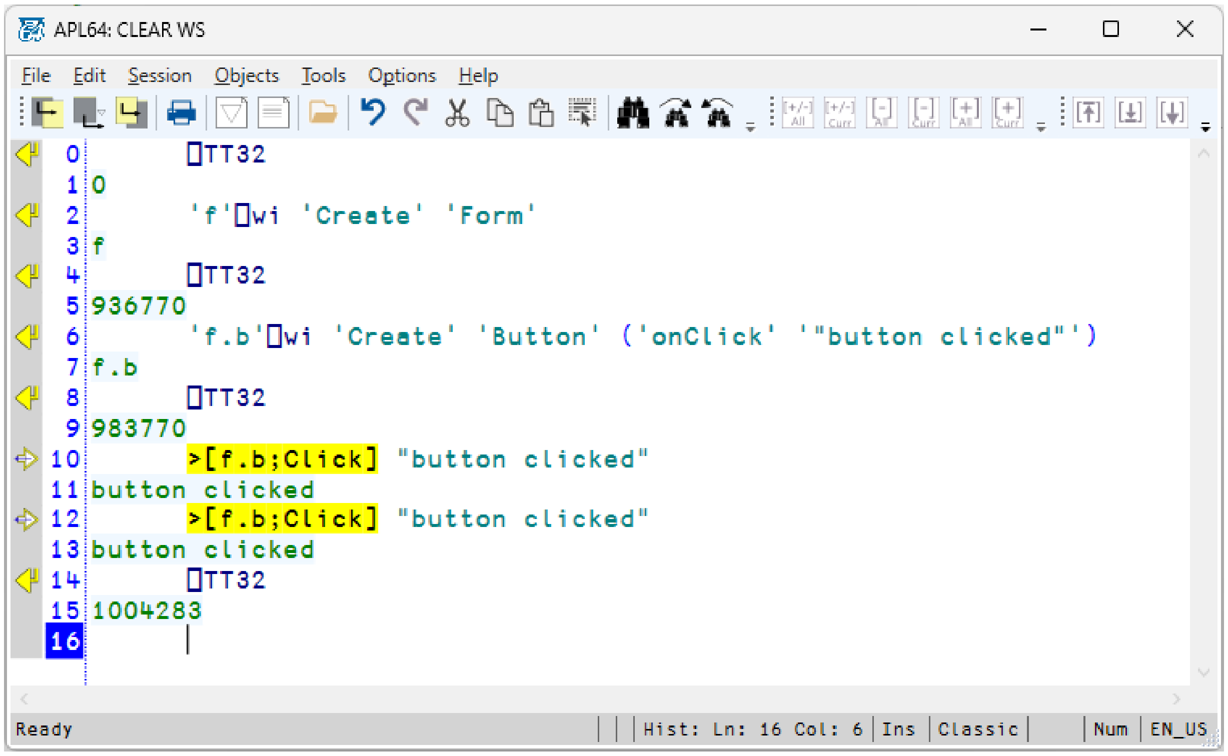
When you divide ⎕tt32 by ⎕tf, the result will be the elapsed time in seconds.

The measurement may be used to measure the performance of specific areas of an APL64-based application system which targets the Windows operating system.

Remarks:
Note that ⎕tt32 time counts the entire time spent inside ⎕wi 'Wait' operations. But the time spent inside a ⎕wi 'Wait" does not get accumulated into the ⎕tt32 total until the Wait ends. During the Wait the ⎕tt32 time does not change unless other ⎕wi operations take place (such as during callbacks). But the ⎕tt32 time continues to "tick" regardless of whether you are inside or outside of a Wait state on a ⎕wi object.

Caution:
The resolution, precision, accuracy and stability of the counter are affected by the characteristics of the low-level hardware clock.

Example:



```
      ⎕TT32
0
      'f'⎕wi 'Create' 'Form'
f
      ⎕TT32
936770
      'f.b'⎕wi 'Create' 'Button' ('onClick' '"button clicked"')
f.b
      ⎕TT32
983770
      >[f.b;Click] "button clicked"
button clicked
      >[f.b;Click] "button clicked"
button clicked
      ⎕TT32
1004283
```

⎕type Determine the Class of an Array

Purpose:
Returns a typical member of the class of arrays to which the argument belongs. This is a cover function for the Evolution Level 1 behavior of monadic epsilon (∊). At Evolution Level 2, monadic epsilon is the Enlist function.

Syntax: result ← ⎕type array

Argument:
array is any array.

Result:
result is an array that has the same nesting and shape as array. Each simple element in result is either a

blank, a 0, a Unicode space or an Object reference. The blanks represent character data; the 0s represent numeric data.

Examples:

```
      A ← 2 2 ρ 'Feather' (⍳3) 'your' ((⍳4) 'nest')

      ]display A                          ]display ⎕type A
.→-------------------------.        .→-------------------------.
↓.→------..→----.          |        ↓.→------..→----.          |
||Feather||1 2 3|          |        ||       ||0 0 0|          |
|'-------''~----'          |        |'-------''~----'          |
|.→---.   .→--------------.|        |.→---.   .→--------------.|
||your|   |.→------..→---.||        ||    |   |.→------..→---.||
|'----'   ||1 2 3 4||nest|||        |'----'   ||0 0 0 0||    |||
|         |'~------''----'||        |         |'~------''----'||
|         '∊--------------'|        |         '∊--------------'|
'∊-------------------------'        '∊-------------------------'

      ]display ' ' = ⎕type A
.→----------------------------------.
↓.→------------..→----.             |
||1 1 1 1 1 1 1||0 0 0|             |
|'~------------''~----'             |
|.→------.      .→-----------------.|
||1 1 1 1|      |.→------..→------.||
|'~------'      ||0 0 0 0||1 1 1 1|||
|               |'~------''~------'||
|               '∊-----------------'|
'∊----------------------------------'
```

⎕ucase Convert Lowercase to Uppercase

Purpose: Converts lowercase characters to uppercase characters.

Syntax: res ← ⎕ucase rArg

Argument:
arg is any APL64 value. The character or string elements of arg will be converted to upper case. Non-text elements of arg will not be modified in res.

Result:
res has the same data type, rank and shape as rArg

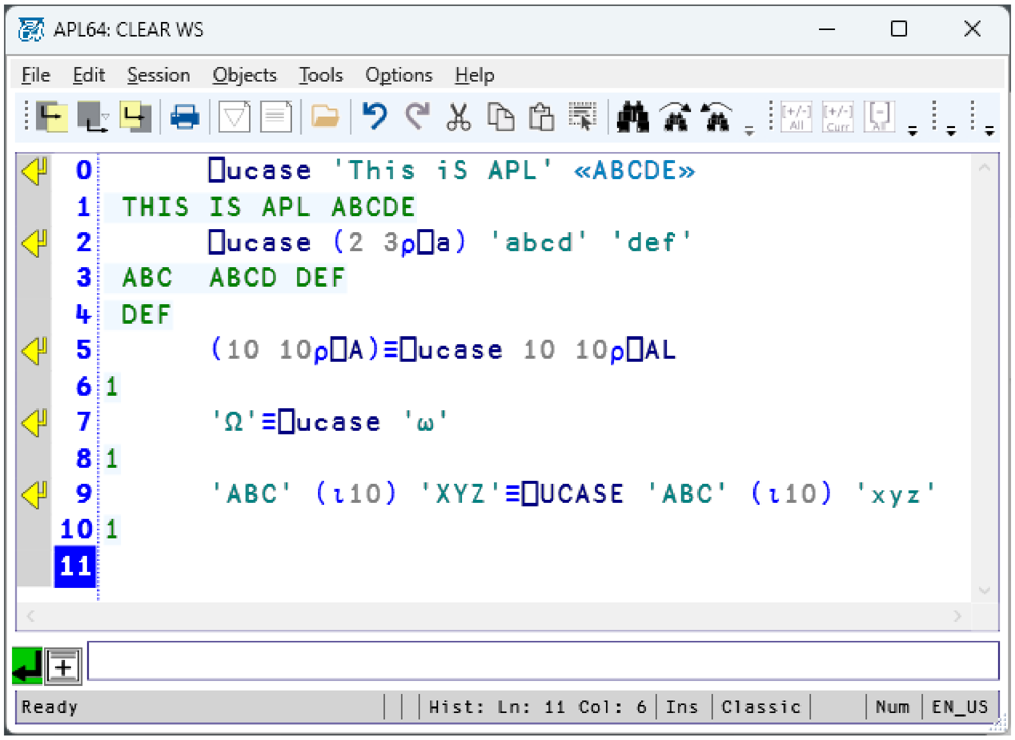
Remarks:
⎕ucase replaces the assembler function UPPERCASE in the ASMFNS.w3 workspace in APL+Win.

⎕ucase uses the .Net String.ToUpperInvariant method.

Example:

```
⎕ucase 'This iS APL' «ABCDE»
```

```
⎕ucase (2 3ρ⎕a) 'abcd' 'def'
(10 10ρ⎕A)≡⎕ucase 10 10ρ⎕AL
'Ω'≡⎕ucase 'ω'
'ABC' (ι10) 'XYZ'≡⎕UCASE 'ABC' (ι10) 'xyz'
```


APL64: CLEAR WS
File Edit Session Objects Tools Options Help
Ready | Hist: Ln: 11 Col: 6 | Ins | Classic | Num | EN_US

```
      ⎕ucase 'This iS APL' «ABCDE»
THIS IS APL ABCDE
      ⎕ucase (2 3ρ⎕a) 'abcd' 'def'
ABC  ABCD DEF
DEF
      (10 10ρ⎕A)≡⎕ucase 10 10ρ⎕AL
1
      'Ω'≡⎕ucase 'ω'
1
      'ABC' (ι10) 'XYZ'≡⎕UCASE 'ABC' (ι10) 'xyz'
1
```

⎕ucmd Execute User Command

Purpose:
Executes a user command under program control.

Syntax: result ← ⎕ucmd 'command'

Argument:
The argument to ⎕ucmd is any defined user command from any accessible user command processor file with or without the right bracket. You must also supply any arguments necessary to the user command.

Result:
The result of ⎕ucmd is the result of executing the command in the right argument. The result is 0 0ρ'' if the command does not produce a result. For example, the two lines below generate the same output in the session; depending on the user command, you may be able to capture the result when using the system function. See the User Command Processor chapter in the User Manual for more information.

```
   ]WSLOC VAR1
   ⎕ucmd 'WSLOC VAR1'
```

There is a row of ⎕si and a column of ⎕idloc to represent the user command processor level in the state indicator.

Example:

```
    ⎕ucmd 'FLIB 1234567'
FORMAT  INPUT  OUTPUT  REPORT
```

⎕ucs Universal Character Set

Purpose:
Converts between the Unicode character set datatype and numeric values representing Unicode code points.

Syntax: result ← [from_format [to_format]] ⎕ucs data

Argument:
from_format is the optional name of encoding used for input code values.
to_format is the optional name of the encoding used to output code values.
data is an array of integer-valued numbers or an array of Unicode characters.

Result:
Monadic ⎕ucs
result is an array with the same rank and shape as data; this will also be a Unicode character array when data is numeric, and a numeric array when data is Unicode characters. See ⎕type for related information.

Dyadic ⎕ucs
result is the translated data as a character or integer vector.

The list and descriptions of encodings that dyadic ⎕ucs supports:

Encoding	**Description**
X-AV-UCS	internal APL UCS datatype.
X-AV	fixed width, local 8-bit ⎕AV data (see Note 2).
X-AV.WIN	fixed width, APL64 8-bit ⎕AV data (see Note 2).
X-AV.VISUAL	fixed width, VisualAPL 8-bit ⎕AV data (see Note 2).
X-APL	fixed width, local 8-bit ⎕AV data tolerant for APL code (see Note 2).
X-APL.WIN	fixed width, APL64 8-bit ⎕AV data tolerant for APL code (see Note 2).
X-APL.WIN	fixed width, APL64 8-bit ⎕AV data tolerant for APL code (see Note 2).
X-TEXT	fixed width, local 8-bit ⎕AV data tolerant for ASCII (Nee Note 2).
X-TEXT.WIN	fixed width, APL64 8-bit ⎕AV data tolerant for ASCII (see Note 2).
UTF-8	multibyte transfer encoding, 1 or more bytes per code point (see Note 3).
UTF-16	multi- 16-bit transfer encoding, 1 or more 16-bit words per code point (see Note 3).
UTF-16BE	using big-endian 16-bit words (see Note 3).
UTF-16LE	using little-endian 16-bit words (see Note 3).
UTF-32	32-bit transfer encoding (see Note 3).

UTF-32BE	using big-endian 32-bit words (see Note 3).
UTF-32LE	using little-endian 32-bit words (see Note 3).
UCS-2	fixed width, 16-bit, UCS data, integer (4-byte) (see Note 4).
UCS-2BE	fixed width, 16-bit, UCS data, character big endian (see Note 4).
UCS-2LE	fixed width, 16-bit, UCS data, character little endian (see Note 4).
UCS-4	fixed width, 32-bit, UCS data, integer (see Note 4).
UCS-4BE	fixed width, 32-bit, UCS data, character big endian (see Note 4).
UCS-4LE	fixed width, 32-bit, UCS data, character little endian (see Note 4).

Note 1: The left arguments are optional; the default is X-AV-UCS.

Note 2: Fixed 8-bit character encodings. On input, these must all be passed as character data. If passed as UCS Character data, they will be coerced to 8-bit character data before being translated. An error is signaled if the coercion fails.

Note 3: Variable width transfer encodings. On input, these must all be passed as character data, with the same caveat in note 1 above.

Note 4: Fixed width representations of Unicode code points as numeric values. On input, UCS-2 and UCS-4 must be APL numeric data coercible to integer. The BE and LE variants must be character data coercible to 8 bit characters.

Remarks:
Use X-TEXT when passing data that does not contain APL characters.

Use X-AV when passing data containing APL characters. This is the encoding to use when outputting data to a source code management system because what goes out should be the same as what comes in.

Use X-APL when passing data native to the user's current APL system.

Examples:

```
      ⎕ucs 'abc'
97 98 99
      ⎕ucs 169 8364 163 165 174 63
©€£¥®?
      ⎕ucs '©€£¥®?'
169 8364 163 165 174 63
      text←'This is sample Unicode string.'
      ⎕ucs text
84 104 105 115 32 105 115 32 115 97 109 112 108 101 32 85 110 105 99
111 100 101 32 115 116 114 105 110 103 46
      text ≡ ('UCS-2' 'X-TEXT') ⎕ucs ('X-TEXT' 'UCS-2') ⎕ucs text
1
```

⎕ul User Load

Purpose:
Returns the number of users currently sharing the system.

Syntax: result ← ⎕ul

Result:
The explicit result of ⎕ul is a numeric scalar 1.

Remarks:
⎕ul exists in APL64 for compatibility with other APL systems, so that you can run programs brought in from these systems without modifying them to eliminate ⎕ul.

⎕userid User Identification

Purpose:
Returns the user identification of a machine on the network.

Syntax: result ← ⎕userid

Result:
The explicit result of ⎕userid is a character vector that contains the user identification of that particular machine on the network. The result depends on your operating system and the network software you are using.

⎕userpath Path to the OS-Specific User Folder

Purpose:
Returns a character vector containing the path defined by .Net.

Syntax: result ← ⎕userpath

Domain:
result is a character vector that contains the user path that the user has read/write access to. This value is the default path for:

(a) ⎕sysconfigfile
(b) ⎕ucmdfile

Example:

```
    ⎕userpath
C:\Users\Joe\AppData\Roaming\APLNowLLC\APL64
```

⎕valence Get Valence for Current Function

Purpose:
Returns the valence for the current function.

Syntax: result ← ⎕valence

Result:
The result of ⎕valence is 0, 1, or 2 for niladic, monadic, or dyadic invocation of the current function. Note that 1 is returned for functions that are declared dyadically but called monadically and 2 is only returned when a function is declared dyadically and called dyadically. It returns 0 if there are no functions on the ⎕SI stack (such as when called from immediate execution with a clear ⎕SI stack).

⎕vget Get Variable Value with Specified Scope

Purpose:
Returns the value of a variable at local or global scope.

Syntax: result ← [scope] ⎕vget varname [default_value]

Argument:
varname is a character scalar or vector that contains one variable name;
default_value is an optional default value to be returned if the variable name in varname is not defined;
scope is an optional numeric argument that specifies the scope at which names are interpreted, where 1 = local (default); and 0 = global. If you omit the argument, it is equivalent to specifying 1 for local scope.

Result:
The result is an array with the same rank, shape and value as the named variable. If you try to access a variable name that is not defined a VALUE ERROR is generated unless you have specified a default value. If a default value has been specified in the second element of right argument, then that value is returned instead of generating an error. This is very useful for evaluation of optional left arguments in function calls.

Example:

```
     ⎕←VAR←'GLOBAL DATA'
GLOBAL DATA
     ⎕VR 'TEST'
   ∇ NAME TEST N;VAR
[1]   VAR←'LOCAL DATA'
[2]   ⎕←N ⎕VGET NAME 'DEFAULT VALUE'
   ∇

     'VAR' TEST 0
GLOBAL DATA
     'VAR' TEST 1
LOCAL DATA
    'NONAME' TEST 1
DEFAULT VALUE
```

⎕vi Verification of Formatted Input

Purpose:
Provides a validity check on an input character vector (often used in conjunction with ⎕fi).

Syntax: result ← ⎕vi 'data'

Argument:
data is a character representation of data that are presumably numeric separated by spaces.

Result:
The result is a Boolean vector with 1s in the positions where groups of characters represent well-formed numbers, and 0s where they do not.

Examples:

```
     A←'666 ¯1.20  .1
314159E¯5'
     ⎕fi A
666 ¯1.2 0.1 3.14159
     ⎕vi A
1 1 1 1

     B←'ANSWER IS 666 LBS.'
     ⎕fi B
0 0 666 0
     ⎕vi B
0 0 1 0
     (⎕vi B)/⎕fi B
666
```

⎕vr Visual Representation of a Function

Purpose:
⎕vr returns the visual representation of a function as a character vector. The APL64 programmer can select the format of the result of ⎕vr. The result of the ⎕vr system function is accepted as an argument to the ⎕def system function.

Syntax: result ← [scope] ⎕vr 'fnname' [op]

Arguments:

scope is an optional numeric argument that specifies the scope at which names are interpreted, where 1 = local (default); and 0 = global**.**

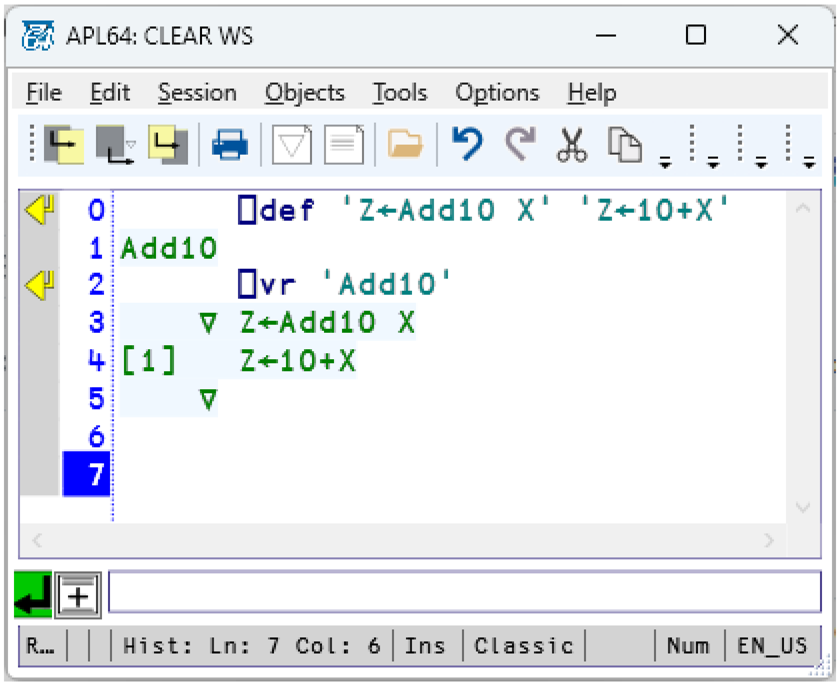
fnname is a character scalar or vector that contains one function name. If fnname is a character scalar or vector but does not contain the name of an unlocked, APL64 developer-created function, result is an empty vector.

op is optional argument indicating the desired format of the result. This argument is not case-sensitive.

op Value	**Description**
op argument not provided	Traditional ⎕vr format (Compatible with APL+Win)

'InclLFD'	Traditional ⎕vr format with last fix date appended
'Xml'	Xml format including editing history
'Json'	Json format including editing history

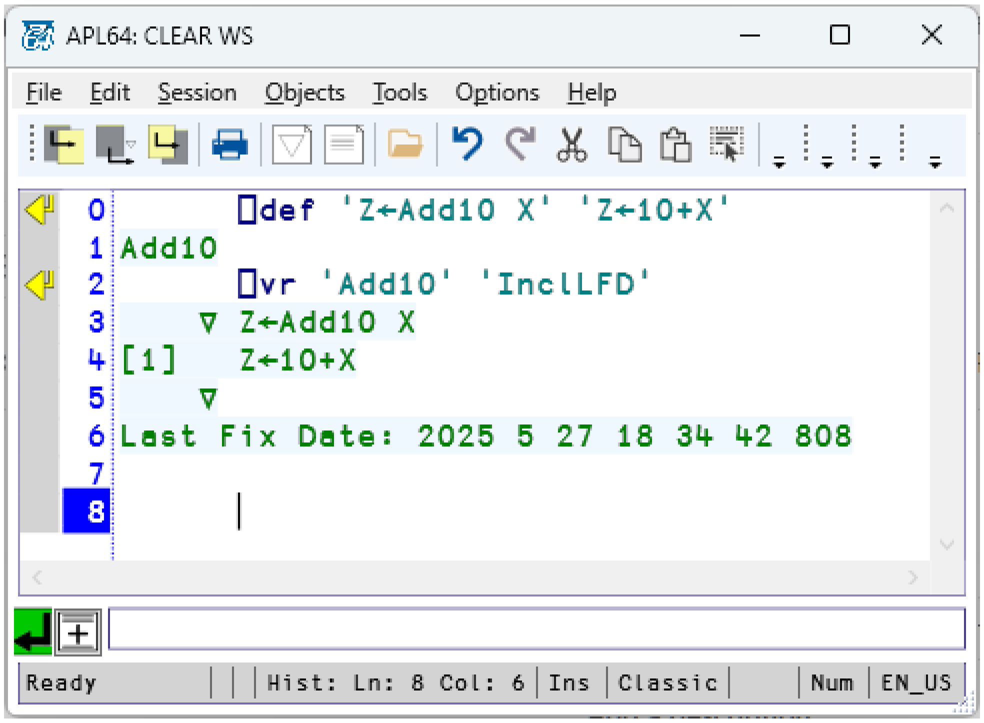
result: No op argument provided: result ← scope ⎕vr 'fnname'
The result is a character vector, visual representation of the function with bracketed line numbers and embedded newline characters separating the function lines. The result is not affected by ⎕pw. The result preserves the 'as edited' text of the function.


APL64: CLEAR WS
File Edit Session Objects Tools Options Help
R… Hist: Ln: 7 Col: 6 Ins Classic Num EN_US

```
      ⎕def 'Z←Add10 X' 'Z←10+X'
Add10
      ⎕vr 'Add10'
    ∇ Z←Add10 X
[1]   Z←10+X
    ∇
```

result: op argument provided as 'InclLFD': result ← scope ⎕vr 'fnname' 'InclLFD'

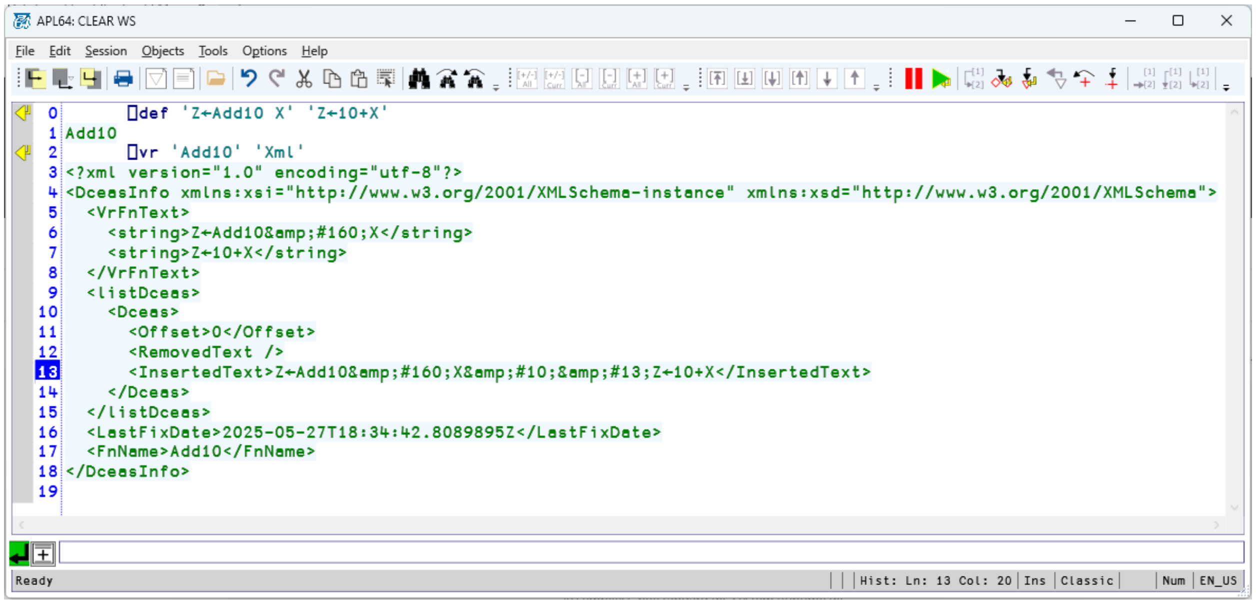
The result is the same as the no op argument provided, except that last fix date (UTC) added at the end of the result:



```
      ⎕def 'Z←Add10 X' 'Z←10+X'
Add10
      ⎕vr 'Add10' 'InclLFD'
    ∇ Z←Add10 X
[1]   Z←10+X
    ∇
Last Fix Date: 2025 5 27 18 34 42 808
```

Result: op argument provided as 'Xml': result ← scope ⎕vr 'fnname' 'Xml'

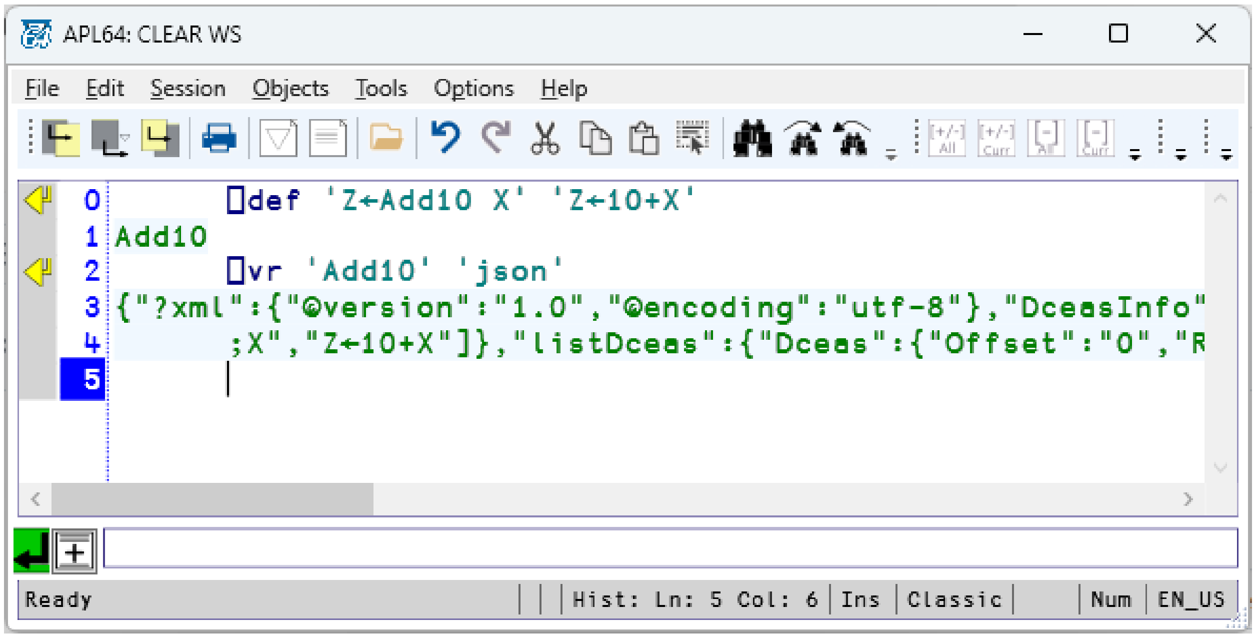
The result is an XML-format text vector which includes the function text, the uncommitted editing changes, the function name and the last fix date. Because the XML format is used, the new line characters are 'escaped'.



```
      ⎕def 'Z←Add10 X' 'Z←10+X'
Add10
      ⎕vr 'Add10' 'Xml'
<?xml version="1.0" encoding="utf-8"?>
<DceasInfo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <VrFnText>
    <string>Z←Add10&amp;#160;X</string>
    <string>Z←10+X</string>
  </VrFnText>
  <listDceas>
    <Dceas>
      <Offset>0</Offset>
      <RemovedText />
      <InsertedText>Z←Add10&amp;#160;X&amp;#10;&amp;#13;Z←10+X</InsertedText>
    </Dceas>
  </listDceas>
  <LastFixDate>2025-05-27T18:34:42.8089895Z</LastFixDate>
  <FnName>Add10</FnName>
</DceasInfo>
```

result: op argument provided as 'Json': result ← scope ⎕vr 'fnname' 'json'

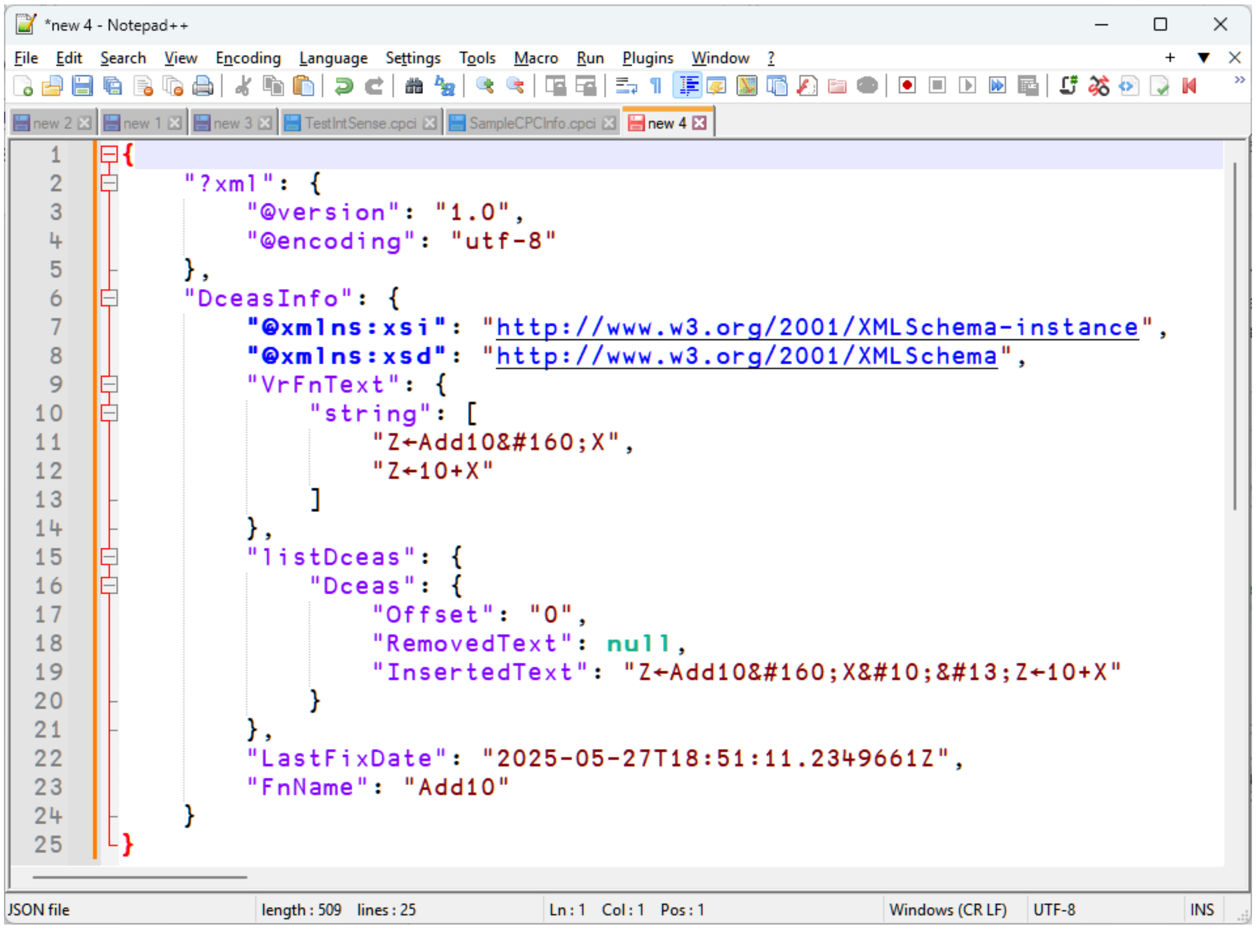
The result is JSON-format text vector which includes the function text, the uncommitted editing changes, the function name and the last fix date. Because the XML format is used, the new line characters are 'escaped'. Json format is generally more compact than Xml format.



```
        ⎕def 'Z←Add10 X' 'Z←10+X'
Add10
        ⎕vr 'Add10' 'json'
{"?xml":{"@version":"1.0","@encoding":"utf-8"},"DceasInfo"
        ;X","Z←10+X"]},"listDceas":{"Dceas":{"Offset":"0","R
```

```
Ready | | Hist: Ln: 5 Col: 6 | Ins | Classic | | Num | EN_US
```

Notepad++ may be used to conveniently view the JSON format result:


*new 4 - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
new 2
new 1
new 3
TestIntSense.cpci
SampleCPCInfo.cpci
new 4

```
{
    "?xml": {
        "@version": "1.0",
        "@encoding": "utf-8"
    },
    "DceasInfo": {
        "@xmlns:xsi": "http://www.w3.org/2001/XMLSchema-instance",
        "@xmlns:xsd": "http://www.w3.org/2001/XMLSchema",
        "VrFnText": {
            "string": [
                "Z←Add10&#160;X",
                "Z←10+X"
            ]
        },
        "listDceas": {
            "Dceas": {
                "Offset": "0",
                "RemovedText": null,
                "InsertedText": "Z←Add10&#160;X&#10;&#13;Z←10+X"
            }
        },
        "LastFixDate": "2025-05-27T18:51:11.2349661Z",
        "FnName": "Add10"
    }
}
```

⎕wa Work Area Available

Purpose:

Returns the current amount of work area available in the active workspace (in bytes).

Syntax: result ← ⎕wa

Result:

The result is a numeric scalar whose value is the current number of unused bytes in the active workspace.

Example:

```
      ⎕wa
1.702108059E10
```

⎕watchpoint Watch Point

Purpose:

Internal system-created function used only in the debugger when debugging with Watch Points.

Syntax: N/A

Effect:
Debugging with Watch Points created the function ⎕watchpoint. You must branch out of the system-created function or past the line that caused the suspension to continue your function.

Note: Refer to the section *Debugging with Watchpoints* in **Chapter 2: Working in an APL64 Session** in the APL64 User Manual for additional information.

⎕wcall Windows Call

Purpose:
Makes calls to Windows libraries (DLLs).

Syntax: ⎕wcall 'function' [arguments]

Arguments:
'function' is either a function identified in the ADF/INI file or one of a small group of built-in functions; arguments are any arguments required for function.

Effect:
When you use ⎕wcall, you call Windows in an event-enabled style; while you remain in this call, you can receive callbacks. Callbacks can be either ⎕wi callbacks (such as execution of an onClick event handler) or ⎕wcall filter-callbacks.

The W_Ini function is used to define or reference the WCALL definition in the APLW.INI and/or APLW.ADF file. When you reference a string value with W_Ini it has historically never differentiated between strings defined in the INI file versus those defined in ADF file. It returned the value found in either INI or ADF file. Referencing a string via W_Ini returns the first definition found in W_Def cache first, then the INI and ADF files (unless the [Config]IniFirst=0 is defined in APLW.INI in which case APLW.ADF is accessed before APLW.INI).

The W_Def function is used in the same way as W_Ini except that it lets you create WCALL definitions and other strings in your application that are stored into a local cache (memory) rather than written into the INI file. This permits setting WCALL definitions in a running application that might have a read-only INI file. These strings should execute faster than W_Ini because no INI file reading or writing is involved. You can get a list of all strings in a section by calling: ⎕WCALL 'W_Def' '[SectionName]' and a list of all strings in all sections by calling: ⎕WCALL 'W_Def' '[]'.

The W_Reset function is used to reset the WCALL definitions created by W_Def so they can be reloaded without restarting APL64.

To learn more about ⎕wcall in the APL64, view the section Calling Dynamic Link Libraries in Help | APL+Win Compatibility | Windows GUI Reference.

⎕wgive Give (Yield) CPU to Windows

Purpose:
Allows you to yield the CPU to Windows during execution of a long-running event handler, callback, or immediate execution calculation.

Syntax: ⎕wgive int

Argument:
int is an integer scalar; valid values are ¯1, 0, or any positive value (note: all positive values are equivalent).

Effect:
When you are inside a callback, ⎕wgive temporarily yields the CPU to Windows. This allows your application to process user input that occurs while your calculation is executing. If you do not specifically give the CPU back to Windows, then your application does not respond to input until that callback completes execution. In a case when your callback may be a long calculation, you may want to know if the user clicks the Cancel button to abort execution.

To yield the CPU to Windows at a specific point in your event handler, use a right argument of 0. This allows Windows to handle any messages that have come in and then returns control to the handler.

If you specify any nonzero integer, you yield the CPU to Windows throughout the event handler at every available opportunity. In practice, an opportunity occurs approximately one to three times for every APL primitive. You may have to experiment with the execution characteristics of your specific application to find the best balance of execution efficiency versus input responsiveness.

When your calculation lends itself to being broken down into small incremental units, it is sometimes better to use a Timer with interval 0 to carry out background processing rather than using ⎕wgive with a nonzero argument. This avoids the complications that can arise with ⎕wgive.

If you do use ⎕wgive with non-zero values, you can reset it to zero; this yields control to Windows. Alternatively, you can use ¯1 to reset the value to zero without actually yielding.

Example:
In this example, ⎕wgive is called with an argument of 0 to yield the CPU just before processing the record.

```
   ∇ Update file
[1] ⍝ Process a list of file updates
     …
[9]  :while rec < limit
[10]    ⎕wgive 0 ⋄ ⎕fhold inputfile
[11]    (Process ⎕fread inputfile, rec) ⎕fappend outputfile
[12] :end
   ∇
```

In this example, processing continues unless the user presses the Cancel button. Using a nonzero argument to ⎕wgive allows the user's input to be recognized.

```
    'fmStatus.bnCancel' ⎕wi 'onClick' 'canceled"1'

    ∇ UpDate file
[1]  ⍝ Process a list of file updates
[2]  ⎕wgive 100 ⋄ canceled"0
     …
[9]  :while rec < limit
[10] :and ~canceled
[11]    ⎕fhold inputfile
[12]    (Process ⎕fread inputfile, rec) ⎕fappend outputfile
[13] :end
    ∇
```

⎕where Determine Element Indices

Purpose:
Determines the indices of the non-zero elements of the argument. (Sensitive to ⎕io.)

Syntax: indices ← ⎕where numvec

Arguments:
The argument is a numeric vector, usually Boolean.

Result:
The result is a vector of indices equivalent to BV/⍳⍴BV for a Boolean vector, or (0≠V)/⍳⍴V for an arbitrary numeric vector.

Remarks:
⎕where is the replacement for the assembler function WHERE in the ASMFNS.w3 workspace in APL+Win.

Example:

```
      ⎕io←1 ⋄ ⎕WHERE 1 0 2 0 0 3 0 0 0 0
1 3 6
```

⎕wi Windows Interface

Purpose:
Creates, controls, and runs Graphical User Interfaces (GUIs.)
To learn more about ⎕wi in the APL64 Developer version go to **Help | APL+Win Compatibility | Windows GUI Reference**.

Syntax: result ← ['object'] ⎕wi 'action' [arguments]

Arguments:
object is the fully qualified object name;
action is the action to be performed on object;
arguments are arguments to the action, which may be required or optional depending on the action.

The left argument is a fully qualified object name containing one or more names separated by periods, such as 'fmMain' or 'fmMain.bnOk'. The special object name '#' is recognized as the System Object. It corresponds to the Windows desktop and has a limited number of properties, methods, and events associated with it. If you omit the left argument, ⎕wi uses the contents of ⎕wself, the current object.

Although APL64 accepts an instance number (an integer enclosed in brackets at the end of a name) for compatibility with APL+Win, this syntax has no special meaning. The brackets and number are simply characters in the name.

If the right argument is a simple character vector, it specifies an action without an argument. If it is a nested vector, the first item is the action and all subsequent items form the arguments to the action.

The action portion of the right argument can also use relative-object-name prefixes before the action name. See the Programming the Windows Interface: Introduction chapter in the Windows Reference Manual for more information.

Result:
The result is 0 0 ρ0 for most methods and for setting a property value. When referencing a property value, the result is the value of that property for the object you specify. Some methods, such as New, also return a result. (See the Graphical User Interface: Methods chapter in the Windows Reference Manual for information on which methods return a result.)

Effect:
⎕wi is the central feature of the interface between your application and the object-oriented user interface facilities in Windows. You use this function to:

- create forms and controls
- set and reference properties
- invoke methods
- associate handlers with events
- wait for user input to trigger event-handler callbacks.

The right argument to ⎕wi determines what action occurs during the call. The left argument determines the object to which the action applies. Properties, event-handler properties, and methods are documented in the Graphical User Interface chapters in the Windows Reference Manual.

When you call ⎕wi, one of the effects is to allow internally generated events to occur. For example, if you set the text property of an Edit control, a Change event occurs. If you define an onChange event-handler, the system executes it before ⎕wi returns from setting the property. Externally generated events (from user input such as keyboard, mouse, or DDE) occur in the application only when the Wait method or ⎕wgive is used or when ⎕si is empty.

Examples:

```
⍝ Create a new form:
    'fmMyform' ⎕wi 'New' 'Form'
⍝ Add a new button to the form with a caption and location:
    'fmMyform.bn1' ⎕wi 'New' 'Button' ('caption' 'My Button') ('where' 8 15 1.5 10)
⍝ Set the button in the previous example to style 1 (default button):
    'fmMyform.bn1 '⎕wi 'style' 1
⍝ Query the value of the where property for the default object:
    'fmMyform.bn1 ⎕wi 'where'
18.625 5.875 15 40
⍝ Specify that the system runs the APL function FOO as the event handler when a Click event
occurs on the button:
    'fmMyform.bn1' ⎕wi 'onClick'  'FOO'
```

⎕wordrepl Replace Words in Character Vector

Purpose:
Replaces existing words in a character vector with new characters, as specified. ⎕WORDREPL does not affect partial words.

Syntax: newvec ← '/old/new/old1/new1/...' ⎕wordrepl 'vec'

Arguments:
The left argument is an alternating series of existing and replacement words. The first character is the delimiter.
The right argument is a character vector that contains the words you want to replace.

Example:

```
    '/HE/HIS/NOW/WHAT/!/?' ⎕wordrepl 'NOW IS THE TIME!'
WHAT IS THE TIME?
```

⎕wre Windows Runtime Executable

Purpose:
 Specified actions of the Windows Runtime Executable (WRE) creation utility in the APL64 developer version may be performed under program control using the ⎕WRE system function. These actions are performed using the memory-based, WRE configuration data set in the current APL64 developer version instance, which is not modified by the ⎕WRE system function actions.

The WRE dialogue and the ⎕WRE system function share the same current WRE data set. The current WRE data set remains available during the current APL64 developer version instance but is not automatically saved when the current APL64 developer version instance ends. If appropriate, use the ⎕WRE system function or the WRE dialogue to save the current WRE data set for future APL64 developer instances with a programmer-selected file name.

Syntax: result ← action ⎕wre argument

Arguments:
 action is the action to be performed. Action argument text is case-insensitive.

argument is the required argument to the action.

Result:
result is a two-element variable where the first element is an integer: 0 = success, ¯ 1 = not successful, and the second element is a character vector describing the action result.

⎕WRE New Action

Syntax: 'New' ⎕WRE ''

The New action will initialize the WRE configuration data set in the current APL64 developer version instance.

⎕WRE Load Action

Syntax: 'Load' ⎕WRE pathToWreConfigurationFile

The Load action will load a pre-prepared WRE xml-format configuration file into the WRE configuration data set in the current APL64 developer version instance.

⎕WRE Create Action

Syntax: 'Create' ⎕WRE ''

The Create action will create a WRE executable using the WRE configuration data set in the current APL64 developer version instance.

⎕WRE Run Action

Syntax: 'Run' ⎕WRE pathToWreExecutable waitForExit commandLineArguments

The Run action will start an instance of the WRE application specified by the three elements of the right argument:

Element #	**Data type**	**Right Argument Element Value**
1	String or Character vector	Full path to WRE executable
2	Integer	Wait For Exit (milliseconds) or 0/Infinite wait
3	String or Character vector	WRE executable command line arguments

The result of the ⎕WRE run action is available only after the earlier of the expiration of the 'Wait For Exit' value or the closing of the ⎕WRE application instance.

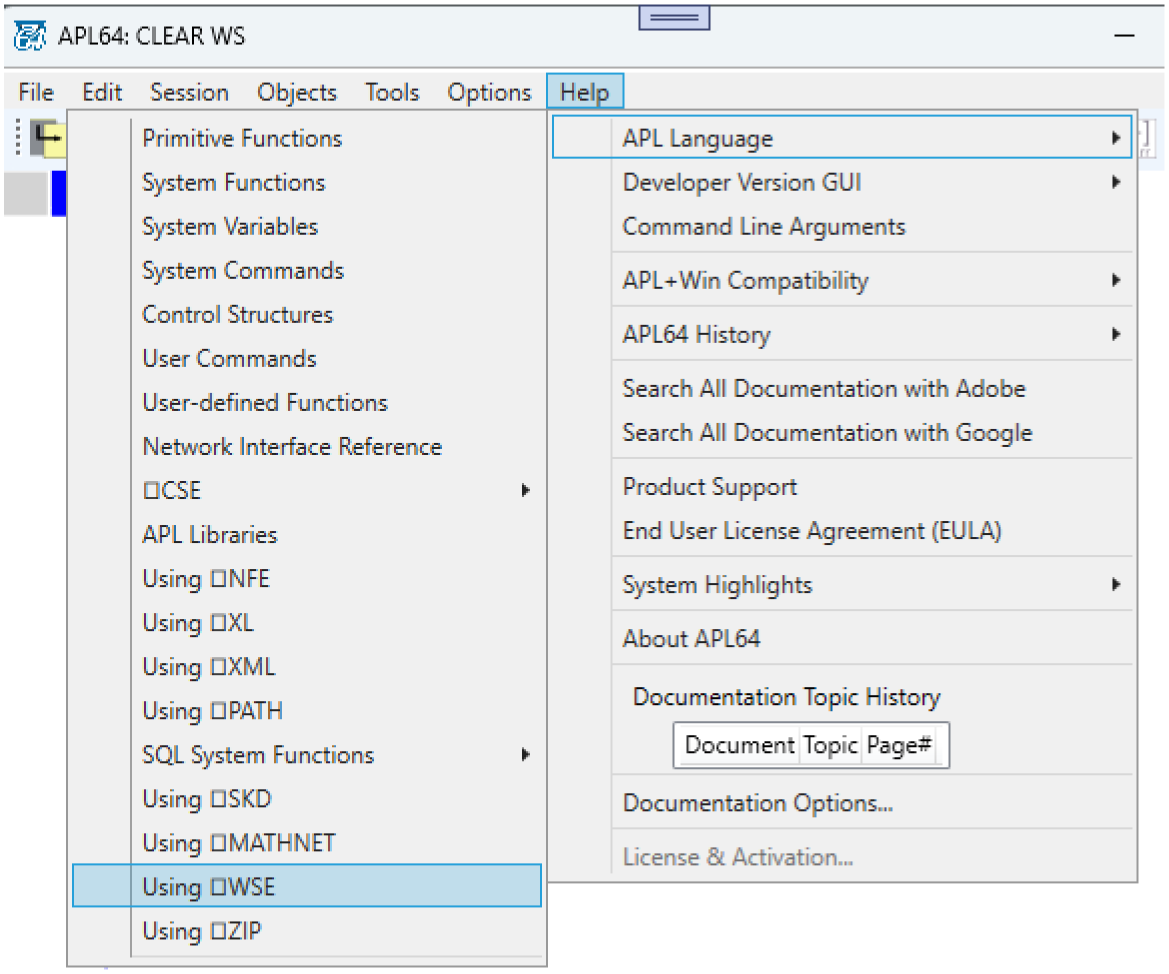
⎕WRE Save Action

Syntax: 'Save' ⎕WRE targetFileName

The Save action will save the WRE configuration data set in the current APL64 developer version instance to the file specified in the right argument.

⎕wse Workspace Engine

The ⎕WSE system function exposes the APL64 workspace engine object. To learn more about ⎕WSE use the **Help | APL Language | Using ⎕WSE** menu item:



⎕wslib Workspace Library List

Purpose:
Returns a character matrix that lists all the workspaces including the .ws64 extension in the designated library.

Syntax:
```
result ← ⎕wslib ''
result ← ⎕wslib 'dir'
result ← ⎕wslib lib
```

Arguments:
dir is the directory you want to search;
lib is the library you want to search.

The argument must be empty, a valid path, or a positive integer that has been associated with a directory by using ⎕libd or the Libraries choice on the Options menu. If the argument is empty, the system uses the current working directory.

Result:
⎕wslib lists in alphabetical order (uppercase Z before lowercase a) the workspaces in either the specified directory (dir) or library (lib), or the default directory. If the argument is a path name, the

result is a matrix of workspace names, left-justified. The number of columns is the length of the longest workspace name in the list.

If you supply a numeric library number, the result is a character matrix that contains one workspace identification in each row. The first ten columns contain the right-justified library number with a space in the eleventh column; the remaining columns contain the left-justified file name.

)wslib produces the same list of workspaces, but it lists them in multiple columns to save lines on the screen, and it lists them without library numbers.

Examples:

```
      ⎕wslib 'C:\APL64\MONTHS'
FEBRUARY.ws64
JANUARY.ws64
MARCH.ws64

      ⎕libd '12 C:\APL64\MONTHS'
      ⎕wslib 12
       12 JANUARY.ws64
       12 FEBRUARY.ws64
       12 MARCH.ws64

      ρ⎕wslib 'C:\APL64\MONTHS' ⋄ ρ⎕wslib 12
3 13
3 24
```

⎕wsowner Workspace Owner

Purpose:
Returns the user number of the user who last saved the current workspace.

Syntax: result ← ⎕wsowner

Result:
The result is an integer scalar that represents the user number (1↑⎕ai) of the user who last saved the workspace. In a clear workspace, result is 0.

Example:

```
      ⎕wsowner
0
```

⎕wssize Workspace Size

Purpose:
Returns the size of the active workspace in bytes.

Syntax: result ← ⎕wssize

Result:
The result is a numeric scalar that contains the total size of the active workspace, including the space used by APL objects, the symbol table, and unused storage (⎕wa).

Examples:

```
      ⎕wssize
3.40920279E10
      ⎕wa
3.404520402E10
```

Determine the approximate number of bytes needed to store this workspace on disk.

```
      ⎕wssize-⎕wa
44726912
```

⎕wsts Workspace Timestamp

Purpose:
Returns the time a loaded workspace was saved or the time of the most recent save or clear operation that was performed on the active workspace.

Syntax: result ← ⎕wsts

Result:
The result is a numeric scalar that contains the time of the most recent)save or)clear that was performed on the active workspace. The time code is in microseconds since 00:00 on 1 January 1900.

Examples:

```
      )load MYWS
2 MYWS SAVED 07/08/88 15:14:00

      ⎕wsts
2.76226284E15

      )copy DATES FTIMEFMT
SAVED 07/04/88 15:17:21

      FTIMEFMT ⎕wsts
07/08/88 15:14:00.000
```

⎕xfcreate Extended Create Component File

Purpose:
Creates a new component file using "extended" file name conventions.
(This is a companion function to ⎕fcreate and should supersede it.)

Syntax: 'fileid.ext' ⎕xfcreate tieno
'fileid.ext' size ⎕xfcreate tieno
'fileid.ext' size compno ⎕xfcreate tieno

Arguments:
fileid is a file identifier formed according to the rules for extended file functions (see the introduction); when size or compno is specified, fileid must be nested. Note that you must explicitly specify any file extension, which is not limited to .SF.
size, optional, specifies the file size limit in bytes; it must be less than 2*32 (4 gigabytes). The default value is the size limit of the file being duplicated; you can specify an empty vector as a place holder when you want the default value for size and you want to explicitly specify the initial component number.
compno is an integer that specifies the initial component number.
tieno is a positive integer file tie number.

Effect:
The ⎕xfcreate function attempts to create a new component file and tie it to tieno. The name of the new file is specified by fileid. The tie is shared. See the Discussion section under ⎕xfstie for more information about extended file ties.

Access:
No file access code is required for ⎕xfcreate.

Examples:

```
'George Washington.sf' ⎕xfcreate 1
(197 ,  'ngstrom.sf') ⎕xfcreate 2
'ú.oil' ⎕xfcreate 3
('G'  , 246 , 'del') 1000000 10 ⎕xfcreate 4
```

The first line attempts to create an APL component file with the name George Washington and the extension .sf.

The second line attempts to create a component file with the name Ångstrom.sf.

The third line attempts to create a file with extension .oil. The operating system may see the file with the English Pound sign (£.oil), depending on how you generate the character. (Note that in the APL session, in a GUI edit box, and in applications such as Windows Notepad, you can get different characters depending on whether you type Alt+163 or Alt+0163.) If you specify the argument as an integer, 163 ⎕xfcreate 3, Windows Explorer sees the file with the English Pound sign.

The fourth line attempts to create a file with the name of Gödel, a size limit of a million bytes, and starting with component number 10. (ANSI character 246 is a lowercase o umlaut). The lower case o umlaut is ⎕av[148]. If you specify the o umlaut in APL, the host system will do something with its character 148. In many fonts this character is the right half of paired double quote marks, but the character is not always available so the system may display an unspecified symbol.

⎕xfdup Extended Duplicate Component File

Purpose:
Creates an exact copy of a file using "extended" file name conventions and, if possible, compacts it to occupy less disk space. (This is a companion function to ⎕fdup and should supersede it.)

Syntax:
```
'fileid.ext'                  ⎕xfdup tieno
'fileid.ext' [size]           ⎕xfdup tieno
'fileid.ext' [size] compno    ⎕xfdup tieno
'fileid.ext'                  ⎕xfdup tieno [passno]
'fileid.ext' [size]           ⎕xfdup tieno [passno]
'fileid.ext' [size] compno    ⎕xfdup tieno [passno]
```

Arguments:
fileid is a file identifier formed according to the rules for extended file functions (see the introduction); when size or compno is specified, fileid must be nested. Note that you must explicitly specify any file extension, which is not limited to .SF.
size, optional, specifies the file size limit in bytes; it must be less than 2*32 (4 gigabytes). The default value is the size limit of the file being duplicated; you can specify an empty vector as a place holder when you want the default value for size and you want to explicitly specify the initial component number.

compno is an integer that specifies the initial component number.
tieno is a positive integer file tie number.
passno is an optional file passnumber.

Effect:
⎕xfdup creates a new file with the specified name (fileid) and copies all the data from the file specified by tieno into it. In the process, it recovers unused space created by replacing records with a different sized component, which may allow the new file to occupy less disk space than the original file. The old file remains unchanged.

You can use the same name as the file being duplicated for the new file. Using the same name replaces the file with the compacted version. The file must be exclusively tied to duplicate it with the same name; otherwise, a NONCE ERROR occurs.

Access:
The file you want to duplicate must be tied using one of the extended file tie functions, the passnumber must match the one in effect, and you must have both duplicate access and the authority to create files in the specified (or default) directory or library. The access code for ⎕xfdup is 16384.

Discussion:
To facilitate compatibility with older software and operating systems, some file systems that support long file names create short alias names so that every file can be referred to by an 8.3 format name. Older operating systems that cannot use long names see only the short names.

Windows generates short names using an algorithm that removes spaces and some unusual characters, truncates the name sufficiently, then appends a tilde (~) and at least one digit. It chooses digits to

ensure the short name is different from other short names in the same directory.
In a particular directory, the files in the first column below could have the short names in the second column.

foo bar number 1.txt stuff	FOOBAR~3.TXT
foo bar number 2.txt	FOOBAR~2.TXT

Note that the association between the names is tenuous. If you duplicate the file to the same name, the relationship between the long name and the short name may be severed.

⎕xferase Extended Erase Component File

Purpose:
Erases a component file tied using "extended" file name conventions.
(This is a companion function to ⎕ferase and should supersede it.)

Syntax:
```
'fileid.ext' ⎕xferase tieno
'fileid.ext' ⎕xferase tieno [passno]
```

Arguments:
fileid.ext is a file identifier formed according to the rules for extended file functions (see the introduction);
tieno is a positive integer file tie number; it must be an exclusive extended component tie;
passno is an optional file passnumber.

The left and right arguments designate the same file. The right argument must be an integer scalar or one- or two-element, integer vector that defines the file you want to erase. It contains a valid positive tie number and optional valid passnumber. If you omit passnumber, the system assumes it is 0.

Effect:
The system erases the component file tied to tieno. The interpreter processes fileid exactly as if it were tying the file. Then it checks that the resulting complete path name matches the one produced at the time tieno was tied. If the path names are different, the function fails and the file is not erased. This is a perfunctory check intended to avoid immediate execution calamities. The interpreter does not attempt to recognize different names for the same file (UNC vs drive letters, long names vs 8.3 aliases, etc.).

Access:
The file must be tied; the system cannot erase the file if any other user also has it tied. The passnumber must match the one in effect and you must have erase access. The access code for ⎕xferase is 4.

Examples:

```
    'TEXTFILE' ⎕xftie 10
    'TEXTFILE' ⎕xferase 10
```

⎕xfnames Names of Extended Tied Component Files

Purpose:
Returns the file identifiers of all component files tied with ⎕xftie or ⎕xfstie.
(This is a companion function to ⎕fnames.)

Syntax: result ← ⎕xfnames

Result:
The result is a character matrix of file identifiers. Each row of ⎕xfnames is a complete path name, including drive letter and colon or UNC prefix, exactly as passed to the host operating system to open the file at the time it was tied. The rows of result have the same order as ⎕xfnums.

Example:

```
      ⎕xfnames
C:\APLWIN\WSS\CHAPTER11
```

⎕xfnums Tie Numbers of Extended Tied Component Files

Purpose:
Displays the tie numbers of all component files tied with ⎕xftie or ⎕xfstie.
(This is a companion function to ⎕fnums.)

Syntax: result ← ⎕xfnums

Result:
The result is a numeric vector of file tie numbers. The tie numbers are in the same order as the file names that ⎕xfnames reports; that is, the order in which they were tied. Note that ⎕xfnums does not include the tie numbers assigned by the traditional component file functions, and ⎕fnums does not include tie numbers assigned by the extended component file functions. Thus, if you get a file tie error, you must check both functions to determine if the file is already tied.

Example:

```
      ⎕xfnums
28 34 18
```

⎕xfrename Extended Rename Component File

Purpose:
Changes the name of a file using "extended" file name conventions.
(This is a companion function to ⎕frename and should supersede it.)

Syntax: 'fileid.ext' ⎕xfrename tieno
            'fileid.ext' ⎕xfrename tieno [passno]

Arguments:
fileid.ext is a file identifier formed according to the rules for extended file functions (see the

introduction);
tieno is a positive integer file tie number; it must be an exclusive extended component tie;
passno is an integer passnumber.

The right argument must be an integer scalar or one- or two-element, integer vector that defines the file you want to rename. It contains a valid positive tie number and optional valid passnumber. If you omit the passnumber, the system assumes it is 0.

Effect:
⎕xfrename changes the file name to the one specified in the left argument, potentially moving it to a different directory on the same disk drive. If the file name already exists, the system signals a FILE NAME ERROR. You cannot rename a file and change its size as you could with the traditional function. You can change the size of a component file using ⎕fresize.

The result of ⎕xfnames reflects the new file identifier. The user who renames the file becomes the new file owner.

Access:
The file must be exclusively tied, the passnumber must match the one in effect, and you must have rename access. The access code for ⎕xfrename is 128.

Examples:

```
     ⎕xflib ''
D:\WORKING\ALGORITHMS\PRIMES.SF
     'PRIMES' ⎕xftie 10
     'PRIMENUMBERS.SF' ⎕xfrename 10
     ⎕xflib ''
D:\WORKING\ALGORITHMS\PRIME NUMBERS.SF
```

⎕xfstie Extended Share Tie a Component File

Purpose:
Ties a component file for shared use using "extended" file name conventions.
(This is a companion function to ⎕fstie and should supersede it.)

Syntax: 'fileid.ext' ⎕xfstie tieno
'fileid.ext' ⎕xfstie tieno [passno]

Arguments:
fileid.ext is a file identifier formed according to the rules for extended file functions (see the introduction);
tieno is a positive integer file tie number;
passno is an integer passnumber.

The right argument must be an integer scalar or one- or two-element, integer vector that specifies the number by which you will define the file to other functions. It contains a valid positive tie number and optional valid passnumber. If you omit the passnumber, the system assumes it is 0.

Discussion:
Formerly, there were two classes of file ties: component file ties and native file ties. Component file tie numbers use positive integers. Native file tie numbers use negative integers.
The extended tie creating file functions introduced a new, orthogonal classification of ties, traditional or extended, according to the function used to establish the tie. Thus, there are now four classes of file ties:

traditional component
extended component
traditional native
extended native.

This list is a classification of ties; it refers to the way a file is being used, not to the file itself. Both traditional and extended component file ties share the positive integers as tie numbers. Traditional and extended native file ties share the negative integers as tie numbers. It is highly recommended that you not use both the traditional and extended file functions for tying files at the same time.

Effect:
The file is share tied; more than one user can share-tie a file simultaneously. File ties are "slippery;" that is, if a file is already tied to one tie number, you can tie that file to the same number or to another unused tie number without untying the file.

Access:
The file must exist and must not be exclusively tied (⎕xftie or ⎕ftie) by anyone, although it can be share-tied by others. The user must have some form of access to the file, and the passnumber must match an appropriate one in the access matrix.

Examples:

```
    'PRIME NUMBERS.SF' ⎕xfstie 37
    'C:\APLW\FILES\MYFILE.XSF' ⎕xfstie 22
```

⎕xftie Extended Tie a Component File (Exclusive)

Purpose:
Ties a component file for exclusive (non-shared) use using "extended" file name conventions.
(This is a companion function to ⎕ftie and should supersede it.)

Syntax:
```
'fileid.ext' ⎕xftie tieno
'fileid.ext' ⎕xftie tieno [passno]
             ⎕xftie tieno
```

Arguments:
fileid.ext is a file identifier formed according to the rules for extended file functions (see the introduction);
tieno is an available positive integer file tie number;
passno is an integer passnumber.

The optional left argument must be a valid file identifier (or a device that the operating system recognizes as a file).

The right argument must be an integer scalar or one- or two-element, integer vector that specifies the number by which you will define the file to other functions. It contains a valid positive tie number and optional valid passnumber. If you omit the passnumber, the system assumes it is 0.

Effect:
The file is exclusively tied. No other user can tie the file as long as it remains exclusively tied. File ties are "slippery;" that is, if a file is already tied to one tie number, you can tie that file to the same number or to another unused tie number without first untying the file.

Monadic ⎕xftie takes a tie number as the right argument and returns a two-element integer vector indicating the file tie state. The value of the first element is the same as the open mode arguments to the dyadic version of ⎕ntie and indicates what the current open mode of the file is. The open mode values are sums of the following:

Access Requested	Access Granted to Other Users
0 = read access	16 = no access allowed (exclusive)
1 = write access	32 = read access allowed, write access denied
2 = read and write access	48 = write access allowed, read access denied
	64 = read and write access allowed

The values are dependent on whether the tie request was for a shared or exclusive tie and whether read or write access is granted.
The second element indicates if the tie was share tie or an exclusive tie. Share ties show a value of 0, exclusive ties show a value of 1.

Access:
The file must exist, it must not be tied by anyone else, the user must have the authority to exclusively tie the file, and the passnumber must match an appropriate one in the access matrix. The access code for ⎕xftie is 2.

Examples:

```
      'PRIME NUMBERS.SF' ⎕xftie 37
      'C:\APLW\FILES\MYFILE.XSF' ⎕xftie 2
```

⎕xl Exchange data with Microsoft Excel Worksheet

Purpose:
⎕XL provides a convenient interface between APL64 and Microsoft Excel workbook files.
To learn more about ⎕XL in the APL64 Developer version go to **Help | APL Language | Using ⎕XL**.

⎕xlib Extended Library List

Purpose:
Returns a character matrix of long filenames of all files in the directory you specify using "extended" file

name conventions. (This is a companion function to ⎕lib; there is no extended file function that corresponds to ⎕flib.)

Syntax: result ← ⎕xlib 'directory'
result ← ⎕xlib 'pattern'

Arguments:
directory is a vector, possibly heterogeneous, formed like a fileid according to the rules for extended file functions (see the introduction). It names a directory to be searched.
pattern is like fileid but may contain the wild-card characters * and ? in the final portion. The question mark represents exactly one character; it can be any valid character. The asterisk represents any number of characters, from zero to the maximum allowable length. You can specify multiple * and ? characters. Only files matching this pattern are included in the result. If you specify a period in the pattern, the period must exist in the internal representation of the file. Thus a designation of *.* will not match a file that has no extension.

Result:
The result is a matrix of the long filenames of all files in the directory you specify or that satisfy a pattern you specify. Note that the extended function preserves the case of filenames as the system records them, whereas the standard functions always use uppercase. Since case matters in sorting, ⎕xlib may return the same list of names as ⎕lib, but in a different order.

Examples:

```
    ⎕xlib '*.sf'
    ⎕xlib '\\APLW\2.0.00\test files\*.sf'
    ⎕xlib '*FOO*.xyz'
    ⎕xlib '??xyz.*'
```

⎕xload Load a Workspace, Bypass the Latent Expression

Purpose:
Replaces the active workspace by loading a designated workspace (under program control), but without executing the latent expression (⎕lx).

Syntax: ⎕xload 'wsid'

Argument:
wsid is a character scalar or vector that specifies the workspace to be loaded; the system assumes the default extension (.ws64). (You can use the file extension .w3 to load an APL+Win workspace.) If you omit the directory name or library number, the system uses the default directory.

Effect:
Loads the specified workspace, making it the new active workspace. The system changes ⎕wsid and does not execute ⎕lx.

Examples:
Executing ⎕lx separately shows that it did not execute when the system loaded the workspace.

```
    ⎕xload 'C:\APL64\APLFRAME'
"C:\APL64\APLFRAME.ws64" LAST SAVED 7/5/2021 3:38:32 PM
    ⎕LX
"WELCOME TO APLFRAME"
```

⎕xml Interface to .Net XML Objects

Purpose:
⎕XML provides a convenient interface between APL64 and .Net XML objects.
To learn more about ⎕XML, in the APL64 Developer version go to **Help | APL Language | Using ⎕XML**.

⎕xncreate Extended Create Native File

Purpose:
Creates a new native file using "extended" file name conventions and ties the file.
(This is a companion function to ⎕ncreate and should supersede it.)

Syntax: 'fileid.ext' ⎕xncreate tieno

Arguments:
fileid.ext is a file identifier formed according to the rules for extended file functions (see the introduction);
tieno is a negative integer file tie number.
The left argument must be a valid native file identifier; the right argument must be a negative integer that designates an available file tie number. You cannot have another file currently tied with this number.

Effect:
The system creates and ties a new file. Microsoft operating systems create files in openmode 66. To change this type of file open, you can use a slippery tie with another mode (such as fileid ⎕ntie tn, 18) right after you create the file.

Examples:

```
    'MEMO.TXT' ⎕xncreate ¯27
    'E:\CIRCUL\MEMO.TXT' ⎕xncreate ¯27
    'B:SCRATCH.GIG' ⎕xncreate ¯25
```

⎕xnerase Extended Erase Native File

Purpose:
Erases a native file tied using "extended" file name conventions and deletes the file name from the disk directory. (This is a companion function to ⎕nerase and should supersede it.)

Syntax: 'fileid.ext' ⎕xnerase tieno

Arguments:
fileid.ext is a file identifier formed according to the rules for extended file functions (see the

introduction);
tieno is a negative integer file tie number. The left and right arguments designate the same file.

Effect:
The system unties and erases the native file tied to tieno. The interpreter processes fileid exactly as if it were tying the file. Then it checks that the resulting complete path name matches the one produced at the time tieno was tied. If the path names are different, the function fails and the file is not erased. This is a perfunctory check intended to avoid immediate execution calamities. The interpreter does not attempt to recognize different names for the same file (UNC vs drive letters, longnames vs 8.3 aliases, etc.). If the erasure is unsuccessful, the system attempts to retie the file.

Examples:

```
      'B:MEMO.TXT' ⎕xntie ¯27
      'B:MEMO.TXT' ⎕xnerase ¯27

      'SCRATCH.GIG' ⎕xntie ¯33
      'SCRATCH.GIG' ⎕xnerase ¯33
```

⎕xnnames Names of Extended Tied Native Files

Purpose:
Returns the file identifiers of all the files currently tied with ⎕xntie.
(This is a companion function to ⎕nnames.)

Syntax: result ← ⎕xnnames

Result:
The explicit result is a character matrix that contains one file identifier per row. Each row of ⎕xnnames is a complete path name, including drive letter and colon or UNC prefix, exactly as passed to the host operating system to open the file at the time it was tied. The rows of the result have the same ordering as the result of ⎕xnnums.

⎕xnnums Tie Numbers of Extended Tied Native Files

Purpose:
Returns the file tie numbersof all the files currently tied with ⎕xntie.
(This is a companion function to ⎕nnums.)

Syntax: result ← ⎕xnnums

Result:
The explicit result is a numeric vector of file tie numbers, in the same order as the result of ⎕xnnames.

Examples:

```
      ⎕xnnums
¯27 ¯52 ¯3 ¯37 ¯4
```

```
      ⎕xnuntie ⎕xnnums
      ρ⎕xnnums
 0
```

⎕xnrename Extended Rename Native File

Purpose:
Changes the name of a native file using "extended" file name conventions.
(This is a companion function to ⎕nrename and should supersede it.)

Syntax: 'fileid.ext' ⎕xnrename tieno

Arguments:
fileid.ext is a file identifier formed according to the rules for extended file functions (see the introduction);
tieno is a negative integer file tie number

The left argument specifies the new name for the file; the right argument must be a negative integer that designates the file you want to rename.

Effect:
The system renames the file tied to the tie number in the right argument to the name specified in the left argument. If a file with the specified name already exists, the system signals FILE NAME ERROR.

⎕xnrename is an extension to the DOS RENAME command. It changes the name of a file, but cannot move the file to a different disk drive. However, it can move the file to another directory on the same disk drive.

⎕xntie Extended Tie Native File

Purpose:
Establishes a file tie for a native file using "extended" file name conventions.
(This is a companion function to ⎕ntie and should supersede it.)

Syntax:

```
'fileid.ext' ⎕xntie tieno
'fileid.ext' ⎕xntie tieno [openmode]
             ⎕xntie tieno
```

Arguments:
fileid.ext is a file identifier formed according to the rules for extended file functions (see the introduction);
tieno is a negative integer file tie number;
openmode is an optional valid integer DOS open mode.
The optional left argument must be a valid native file identifier (or a device that the operating system recognizes as a native file).

The right argument is a negative integer scalar or a two-element, integer vector, of which the first element is negative. The argument designates the file tie number and, optionally, a DOS open mode.

The open mode allows access to native files by multiple users or processes. The open mode values consist of the sum of one code that indicates the type of access you want and a second code that indicates the type of access you grant to others while you have the file tied. If another user has opened the file first, the access you want must be compatible with the access granted by the other user, and the access you grant must be compatible with the access that user requested.

If you specify open mode explicitly, the domain for this function is a value 0 through 255 inclusive; however, as of DOS 3.3, the only reasonable values are 16, 17, 18, 32, 33, 34, 48, 49, 50, 64, 65, or 66. Thus, if you want to be able to read and write to a file but allow other users read-only access while you have the file tied, you would specify 34 for open mode.

```
    'C:\DRAFT\PROPOSAL.TXT' ⎕xntie ¯1 34
```

Note that the word "reasonable" above, has limited scope. If you specify open mode 49, any user could write to the file, but no user could read it.

Access Requested	**Access granted to other users**
0 = read access	16 = no access (exclusive)
1 = write access	32 = read access
2 = read and write access	48 = write access
	64 = read and write access

If a second user attempts to tie the file with an open mode that is incompatible with the open mode specified by a first user, the system signals a HOST ACCESS ERROR or an XFHOST ERROR. The question of compatibility involves both the access requested value and the access granted value. If the first user includes 16 in open mode, then no second user can tie the file. If the first user specifies 32 through 34, then the second user can request only read access (zero); any value that includes 1 or 2 results in an error. If the first user specifies 48, 49 or 50, then the second user can request only write access (1). If the first user specifies 64 through 66, the second user can request read and/or write access.

The second user must also specify an access granted value that is compatible with the first user's requested access. If the first user has read-only access, the second user cannot include 48 in his open mode value. If the first user has write access, the second user cannot include 32. If the first user requested both read and write access, the second user must include 64. A second user cannot successfully specify exclusive access.

If you specify one of the access requested values, but do not specify an access granted value (that is, open mode of zero, 1 or 2), the effect for APL is the same as specifying exclusive access; another instance of APL cannot tie the file. If you do not specify open mode at all, the default value is 66. APL64 tries open mode 64 if an ACCESS DENIED error occurs. (Except see the historical note below.)

Note that you can also trigger an error for access reasons unconnected with another user, for example, requesting write access (including 1 or 2) on a file which the operating system considers to be read-only.

Monadic ⎕xntie takes a tie number as the right argument and returns a two element integer vector indicating the file tie state. The value is the same as the open mode arguments to the dyadic version of ⎕ntie and indicates what the current open mode of the file is. If an open mode argument was specified when the file was tied, the result of monadic ⎕xntie is that value. If no open mode argument was specified the return value is dependent on whether read or write access is granted and the value of "Network" specified in the APL configuration file. The value of the second element is applicable only to component file ties. Therefore, the second element value is meaningless for native file ties.

Historical Baggage:
In the days of PC DOS, there was an option called "Network" that you could (can) specify in the APLW.INI file. By default, when not specified, this is "on." There is no known situation where it is currently useful to specify this option, but it remains in the system to protect old code. If "Network" is off, and you do not specify openmode, the system attempts read/write access with an exclusive tie, which blocks anyone else from tying the file and fails if another has the file tied in any mode. If ⎕xntie fails in this situation, the system tries exclusive read-only.

Effect:
The system ties the native file. You can retie a file that you already have tied without first using ⎕nuntie. The tie number can be the same number or a different number. The only restrictions are that no other file can already be tied with the new tie number. You can use this "slippery" tie to ensure that a file is tied without looking up its name in ⎕xnnames.

If the needed access is not available or if the open mode is not acceptable to the operating system, an ACCESS ERROR results. A slippery tie closes the file and reopens it, possibly with a different specified open mode. Another user could conceivably open the file at that precise moment and block the retie. Caution: By manipulating directories with ⎕chdir and ⎕libd, you conceivably can tie the same file to two different tie numbers without the system detecting it. The result of this condition is unpredictable; you should avoid creating it.

Examples:

```
      'A:PRIMES.INT' ⎕xntie ¯37
      'C:\SPREAD\PRINT FN.OUT' ⎕xntie ¯1 34
```

⎕xso Excess Output

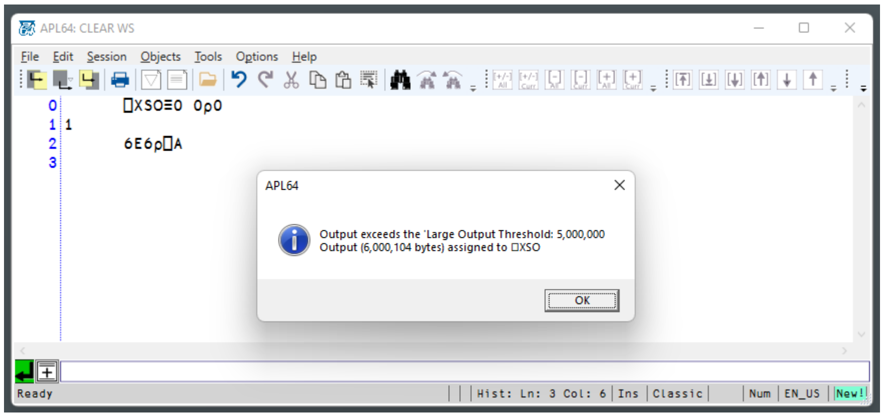
Purpose:
This workspace-related system function stores large excess output to be rendered to the history pane.

Syntax: result ← ⎕xso

Domain:
The ⎕xso system function has a default value of 0 0ρ0 in a clear workspace.

Result:
Rendering large excess output to the history pane may take considerable time. To avoid such a delay, use the ⎕xso system function to the store large output to be rendered to the history pane, prior to formatting, whenever the **Session | Rendering Large Output To History | Large Output Threshold Applies** is checked and the size of the output is greater than the programmer-specified setting in **Session | Rendering Large Output To History | Large Output Threshold**.

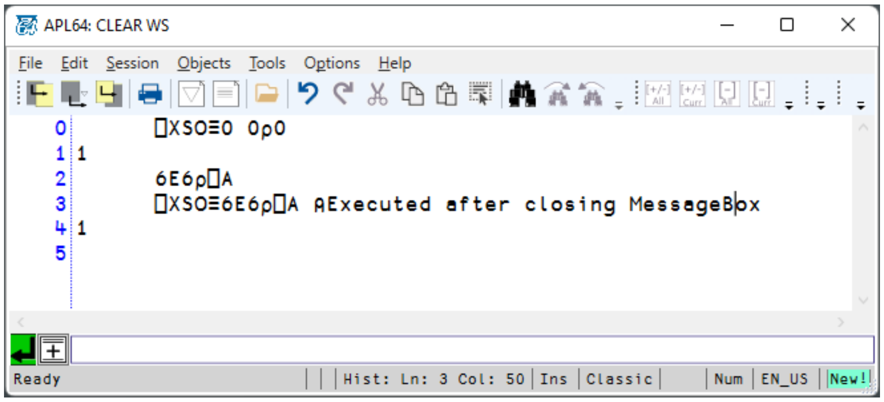
Example:



```
      ⎕XSO≡0 0⍴0
1
      6E6⍴⎕A
```

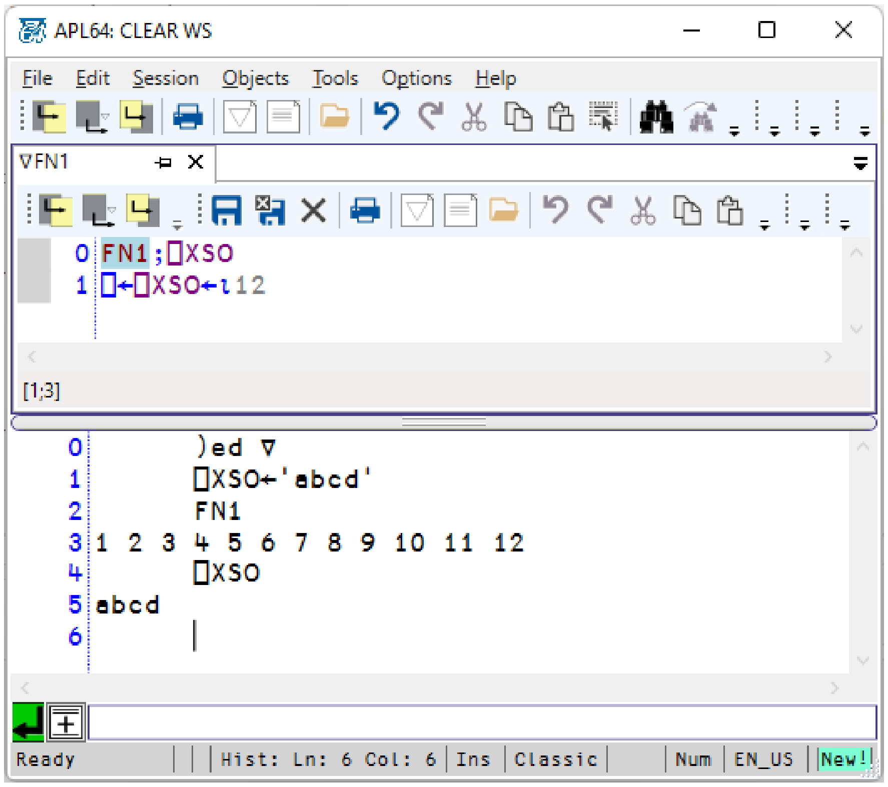
APL64

Output exceeds the 'Large Output Threshold: 5,000,000
Output (6,000,104 bytes) assigned to ⎕XSO



```
      ⎕XSO≡0 0⍴0
1
      6E6⍴⎕A
      ⎕XSO≡6E6⍴⎕A ⍝Executed after closing MessageBox
1
```

⎕XSO can be localized in an APL programmer-defined function:


APL64: CLEAR WS
File Edit Session Objects Tools Options Help
∇FN1

```
0 FN1;⎕XSO
1 ⎕←⎕XSO←⍳12
```

[1;3]

```
        )ed ∇
        ⎕XSO←'abcd'
        FN1
1 2 3 4 5 6 7 8 9 10 11 12
        ⎕XSO
abcd
```

Ready | | Hist: Ln: 6 Col: 6 | Ins | Classic | | Num | EN_US | New!

⎕zip Build and Manipulate File Archives

Purpose:

Supports creating and using .Net dictionary instances in APL64.

For detailed documentation use the **Help | APL Language | Using ⎕ZIP** menu item in the APL64 developer version.