APL64 SYSTEM VARIABLES

Basic Information

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

System variables are a special class of APL variables that manage the interaction between the APL interpreter and the active workspace. They hold information that you, your programs, or the system can always find in any workspace. To you, system variables behave like ordinary variables with some restrictions on domain and shape; to the system, they are a set of parameters controlling the interface with you. System variables are always available. You cannot erase or copy them. You can reference them, assign values to them, and localize them in functions. You cannot use the names of system variables as function names or as the names of labels, arguments, or results.

System variables are like other variables in functions with the exception of localization. When execution depends on a system variable that is localized but has no assigned value, the system assumes the value that the variable had at a previous level. This behavior is known as pass-through localization and applies to all system variables except ⎕sa.

Various primitive functions depend upon the value of one or more of these variables.

Some system variables are related to your APL session. Session-related variables change only when you explicitly reset them or when you start a new APL session. Other system variables are related to a workspace. The system stores workspace-related system variables with the workspace; therefore, their values may change after a)load or)clear.

Contents

□alx Attention latent expression

Purpose:
This workspace-related system variable contains the APL expression you want to execute in the event of an attention exception.

Syntax: value ← □alx
□alx ← 'statement'

Domain:
value is a character vector or singleton;
statement is an APL character vector or singleton containing an APL statement that replaces the current value. In a clear workspace, the default value of □alx is '□dm'.

Effect:
When an attention exception occurs during the execution of an APL statement or function, the system executes the most local value of the statement stored in □alx using the Execute primitive function (⍎). An attention exception occurs whenever execution suspends at the start of a function line because of a weak interrupt. You can usually generate a weak interrupt by pressing Pause once. The system interprets this as a request to stop execution as soon as it executes the current line.
You may be able to generate a strong interrupt by pressing Pause twice in rapid succession. The system interprets this as a request to stop execution immediately. Note that a strong interrupt does not trigger an attention exception whereas a weak interrupt does.
In addition, any APL error can occur during execution of □alx.

Examples:
In SAMPLE1, □alx protects a critical function from suspension when an interrupt is signaled by automatically restarting the current interpreter statement. Note that □lc has no element corresponding to the Execute symbol (⍎) that would show in the state indicator [see □si or)si] during the execution of the statement ⍎ □alx.

```
   ∇ SAMPLE1;□alx
[1] □alx←'→□lc'
          .
          .
          .
   ∇
```

```
   ∇ SAMPLE2;□alx
[1] □alx←'□error "ATTN"'
          .
          .
          .
   ∇
```

The function SAMPLE2 uses □alx to pass a special error exception to the calling function so that □elx can handle both errors and attentions. The calling function can then determine that the error resulted from an attention exception and take appropriate action.

□carg APLNext C# .Net custom event information

Purpose:
This system variable contains information which is useful when an APL64 function is executing as the event handler for a .Net custom event in the APLNext C# Script Engine (CSE).

Syntax: res ← □carg

Domain:
In a clear workspace or if no .Net custom event is being handled by an executing APL64 event handler function subscribed to that event, the value of ⎕carg is 0ρ0 (zilde).

When an APL64 event handler function which has been subscribed to a .Net custom event is currently executing, the value of ⎕carg is a four-element vector which is local to the scope of that APL64 event handler function:

Element	Description
[1]	Name of the instance of the associated CSE object.
[2]	Name of the C# object which fired the subscribed event.
[3]	Name of the C# event fired by the C# object.
[4]	APL64 object containing the custom event args values.

Note: Future versions of the CSE may increase the number of elements in ⎕carg.

The event args values are defined by the .Net delegate specification for a .Net custom event using .Net data types which have direct representations as APL64 data types.

An APL64 function can be subscribed to a .Net custom event using the CSE AddCustomEventHandler or AddCustomEventHandlerEx method.

If an APL64 event handler function is subscribed to a .Net routed event using the CSE AddEventHandleror AddEventHandlerEx methods, when that APL64 event handler function is executing, the value of ⎕carg is zilde because routed events provide for 'event arguments' which may not have a direct representation as APL64 data types.

See the APL64 C# Script Engine Manual for more information.

⎕cself Current APLNext C# Script Engine Object

Purpose:
This session-related system variable contains the fully qualified name of the current APLNext C# Script Engine (CSE) object.

Syntax: obj ← ⎕cself
⎕cself ← 'newobj'

Domain:
obj is the name of the current CSE object. The default value when no callbacks are pending is an empty character vector unless you have assigned it in immediate execution when no callbacks were pending. You can assign a character vector with the name of the CSE object you want to become current.

See the APL64 C# Script Engine Manual for more information.

Effect:
When processing a callback, APL sets ⎕cself to the fully qualified name of the current object. You can use this in an event handler to identify the object that the event applies to. This also allows you to write

a handler that will work with any number of different objects, since any actions you take using monadic □cse apply to the current object.

You can change the value of □cself during a callback or in immediate execution mode. If you assign □cself to the name of an object, you can refer to the object implicitly by using □cse without a left argument.

Delocalization of □cself to a non-existent name sets □cself to an empty vector.

Example:
In the following example, a CSE object named C is assigned to □cself. Then □cse is invoked without having to specify the left argument.

```
    C ← 'C' □cse 'Init' 'System'
    '' ≡ □cself
1
    □cself ← C
    □cse 'ExecStmt' 'string s = "abcd";'
0
    □cse 'GetValue' 's'
abcd
```

□ct Comparison tolerance

Purpose:
This workspace-related system variable specifies the maximum relative difference allowed between two numbers for them to be considered equal.

Syntax: value ← □ct
□ct ← value

Domain:
value is any single numeric value such that $0 \leq \square CT \leq 1E^{-}10$. In a clear workspace, the default value is $1E^{-}13$. When referenced, □ct is always a numeric scalar.

Effect:
Because inexact internal representation of non-integer values and cumulative rounding errors are inherent in computer arithmetic, two numbers that should theoretically be equal may be slightly different. Comparison tolerance ameliorates (or disguises) some of these problems by establishing a threshold for ignoring small differences between two numbers that may be deemed inconsequential for practical purposes.

All comparisons with the number 0 are exact and are independent of □ct. Otherwise, the system considers two numbers equal if their relative difference is less than or equal to □ct. This means that A and B are considered equal if: (|A-B)≤□ct×(|A)⌈|B. Other comparisons are derived from that property. The system uses □ct when computing the result of any of these primitive functions using floating-point data:

floor (⌊)	index of (⍳)	numeric relations (<≤=≥>)
ceiling (⌈)	member of (∈)	match (≡)
residue (∣)	find (⍷)	without (~)

Examples:

```
      ⎕ct
1.0E¯13
      3=3+.000000000001
0
      ⎕ct←.00000000001
      3=3+.000000000001
1
```

Caution:
If ⎕ct is 0, all comparisons are exact. Setting ⎕ct to 0 may produce surprising results from floating-point calculations on real numbers due to the way numbers are represented and the effects of rounding. If the results of your computations depend on the value of ⎕ct, you should reconsider your analysis.

Examples:

```
      ⎕ct
1.0E¯13
      a←4.3357
      b←13.63
      b=a+b-a
1
      ⎕ct←0
      b=a+b-a
0
```

⎕devcap Developer version MainWindow caption

Purpose:
This session-related system variable contains the text appearing in the title bar to the left of CLEAR WS or the workspace path for your APL64 session.

Syntax: caption ← ⎕devcap
⎕devcap ← caption

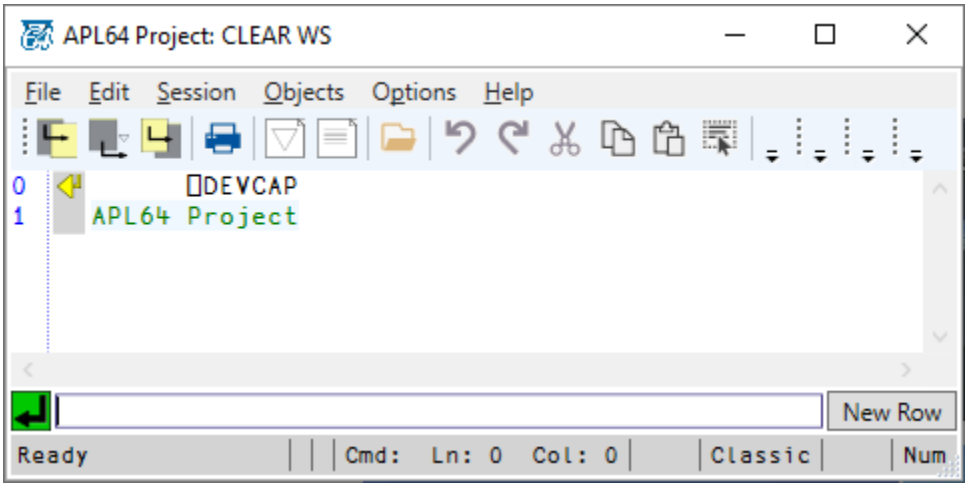
Domain:
caption: Read/Write character vector

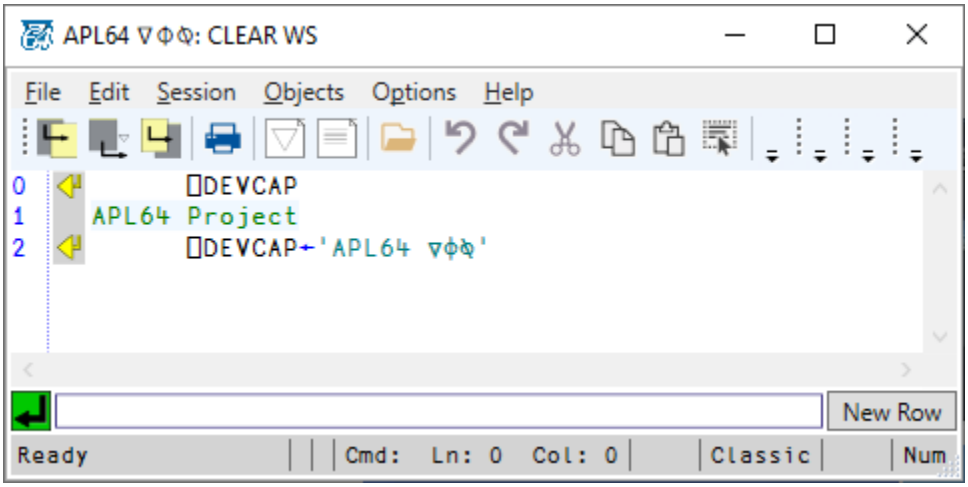
The value of the ⎕DEVCAP system variable:

- Applies only to the APL64 Developer version
- Is an APL64 character vector or scalar
- Has default value 'APL64 Project'
- Is not saved in the APL64 xml-format configuration file

- Is a component of the main window and subordinate window titles in the APL64 Developer GUI
- Is a component of the APL64 Developer version tab in the Windows system tray
- Is a component of the log file path used by the ⎕LOG system function

Examples:





⎕dm Diagnostic Message

Purpose:
Returns the last diagnostic message recorded in the workspace. The system produces a diagnostic message for any event that halts execution, such as an APL error or a user interrupt.

Syntax: result ← ⎕dm
⎕dm ← 'statement'

Arguments:
statement is a character vector or scalar value that contains an APL expression. If the value being set contains any embedded ⎕tcnul, it is truncated at the first ⎕tcnul.

Result:
The result is a character vector that contains the diagnostic message associated with the last error or interrupt that occurred. If an error occurs, the message consists of three parts: the error type, the code that caused the error, and one or more carets pointing to what may be the error or errors. ⎕tcnl characters separate the three parts. If an interrupt occurs, the message contains the line number of the function that the system was executing when the interrupt occurred.

Effect:
Displays the diagnostic message associated with the last weak interrupt, strong interrupt, or trapped error that occurred in the workspace. Except for INTERRUPT, ⎕dm does not reflect the diagnostic message displayed after an untrapped error or attention. For more information on exceptions, see ⎕alx, ⎕elx, and ⎕error in this chapter and the Exception Handling chapter in this manual.

The system saves the diagnostic message reported by ⎕dm when you save the workspace. If there is not enough workspace storage available when an error or attention occurs, the system displays NO SPACE FOR ⎕DM followed by the diagnostic message. ⎕dm is empty after this type of error.

⎕dm can be localized to preserve its value upon exit from a function. This allows the state of ⎕dm to remain unchanged upon exit from a function that may have produced errors and changed the state of ⎕dm during its execution.

Caution:
System-produced diagnostic messages may be altered or extended in the future. Therefore, you should design applications that analyze the result of ⎕dm to allow easy modification. One such technique is to use the same function for analyzing the diagnostic message throughout an application.

Examples:
⎕dm is empty in a clear workspace.

```
    )CLEAR
CLEAR WS
    ⍴⎕dm
0
```

If you generate an APL error, the system displays the normal diagnostic message since ⎕elx←'⎕dm'.

```
    3+A
VALUE ERROR
    3+A
      ^
```

⎕dm now contains the diagnostic message associated with the last error exception. It is a character vector that includes the newline characters.

```
    ⍴⎕dm ⋄ ⎕dm
32
VALUE ERROR
    3+A
      ^
```

If you save the workspace, the system saves ⎕dm with the workspace. You can see this if you clear the workspace, observe that ⎕dm is empty, and then reload the previous workspace.

```
    )SAVE TEMP
 TEMP SAVED . . .
    )CLEAR
CLEAR WS
    ρ⎕dm
0

    )LOAD TEMP
 TEMP SAVED . . .
    ⎕dm
VALUE ERROR
    3+A
      ^
```

If you set ⎕elx to do nothing, the system displays no error message on obvious APL errors. However, the last error message is in ⎕dm.

```
    ⎕elx←''
    5÷0
    'A' + 1
    ⎕dm
DOMAIN ERROR
    'A' + 1
      ^
```

If you use the system function ⎕error, you can set the value of ⎕dm.
⎕elx ← '⎕error "OOPS - Try Again." '
The same statement causes an error, but the system displays your message

```
    'A' + 1
OOPS - Try Again.
    ⎕dm
OOPS - Try Again.
```

⎕elx Error latent expression

Purpose:
This workspace-related system variable contains the APL expression you want the system to execute in the event of an error exception.

Syntax: statement ← ⎕elx
⎕elx ← 'statement'

Domain:
Character vector or singleton that contains an APL expression. The default value of ⎕elx is '⎕dm' in a clear workspace.

Effect:
Whenever an exception occurs during execution of an APL expression or function, the system executes the statement that is stored in the most local value of ⎕elx. Thus, if ⎕elx has its default value ('⎕dm') when an error occurs, the system displays the diagnostic message (see the description of ⎕dm) and returns to immediate execution mode. If you want the system to take an alternate action in the event of an error, you can define that action as an APL statement and assign it to ⎕elx. See the Exception Handling chapter in this manual for more information.

An exception is a trapped error or an attention. The system traps any error that results when an APL statement is ill-formed or when the environmental conditions necessary to complete it are not met. The system also traps any error exceptions signaled by the system function ⎕error.

Errors that are not trapped (and therefore do not trigger ⎕elx) include:
- input errors, including errors in expressions evaluated for quad (⎕) input
- errors that result from system commands
- errors signaled by an ill-formed statement in ⎕elx
- system errors (internal errors in the APL64 itself).

If an error occurs during execution of the actual statement in ⎕elx, the system displays the diagnostic message and returns to immediate execution input. If, however, the error handler calls a function, errors signaled within that function trigger execution of ⎕elx.

If an error occurs while the system is evaluating quad (⎕) input, the system displays the diagnostic message associated with the error and prompts again for input; the system does not change ⎕dm nor execute ⎕elx. If you call a function in quad input, errors occurring within the called function do trigger execution of ⎕elx.

Examples:
In the function SAMPLE1, ⎕elx branches to the error-processing part of the function if an error occurs; the function performs different actions depending on what type of error occurred by using a selection control structure.

```
   ∇ SAMPLE1;⎕elx
[1]   ⎕elx←' →ERR '
.
.
.
[m]   ERR:  type← 6↑⎕dm
[m+1]   :select type
[m+2]   :case 'DOMAIN'
[ ]   . . .
[m+m]   :case 'INDEX '
[ ]   . . .
[m+n]   :case 'LENGTH'
[ ]   . . .
[p]     :case 'RANK E'
[ ]   . . .
[p+m]   :case 'SYNTAX'
[ ]   . . .
```

```
[p+n]  :else
[ ]  ...
[ ]    :endselect
.
.
.
  ∇
```

In the function SAMPLE2, ⎕elx invokes an error in the function that called it rather than having the error occur in the called function.

```
   ∇ SAMPLE2;⎕elx
[1]  ⎕elx←'⎕error((⎕dm⍳⎕tcnl)-⎕io)↑⎕dm'
.
.
.
   ∇
```

⎕io Index origin

Purpose:
This workspace-related system variable sets or retrieves the value of the index origin. Changing the index origin allows you to start counting with zero instead of 1.

Syntax: value ← ⎕io
⎕io ← value

Domain:
value can be either 0 or 1. In a clear workspace, the default value for ⎕io is 1.

Effect:
When generating or referencing index values, the system assumes that indices are numbered starting at ⎕io. The value of ⎕io is used to:

- compute the result of Index generator (monadic ⍳) and Index of (dyadic ⍳)
- compute the result of Roll (monadic ?) and Deal (dyadic ?)
- compute the result of Grade up (⍋) and Grade down (⍒)
- index the elements of an array (A[...])
- specify an axis to apply to a primitive function or operator (⍤[...]A)
- interpret the left argument to Dyadic Transpose (...⍉A)
- interpret the left argument to Pick (...⊃A)
- compute the result of ⎕def and ⎕fx when an invalid argument is used.

Example:
These examples show the effect of ⎕io on various operations.

```
      ⎕io←1                    ⎕io←0
      ⍳5                       ⍳5
1 2 3 4 5                0 1 2 3 4
```

```
      X←5+⍳5 ⋄ X
6 7 8 9 10

      X[3]
8

      X[5]
10

      X[0]
INDEX ERROR
      X[0]
      ^^

      1 2 3 4 [3]
3
```

```
      X←5+⍳5 ⋄ X
5 6 7 8 9

      X[3]
8

      X[5]
INDEX ERROR
      X[5]
      ^^

      X[0]
5

      1 2 3 4 [3]
4
```

```
      ⎕io←1
      'ABCDEF'[2+⍳3]
CDE

      V←6 23 11 4 ¯6
      ⍋V
5 4 1 3 2

X,[0.5] V
 6  7  8  9 10
 6 23 11  4 ¯6

      3 ? 3
3 1 2
```

```
      ⎕io←0
      'ABCDEF'[2+⍳3]
CDE

      V←6 23 11 4 ¯6
      ⍋V
4 3 0 2 1

    X,[0.5] V
5  6
6 23
7 11
8  4
9 ¯6

      3 ? 3
2 0 1
```

⎕lx Latent expression

Purpose:

This workspace-related system variable stores an APL expression that you want the system to execute after you load the workspace. This provides a convenient way to start an application automatically once it has been loaded.

Syntax: expr ← ⎕lx
⎕lx ← 'expr'

Domain:

expr is a character vector that contains a valid APL expression. In a clear workspace, the default value for ⎕lx is an empty vector ('').

Effect:
Stores a statement that the system executes whenever it loads the workspace [except by ⎕xload or)xload]. If, at load time, ⎕lx represents an invalid APL statement, the system reports an error and suspends execution as if the statement were a line you entered in immediate execution mode.

Example:
The following example illustrates a typical latent expression. When you load the workspace, the system executes Autostart, which displays the main form.

```
   ∇ Autostart; Main
[1]   Main←'fmMain' ⎕wi 'New' fmMain_def
[2]   z←Main ⎕wi 'Wait'
   ∇

     ⎕lx←'Autostart'
     )save STARTWS
     )load STARTWS
C:\INIT\STARTWS SAVED...
{the main form is displayed}
```

⎕narg Network argument

Purpose:
This session-related system variable contains the arguments associated with a network socket event.

Syntax: res ← ⎕narg
⎕narg ← argument

Domain:
The descriptions of each socket event in this manual specify the domain for this variable. When no callbacks are pending, ⎕narg is an empty numeric vector, unless you have assigned it in immediate execution when no callbacks were pending.

Effect:
When a callback is pending, the system sets ⎕narg to the appropriate value for the event at the time that the callback is invoked. These arguments are implicitly localized during a callback and restored to their prior values when the callback finishes.

You can change the value of ⎕narg during a callback or in immediate execution mode.

⎕nevent Network socket event

Purpose:
This session-related system variable contains the name of the current socket event during a callback.

Syntax: event ← ⎕nevent
⎕nevent ← 'event'

Domain:
event is a character vector with the event name. The default value when no callbacks are pending is an empty character vector unless you have assigned it in immediate execution when no callbacks were pending.

Effect:
When a sockets callback occurs on an object, ⎕nevent contains the name of the event for the system to handle. For example, if you have defined an onAcceptNotify event-handler expression, during the callback, ⎕nevent has the value AcceptNotify. This variable is analogous to ⎕wevent for the Windows interface.

⎕newline OS-dependent new line characters

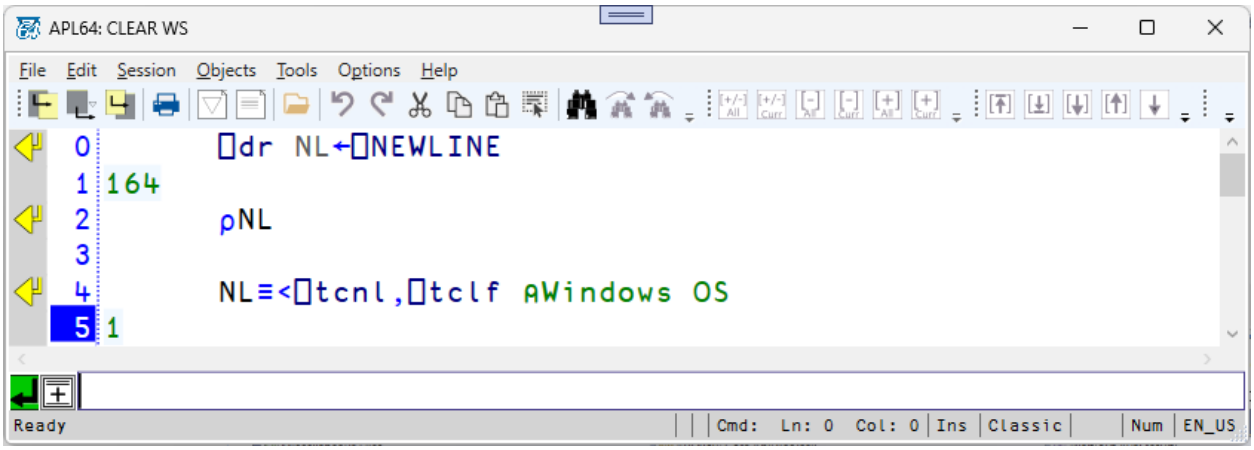
Purpose:
Returns operating system dependent new line characters.

Syntax: result ← ⎕newline

Results:
result is a string which can be used to create a string which will be rendered as multi-line text.

Remarks:
Refer to the ⎕setnewlines system function for additional information.

Examples:



⎕nself Current network socket

Purpose:
This session-related system variable contains the number of the current network socket.

Syntax: soc ← ⎕nself
⎕nself ← newsoc

Domain:
soc is the number of the current network socket. The default value when no callbacks are pending is an empty numeric vector unless you have assigned it in immediate execution when no callbacks were

pending. You can assign a numeric singleton with the number of the socket you want to become current.

Effect:
When processing a callback, APL sets ⎕nself to the number of the current socket. You can use this in an event handler to identify the socket that the event applies to. This also allows you to write a handler that will work with any number of different sockets, since any actions you take using monadic ⎕ni apply to the current socket.

⎕pathcase Case-sensitivity of file paths

Purpose:
This system variable is applicable only when the APL64 programmer's application will access a case sensitive file folder.

Syntax: value ← ⎕pathcase
⎕pathcase ← value

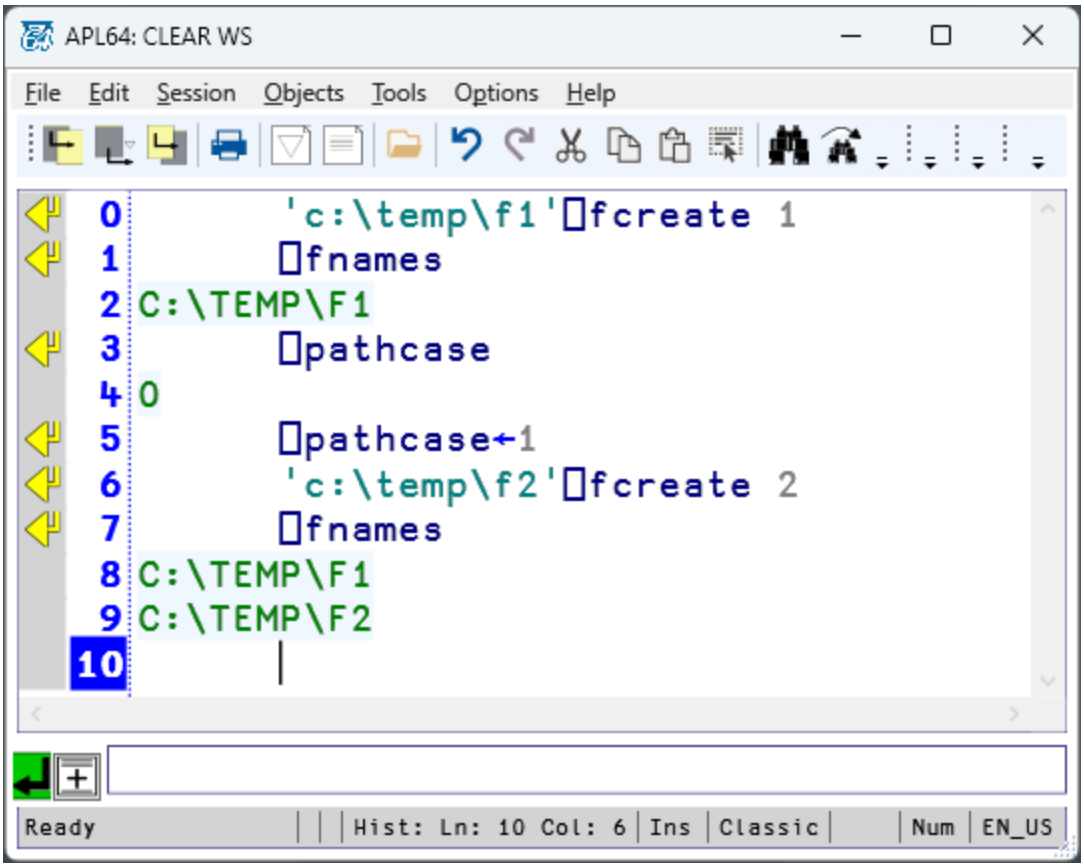
Domain:
The ⎕pathcase value can be either 0 or 1. The default value is zero. The value is saved in the APL64 xml-format configuration file. Also, can be set via the PATHCASE APL64 command line argument when the APL64-based application starts.

Effect:

For a few purposes legacy APL+Win modified user-input paths by trimming leading/following spaces and converting to upper case. When the application did not access case sensitive paths, this 'feature' may have been deemed a 'convenience'.

If the value of ⎕pathcase is zero, APL64 operates the same as legacy APL+Win.

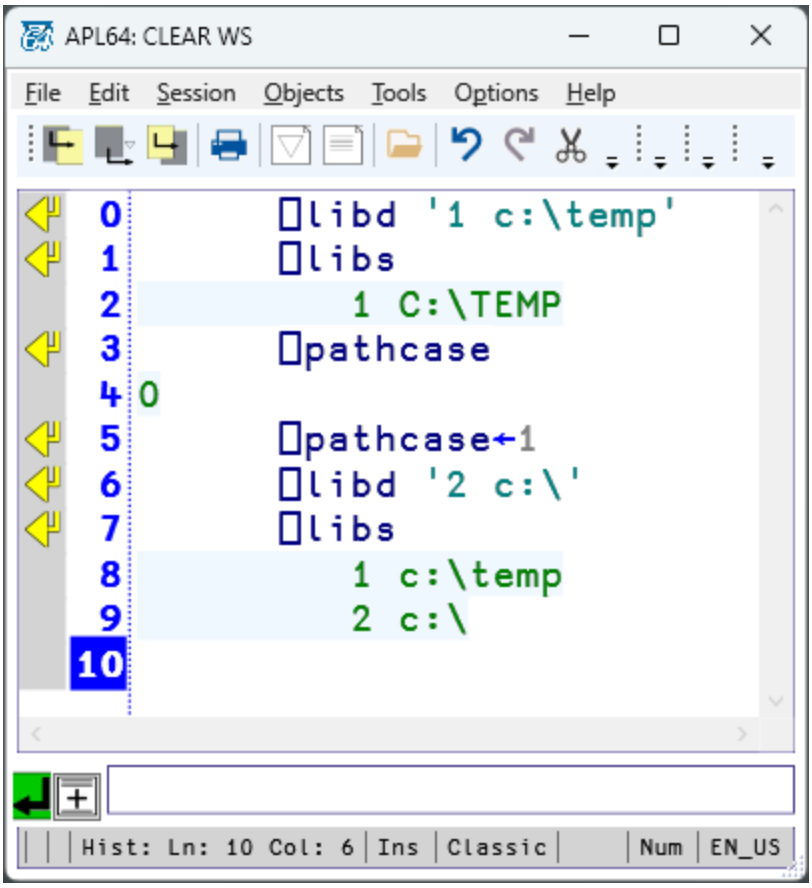
Example ⎕fnames:



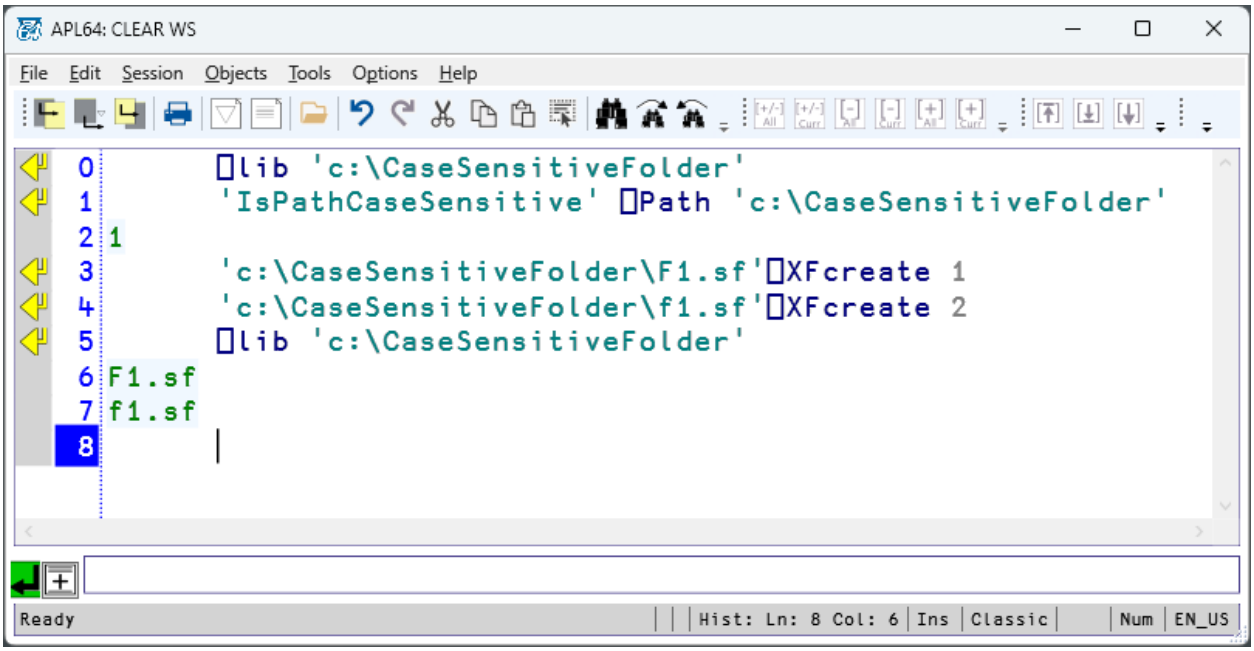
Although a case sensitive path in Windows is rare, in a cross-platform environment, case sensitive paths often occur. To provide cross-platform support in APL64, the APL64 ⎕pathcase system variable is available. When an APL64-based application is used in a case sensitive path environment, the value of ⎕pathcase should be 1.

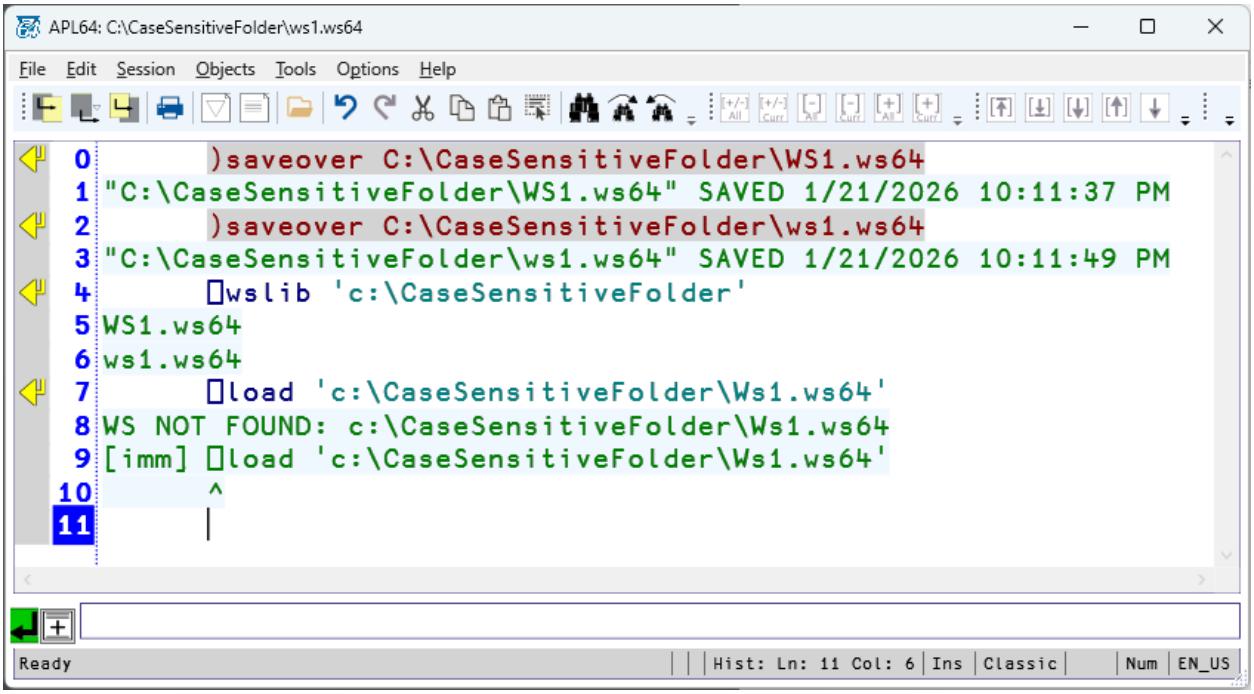
In APL64 programmer-provided paths are accepted and stored without modification, and this behavior is not affected by the value of ⎕pathcase. The value of ⎕pathcase does affect the use of such paths in APL64.

Example ⎕libs:



Example: Accessing a case-sensitive folder:





Example: Create a case-sensitive file folder, c:\Test1\Test2, in Windows:

- Start Windows 11 on the workstation
- Open the Windows PowerShell dialog as Administrator and run the command:
  Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Windows-Subsystem-Linux
- Restart Windows
- Open the Windows PowerShell dialog as Administrator and run the command:
  fsutil.exe file setCaseSensitiveInfo "c:\Test1\Test2" enable

⎕pp Print precision

Purpose:
This workspace-related system variable specifies the maximum number of significant digits that the system uses when it displays numeric data.

Syntax: result ← ⎕pp
⎕pp ← number

Domain:
You can assign ⎕pp an integer value from 1 to 17 inclusive. The default value is 10 in a clear workspace.

Effect:
The system uses the value of ⎕pp to represent the result of monadic format (o) or any system-displayed numbers. The system uses up to ⎕pp significant digits to represent the numbers. If the system cannot exactly represent a value with ⎕pp digits, it rounds the display to ⎕pp digits. This rounding affects only the display of the numbers; subsequent computation is unaffected.

Note: ⎕pp←17 permits the system to display full internal precision, with every internal floating-point value distinguishable from its nearest neighbors. The final digit may not be otherwise significant.

Examples:

```
    ⎕pp
10
    ÷3
0.3333333333
    2÷3
0.6666666667
    ÷8
0.125
    ÷64
0.015625

    ⎕pp←3
    ÷64
0.0156
```

⎕pr Prompt replacement

Note: ⎕pr has been deprecated in APL64.

⎕pw Print width

Purpose:
This session-related system variable sets the maximum number of character positions or columns available for output to the history pane.

Syntax: result ← ⎕pw
⎕pw ← number

Domain:
You can assign ⎕pw an integer value from 30 to 2147483647, inclusive. The default value at the start of an APL session is 200. Two special values are also possible: 0 and ¯1. Specifying ¯1 sets ⎕pw to the current session history pane width. The value 0 is a synonym for the maximum width 2147483647. So, if you set ⎕pw to 0 it is setting it to 2147483647 but when this value is set, it is returned as value 0. However, internally it is set to 2147483647. Enable the menu option **Session | Set ⎕PW Based on Window Width** to set ⎕pw to match the width of the current history pane.

Effect:
The system uses no more than the first ⎕pw print positions on each line during output. Output that would extend beyond this number of positions is "folded" onto subsequent lines that are indented six spaces. The display of numeric data is folded between numbers.

The system "slices," rather than folds, the display of nested and heterogeneous arrays, producing a more readable display. This means that the system prints the first row of homogeneous data completely, using multiple lines if necessary, before it begins printing the second row. However, when

printing heterogeneous and nested data, the system prints as many columns as it can print, starting from the left, of all the rows of data (a vertical slice of the output). It then continues below the last row of output with more columns of data.

Examples:

```
      ⎕pw
200
      ⎕pw←30
      60⍴'⎕'
⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕
      ⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕
      ⎕⎕⎕⎕⎕⎕

      2 52⍴'ABCDEFGHIJ'
ABCDEFGHIJABCDEFGHIJABCDEFGHIJ
      ABCDEFGHIJABCDEFGHIJABCDEFGHIJAB
CDEFGHIJABCDEFGHIJABCDEFGHIJABCDEFGHIJAB
      CDEFGHIJABCDEFGHIJABCD

(⊂2 9⍴'AB'),(⊂2 8⍴'CDE'),(⊂2 7⍴'FGHI'),(⊂2 6⍴'J')
 ABABABABA  CDECDECD  FGHIFGH
 BABABABAB  ECDECDEC  IFGHIFG

      JJJJJJ
      JJJJJJ
```

⎕qqo Quote-quad output

Purpose:
This workspace-related system variable specifies when quote-quad output will be rendered to the history pane.

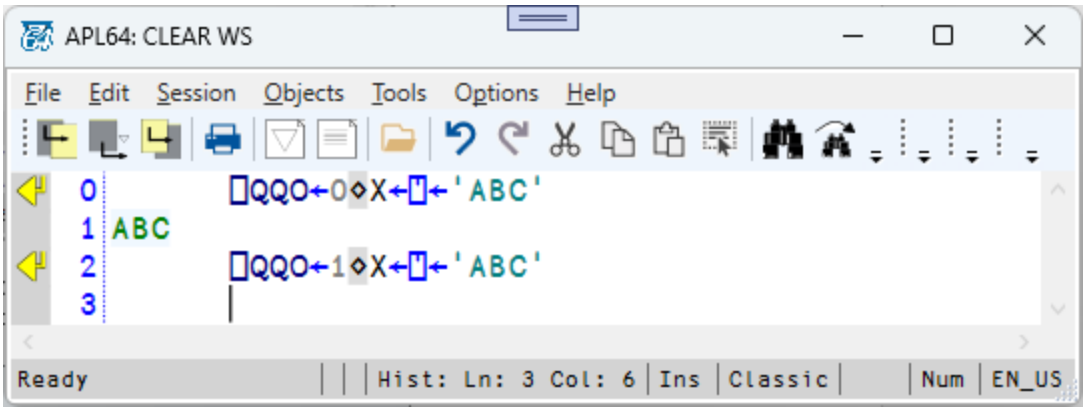
Syntax: result ← ⎕qqo
⎕qqo ← value

Domain:
You can assign ⎕qqo an integer value of 0 or 1. The default value at the start of an APL64 session is 0, to suppress quote-quad output to the history pane. Specifying 1 will render quote-quad output the history pane.

Note: Refer to the menu Help | Developer Version GUI | Quad and Quote-Quad for additional information on this variable.

Effect:
The APL+Win behavior of quote-quad output is to render the text to the history pane, even if quote-quad output is assigned to an APL variable. The APL+Win behavior of quote-quad output may be overridden by the programmer using the ⎕qqo system variable or the **Session | Suppress Unassigned History Output from Quote-Quad Output** menu option.

Examples:



⎕rl Random link

Purpose:
This workspace-related system variable sets or gets information about the type of the pseudo-random number generation algorithm used by the system, the current state of that pseudo-random number generator, and the optional discard value used by the pseudo-random number generator in advancing its internal state.

Syntax: result ← ⎕rl
⎕rl ← an integer scalar or a vector of two or three elements

Domain:
The result and the range of permissible values of ⎕rl depend on the pseudo-random number generation algorithm in use by the system.

The default value of ⎕rl in a clear workspace is scalar 7*5. This indicates that the historical pseudo-random number generation algorithm, Multiplicative Linear Congruential with multiplier 16807 (7*5) will be used. Setting ⎕rl to any scalar positive integer will cause the system to use the historical algorithm. Setting ⎕rl to a scalar positive integer will modify the internal state of the historical algorithm. This definition of ⎕rl preserves the historical syntax and features of ⎕rl.

Several additional pseudo-random number generation algorithms may be used by the system in which case the result and the range of permissible values of ⎕rl must be a two or three element (possibly nested) vector. Additional pseudo-random number algorithms may be implemented in a future version of APL64.

Pseudo-random Number Algorithm	**Element #1**	**Element #2**	**Element #3**
Historical Multiplicative Linear Congruential Multiplier 16807	positive scalar integer indicating the internal state of the historical algorithm	n/a in this case ⎕RL is a scalar	n/a in this case ⎕RL s a scalar
Extended Historical Multiplicative	zero	Internal state	optional positive integer discard value

Linear Congruential Multiplier 16807			
Mersenne Twister 19937 (32-bit)	¯1	nternal state	optional positive integer discard value
Subtract with Carry (24-bit)	¯2	nternal state	optional positive integer discard value
Multiplicative Linear Congruential Multiplier 48271	¯3	nternal state	optional positive integer discard value

For pseudo-random number generation algorithms other than the non-extended historical algorithm, the value of ⎕rl must be a two or three element vector. The first element is a non-positive integer which indicates the pseudo-random number generation algorithm to be used by the system.
The second element of ⎕rl is used to set or restore the internal state of the pseudo-random number generation algorithm. If the second element of ⎕rl is ⍬ the default internal state will be used by the algorithm. The second element of ⎕rl can be a scalar or homogeneous vector, containing integer, floating point, character, Boolean, or a UCS character vector containing the representation of the internal state along with other information pertaining to the pseudo-random number generator. For the extended historical algorithm, the internal state is represented by a scalar.

The default internal state is described in the following table.

Algorithm	**Default Internal State**
Historical Multiplicative Linear Congruential Multiplier 16807	16807
Extended Historical Multiplicative Linear Congruential Multiplier 16807	16807
Mersenne Twister 19937 (32-bit)	obtained by capturing the second element of ⎕rl ¯1 ⍬
Subtract with Carry (24-bit)	obtained by capturing the second element of ⎕rl ¯1 ⍬
Multiplicative Linear Congruential Multiplier 48271	48271

In some cases, the internal state of the algorithm is returned in the second element of ⎕rl as a vector of Unicode characters because the information necessary to represent the native-C++ internal state of the algorithm cannot be represented in the form of an APL64 data type. In this case, the first four elements of the vector comprise an MD5 checksum. The fifth element of the vector contains the algorithm type code. The sixth element of the vector is the discard value. The seventh element of the vector contains the size of the algorithm state in bytes. The eighth and subsequent elements comprise the native-C++

internal state of the algorithm. The checksum is based on the algorithm type, the discard value, the state size and the state data and is used to validate the data. This UCS-format internal state can be captured by the APL64 programmer and subsequently used to set the internal state of the algorithm.

The third element of □rl is an optional non-negative integer scalar in the range [0, ¯1+2*32], which is the specified number of pseudo-random numbers to be discarded when the selected algorithm is first used after it has been selected. The default value of the discard value is zero of it is not otherwise specified. For certain pseudo-random number generation algorithms, the discard value is useful to eliminate the initial values generated which, by design, may not be reasonably distributed. The maximum allowable optional discard value can be specified by adding the MaxDiscard entry in the [RNG] section of the APLW.INI file, which by default, this maximum value is set to ¯1+2*32.

Effect:
The system uses the value of □rl to select the type of pseudo-random number generator, set the internal state of that generator, and optionally discard initial generated values for computing the result of the APL64 roll (monadic ?) and deal (dyadic ?) functions.
The value of □rl is preserved when an APL64 workspace is saved.
As the system generates pseudo-random numbers, the internal state of the generation algorithm is updated in □rl. The captured value of □rl may be used to set the value of □rl in order to reproduce the same sequence of pseudo-random numbers.

Examples:

```
    )clear
CLEAR WS
```
By default the system uses Algorithm 0.
```
    □rl
16807
```
Generate three random numbers from 1 to 100:
```
    ?3⍴100
14 76 46
    □rl
984943658
```
When the historical algorithm is selected, □rl[□io] may only be set to a non-negative integer in the range [0, ¯2+2*32].
```
    □rl
16807
```
Select Algorithm 0 and specify the use of the default state:
```
    □rl← 0 ⍬
```
Generate three random numbers from 1 to 100:
```
    ?3⍴100
14 76 46
```
Select Algorithm 0, set the use of the default state and specify an initial discard value:
```
    □rl← 0 ⍬ 3
    state← □rl
    state
0 <<<UCS Characters>>> 3
```

Note: In the example above, if the discard value is 0 instead of 3, the internal state returned would have been an integer scalar. To preserve the discard value, the ⎕rl returns a vector.
Generate three random numbers from 1 to 100:

```
    ?3⍴100
54 22 5
```

Note: In the example above, the numbers were generated after the internal state of the generator has been advanced three times.
Select the Mersenne Twister generator, specify the use of the default state and no optional discard value and capture the value of ⎕rl:

```
    ⎕rl← ¯1 ⍬
    state ← ⎕rl
    state
¯1 <<<UCS Characters>>> 0
```

Note: In the example above the optional discard value has been set to zero and ⎕rl returns a three-element vector.
The UCS character vector above contains:

```
    elm2← ⊃state[2]
    data← ⎕ucs elm2
    data[⍳8]
    ¯405587845 ¯1425600422 ¯143747245 1727617676 ¯1 0 5000 624
```

Note: The first four elements of the data vector contains the checksum; the fifth element contains the algorithm type (¯1); the sixth element contains the discard value (0); the seventh element contains the state size (5000); the eighth element (624) marks the beginning of the state data.
Generate three random numbers from 1 to 100:

```
    ?3⍴100
82 14 91
```

Reset the state of the Mersenne Twister generator:

```
    ⎕rl← state
```

Generate three random numbers from 1 to 100:

```
    ?3⍴100
82 14 91
```

Select Algorithm 0 and specify the initial internal state:

```
    ⎕rl← 0 16807
    ⎕rl
16807
```

Select Algorithm 0 and specify the initial internal state:

```
    ⎕rl← 0 'a'
    ⎕rl
2027626332
```

Note: In the example above, character 'a' is used by a seed sequence object to set the internal state of the historical algorithm.

Select Algorithm 0 and specify the initial internal state:

```
      ⎕rl← 0 ⎕ts
```

Note: In the example above, the value of ⎕ts (an integer vector) is used by a seed sequence object to set the internal state of algorithm.

Select Algorithm ¯1 and specify the initial internal state:

```
      ⎕rl← ¯1 ⎕guid
```

Note: In the example above, the value of ⎕guid (a character vector containing a Global Unique Identifier) is used by a seed sequence object to set the initial internal state of the algorithm.

Select Algorithm ¯2 and specify the initial internal state:

```
      ⎕rl← ¯2 (12.45 ¯23.67 34.125)
      ⎕rl
¯2 <<<UCS Characters>>> 0
```

Note: In the example above, the floating-point vector is used by a seed sequence object to set the initial internal state of the algorithm.

Select Algorithm ¯2 and specify the initial internal state:

```
      ⎕rl← ¯2 (10 ⍴ 1 0)
      ⎕rl
¯2 <<<UCS Characters>>> 0
```

Note: In the example above, the Boolean vector is used by a seed sequence object to set the internal state of the algorithm.

Select Algorithm ¯3 and specify the initial internal state:

```
      ⎕rl← ¯3 234.56
      ⎕rl
¯3 <<<UCS Characters>>> 0
```

Note: In the example above, the floating-point value is used by a seed sequence object to set the internal state of the algorithm.

⎕sa Stop action

Purpose:

This workspace-related system variable is provided mostly for compatibility with code from older systems. It is not useful to set ⎕sa when you are writing an application using the windows interface facilities, because the system can trigger ⎕sa inappropriately at the end of callbacks.

In an application that does not use ⎕wi or ⎕ni, ⎕sa specifies the action for the system to take whenever execution stops for immediate execution input.

Syntax:

```
result ← ⎕sa
⎕sa ← 'action'
```

Domain:

The domain for assignment to ⎕sa is limited to an empty vector or one of the following character vectors: 'CLEAR', 'EXIT', 'OFF', or 'ERROR'. In a clear workspace, the default value of ⎕sa is an empty character vector (''). The system ignores superfluous leading and trailing blanks and treats an all-blank vector as empty.

Effect:
Specifies the action for the system to take whenever execution stops for immediate execution input. The following table shows the effect of the valid values of ⎕sa.

System Actions for Various Values of ⎕sa

Value	Effect
`''`	Takes no special stop action. Execution suspends in the local environment, and the system accepts immediate execution input.
`'CLEAR'`	Clears the active workspace.
`'EXIT'`	Strips the state indicator back to an environment where ⎕sa is not `'EXIT'`. If the value of ⎕sa in the resulting environment is `'CLEAR'`, clears the workspace.
`'OFF'`	Terminates the APL session with normal untying of any tied files; returns to the operating system.
`'ERROR'`	Produces an error upon entry into immediate execution mode, which ⎕elx can trap. The system sets the local ⎕sa value to empty, places the message `STOP ACTION` in ⎕dm, and executes ⎕elx at the local level in the state indicator.

After the system takes the stop action (except for 'OFF' and possibly for 'ERROR'), it accepts immediate execution input. If you interrupt execution at a point where ⎕sa has been localized but not assigned, the system strips back the state indicator to an environment where ⎕sa is defined.

Examples:
These examples show the effect of each of the settings of ⎕sa in the global environment. For illustration, ⎕sa is not localized in any of the functions called and no other exception handlers are used.

An error occurs with ⎕sa set to its default value, and the system suspends execution at the point of error. ⎕sa is set to 'EXIT' and the function is executed again; the error recurs and the state indicator is cleared. ⎕sa is set to 'CLEAR' and the function is executed again. The error occurs once more, but the entire active workspace is cleared.

```
    )wsid
IS PROCESS
    ⎕si

    ⎕sa←''

    PROCESS 'PAYROLL'
INDEX ERROR
LOOKUP[4] ∀
    ⎕si
LOOKUP[4]*
DSEARCH[14]
XQT[8]
PAYUPDATE[38]
PROCESS[12]

    )reset
    ⎕si
```

```
      ⎕sa←'EXIT' ⋄ PROCESS 'PAYROLL'
INDEX ERROR
LOOKUP[4] ∇
      PROCESS 'PAYROLL'
      ρ⎕si
0 0

      ⎕sa←'CLEAR' ⋄ PROCESS 'PAYROLL'
INDEX ERROR
LOOKUP[4] Ã∇
CLEAR WS
      )wsid
IS CLEAR WS
```

⎕sinl State indicator with names localized

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 ← ⎕sinl

Domain:
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.

Note: Unlike APL+Win, all local variable names are indented uniformly and sorted in alphabetical order for each function before displaying them.

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
```

⎕sys System parameters

Purpose:
This session-related system variable returns and sets the status of certain internal system flags.

Syntax: result ← ⎕sys
⎕sys[i] ← data

Result:
The result is a 34-element integer vector that contains information about the interpreter's internal state; you can assign some of the elements. □sys can be localized. The meanings of the elements are:

Element	Description
[1]	Reserved for future enhancement.
[2]	This parameter controls whether a file commit occurs immediately after you write to an APL64 component file (the default; values 1 or 2) or if component file writes are buffered (to be committed later; value 0). If you set this value to 0, applications that have extensive file activity will run much faster; however, there may be timing implications, particularly in a file sharing situation. Possible values are (see Note 1): **0** no fsync() on writes **1** perform sync() on each write **2** perform fsync() on each write
[3 4 5]	Interpreter diagnostic features; for use only as directed by customer support.
[6 7]	Not used by APL64.
[8-11]	Interpreter diagnostic features; for use only as directed by customer support.
[12]	Current Evolution Level. Possible values are 0, 1, or 2 (default). To change the Evolution Level, use the)evlevel system command.
[13]	Lowest nonzero Evolution Release Level (always 1 in this release).
[14-17]	Not used by APL64.
[18]	Not used by APL64.
[19]	System version. Return a seven-digit integer, where the first four digits represent the major component of the version number; the fifth digit represents the minor component of the version number in the range of 0 – 9; and the sixth and seventh digits represent the build component of the version number in the range of 0 – 99.
[20]	The saved workspace version number. Below is the table of the workspace version and the corresponding APL64 version: **Value** | **APL64 Version** 202 | 2025.0.8
[21]	Runtime flag. 0 if development system; 1 if runtime system (WRE or CPC).
[22]	Arbitrary APL return code. You can set this value (see Note 2).
[23]	Reserved for future use.
[24]	Not used by APL64.
[25-26]	Not used by APL64.
[27]	Not used by APL64.
[28]	Not used by APL64.
[29]	Not used by APL64.
[30]	Controls the debug mode execution setting. Possible values are 0 or 1 (default).
[31]	Not used by APL64.
[32]	Not used by APL64.
[33]	Controls whether □inbuf is immediately cleared when an error occurred. Possible values are 0 and 1 (default); set to 0 to restore APL+Win behavior. (see Note 3).
[34]	Current CHECK state of the workspace. Possible values are 0 (OFF) and 1 (ON). To change the CHECK state, use the)check system command.

Note 1: The FSYNC value can be specified at startup with the command line argument COMPFILECOMMITSYNC or abbreviated CFCS. This info will be saved in the APL64 xml-format configuration file.
Note 2: You can set the 22nd element with an arbitrary value to create a "return code" except in a CPC (Cross-platform Component). When APL64 exits, this value is read by the operating system and returned to the process that invoked APL. A calling application can capture this value using the Windows GetExitCodeThread API or the DOS batch errorlevel parameter.
Note 3: The clearing of ⎕inbuf will be indicated by the presence of this line immediately following the error message: <<<<<<<<<<Inbuf Buffer Cleared>>>>>>>>>>

⎕silimit State indicator limit

Purpose:
This session-related system variable returns and sets the state indicator entries limit value.

Syntax: result ← ⎕silimit
⎕silimit ← value

Domain:
result is a scalar that displays the size of the SI stack. The default value is 10000.

Example:

```
      ⎕silimit
10000
```

⎕ucmdfile Path to initial user command file

Purpose:
This session-related system variable returns and sets the location of the root user command file.

Syntax: result ← ⎕ucmdfile
⎕ucmdfile ← data

Domain:
result is a character vector containing the path of the UCMDS.SF user command file used by the current APL64 instance. The full path of this file can be set by assigning data to ⎕UCMDFILE or by using the UCMDFILE command line argument.

The default path to this file is provided by ⎕USERPATH.

The result is 0⍴'' if the ⎕ucmdfile is not defined.

Example:

```
      ⎕ucmdfile
C:\Users\Joe\AppData\Roaming\APLNowLLC\APL64\UCMDS
```

⎕warg Argument for a Windows event or method

Purpose:
This session-related system variable contains the arguments associated with an event or method.

Syntax: res ← ⎕warg
⎕warg ← arguments

Domain:
The descriptions of each event and method in the Windows Reference Manual specify the domain for this variable. When no callbacks are pending, ⎕warg is an empty numeric vector, unless you have assigned it in immediate execution when no callbacks were pending.

Effect:
When a callback is pending, the system sets ⎕warg to the appropriate value for the event or method at the time that the callback is invoked. These arguments are implicitly localized during a callback and restored to their prior values when the callback finishes.

You can change the value of ⎕warg during a callback or in immediate execution mode.

Example:
In the following example, each time you click a mouse button on the form named fmForm (but not necessarily on one of the children of the form), the system displays the result of ⎕warg in your session. In this case, the appropriate values include the coordinates of the mouse, which button was clicked, which mouse buttons were pressed, and the keyboard shift state.

```
'fmForm' ⎕wi 'onMouseDown' '⎕warg'
```

⎕watchpoints

Purpose:
This session-related system variable is a debugging tool that suspends program execution when a variable changes (or could change) and a condition you specify is met.

Syntax: condition ← ⎕watchpoints
⎕watchpoints ← condition

Domain:
condition is a two-column nested array of character vectors. The first column holds the name of the variable whose imminent change in value will trigger evaluation of the expression in the second column; if that expression evaluates to true (non-zero) execution will be suspended.

Effect:
The value of ⎕WATCHPOINTS in a clear workspace is a zero-row matrix. The ⎕WATCHPOINTS system variable is saved with and loaded with a workspace.

Note: Refer to the section *Debugging with Watchpoints* in **Chapter 2: Working in an APL64 Session** in the APL64 User Manual for additional information.

⎕wevent Windows event

Purpose:
This session-related system variable contains the name of the current event during a callback.

Syntax: event ← ⎕wevent
⎕wevent ← 'event'

Domain:
event is a character vector with the event name. The default value when no callbacks are pending is an empty character vector unless you have assigned it in immediate execution when no callbacks were pending.

Effect:
When a callback occurs on an object, ⎕wevent contains the name of the event or method for the system to handle. For example, if you have a form with an OK button, when the user clicks the mouse on that button, the system handles that event by executing the onClick event-handler expression. During the callback, ⎕wevent has the value Click.

If the callback occurs because of a ⎕wcall-defined filter, ⎕wevent has the value '?'. If the callback is due to the invocation of the ⎕wi 'Exec' method, ⎕wevent has the value Exec.

Example:
In the following example, each time you click a mouse button on the form named fmForm (but not necessarily on one of the children of the form), the system displays the result of ⎕wevent in your session.

```
'fmForm' ⎕wi 'onMouseDown' '⎕wevent'
```

⎕wordrepla Word forming characters

Purpose:
This workspace-related system variable contains the characters ⎕wordrepl considers words in APL.

Syntax: vector ← ⎕wordrepla
⎕wordrepla ← vector

Domain:
Character vector or singleton that contains the characters which can constitute word elements in the ⎕wordrepl system function. In a clear workspace, the default value of ⎕wordrepl is 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789Δ∆_⎕ '.

Effect:
⎕WORDREPLA may be modified by the APL64 programmer.

⎕WORDREPLA may be localized in an APL64 programmer-defined function.

When an APL64 instance is created, a workspace is loaded or the workspace is cleared, ⎕WORDREPLA is initialized to the default value.

⎕WORDREPLA may be saved in an saved APL64 workspace.

Examples:

```
CLEAR WS
      □wordrepla
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789Δ⍙_□
      □wordrepla←□wordrepla,'!'
      □wordrepla
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789Δ⍙_□!
      '/Δhello/Δworld/hello/you/!/#' □WORDREPL 'Δhello hello !!'
Δworld you !!
      '/Δhello/Δworld/hello/you/!/#' □WORDREPL 'Δhello hello !'
Δworld you #
```

□wres Windows result

Purpose:
This session-related system variable contains the explicit result, if any, that a Windows method or event handler returns.

Syntax: result ← □wres
□wres ← result

Domain:
result is any scalar, vector, or array with the result of the method or event handler. The default value when no callbacks are pending is zero unless you have assigned it in immediate execution when no callbacks were pending.

Effect:
□wres is used to determine the explicit result (usually 0 0 ρ0) that is returned as the result of □wi from a method or event handler. In some cases, your code sets □wres to control how a user action is processed, for example, to filter or translate keystrokes. See the descriptions of specific event-handler properties and methods in the Windows Reference Manual for more information.

□wself Current Windows object

Purpose:
This session-related system variable contains the fully qualified name of the current Windows object.

Syntax: wobj ← □wself
□wself ← 'newobj'

Domain:
wobj is the name of the current Windows object. The default value when no callbacks are pending is an empty character vector unless you have assigned it in immediate execution when no callbacks were pending. You can assign a character vector with the name of the Windows object you want to become current.

Effect:
When processing a callback, APL sets □wself to the fully qualified name of the current object. You can

use this in an event handler to identify the object that the event applies to. This also allows you to write a handler that will work with any number of different objects, since any GUI actions you take using monadic ⎕wi apply to the current object.

If the callback occurs because of a ⎕wcall-defined filter, ⎕wself is empty ('').
You can change the value of ⎕wself during a callback or in immediate execution mode. If you assign ⎕wself to the name of an object, you can refer to the object implicitly by using ⎕wi without a left argument.

Delocalization of ⎕wself to a non-existent name sets ⎕wself to an empty vector.

Example:
In the following example, a form named fmForm has two buttons; each time you click a mouse button, the system displays the name of the object that the mouse was on when you clicked it.

```
'fmForm' ⎕wi 'onMouseDown' '⎕wself'
'fmForm.bn1' ⎕wi 'onMouseDown' '⎕wself'
'fmForm.bn2' ⎕wi 'onMouseDown' '⎕wself'
```

⎕wsid Workspace identification

Purpose:
This workspace-related system variable reports or changes the active workspace identification.

Syntax: result ← ⎕wsid
⎕wsid ← 'wsid'

Domain:
⎕wsid contains any well-formed workspace name, optionally preceded by a directory or library designation; the system assumes the default extension (.ws64).

Result:
⎕wsid returns the workspace identification. If the workspace name is a clear workspace, the system returns an empty vector. The actual format depends on whether you have defined libraries. If the directory is not associated with a library number, ⎕wsid contains the workspace name in directory form.

If the directory for the workspace name is currently associated with a library number in ⎕libs, ⎕wsid is a character vector. 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.

When an APL64 application is based on a WRE (Windows Runtime Executable) or CPC (Cross-platform Component) the value returned from ⎕wsid is "{LoadedFromStream}", indicating that the workspace being used by the application was embedded in a WRE (.exe) and is not a physical .ws64 file. To obtain the WRE (.exe file) at runtime, use ⎕exepath.

Examples:

```
    )wsid
IS C:\T\ANSWER
```

```
    ⎕wsid ⋄ ρ⎕wsid
C:\T\ANSWER
11
    ⎕wsid←'11 TOOLS' ⋄ ⎕wsid ⋄ ρ⎕wsid
 11 TOOLS
22
```

⎕wsname Workspace name

Purpose:
This workspace-related system variable reports or changes the active workspace identification.

Syntax: result ← ⎕wsname
⎕wsname ← 'wsname'

Domain:
⎕wsname contains any well-formed workspace name.

Result:
⎕wsname returns the workspace identification always preceded by a directory and the default extension (.ws64) when an extension isn't present. ⎕wsname, unlike ⎕wsid, is not affected by library number definitions and therefore always returns a full file path name.

If the workspace name is a clear workspace, the system returns an empty vector. The actual format depends on whether you have defined libraries.

When an APL64 application is based on a CPC (Cross-platform Component) the value returned from ⎕wsname is "{LoadedFromStream}", indicating that the workspace being used by the application was embedded in a CPC (Nuget package) and is not a physical .ws64 file. To obtain the CPC file location at runtime, use ⎕exepath.

Examples:

```
    )wsname
IS C:\T\ANSWER.ws64
    ⎕wsname ⋄ ρ⎕wsname
C:\T\ANSWER.ws64
16
    ⎕wsid←'11 TOOLS' ⋄ ⎕wsname ⋄ ρ⎕wsname
C:\T\11 TOOLS.ws64
18
    ⎕wsid←'11 TOOLS.w3' ⋄ ⎕wsname ⋄ ρ⎕wsname
C:\T\11 TOOLS.w3
16
```