APL64 System Commands

System commands are instructions to the APL64 system rather than facilities of the APL language interpreter. System commands all begin with a right parenthesis,), to distinguish them from APL language statements. System commands are not case sensitive, and you can abbreviate them to the first four letters. The commands are listed below in categories related to their purposes.

Contents

Active Workspace Environment

)check Displays or changes the check execution mode

Purpose:
Displays or changes the check execution mode

Syntax:

```
)check
)check ON
)check OFF
```

Effect:

This command controls whether the XCHECK overhead is performed that recompiles any APL statement xcodes in the workspace. Previously executed statements in the workspace generate xcodes which reference a symbol table entry. This can be out of sync, for example when an APL statement is executed and calls a function that didn't exist in the workspace. Enabling check execution mode will re-build the xcode, reminiscent of a ⎕def ⎕vr of a function, for every function in the workspace to allows functions to run optimally. This is also referred to as "Reflowing" the workspace.

Displays or sets the check execution mode. This command allows the programmer to turn check execution mode off or on. This setting is workspace relative. It is saved with the workspace on a)Save command and loaded with the workspace on a)Load command. The)Clear command resets it to the default value.

Note: The)check command cannot be executed unless the)SI level is clear. If functions are suspended, you cannot change their settings.

The three states are set by the following arguments to the system command:

)check

This command prints the current check execution mode state.

)check ON

Check execution mode is enabled.

)check OFF

This is the default state. Check execution mode is disabled.

Similar to)debug, whenever check mode changes, all functions in the workspace are "reflowed". The current check state of the workspace and be queried with ⎕sys[34].

Remarks:
In APL+Win, the "binding valence" of every object named in a function statement was checked before executing that statement. This pre-execution overhead was encountered each time a statement was executed to make sure a variable hadn't been redefined as a function or vice versa, or that the valence of a function hadn't change (such as a monadic function being changed to a dyadic function). This must

be done before the statement executes so that if the valence has changed, the statement's xcode can be rebuilt before running that code. But this adds overhead to statement execution. To reduce this overhead, the)CHECK system command was introduced in APL64 to give developers control over this behavior. In most cases the valences of names do not change during execution of a particular function statement and this pre-execution overhead can be eliminated. So, the default is OFF, removing this xcode valence check overhead from function execution.

It is unusual in code for a name to change valence between statement executions. But sometimes during development of code, you might execute a statement that references a name that doesn't exist leading to a VALUE ERROR. And this might simply be because a function you intended to execute hadn't yet been defined in the workspace. In this case, the missing variable is going to be seen (and bound into the xcode) as a variable. So, a first execution will construct the xcode for that statement with the name assumed to be a variable. If you hit this VALUE ERROR and then define or copy the intended function into the workspace and try executing the original function again, this may result in a TYPE CHECK ERROR. To resolve this condition, you can execute either)CHECK ON or)CHECK OFF. Either action will reflow all functions in the workspace, removing previously compiled xcode, and allowing correct execution if you try running the code again.

The recommendation is to set)CHECK ON during development. This will make the code run slightly slower but will avoid such TYPE CHECK ERROR conditions. But before running the code in a production context you should execute)CHECK OFF to disable valence checking and make the code run as fast as possible.

)clear Clears the active workspace

Purpose:
Clears the active workspace.

Syntax:)clear

Effect:
Discards the contents of the active workspace and resets the workspace-related system variables to their default values. The)clear operation does not affect file ties or session-related system variables. When you clear a workspace, the system erases the contents and deletes the workspace name, if any. Even after you create new functions or variables in the workspace, the system identifies it as the CLEAR WS until you assign it a name.

You can clear the workspace under program control by using: ⎕sa←'CLEAR' ⋄ →

Example:
The example below shows a named workspace that contains user-defined variables, with nondefault settings for one session-related and one workspace-related system variable. After the workspace is cleared, the user-defined variables are deleted and the workspace-related variable (⎕io - index origin) is reset to its default value. The session-related system variable (⎕pw - printing width) retains its nondefault value.

```
      )wsid
IS D:\APL64\EXAMPLE
      ⎕pw"56
```

```
    ⎕io"0
    )vars
A   B    C    DAY   E
F   G    H    I
 )clear
CLEAR WS
    )vars
    )wsid
IS CLEAR WS
    ⎕pw
56
    ⎕io
1
```

)csave Commits function editing history records and saves workspace

Purpose:
Commits existing function editing history records for functions in the current workspace and saves the workspace.

Syntax:)csave

Effect:
Creates a copy of the active workspace as a file on disk with the name: wsid.ws64 after first having committed any existing function editing history records to the workspace. **Note**: This action cannot be reversed.

For maximum safety during the)csave operation, the system builds a new workspace file as a temporary file with a unique DOS file name. After the system successfully saves the entire workspace in the temporary file, it erases the old workspace file and renames the temporary file. If a disk error or system crash occurs during the save process, the original version of the saved workspace remains intact on the disk.

)debug Displays or changes the debug/release execution mode

Purpose:
Displays or changes the debug/release execution mode.

Syntax:
```
)debug
)debug ON
)debug OFF
```

Effect:

Displays or sets the debug execution mode. This command allows the programmer to turn debug execution mode off or on. This setting is workspace relative. It is saved with the workspace on a)Save command and loaded with the workspace on a)Load command. The)Clear command resets it to the default value.

Note: The)debug command cannot be executed unless the)SI level is clear. If functions are suspended, you cannot change their settings.

The three states are set by the following arguments to the system command:

)debug

This command prints the current debug execution mode state.

)debug ON

This is the default state. Debug execution mode is enabled.

)debug OFF

Debug execution mode is disabled.

When debug mode is enabled the :DEBUG, :TRACE, :ASSERT, and :IFDEBUG statements behave as described below. Whenever debug mode changes, all functions in the workspace are "reflowed". In release mode this causes debug statements to become logically invisible as if they were comments. Reflowing calculates the next-statement pointer for normal statements to make debug statements completely skipped without incurring any execution overhead, not even one machine cycle. In release mode, debug statements are not just inexpensive, they are literally free. ⎕IT's Internal Representation (IR) command lets us examine this in detail. When debug mode is OFF we have the following:

```
       )debug off
       'IR' ⎕IT 'Foo'
... Next/False Stmt [Line] Code
...    4          0 [0]    Foo
...    4          1 [1]    :debug ⎕←'debug'
...    4          2 [2]    :trace ⎕←'trace'
...    +          3 [3]    :assert 0<ρ⎕←'assert'
...    8          4 [4]    ⎕←'normal'
...    +/8        5 [5]    :ifdebug
...    8          6 [6]      ⎕←'ifdebug'
...    +          7 [7]    :end
```

Some output from these commands is omitted for brevity. The Next/False column shows the next statement to execute following each statement. The /False part only exists for "decision" lines (such as line [5]), which must make true/false decisions about what statements come next. The next statement for line [0] tells us the first line to execute. This is set to 4 meaning the first three lines of debug statements will be completely skipped and execution will start on line [4] with the "normal" line. Its next statement is 8. This skips over the remaining debug statement and exits from the function. When debug mode is ON, this flow pattern changes completely as shown below:

```
       )debug on
       'IR' ⎕IT 'Foo'
... Next/False Stmt [Line] Code
...    +          0 [0]    Foo
...    +          1 [1]    :debug ⎕←'debug'
...     +           2 [2]    :trace ⎕←'trace'
...     +           3 [3]    :assert 0<⍴⎕←'assert'
...     6           4 [4]    ⎕←'normal'
...     +/8         5 [5]    :ifdebug
...     8           6 [6]      ⎕←'ifdebug'
...     +           7 [7]    :end
```

A plus (+) is displayed in the Next/False column when the next statement immediately follows the previous one. The (+) shown as the next statement for line [0] means execution begins at line [1] with the first debug statement. This is followed by two more debug statements until we reach the "normal" line and then continue with the final debug statement.

Even when debug mode is ON the flow optimizer avoids executing unnecessary lines (for example, lines [5] and [7] are never executed). When line [4] finishes execution we continue on line [6] without executing line [5]. Logically we need to execute line [5] to make a debug or non-debug decision about whether line [6] should to be executed or not. However, when we reflow the function for debug mode, we already know the decision on line [5] will always be true so we can skip that line and jump directly to line [6] for any line that would normally flow into line [5]. Similarly, when line [6] finishes execution it skips over line [7] because the flow optimizer knows there is nothing to execute there.

Execution of this code with debug mode OFF and ON illustrates these modes in action:

```
      )debug off
WAS ON
NOW OFF
      Foo
normal
      )debug on
WAS OFF
NOW ON
      Foo
debug
trace
assert
normal
ifdebug
```

)drop Deletes a saved workspace from disk

Purpose:
Erase a saved workspace from disk storage.

Syntax:)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 ⎕drop system function provides a similar capability under program control.

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

Examples:

```
    )drop TEMPWS
05/25/2021 12:17:13
    )drop
C:\TEMP\OLDWS
05/24/2021 10:50:51
```

)flib Displays a list of component files in a directory or library

Purpose:
Lists the names of APL component files with the .SF extension in a library or directory.

Syntax:
```
)flib
)flib dir
)flib lib
```

Arguments:
dir is the directory you want to search.

lib is the library number of the directory you want to search.

Effect:
Lists all files that have the extension .SF, assuming them to be component files, stored in the specified directory or library, even if the user has no access to them. If you do not specify a library number or directory name,)flib searches the current working directory. You can specify a full path name even if you used ⎕libd to define the path as a library. The listing of names does not include the extension. ⎕flib provides a similar capability you can use under program control.

Examples:

```
    )flib
DATEBOOK  TAXDATA
    ⎕libd '3 C:\MYLIB'
    )flib 3
ORACLE    REPORTS
    )flib C:\APLW
PRINTERS  SLT      SERUPFNS
```

)fns Displays the names of functions in the active workspace

Purpose:
Lists the names of all user-defined functions in the active workspace.

Syntax:
```
)fns
)fns pattern
)fns @wsname
```

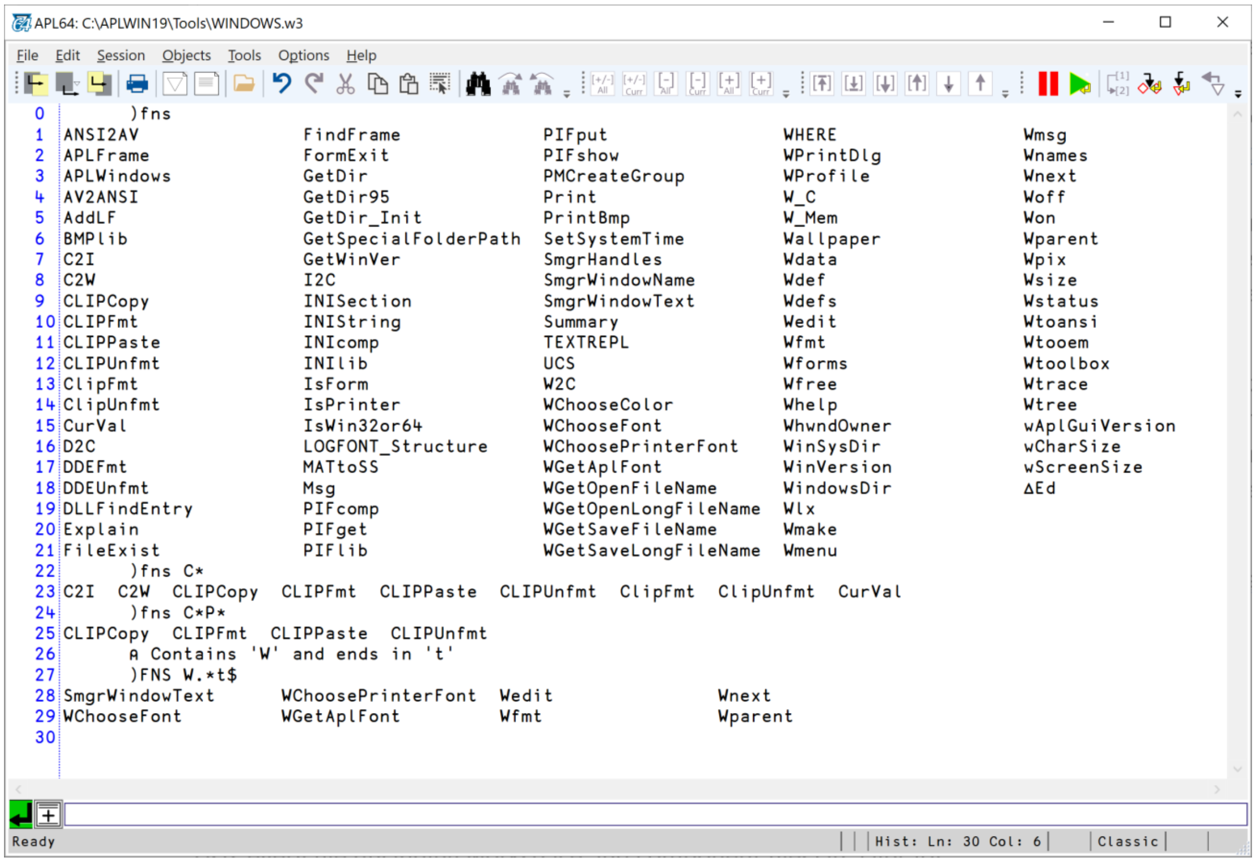
Argument:
pattern is a list of zero or more patterns, separated by blanks, for filtering the names; extended wildcard notation and regular expression (regex) filter patterns are supported;
@wsname is the name of a saved workspace that to you want to query for its list of user-defined functions. **Note**: Specify an APL+Win workspace with the .w3 extension.

Effect:
Displays a list, in alphabetical order, of the user-defined functions in the active workspace. Specifying the optional pattern returns the names that match the argument pattern. The pattern can contain literal characters that are valid in names such as A-Z, a-z, 0-9, _, Δ, or ⍙. In addition, a * (asterisk) can be used to denote zero or more of any character and a ? (question mark) can be used to denote any single character. Any other characters kick the phrase into extended regex notation. In this case, the entire expression is considered to be a "regular expression" with the syntax rules as described here: https://docs.microsoft.com/en-us/dotnet/standard/base-types/regular-expression-language-quick-reference. Remember that in regular expressions, a . (period) selects any character, and .* (asterisk) selects zero or more occurrences of any character, and ^ character denotes the beginning of the name and $ character denotes the end of the name. So in this case the * and ? in the argument patterns are treated as standard wildcard characters where the * means zero or more occurrences of any character and ? means 1 occurrence of any character.

With regular expression syntax, implicitly include a leading ^ character before the regex specification and a trailing $ character after it. This means that if you specify AB[CZ].*X as a regex pattern the full regular expression will actually be ^AB[CZ].*X$. The ^ character means the start of the line (but in the pattern matching context for names this means the start of a name). The $ character means the end name.

⎕nl and ⎕idlist provide a similar capability you can use under program control. The user command]fns provides a similar but more flexible capability.

Examples:


APL64: C:\APLWIN19\Tools\WINDOWS.w3
File Edit Session Objects Tools Options Help
Ready
Hist: Ln: 30 Col: 6
Classic

```
      )fns
ANSI2AV               FindFrame             PIFput                WHERE                 Wmsg
APLFrame              FormExit              PIFshow               WPrintDlg             Wnames
APLWindows            GetDir                PMCreateGroup         WProfile              Wnext
AV2ANSI               GetDir95              Print                 W_C                   Woff
AddLF                 GetDir_Init           PrintBmp              W_Mem                 Won
BMPlib                GetSpecialFolderPath  SetSystemTime         Wallpaper             Wparent
C2I                   GetWinVer             SmgrHandles           Wdata                 Wpix
C2W                   I2C                   SmgrWindowName        Wdef                  Wsize
CLIPCopy              INISection            SmgrWindowText        Wdefs                 Wstatus
CLIPFmt               INIString             Summary               Wedit                 Wtoansi
CLIPPaste             INIcomp               TEXTREPL              Wfmt                  Wtooem
CLIPUnfmt             INIlib                UCS                   Wforms                Wtoolbox
ClipFmt               IsForm                W2C                   Wfree                 Wtrace
ClipUnfmt             IsPrinter             WChooseColor          Whelp                 Wtree
CurVal                IsWin32or64           WChooseFont           WhwndOwner            wAplGuiVersion
D2C                   LOGFONT_Structure     WChoosePrinterFont    WinSysDir             wCharSize
DDEFmt                MATtoSS               WGetAplFont           WinVersion            wScreenSize
DDEUnfmt              Msg                   WGetOpenFileName      WindowsDir            ∆Ed
DLLFindEntry          PIFcomp               WGetOpenLongFileName  Wlx
Explain               PIFget                WGetSaveFileName      Wmake
FileExist             PIFlib                WGetSaveLongFileName  Wmenu
      )fns C*
C2I  C2W  CLIPCopy  CLIPFmt  CLIPPaste  CLIPUnfmt  ClipFmt  ClipUnfmt  CurVal
      )fns C*P*
CLIPCopy  CLIPFmt  CLIPPaste  CLIPUnfmt
      ⍝ Contains 'W' and ends in 't'
      )FNS W.*t$
SmgrWindowText      WChoosePrinterFont  Wedit               Wnext
WChooseFont         WGetAplFont         Wfmt                Wparent
```

)lib Displays a list of all files in a directory or library

Purpose:
Lists every file (including workspace and component files) in a library.

Syntax:)lib
)lib dir
)lib lib

Arguments:
dir is the directory you want to search;

lib is the number of library you want to search.

Effect:
Lists all files stored in the specified directory or library. If you do not specify an argument,)lib lists the files in the current working directory. This list, unlike those generated by)flib and)wslib, includes the extensions .ws64 for saved APL workspaces and .SF for APL component files saved using a traditional file function. You can specify a full path name even if you used ⎕libd to define the path as a library. ⎕lib provides a similar capability you can use under program control.

Examples:

```
    )lib
DATES.ws64   TEST.SF
```

```
      ⎕libd '3 C:\APLW\MYSTUFF'
      )lib 3
JUNK.SF    TEST.ws64
      )lib C:\APLW\WSS
ADDSUB.C   DEMO.WS     MOVEFILE.WS
APL        FORMAT.WS   XDEMO.VF
CORE       MAKEFILE
```

)libs Library to directory correspondence

Purpose: Displays the definitions of the libraries in use during this session.

Syntax:)libs

Effect:
Displays the library definitions in use during this session. If)libs returns no output , it indicates that you did not define any libraries. If you assign any library numbers to directories, then)libs lists the library-to-directory correspondences. You can use each library number as a substitute for the corresponding directory name.

⎕libs provides a similar capability that you can use under program control.

Examples:

```
      )libs
      ⎕libd '11 C:\APLWIN\TOOLS'
      )libs
11 C:\APLWIN\TOOL)
```

)load Loads a saved workspace from disk

Purpose:
Activates a saved workspace by replacing the active workspace with a copy of a workspace stored on disk.

Syntax:)load wsid

Argument:
wsid is the workspace identifier.

Effect:
Replaces the active workspace with a copy of the specified saved workspace (wsid) and displays the time and date that the workspace was saved. Once loaded, the system automatically executes the latent expression (⎕lx). In a workspace saved with a nonempty state indicator, ⎕lx could be a localized latent expression.

The workspace can be in any directory. If you do not specify a directory,)load assumes the current directory. If the system does not locate the specified workspace in the specified directory, it displays a WS NOT FOUND message. The)load operation does not affect file ties or session-related system variables.

Note: In cases where another user has just resaved the workspace you are trying to load, you may get a WS NOT FOUND message if the timing places your load operation between the erasing of the old workspace and the renaming of the temporary workspace. Simply repeat the operation.

⎕load provides a similar capability you can use under program control. See also the descriptions of)xload, ⎕qload, and ⎕xload.

)names Displays the names of variables and functions in the workspace

Purpose:
Lists the names of all variables and user-defined functions in the active or saved workspaces.

Syntax:
```
)names
)names pattern
)names @wsname
```

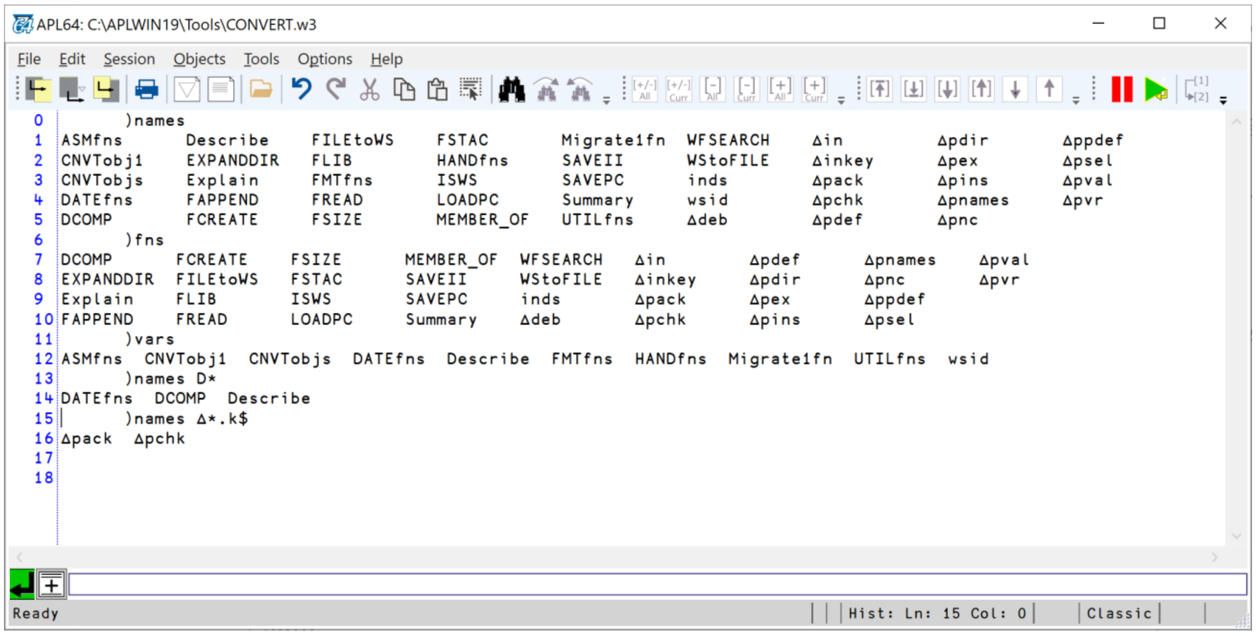
Argument:
pattern is a list of zero or more patterns, separated by blanks, for filtering the names; extended wildcard notation and regular expression (regex) filter patterns are supported;
@wsname is the name of a saved workspace that to you want to query for its list of variables and user-defined functions. **Note**: Specify an APL+Win workspace with the .w3 extension.

Effect:
Displays a list, in alphabetical order, of the user-defined functions and variables in the active or saved workspaces. Specifying the optional pattern returns the names that match the argument pattern. The pattern can contain literal characters that are valid in names such as A-Z, a-z, 0-9, _, ∆, or ⍙. In addition, a * (asterisk) can be used to denote zero or more of any character and a ? (question mark) can be used to denote any single character. Any other characters kick the phrase into extended regex notation. In this case, the entire expression is considered to be a "regular expression" with the syntax rules as described here: https://docs.microsoft.com/en-us/dotnet/standard/base-types/regular-expression-language-quick-reference. Remember that in regular expressions, a . (period) selects any character, and .* (asterisk) selects zero or more occurrences of any character, and ^ character denotes the beginning of the name and $ character denotes the end of the name. So in this case the * and ? in the argument patterns are treated as standard wildcard characters where the * means zero or more occurrences of any character and ? means 1 occurrence of any character.

With regular expression syntax, implicitly include a leading ^ character before the regex specification and a trailing $ character after it. This means that if you specify AB[CZ].*X as a regex pattern the full regular expression will actually be ^AB[CZ].*X$. The ^ character means the start of the line (but in the pattern matching context for names this means the start of a name). The $ character means the end name.

⎕nl and ⎕idlist provide a similar capability you can use under program control. The user command]fns provides a similar but more flexible capability.

Examples:


APL64: C:\APLWIN19\Tools\CONVERT.w3
File Edit Session Objects Tools Options Help
Ready
Hist: Ln: 15 Col: 0
Classic

```
      )names
ASMfns      Describe    FILEtoWS    FSTAC       Migrate1fn  WFSEARCH    ∆in         ∆pdir       ∆ppdef
CNVTobj1    EXPANDDIR   FLIB        HANDfns     SAVEII      WStoFILE    ∆inkey      ∆pex        ∆psel
CNVTobjs    Explain     FMTfns      ISWS        SAVEPC      inds        ∆pack       ∆pins       ∆pval
DATEfns     FAPPEND     FREAD       LOADPC      Summary     wsid        ∆pchk       ∆pnames     ∆pvr
DCOMP       FCREATE     FSIZE       MEMBER_OF   UTILfns     ∆deb        ∆pdef       ∆pnc
      )fns
DCOMP      FCREATE    FSIZE      MEMBER_OF  WFSEARCH   ∆in        ∆pdef      ∆pnames    ∆pval
EXPANDDIR  FILEtoWS   FSTAC      SAVEII     WStoFILE   ∆inkey     ∆pdir      ∆pnc       ∆pvr
Explain    FLIB       ISWS       SAVEPC     inds       ∆pack      ∆pex       ∆ppdef
FAPPEND    FREAD      LOADPC     Summary    ∆deb       ∆pchk      ∆pins      ∆psel
      )vars
ASMfns  CNVTobj1  CNVTobjs  DATEfns  Describe  FMTfns  HANDfns  Migrate1fn  UTILfns  wsid
      )names D*
DATEfns  DCOMP  Describe
      )names ∆*.k$
∆pack  ∆pchk
```

)output Helps manage implicit out in the APL history

Purpose:
Manage implicit output in the APL history.

Syntax:)output
)output ON
)output OFF
)output ERROR
)output STRICT
)output NOCALLBACK

Effect:
Helps to manage implicit output in the APL history from APL code that generates unintended implicit output. This command allows the programmer to turn implicit output off or on. It also provides a means for locating code which performs implicit output. The command sets a four state switch in the active workspace. This switch is workspace relative. It is saved with the workspace on a)Save command and loaded with the workspace on a)Load command. The)Clear command resets it to the default value.

Implicit output states only apply to expressions executed within a function or an event handler. They do not apply to immediate execution statements.

The four states are set by the following arguments to the system command:

)output ON

This is the default state. Implicit output is generated by any APL expression that does not assign its result.

)output OFF

In this state Implicit output is not generated.

)output ERROR

This state is used to locate expressions that generate implicit output. In this state execution of any expression that would generate implicit output in the default state causes an IMPLICIT OUTPUT ERROR. An expression beginning with 0 0ρ by definition generates no output and so does not cause an IMPLICIT OUTPUT ERROR.

)output STRICT

This state is also used to locate expressions that generate implicit output, but gives slightly different behavior. The)output ERROR state may produce IMPLICIT OUTPUT errors on some implicit output that is not visible. The)output STRICT state produces errors only when visible implicit output is generated. In addition, this state produces the IMPLICIT OUTPUT error message after the output has been generated, so the effect of the output can be seen.

)output NOCALLBACK

This state suppresses the callback info with **>[object;event]** prefix from displaying in the APL64 history. In APL64, all callbacks generate info with **>[object;event]** prefix before the executable statement. The purpose of this is to accurately reflect what occurred during an instance of the APL64 Developer version. To maintain this accurate history, the cause and effect of an action are displayed in the APL64 history pane. Examples of this are a user-entered APL executable statement, interpreter callbacks, invocation of a user-defined tool, etc. In each case the cause and the result, if any, are displayed in the APL64 history.

)output

This command prints the current implicit output state. The other four commands print the previous implicit output state.

Remarks:
Implicit output from ⎕elx and ⎕alx are not affected by the)OUTPUT command.

)profile Profile Performance

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 Saves the active workspace unless one exists under that name (protected save)

Purpose:
Saves the active workspace after first checking that saving the workspace will not replace an existing workspace with the same name.

Syntax:)psave
)psave wsid

Argument:
wsid is the workspace identifier for the saved workspace.

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. If a workspace already exists with the supplied name, a WS NAME ERROR occurs. If you supply a directory name or library number,)psave saves the file in the specified directory; otherwise, it saves the file in the current directory. If you do not specify a wsid, the system uses the active workspace identifier (⎕wsid), including its library number or directory name. You cannot save a clear (that is, unnamed) workspace; you must name it first or include the name in the argument to)psave.

If wsid is different from the workspace name, the system updates ⎕wsid, ⎕wsts, and ⎕wsowner to match the values of the saved workspace.

For maximum safety during the)psave operation, the system builds the new workspace file as a temporary file with a unique DOS file name. After the system successfully saves the entire workspace in the temporary file, it erases the old workspace file and renames the temporary file. If a disk error or system crash occurs during the save process, the original version of the saved workspace remains intact on the disk.

⎕psave provides a similar capability you can use under program control. See also the descriptions of)save and ⎕save.

)reset Clears the state indicator

Purpose:
Clears the state indicator of the active workspace.

Syntax:
```
)reset
)reset n
```

Argument:
n is the number of suspensions you want to clear from the state indicator.

Effect:
)reset clears the state indicator completely, as opposed to → or)reset 1, which clear only the most recent suspension.

If you specify n, the system clears the state indicator for n suspensions.

⎕sa provides a similar capability that you can use under program control with (⎕sa←'EXIT' ⋄ →).

Examples:

```
    )si
SUBFN[6]*
STARTUP[2]
SUBFN[5]*
STARTUP[2]
SUBFN[4]*
STARTUP[2]
    →
    )si
```

```
SUBFN[5]*
STARTUP[2]
SUBFN[4]*
STARTUP[2]
    )reset
    )si
```

)rsave Runtime save of a workspace (obsolete)

This system command is deprecated in APL64.

)save Saves the active workspace

Purpose:
Saves a copy of the active workspace on disk under the specified name.

Syntax:
```
)save
)save wsid
```

Argument:
wsid is the workspace identifier.

Effect:
Creates a copy of the active workspace as a file on disk with the name: wsid.ws64 that can be loaded in either the development or runtime system.

If you also supply the directory name or library number, the system saves the file in the specified directory; otherwise, it saves the file in the current directory. If you do not specify a wsid, the system uses the active workspace identifier (⎕wsid), including its library number or directory name. You cannot save a clear (that is, unnamed) workspace; you must name it first or include the name in the argument to)save.

If wsid is different from the active workspace name, the system updates ⎕wsid, ⎕wsts, and ⎕wsowner to match the values of the saved workspace. If the active workspace name is different from wsid and a workspace is already saved on disk with a name of wsid, the system displays a NOT SAVED THIS WS IS... message.

For maximum safety during the)save operation, the system builds a new workspace file as a temporary file with a unique DOS file name. After the system successfully saves the entire workspace in the temporary file, it erases the old workspace file and renames the temporary file. If a disk error or system crash occurs during the save process, the original version of the saved workspace remains intact on the disk.

⎕save provides a similar capability you can use under program control, but it does not check whether the name already exists. See also the descriptions of)psave and ⎕psave.

)saveover Save over the active workspace

Purpose:
Saves a copy of the active workspace on disk using the name of a workspace that already exists.

Syntax:)saveover
)saveover wsid

Argument:
wsid is the workspace identifier.

Effect:
Creates a copy of the active workspace as a file on disk with the name: wsid.ws64

If you also supply the directory name or library number, the system saves the file in the specified directory; otherwise, it saves the file in the current directory. If the active workspace name is different from wsid and a workspace is already saved on disk with a name of wsid, the system erases the existing file and replaces it with the active workspace.

)si Displays state indicator

Purpose:
Displays the state indicator of the active workspace, showing which functions are pendent or suspended.

Syntax:)si

Effect:
Displays the state indicator starting with the most recent entry. The state indicator includes the status of:

- suspended functions
- pendent functions
- user commands (])
- executes (-)
- evaluated input (⎕) calls
- debugger suspensions (#)
- callouts from APL using ⎕wi, ⎕call, or ⎕wgive (>)
- callbacks from the APL Windows interface (<).

The list shows the name of the function and the number of the statement at which execution suspended. 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.

In the cases of callouts or callbacks, the list shows the name of the object and action or event.

⎕si provides a similar capability you can use under program control.

Example:

```
    )si
<[fmMain;Wait]
SUBFN[7]*
REPORT[3]
SUBFN[7]*
```

```
STARTUP[11]
 ⍎
```

)sic Clears the state indicator

Purpose:
Clears the state indicator of the active workspace.

Syntax:)sic

Effect:
Clears the state indicator completely, as opposed to →, which clears only the most recent suspension. The system command)reset performs the same function as)sic. ⎕sa provides a similar capability you can use under program control (⎕sa← 'EXIT' ⋄ →).

Examples:

```
     )si
SUBFN[6]*
STARTUP[2]
SUBFN[5]*
STARTUP[2]
SUBFN[4]*
STARTUP[2]
     →
     )si
SUBFN[5]*
STARTUP[2]
SUBFN[4]*
STARTUP[2]
     )sic
     )si
```

)sinl Displays state indicator

Purpose:
Displays the state indicator of the active workspace, showing which functions are pendent or suspended and which names are localized within each function.

Syntax:)sinl

Effect:
Displays the same information as)si with the addition of localized names at each level of the stack. ⎕sinl provides a similar capability you can use under program control.

Example:

```
     )sinl
SUBFN[5] * L1  L2  X  ⎕io
REPORT[3]  X  Y  ⎕ELX
```

```
SUBFN[5] * L1  L2  X  ⎕io
STARTUP[11] RESULT  MORE  DONE
```

)symbols Displays the size of the symbol table

Purpose:
Displays or changes the number of symbol table entries for which there is space reserved in the active workspace.

Syntax:)symbols
)symbols @wsname

Argument:
@wsname is the name of a saved workspace that to you want to query for its number of symbol table entries. **Note**: Specify an APL+Win workspace with the .w3 extension.

Effect:
)symbols reports the current number of entries possible in the symbol table of the active workspace and the number in use. The system also automatically enlarges the symbol table when it requires additional symbol space.

⎕symb provides a reporting capability you can use under program control.

Example:

```
    )clear
CLEAR WS
    )symbols
IS 0
    A←B←C←55
    )symbols
IS 3
```

Note: Entries in the symbol table of the active workspace never get erased during an APL session instance.

)vars Displays variable names

Purpose:
Lists the names of the variables in the active workspace.

Syntax:)vars
)vars pattern
)vars @wsname

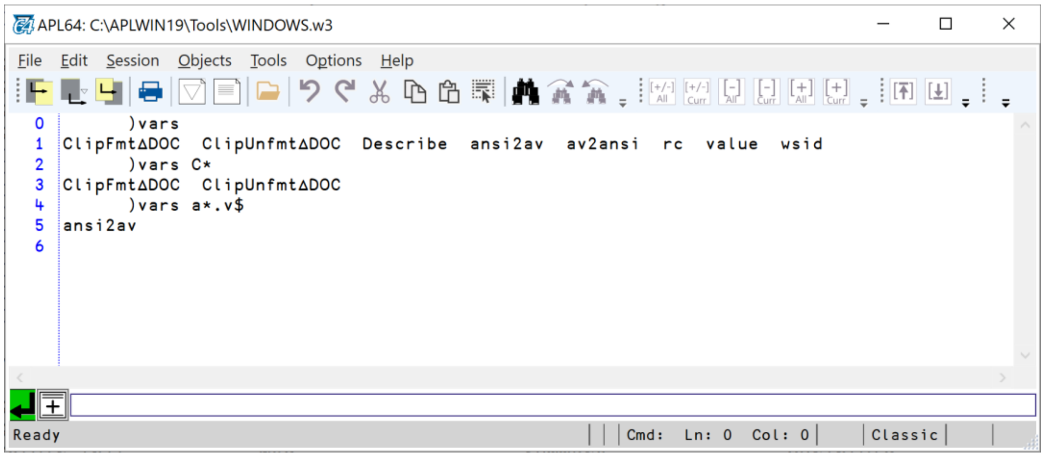
Argument:
pattern is a list of zero or more patterns, separated by blanks, for filtering the names; extended wildcard notation and regular expression (regex) filter patterns are supported;
@wsname is the name of a saved workspace that to you want to query for its list of variables.
Note: Specify an APL+Win workspace with the .w3 extension.

Effect:
Displays a list, in alphabetical order, of the variables currently in the local environment of the active workspace. Specifying the optional pattern returns the names that match the argument pattern. The pattern can contain literal characters that are valid in names such as A-Z, a-z, 0-9, _, ∆, or ⍙. In addition, a * (asterisk) can be used to denote zero or more of any character and a ? (question mark) can be used to denote any single character. Any other characters kick the phrase into extended regex notation. In this case, the entire expression is considered to be a "regular expression" with the syntax rules as described here: https://docs.microsoft.com/en-us/dotnet/standard/base-types/regular-expression-language-quick-reference. Remember that in regular expressions, a . (period) selects any character, and .* (asterisk) selects zero or more occurrences of any character, and ^ character denotes the beginning of the name and $ character denotes the end of the name. So in this case the * and ? in the argument patterns are treated as standard wildcard characters where the * means zero or more occurrences of any character and ? means 1 occurrence of any character.

With regular expression syntax, implicitly include a leading ^ character before the regex specification and a trailing $ character after it. This means that if you specify AB[CZ].*X as a regex pattern the full regular expression will actually be ^AB[CZ].*X$. The ^ character means the start of the line (but in the pattern matching context for names this means the start of a name). The $ character means the end name.

⎕nl and ⎕idlist provide a similar capability you can use under program control. The user command]vars provides a similar but more flexible capability.

Examples:



```
      )vars
ClipFmt∆DOC  ClipUnfmt∆DOC  Describe  ansi2av  av2ansi  rc  value  wsid
      )vars C*
ClipFmt∆DOC  ClipUnfmt∆DOC
      )vars a*.v$
ansi2av
```

)wload Loads a saved APL+Win workspace from disk

Purpose: Activates a saved APL+Win workspace by replacing the active workspace with a copy of a workspace stored on disk.

Syntax:)wload wsid

Argument: wsid is the workspace identifier including the .w3 extension (not optional).

Effect: Replaces the active workspace with a copy of the specified saved workspace (wsid) and displays the time and date that the workspace was saved. Once loaded, the system automatically executes the latent expression (⎕lx). In a workspace saved with a nonempty state indicator, ⎕lx could be a localized latent expression. The workspace can be in any directory. If you do not specify a directory,)wload assumes the current directory. If the system does not locate the specified workspace in the specified directory, it displays a WS NOT FOUND message. The)wload operation does not affect file ties or session-related system variables.

Examples:

```
      )wload c:\aplwin19\tools\windows.w3
"c:\aplwin19\tools\windows.w3" LAST SAVED 4/22/2019 6:54:16 PM
      >[⎕LX] Describe
This is the WINDOWS workspace distributed with the APL+Win.
It contains a variety of utility functions to help you interact
with the Microsoft Windows environment and the APL+Win Windows
Interface.

A description of each function may be obtained by entering
Explain 'fnname'.  Run "Summary" for a synopsis of all of the
functions in the workspace.

Copyright 2005-2019, APLNow LLC.
```

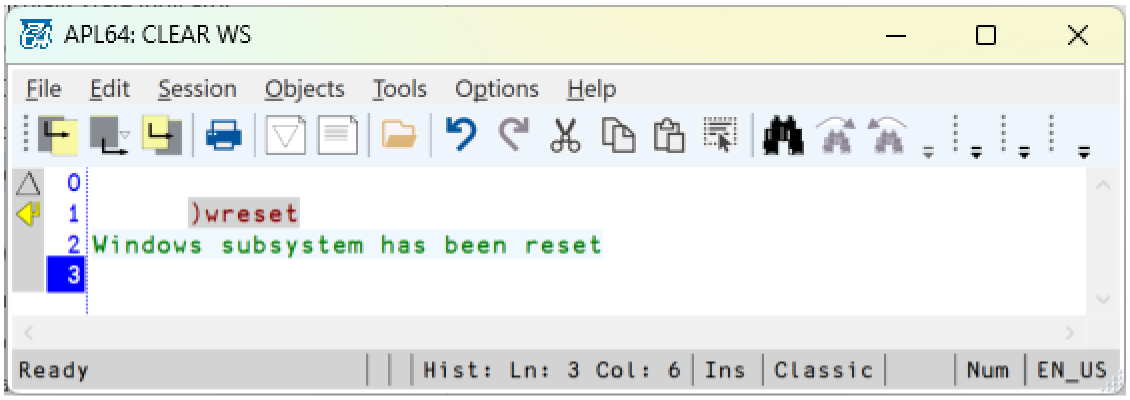
)wreset Resets Windows subsystem state

Purpose:
Resets the Windows subsystem state.

Syntax:)wreset

Effect:
)wreset resets the Windows subsystem state completely reinitialize the active ⎕wi, ⎕wcall, and 32-bit ⎕na states. Th action will also reset the SI (state indicator) state. This can be executed anytime, but especially when the link between the active APL64 session and the Windows subsystem is broken.

Examples:



```
      )wreset
Windows subsystem has been reset
```

)wslib Displays a list of workspaces

Purpose:
Lists the names of the workspaces including the .ws64 extension in a library or directory.

Syntax:
```
)wslib
)wslib dir
)wslib lib
```

Arguments:
dir is the directory.

lib is the library number.

Effect:
Lists all files that have the extension .ws64, assuming them to be workspace files, stored in the specified directory or library, even if the user has no access to them. To list all files that have the extension .w3, specify this extension prepended with the wildcard character, e.g., *.w3.

If you do not specify a library number or directory name,)flib searches the current working directory. You can specify a full path name even if you used ⎕libd to define the path as a library. The listing of names does not include the .ws64 extension.

⎕wslib provides a similar capability you can use under program control.

Examples:

```
    )wslib
GAMES.ws64   MONTHS.ws64   UTILITY.ws64
    )wslib C:\APLW\TIMEWSS
DATES.ws64
    ⎕libd '3 C:\APLW\GRFX'
    )wslib 3
GRAPH.ws64   PRINT.ws64
```

)wsid Workspace identification

Purpose:
Displays or resets the name associated with the active workspace.

Syntax:
```
)wsid
)wsid name
```

Argument:
name is the workspace identifier.

Effect:
Displays the workspace identification without changing it.

When used with name,)wsid sets the name of the active workspace to the workspace identification provided.

⎕wsid provides a similar capability you can use under program control.

)wsname Workspace identification

Purpose:
Displays or resets the full path name associated with the active workspace.

Syntax:)wsname
)wsname name

Argument:
name is the workspace identifier.

Effect:
Displays the workspace identification without changing it.

When used with name,)wsname sets the name of the active workspace to the workspace identification provided. Unlike)wsid,)wsname is not affected by library number definitions and therefore always returns a full file path name.

⎕wsname provides a similar capability you can use under program control.

)wssize Workspace size

Purpose:
Returns the size of the active workspace in bytes.

Syntax:)wssize

Effect:
This command returns 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. In this system, the value is determined by the initial workspace size specified on the command line or in a configuration file.

⎕wssize provides a similar capability you can use under program control.

Example:

```
      )wssize
IS 17053691904
```

)wxload Load a saved APL+Win workspace, suppressing execution of ⎕lx

Purpose:
Retrieves a saved APL+Win workspace without executing its latent expression.

Syntax:)wxload wsid

Argument:
wsid is the workspace identifier including the .w3 extension (not optional).

Effect:
Replaces the active workspace with the saved workspace you specify and displays the time and date

that the workspace was saved, but does not execute the latent expression (⎕lx). In a workspace saved with a nonempty state indicator, ⎕lx can be a localized latent expression.

If the system cannot locate the specified workspace, it displays a WS NOT FOUND message.

The)xload operation does not affect file ties or session-related system variables.

Example:
This example shows that)wxload did not execute ⎕lx.

```
    )wxload c:\aplwin19\tools\windows.w3
"c:\aplwin19\tools\windows.w3" LAST SAVED 2/11/2021 5:05:10 PM
```

)xload Load a saved workspace, suppressing execution of ⎕lx

Purpose:
Retrieves a saved workspace without executing its latent expression.

Syntax:)xload wsid

Argument:
wsid is the workspace identifier.

Effect:
Replaces the active workspace with the saved workspace you specify and displays the time and date that the workspace was saved, but does not execute the latent expression (⎕lx). In a workspace saved with a nonempty state indicator, ⎕lx can be a localized latent expression.

If the system cannot locate the specified workspace, it displays a WS NOT FOUND message.

The)xload operation does not affect file ties or session-related system variables.

⎕xload provides a similar capability you can use under program control.

Example:
This example shows that)xload did not execute ⎕lx.

```
    )xload MYWS
SAVED 03/11/2004 10:26:22
    ⎕lx
'BOO HOO'
```

Object Manipulation

)copy Copy from saved workspace

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

Syntax:)copy wsid
)copy wsid objlist

Arguments:
wsid is the workspace identifier;

objlist is the list of functions or variables you want to copy.

Effect:
Copies global objects from the saved workspace (wsid) into the global environment of the active workspace and displays a SAVED message with the time and date that wsid was saved. Replaces identically named global objects already in the active workspace. If you do not specify objlist,)copy copies all the APL variables and functions in the saved workspace into the active workspace.

If copying cannot be completed because an object is too large to fit into the active workspace, the system displays a NOT COPIED: message along with the names of the objects that could not be copied. If)copy cannot find an object in the specified workspace, it displays a NOT FOUND: message along with the names of the objects that could not be found. In both cases, copying continues with the remaining objects in the list.

If the free space in the active workspace is insufficient for the copy process, the system may display one of the following messages:

```
WS FULL
WS TOO LARGE
```

Copying a function copies only the source form of the function;)copy does not copy any intermediate code normally saved to improve that function's performance. During the copy process,)copy also discards all ⎕stop and ⎕trace settings in effect for a copied function.

When using)copy on a workspace file name that include one or more spaces, you can must delineate the workspace name from the object(s) using one of the two options listed below:

- append a semi-colon to the long file name (like in APL+Win)
- specify the long file name inside a pair of single or double quotes (new in APL64)

If you want to copy the functions alpha and beta from the workspace foo, you will use:

```
 )copy foo alpha beta
```

If you want to copy the function beta from the workspace named foo alpha, you will use:

```
  )copy foo alpha; beta
```

If you want to copy the workspace named foo alpha beta, you will use:

```
 )copy foo alpha beta;
```

⎕copy provides a similar capability that you can use under program control, but it copies local objects into the local environment; that is, it both copies and creates or replaces the local value of a variable that is localized within a function that is running or suspended when you invoke ⎕copy.

)edit Invokes an edit session on an object

Purpose:
Invokes an edit session where you can modify a function, character vector, character matrix, or numeric variable.

Syntax:
```
)edit object
)edit ∇object
)edit #object
```

Argument:
object 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 omitted,)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 numeric variable.

Effect:
Activates the editor for the named object. If the APL object exists, it must be either an unlocked function or a simple character or numeric variable whose rank is two or less (a scalar, vector, or matrix).

Note: The)edit command cannot be executed in a multi-row executable statement in the APL command line.

)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**.

)erase Erases functions and variables from the active workspace

Purpose:
Erases functions and variables from the active workspace.

Syntax:)erase objlist

Argument:
objlist is the list of functions or variables you want to erase.

Effect:
Erases the specified objects from the active workspace. If the system cannot erase any of them, it displays the message NOT ERASED: followed by the names of the objects that were not erased.
You can erase functions that are suspended or pending, but the system does not reclaim the storage they occupy until they complete execution or you clear the stack [see the description of)sic in this chapter].

⎕ex and ⎕erase provide a similar capability you can use under program control.

Example:

```
      )erase JANDATA TRIALFN NOSUCH
NOT ERASED: NOSUCH
```

Note: Erasing a variable in the active workspace does not remove it from the symbol table. Instead, it will have changed its definition from a value binding to a NoValue binding.

)pcopy Protected copy from a saved workspace

Purpose:
Copies APL functions and variables from a saved workspace into the active workspace provided that the copy does not replace any objects in the active workspace.

Syntax:
```
)pcopy wsid
)pcopy wsid objlist
```

Arguments:
wsid is the workspace from which to copy functions or variables.

objlist is the list of functions or variables you want to copy.

Effect:
Copies global objects from the saved workspace (wsid) into the global environment of the active workspace and displays a SAVED message. The system lists objects that do not exist in the saved workspace after it displays a NOT FOUND: message. If you do not specify any objects,)pcopy copies all of the variables and functions, but does not replace identically named objects in the workspace. Objects that)pcopy finds but does not copy are flagged with a NOT COPIED message. This may be because the workspace contains an existing object with the same name or has insufficient space to store the object. Copying continues with the remaining objects on the list.

When using)pccopy for a workspace file name that include one or more spaces, you can must delineate the workspace name from the object(s) using one of the two options listed below:

- append a semi-colon to the long file name (like in APL+Win)
- specify the long file name inside a pair of single or double quotes (new in APL64)

If you want to copy the functions alpha and beta from the workspace foo, you will use:

```
)pcopy foo alpha beta
```

If you want to copy the function beta from the workspace named foo alpha, you will use:

```
)pcopy foo alpha; beta
```

If you want to copy the workspace named foo alpha beta, you will use:

```
)pcopy foo alpha beta;
```

⎕pcopy provides a similar capability you can use under program control, but it both copies and creates local objects into the local environment. That is, the system uses the local value for a variable if the variable is localized in the source workspace, and it creates the local value in the target workspace for a variable that is localized but undefined within a function that is running or suspended when you invoke ⎕pcopy.

)wcopy Copy from saved APL+Win workspace

Purpose:
Copies APL functions and variables from a saved APL+Win workspace to the active workspace.

Syntax:

```
)wcopy wsid
)wcopy wsid objlist
```

Arguments:
wsid is the APL+Win workspace identifier including the .w3 extension (not optional).

objlist is the list of functions or variables you want to copy.

Effect:
Copies global objects from the saved workspace (wsid) into the global environment of the active workspace and displays a SAVED message with the time and date that wsid was saved. Replaces identically named global objects already in the active workspace. If you do not specify objlist,)wcopy copies all the APL variables and functions in the saved workspace into the active workspace.

If copying cannot be completed because an object is too large to fit into the active workspace, the system displays a NOT COPIED: message along with the names of the objects that could not be copied. If)wcopy cannot find an object in the specified workspace, it displays a NOT FOUND: message along with the names of the objects that could not be found. In both cases, copying continues with the remaining objects in the list.

If the free space in the active workspace is insufficient for the copy process, the system may display one of the following messages:

WS FULL
WS TOO LARGE

Copying a function copies only the source form of the function;)wcopy does not copy any intermediate code normally saved to improve that function's performance. During the copy process,)wcopy also discards all ⎕stop and ⎕trace settings in effect for a copied function.

When using)wcopy on a workspace file name that include one or more spaces, you can must delineate the workspace name from the object(s) using one of the two options listed below:

- append a semi-colon to the long file name (like in APL+Win)
- specify the long file name inside a pair of single or double quotes (new in APL64)

If you want to copy the functions alpha and beta from the workspace foo, you will use:

```
)wcopy foo.w3 alpha beta
```

If you want to copy the function beta from the workspace named foo alpha, you will use:

```
)wcopy foo alpha.w3; beta
```

If you want to copy the workspace named foo alpha beta, you will use:

```
)wcopy foo alpha beta.w3;
```

)wpcopy Protected Copy from Saved APL+Win workspace

Purpose:
Copies APL functions and variables from a saved APL+Win workspace into the active workspace provided that the copy does not replace any objects in the active workspace.

Syntax:
```
)wpcopy wsid
)wpcopy wsid objlist
```

Arguments:
wsid is the APL+Win workspace including the .w3 extension (not optional) from which to copy functions or variables.

objlist is the list of functions or variables you want to copy.

Effect:
Copies global objects from the saved workspace (wsid) into the global environment of the active workspace and displays a SAVED message. The system lists objects that do not exist in the saved workspace after it displays a NOT FOUND: message. If you do not specify any objects,)wpcopy copies all of the variables and functions, but does not replace identically named objects in the workspace. Objects that)pcopy finds but does not copy are flagged with a NOT COPIED message. This may be because the workspace contains an existing object with the same name or has insufficient space to store the object. Copying continues with the remaining objects on the list.

When using)wpccopy for a workspace file name that include one or more spaces, you can must delineate the workspace name from the object(s) using one of the two options listed below:

- append a semi-colon to the long file name (like in APL+Win)
- specify the long file name inside a pair of single or double quotes (new in APL64)

If you want to copy the functions alpha and beta from the workspace foo, you will use:

```
)wpcopy foo.w3 alpha beta
```

If you want to copy the function beta from the workspace named foo alpha, you will use:

```
)wpcopy foo alpha.w3; beta
```

If you want to copy the workspace named foo alpha beta, you will use:

```
)wpcopy foo alpha beta.w3;
```

Operating Environment

)cmd Executes a DOS command or invokes the DOS shell to execute a command

Purpose:
Executes a DOS command.

Syntax:
```
)cmd
)cmd command
```

Argument:
command is the DOS command you want to execute.

Effect:
)cmd creates a new window in which DOS runs the command. Output that)cmd produces is not captured in the APL session.

If you specify the command in the APL statement, when DOS finishes executing the command, the system destroys the window immediately; usually before you can read the result. You can capture the output by directing it to a file (for example,)cmd dir > tmp.txt) and then reading the file with ⎕nread. If you do not specify command, you enter a DOS window, and DOS waits for you to enter a command. Any output that results appear in the DOS window. Type exit to return to the APL history.

The system loads the DOS command processor, normally cmd.exe, from disk. This file must be present in the folder specified by the DOS COMSPEC= variable. To see the current setting, execute SET from DOS. You cannot usefully set the COMSPEC= parameter during a)cmd session because the setting lasts only until you exit from DOS back to APL; to be effective, you must set this parameter from DOS before you start Windows.

⎕cmd provides a similar capability that you can use under program control.

Notes:
Normally, APL is blocked while DOS runs the command. If you have the Multiple Execution choice on the Debug menu checked, neither APL nor DOS is blocked; multiple execution can lead to surprising results.

Examples:

Leave APL; list the current DOS folder; and return to APL.

```
    )cmd
C>dir
    [→Output suppressed for this example→]
C>exit
```

Format a disk and return to APL.

```
    )cmd FORMAT B:
```

)evlevel Sets the evolution level for the active workspace

Purpose:
Displays or sets the evolution level for changed language features in APL.

Syntax:
```
)evlevel
)evlevel n
```

Argument:
n is the evolution level you want to set; it can be 0, 1, or 2 in this version.

Effect:
Displays or sets the evolution level. APL operates by default at Evolution Level 2. Certain functions and language features had different behaviors at Evolution Level 1. If you are converting applications from older systems, you can use this command to contrast the new and old behavior of changed features during a session, and to search your code interactively for dependencies on old behavior. This setting is workspace relative. It is saved with the workspace on a)Save command and loaded with the workspace on a)Load command. The)Clear command resets it to the default value.

Note: The)evlevel command cannot be executed unless the)SI level is clear. If functions are suspended, you cannot change their settings.

If you are not converting applications, you should always operate at Evolution Level 2.

This version of APL64 allows the interpreter to behave at three different evolution levels.

- At Evolution Level 2, the interpreter uses the new behavior of a changed feature.
- At Evolution Level 1, the interpreter uses the old behavior of a changed feature.
- At Evolution Level 0, the interpreter signals an EVOLUTION ERROR when you use a changed feature. Running existing applications at this evolution level allows you to search your code interactively for dependencies on old behavior and makes it easier to find the places where you must change your code.

However, new in APL is that the evolution level is stored in the APL saved workspace. The current evolution level is also reset to 2 when clearing the workspace.

Example:
The example below displays a nested matrix containing both character and numeric values. At Evolution Level 1, monadic epsilon (∊) displays the old behavior and acts as the type function. When you set the evolution level to 0, monadic epsilon (∊) produces an EVOLUTION ERROR. If you set the evolution level to 2, which means that epsilon (∊) uses the new behavior, monadic epsilon (∊) acts as the Enlist function. Enlist rearranges an array into a vector, regardless of the array's structure.

```
      V←3 2⍴'MARY' (2 2⍴4) (⍳5) 0 (2 2⍴'HA') 'JO'
      ]DISPLAY V
.→-------------------.
↓ .→---.      .→---. |
| |MARY|      ↓ 4 4| |
| '----'      | 4 4| |
|             '~---' |
| .→--------.        |
| |1 2 3 4 5|  0     |
| '~--------'        |
| .→-.        .→-.   |
| ↓HA|        |JO|   |
| |HA|        '--'   |
| '--'               |
'∊-------------------'
      )evlevel
IS 1
      ]DISPLAY ∊V
.→-------------------.
```

```
↓ .→---.      .→---. |
| |    |      ↓ 0 0| |
| '----'      | 0 0| |
|             '~---' |
| .→--------.        |
| |0 0 0 0 0|  0     |
| '~--------'        |
| .→-.        .→-.   |
| ↓  |        |  |   |
| |  |        '--'   |
| '--'               |
'∊-------------------'
      )evlevel 0
WAS 1
NOW 0
      ∊V
EVOLUTION ERROR
      ∊V
      ^

      )evlevel 2
WAS 0
NOW 2
      ∊V
MARY 4 4 4 4 1 2 3 4 5 0 HAHAJO
      ]DISPLAY ∊V
.→------------------------------.
|MARY 4 4 4 4 1 2 3 4 5 0 HAHAJO|
'+------------------------------'
```

)off Ends the current APL session

Purpose:
Ends the current APL session.

Syntax:)off
)off YES
)off NO

Effect:
Terminates an APL session and returns you to the operating system. The system does not preserve the contents of the active workspace and automatically unties any tied files. If you enter)off when you have an edit session pending that has changed, the system displays:
Edit sessions have changed; abandon changes?

To end your APL session without saving your changes, select OK; the system discards all unsaved changes in your edit session(s). To save your changes, select Cancel; the system returns you to the current APL session.

You can bypass the system prompt using)off YES, which terminates the current APL session regardless of any pending conditions.)off NO terminates the session only if no active edit sessions have changed, you are not in quad input mode, and, if APL is an ActiveX server, does not have a client connected.

⎕sa provides a similar capability you can use under program control (⎕sa←'OFF' ⋄ →).

)tempdir Temporary Directory (obsolete)

This system command is deprecated in APL.