LOGOS

Reference Manual

REUTERS

LOGOS

Reference Manual

REUTERS

ACKNOWLEDGEMENTS
LOGOS is a trademark of Reuter:file Ltd.

Printed in Canada
Publication code: 1311 9008 E1
Version: 2.0

CONTENTS

HOW TO USE THIS MANUAL

This book is a reference guide to LOGOS. It is intended to be used for checking the definition of a command or a specific LOGOS component, but not for learning how to use the system.

Conventions Used In This Document

The following conventions are used in this manual.

alias	Commands you type as shown are in lowercase italic font.
save **pathname** +*value*=**text**	Values you must supply in order to execute a command are shown in lowercase bold font.
save objects	Values for which you can supply a single item or a list of items are indicated by the plural form of the noun.
modifiers	New terms appear in the text in bold face.
Ctrl-S	Keys on your keyboard that you must press simultaneously are separated by a hyphen (-).
F1	Function keys are shown as a capital F suffixed with the number of the function key you must press.
new password: ***apples***	Values entered by the user in sample sessions appear in bold face, italic font.
[=*pn, type,...*]	Lists that contain several items and cannot be shown in full are shown with trailing dots (...) and explained later in the section.
■	Instructions that are buried within a section of text are flagged with a black box.

Related Manuals

For more information on LOGOS, see the following manuals:

LOGOS Pocket Reference (Publication code: 1313 9008 E1)

LOGOS User's Guide (Publication code: 1310 9008 E1)

LOGOS COMMANDS

About LOGOS Commands

Command results are always character vectors, with embedded carriage returns where the display spans more than one line. Normally, the result of a command is displayed by default. However, some commands (such as *delete* and *import*) do not display their result. You can use ⎕← on the command line to see any result, or assignment to a variable to save it.

LOGOS Commands: ∇

The ∇ command opens LOGOS objects for editing. This command is synonymous with the *edit* command. See the section on the *edit* command for details.

LOGOS Commands: *?*

The *?* command displays information about LOGOS commands.

Syntax

`?[?][command]`

?	Displays the names of all LOGOS commands.
??	Displays syntax information about all LOGOS commands.
*?***command**	Displays syntax information about **command**.
*??***command**	Displays general, syntax, and usage information about **command**.

Result

The command returns the requested information as the result.

Examples

```
υ??
```

Displays syntax information for every LOGOS command.

```
υ ?display
```

Displays syntax information for the *display* command.

```
υ ??display
```

Displays general, syntax, and usage information for the *display* command.

LOGOS Commands: ⍎

The ⍎ command executes APL expressions from within LOGOS.

Syntax

⍎ [**expression**]

expression is the APL expression to be executed.

Result

The command returns the result of the executed expression.

Usage

Used within parentheses as an argument, ⍎ provides a means of interpolating APL values into LOGOS command lines.

If you omit **expression**, you are prompted for an expression with the prompt ⍎⍞. If you enter an empty line, you are prompted again. If you enter an expression, it is executed and you are prompted for another line. To terminate the ⍎ command, press the space bar and then press Enter.

The ⍎ command does not recognize the LOGOS delimiters as special characters. For example, ⍎3 *+f* is treated as 3+*f*. *+f* is not treated as a modifier.

Examples

```
∪ ⍎⎕ts
1989 8 17 12 12 56 340
∪ a←list temp.dir ∪ ⍎a ⎕append 10
∪ ⍎
⍎⍞ ⎕size 10
1 6 12064 100992
⍎⍞ 15000 ⎕resize 10
⍎⍞ <Space> <Return>
∪
```

```
∪ x← list .public.util.fr?*
∪ edit (⍎x)
```

LOGOS Commands: ⍕

The ⍕ command aids in the preparation of arguments for certain LOGOS commands, such as *send* and *exit.*

Syntax

⍕ expression

expression is an APL expression.

Result

The command returns the argument as the result.

Usage

The ⍕ command provides a means of passing APL expressions which contain certain special characters to other commands, without processing the special characters. This command has *unprocessed scope* and does not process plus signs, quotes, parentheses, backslashes, or braces in its argument. Thus, the result of the command is the value of its argument, and no characters in the argument are treated as having any special properties. For example:

```
⍕'abc' {def} +ghi
'abc' {def} +ghi
```

Quotes, braces, and plus signs are ignored.

Examples

The ⍕ command is often used with the *send* command, which has ordinary *long* scope and processes quotes, parentheses, and other special characters. In an expression like this one:

```
'can''t continue' ◇ p←v\m
```

you would have to place quotes around the entire expression to prevent LOGOS from removing a level of quotes and attempting to process \m as a reference to a keyword. You would also have to double the existing quotes, as follows:

```
u send '''can''''t continue'' ◇ p←v\m'
```

Using the ⍕ command, you can get the same result by typing:

```
u send (⍕'can''t continue' ◇ p←v\m)
```

LOGOS Commands: *alias*

The *alias* command allows you to adopt a new alias.

Syntax

```
a[lias] alias [password]
        [+newpass[=password]]
```

alias Is the new alias. *It must be an alias you are authorized to use.*

password Is the password required to gain access to that alias. If you omit **password** but one is required, you are prompted for the password. When the new alias is the same as your current one, no password is required unless you are also changing the password with *+newpass*.

+newpass[=**password**]
Specifies a new password to alias. If you omit =**password**, you are prompted for the password. If you specify *+newpass=*, the current password is removed.

Result

When you use *alias* with no argument, the result is the current alias. When you provide an argument, no result is returned.

Usage

Changing the alias to your primary alias refreshes your user profile parameters, such as your working directories and command separator.

Examples

ᵁ *alias (alias)*

Refreshes the profile parameters of the current alias.

ᵁ *alias john*

Switches your alias to *john*, and establishes *john*'s profile. If *john* was already your alias, all profile parameters are refreshed.

```
u alias john yellow
```

Switches your alias to *john*, which has a password of *yellow*. If a password is required and none is supplied, LOGOS prompts for it before it prompts for the new password.

```
u alias john +newpass
new password:
repeat new password:
```

Switches your alias to *john*, and sets a new password. The new password is requested twice, to reduce the chance of a typing error. If *john* already has a password and none is specified, LOGOS prompts for it.

```
u alias john yellow +newpass=
```

Specifies no password for alias *john*.

LOGOS Commands: *build*

The *build* command generates a collection of objects known as a *cluster*. Once generated, the cluster can be placed into one or more end environments. The objects which compose the cluster can be dispersed in the active workspace, or placed as a package into the workspace, a component of a file, a pathname in the LOGOS file system, or a node within a LOGOS paging area.

The *build* command can perform static tree analysis on the objects specified in the root pathname argument.

If the destination of the cluster is a LOGOS paging area, then:

- The actions specified by the various *build* commands are deferred until a *filesave* command is issued.
- Exclusion by inference takes place (by default). Tree analysis for a particular node stops when it reaches the name of one of the other nodes in the paging area, and the contents of this subordinate node are not included in the calling node.

Syntax

```
bu[ild] [destination [pathnames]]
        [+audit=filename]
        [+compile=directives]
        [+depth[=all|n]]
        [+exclude=[,|/]names]
        [+file=filename[cn]]
        [+inference=yes|no]
        [+keepin=n]
        [+lock=passnumber]
        [+overwrite[=audit|,|buffer|,|dest]]
        [+protect]
        [+recursive[=1|2|all]]
        [+size=n]
        [+task[=task]]
        [+update[=nodes]]
        [+workdir=pathnames]
```

destination — Specifies both the destination of the built object, and its form. If you omit **destination**, you must also omit **pathnames**. If you omit both **destination** and **pathnames**, the *build* command sets new default values for any modifiers which were specified; see the usage note, below.

destination may be any of the following:

node	Builds a node of a paging file and names it **node**. (Any undelimited name that meets the rules for APL identifiers is considered a node name.) Using ⋆ instead of a node name indicates the base node, whose objects are always resident in the workspace. Several nodes may be built with one use of *build*, if the names are separated by blanks and enclosed in quotes.
pathname.	Builds a cluster and puts it in **pathname** in the LOGOS file system. (Any name that includes a dot (.) is considered a pathname.)
<name>	Builds a package and places it in the active workspace with the name **name**. (Any name delimited by angle brackets (<>) that meets the rules for APL identifiers is considered a package name.) If *+task* is specified or a task name has been set in the *environment task* parameter, the package is placed into the workspace controlled by the auxiliary task.
<>	Builds a cluster and disperses it in the active workspace. If *+task* is specified or a task name has been set in the *environment task* parameter, the cluster is dispersed into the workspace controlled by the auxiliary task.
<component-number>	Builds a package and places it in a specific component of the file indicated by *+file*. If the destination component number in the specified file is beyond the end of the file, the appropriate number of pad components is appended to the end of the file, each containing the text *unused*.
component-number	If the result of *build* is a single variable, places the value of the variable, not the variable packaged, into the specified component. If the result is a package, it is placed into the component without being re-packaged. If the result is a cluster, it is placed into the component as a package. If the destination component number is beyond the end of the specified file, the appropriate number of pad components is appended to the end of the file, each containing the text *unused*.

pathnames Is the pathname of the root function of the node, cluster, or package. **pathnames** must be the LOGOS pathname of a function, script, or variable. You can specify more than one pathname in a single *build* command. The first one specified is the *primary root* of the node, and those following are *ancillary roots*. (If you subsequently build a shell around this node, the primary root is the one used to determine the default characteristics of the shell.) The calling trees are analyzed in the order of their specification.

If you omit **pathnames**, it is assumed to be the same as **destination**. **destination** in this case must specify a node in a paging file.

+audit=**filename** Identifies an audit file to contain information about this generation. If the audit file you specify does not exist, LOGOS creates it. If you omit *+audit*, the file specified in *environment audit* is used. If no file is specified there, no audit file is used.

+compile=**directives**

Specifies compilation directives to be applied to objects fetched from the LOGOS file system. If you omit *+compile*, only compilation directives specified in the objects or latent in your environment are used.

+depth[=*all* | **n**] Specifies the number of levels in calling trees to analyze. **n** may be any non-negative integer. *+depth* (without a value), *+depth=all*, and *+depth=0* specify that no limit is to be applied to calling-tree analysis. *+depth=1* specifies that calling trees are not to be analyzed; consequently, the built object will contain only the root object. *+depth=2* specifies that calling tree analysis is to be performed on only one level down from the root. Use a value for **n** appropriate to the level of analysis you want. If you omit *+depth*, *+depth=1* is assumed.

+exclude=[, | /]**names**

Specifies objects found in tree analysis which are to be excluded from the node, package, or cluster generated by the *build* command. If you omit *+exclude*, all objects found in calling-tree analysis are included.

,	Adds the objects specified in **names** to any exclusion list established by a global *build* command. If you omit both , and / , **names** forms the complete exclusion list.
/	Removes the objects specified in **names** from any exclusion list established by a global *build* command. If you omit both , and /, **names** forms the complete exclusion list.
names	Is a list of objects forming the exclusion list, or to be added to or removed from the global exclusion list. Separate names by blanks.

+file=**filename** [**cn**]

Identifies the destination file, and optionally an area in the file. If the destination file you specify does not exist, LOGOS creates it. *+file* is relevant only when the destination is a component of a file or a node in a paging area. When building a paging area, you can omit *+file* from the *build* command if you specify it in the *filesave* command. **cn** indicates the starting component number to be used for the paging area. If you omit **cn**, the paging area starts at the first available component of **filename**.

+inference=yes | *no*

Indicates whether nodes referenced by the calling tree analysis used to generate this node are to be excluded by inference.

+inference=yes indicates that called nodes are to be excluded from this node. *+inference=no* indicates that called nodes are to be included in this node. If you omit *+inference*, *+inference=yes* is assumed.

+keepin=**n** Specifies the node's keep-in priority. **n** can be any non-negative integer. The higher the keep-in value assigned to a node, the less likely it is to be paged out. If you omit *+keepin*, *+keepin=0* is assumed (and nodes with a keep-in priority of 0 will be the first to be paged out). Note that if *+keepin* is not used on any nodes, then priority does not enter into the page-out heuristics.

+lock=**passnumber**

Specifies the destination file's passnumber. If *+lock* is not specified, no passnumber is used for the file.

+overwrite[*=audit* | , | *buffer* | , | *dest*]

Specifies one or more working areas to be overwritten.

+overwrite=audit

Generates a new audit record.

+overwrite=buffer

Overwrites the internal build buffer. The build buffer is an internally maintained buffer containing page file information to be used by the *filesave* command. It should only be overwritten if a paging file generation has terminated abnormally (and should definitely be overwritten in that case).

+overwrite=dest

Overwrites the destination paging area.

If you specify *+overwrite* or *+overwrite=audit*, *buffer*, *dest* all working areas are overwritten. If you omit *+overwrite*, no working areas are overwritten.

+protect Protects objects in the workspace from being overwritten if the destination is <**name**> or <>.

+recursive[*=1* | *2* | *all*]

Controls iteration through directory levels encountered. This iteration only occurs for directories which are specified as root pathnames. If you do not specify *+recursive*, *+recursive=2* is assumed.

+recursive=all Indicates that the named level and all its descendants are to be used. Also *+recursive=0* and *+recursive* with no value.

+recursive=1	Signifies only the named level.
+recursive=2	Signifies the direct descendants of the named level, excluding the named level.

+size=**n** Specifies the maximum size, in bytes, of any node. **n** can be any positive integer. When a node's size exceeds the value of this modifier, LOGOS automatically splits its contents into two or more separate nodes. If you omit *+size*, node size is not limited, except by workspace size.

+task[=**task**] Specifies the task whose active workspace is to receive the objects in the destination <**name**> or <>. If *+task* is specified without an argument, the default auxiliary task *aux* is assumed. If you don't specify *+task*, the task specified in the environment task parameter is assumed.

+update[=**nodes**]
Specifies which nodes of the paging area are to be updated, avoiding regeneration of the entire paging area. Specifying *+update* overrides *+overwrite*=**dest**, so only the specified destination node is saved.

+workdir=**pathnames**
Specifies new working directories for this command. The evaluation of any paths implied by the arguments to the *build* command or its calling tree analysis is performed under the new working directories. If you omit *+workdir*, your current working directories are used.

Result

The command returns a message informing you of the number of objects clustered, and the end environments in which they were placed.

Usage

A *paging area* is a set of contiguous components in a SHARP APL file containing a series of named packages which can be retrieved with the LOGOS paging utilities. This file can contain anything else -- including other paging areas.

Each paging area includes a header and a *base node*. The base node, whose nodename is ⋆, is the system kernel. It is always paged into the application's active workspace and is at the top of the paging tree.

The node name of a non-base node usually matches the name of the root function, but it does not have to. For example, terminal-driver nodes for a repertoire of terminals will usually have names reflecting the driven terminals; the functions contained by the nodes will usually have the same names and syntax as functions in other terminals' nodes.

If you issue a *build* command without specifying **destination** or **pathnames**, no cluster is built but default values are set for the modifiers given.

The *build* command allows you to specify multiple destinations. You can, for example, generate four separate pages by entering *build 'a b c d'*. When generating pages, you may not specify another type of end environment in the same *build* statement. However, you can mix the other destination types. For example:

build '10 .klh.util.vtom' roots

The *build* command runs better, faster, and with reduced chance of workspace storage problems if you specify an audit file. Tree analysis requires the use of information which can be moved to the audit file when one is specified. Audit files also allow you to use *distribute* to make incremental modifications to any end environments you have created using them.

Examples

```
∪ build <> input +depth=all +exclude=Input∆02
```

Deposits the function *input* and all functions and global variables it needs, except *Input∆02*, in the active workspace.

```
∪ build <⎕sp> input +depth=all
∪ ⍎⍴⎕pnames ⎕sp
7 13
```

Builds the function *input* and all functions and global variables it needs into a package, and puts the package into ⎕*sp*.

```
∪ build +audit=1234567 genaudit +compile=x,p +depth=all +workdir=.john.terms.x
```

Sets global default values for the *+audit*, *+compile*, *+depth*, and *+workdir* modifiers. Subsequent *build* commands in this session do not need to specify these modifiers again to use these values. However, these default values can be overridden by specifying a new value for any of those modifiers with a specific *build* command.

```
∪ build test.input input
```

Builds a cluster from function *input* (and all functions and global variables it needs because of the global *+depth* set in the previous example), and puts it in path *test.input.*

```
∪ (display test.input +nopathname)
```

test.input is the cluster built in the previous example. The display of a cluster is the *build* command which was used to generate it. In this example, the expression in parentheses is evaluated and the cluster is regenerated.

```
∪ build <10> input output +file=termdrivers
```

Builds packages from root functions *input* and *output,* and puts them in component 10 of the user's file *termdrivers.*

```
∪ save .klh.misc.term +value=hds108
.klh.misc.term[v1]
∪ build 1 .klh.misc.term +file=1234567 temp
1 variable placed into component 1 of file 1234567 temp
∪ ⍎⎕read 10 1
hds108
```

Places the value of *.klh.misc.term,* unpackaged, into component 1 of the file *1234567 temp.*

```
u build <1> .klh.misc.term +file=1234567 temp
1 clustered object placed into component 1 of file 1234567 temp
u ♠⎕read 10 1
**package**
u ♠⎕pnames ⎕read 10 1
term
```

Places a package containing *.klh.misc.term* into the same component.

```
u build ibm3279 input +overwrite
u filesave +file=1234567 term page 40
```

Rebuilds node *ibm3279* from root *input*, first clearing the contents of the various working areas. The *filesave* command supplies the name of the paging file and area where the node will be stored.

LOGOS Commands: *calls*

The *calls* command performs static tree analysis on a list of root pathnames. This command is very much like *build*, but computes only the names resulting from tree analysis, rather than the objects themselves.

Syntax

call[*s*] **pathnames**
  [+*compile*=**directives**]
  [+*depth*[=*all* | **n**]]
  [+*exclude*=**names**]
  [+*notfound*]
  [+*pathnames*]
  [+*workdir*=**pathnames**]

pathnames Is the pathnames on which tree analysis is to be performed. Separate the pathnames you enter with spaces.

+*compile*=**directives**
Specifies compilation directives to be applied to the objects. If you omit +*compile*, only compilation directives in the objects, or in your environment, are applied.

+*depth*[=*all* | **n**] Specifies the maximum number of levels to be examined in calling tree analysis. **n** may be any non-negative integer. +*depth* (without a value), +*depth=0* or +*depth=all* specify that no limit is to be applied to calling tree analysis. +*depth=1* specifies that calling trees are not to be analyzed. +*depth=2* specifies that calling tree analysis is to be performed on only one level down from the root. Use a value for **n** appropriate to the level of analysis you want. If you do not specify +*depth*, +*depth=1* is assumed.

+*exclude*=**names**
Specifies objects which are to be excluded from tree analysis.

+*notfound* Causes the result to consist of a list of names which were not found during calling tree analysis, rather than those that were.

+*pathnames* Causes the result to include the full pathnames (along with type and version number) of objects found during tree analysis, rather than just the object names.

+*workdir*=**pathnames**
Specifies new working directories for this command. If +*workdir* is not specified, the working directories previously set in your session or specified in your LOGOS environment are assumed.

Result

The command returns the object names which are either directly or indirectly referenced by the root pathname (or if *+notfound* is selected, returns the names of those not found).

Usage

Because *calls* is a tree analyzer, the only compilation directives that are likely to be useful in conjunction with the command are those which can introduce or remove code fragments (such as *a*, *c*, *e*, *i*, or *z*).

Examples

```
∪ cl←calls ask +depth
not found: lastinput
∪ ⍎cl
ask
cmdtable
qprime
validate
```

Returns the objects referenced either directly or indirectly by the *ask* function as a result.

```
∪ calls ask +depth=2 +exclude=lastinput
ask
qprime
validate
```

Includes only the objects directly referenced by the *ask* function in the result (due to the *+depth* modifier) and excludes the *lastinput* object from tree analysis.

```
∪ calls ask +depth +notfound
lastinput
```

Returns the list of objects which could not be found as a result of tree analysis.

```
ᵁ calls ask +depth +pathnames +workdir=john.sys.tables .klh.src.util
.klh.src.util.ask[f1]
.john.sys.tables.cmdtable[v2]
.klh.src.util.qprime[f1]
.klh.src.util.validate[f1]
```

Returns the full pathname of the objects found and their types and version numbers.

LOGOS Commands: *cmddir*

The *cmddir* command establishes or inquires about your command directories. Command directories specify where LOGOS will look for a script when you use its name as a command but have not qualified it from the root of the file system.

The command directory in effect when you are first enrolled in LOGOS is *.public.logos.cmds.*

Syntax

```
c[mddir] [pathnames]
        [+reset]
```

pathnames	Is a list of pathnames to be established as command directories. If you omit **pathnames,** your command directories are not altered.
+reset	Resets the command directory to the value saved in your profile. This modifier is only useful when the command is used without an argument.

Result

The command returns the command directories in effect after the operation completes as the result.

Usage

Command directories are searched for scripts in the same order you specify them. Use *environment +profile* to save your command directories in your profile.

Examples

```
u cmddir
.public.logos.cmds
```

Displays the current command directories.

```
u cmddir (cmddir) .sys.util.cmds
.public.logos.cmds .sys.util.cmds
```

Respecifies the command directories by adding a second entry after the existing one.

```
u cmddir +reset
.public.logos.cmds
```

Resets the command directory to the value stored in your profile.

LOGOS Commands: *compare*

The compare command compares two versions of the same object or directory, or two distinct objects or directories.

Syntax

```
com[pare] primaries [secondaries]
        [+attributes[=c|d|j|n|s|t]]
        [+compile[=directives]]
        [+display]
        [+extended]
        [+flags[=b|l|s]]
        [+interleave]
        [+lines]
        [+match]
        [+recursive[=1|2|all]]
        [+show]
```

primaries Is a list of pathnames (objects and directories) to be used as the reference in performing the comparison. **primaries** must contain at least one object.

secondaries Is a list of pathnames (objects or directories) which are compared against the **primaries.**

+attributes[=*c* | *d* | *j* | *n* | *s* | *t*]

Specifies the object attributes to compare. Valid attributes are:

c	Compilation directives
d	Documentation
j	Journal
n	Note
s	Source
t	Tag

If you specify *+attributes* with no argument, all object attributes are assumed.

+compile[=**directives**]

Specifies compilation directives to be applied to a function's or a variable's source before the comparison. If you omit an argument, the source is compiled using the global compilation directives, merged with those saved in the object's *c* attribute.

+display	Displays altered lines from both the primary and the secondary objects. To distinguish between added, deleted, and changed lines, line numbers are prefixed with the characters +, −, or ≠, respectively. Where a changed line occupies a different position in the two objects, both line numbers are given. If *+match* is specified, only lines that have not been altered are displayed, marked by the character =.
+extended	During a directory compare, *+extended* prints the names of objects added to and deleted from the primary directory. To distinguish between added, deleted, and changed paths, names are prefixed with the characters +, −, ≠, respectively.
+flags[=*b*|*l*|*s*]	Controls the behaviour of the command. Valid flags are:

b	Ignores trailing blanks in character objects.
l	Ignores differences in line labels. This works as long as the spellings are used consistently within each program being compared. This is particularly useful when comparing different versions of a program, where one has had its line labels renamed using the editor *resequence* command.
s	Performs a syntactic/semantic level comparison. LOGOS converts both the primary and secondary objects into a canonical format which:

- ignores comments
- treats statement separation using diamonds the same as line breaks
- disregards the spelling of names local to the function being compared. The comparison is based upon consistent use of a name set within each function, and the name sets across functions being compared need not be the same.

These rules allow functions that have been compiled using any combination of decommenting (*x*), diamondization (*d*), local renaming (*r*), or that have been changed with the editor *sort* command, to be compared meaningfully to their sources. Therefore, the two functions compared using *+flags=s* may appear different visually, but compare identically because they represent the same APL code. Specifying *+flags* without an argument assumes all flags.

For example:

```
    ∇ fn1  a;b              ∇ fn2  b;a
[1] b  c  0             [1] a  c  0
[2] a                   [2] a
    ∇                       ∇
```

If *+f=s* is not set, lines 0 and 1 do not match and line 2 does. If *+f=s* is set, the opposite is true: lines 0 and 1 match and line 2, although visually identical, does not match.

+interleave Produces output similar to *+display*, but the lines of each object are interleaved for easier visual comparison of differences.

+lines Displays the line numbers of differing lines with respect to the secondary object. To distinguish between added, deleted and changed lines, line numbers are prefixed with the characters +, −, or ≠, respectively. If *+match* is specified, only lines that have not been altered are displayed, marked by the character =.

+match Reports information on equal objects or, if *+display* or *+lines* is also specified, on equal lines of objects.

+recursive [=*1* | *2* | *all*]
Enumerates the contents of any directories below the path you specify. The argument to this modifier may be *1*, *2*, or *all*, signifying only the named level of the hierarchy, its direct descendants but excluding the named level, or the named level and all it descendants, respectively.

If you do not specify this modifier, *+recursive=2* is assumed. If you specify *+recursive* without a value, *+recursive=all* is assumed.

+show Displays the changed lines and indicates where changes begin on a line. If there is only one change on the line, a caret (^) points to the change. If there is more than one change, an angle bracket (>) points to where the changes begin on the line. If neither *+display* or *+interleave* is specified, *+interleave* is assumed.

If neither *+display, +interleave, +lines*, nor *+show* is selected and you are comparing objects, *compare* returns the primary pathnames and version numbers of differing objects. If you are comparing directories, *compare* also returns added objects (those in the primary but not the secondary). Both added and deleted pathnames are buffered until the last object in the directory is compared, and then they are listed.

Result

The command returns the differences between the items you are comparing. Entries in the primary list that do not have matching objects in the secondary list are reported as *not found in secondaries*. Entries in the secondary list that are not found in the primary list are ignored.

Examples

ᵁ *compare* **object**

Compares an object with the previous version of the same object. This command is equivalent to:

ᵁ *compare* **object**[*0*] **object**[¯*1*]

ᵁ *compare* **object1 object2**

Compares one object to another. In this case, only one object or directory can be specified for both primary and secondary arguments. The object types must match even if the names do not.

ᵁ *compare* **directory**

Compares each object in a directory to its previous version. This command is functionally equivalent to:

ᵁ *compare* **directory**.*?**[*0*] **directory**.*?**[¯*1*]

The output, however, takes a different form.

ᵁ *compare* **directory** *+recursive*

Compares all of the objects in a hierarchy to their previous versions.

ᵘ *compare* **directory1 directory2**

Compares two hierarchies. The directories are compared as well as the objects within the directories. If used with the *+recursive* modifier, this command compares entire structures.

ᵘ *compare* **object directory**

Compares an object with any object of the same name in a particular directory.

LOGOS Commands: *copy*

The *copy* command duplicates objects or hierarchies.

Syntax

```
cop[y] pathnames destination
       [+in]
       [+makedir]
       [+override]
       [+protect]
       [+versions[=n]]
```

pathnames	Is a list of paths to be copied.
destination	Specifies the directory into which the paths are to be copied.
+in	Cancels out-registration of destination paths, if you set the registration.
+makedir	Allows the creation of directories in the destination path. If you omit *+makedir,* new directories are not created.
+override	Overrides the registration of registered paths (set by other users) which are being copied over in the destination hierarchy. This also has the effect of registering the paths out to you.
+protect	Indicates that existing paths in the destination hierarchy are not to be overwritten.
+versions[=**n**]	Specifies a range of versions to be copied. If you omit *+versions*, only the latest version of each selected object is copied. If you omit a value, all versions are copied. Where **n** is a positive integer, specifies that the oldest **n** versions of the selected objects are to be copied. Where **n** is a negative integer (for example, *+versions=¯5*), specifies that the **n** most recent versions of the selected objects are to be copied. *+versions* without a value, *+versions=0*, or *+versions=all* specifies that all versions of the selected objects are to be copied.

Result

The command returns the pathnames of the saved objects, including type and version number, as the result.

Usage

When **pathnames** is a directory, the directory and all objects beneath it are copied under the **destination** directory.

If you specify the [*d*] object type in **pathnames**, none of a directory's descendants is copied, but its attributes are.

If you want to copy specific, individual, noncurrent versions of an object, specify the version numbers in the pathname.

Examples

```
u copy alpha.?* beta
```

Copies all descendants of *alpha* under *beta* (which must exist). An object with the name *alpha.a* is copied to *beta.a.*

```
u copy alpha beta +makedir
```

Copies directory *alpha* and all its descendants under directory *beta.* An object with the name *alpha.a* becomes *beta.alpha.a* in the new directory *beta.* Directories that do no exist in the destination path will be created as needed.

```
u copy alpha[d] beta +makedir
```

Copies only directory *alpha* under *beta.*

```
u copy alpha.a[2] beta
```

Copies only version 2 of *alpha.a* under *beta.*

LOGOS Commands: *delete*

Use the *delete* command to remove paths or versions of objects from the LOGOS file system.

Syntax

delete **pathnames**
    [+*confirm*]
    [+*noncurrent*]
    [+*override*]
    [+*unused*]
    [+*warn*=**n**]

The *delete* command must be spelled out completely.

pathnames	Is a list of paths to delete. You can qualify a pathname with version numbers but an unqualified pathname refers to all versions, rather than only the latest.
+*confirm*	Prompts you to confirm the deletion of each object. (Confirmation is described in a usage note below.) If you omit +*confirm*, objects are deleted without confirmation.
+*noncurrent*	Deletes only noncurrent versions. If you specify +*noncurrent* and pathnames with version numbers, only versions preceding the specified version are deleted. If you omit +*noncurrent*, all versions are deleted if the pathname is not qualified by version numbers, or only the specified versions are deleted if the pathname is qualified by version numbers.
+*override*	Overrides any existing registrations for the specified paths. If you omit +*override*, registered paths are not deleted.
+*unused*	Deletes only unused paths (paths not referenced in any application). If you omit +*unused*, both used and unused paths are deleted.
+*warn*=*n*	Prompts you to confirm deletion if the number of paths to be deleted is greater than or equal to *n*. (Warning prompts are described in a usage note below.) If you specify +*warn=0*, no warning is issued. If you omit +*warn*, a warning is issued if any objects or paths are being deleted in their entirety.

Result

The command returns the pathnames of the deleted objects as the result. The result is not displayed unless requested via assignment.

Usage

delete rejects its **pathnames** argument if any pathname and version qualifier are inadvertently separated by a blank.

The *delete* command proceeds itemwise: the operation on each pathname you specify is separate. If you specify three pathnames, for example, this has the same effect as executing the *delete* command for each of the three pathnames in turn.

If you specify a directory, that directory and all its descendants are deleted. *delete* informs you if it has deleted any of your current working directories.

If you specify *+unused,* the used status is checked for each path subject to deletion. This can be somewhat expensive.

Warning Prompts

A warning is issued if you do not specify *+warn* and any pathnames are being deleted in their entirety, or if the number of paths to be deleted exceeds or is equal to the value of the *+warn* modifier. If you specify *+warn=0,* no warning is issued. A warning prompt might look like this:

delete 15 paths?

The number of paths to be deleted is the number of pathnames; versions are not counted. Consequently, this prompt is never issued when you specify *+noncurrent,* since specification of this modifier always causes the most recent version to be retained.

Reply *yes, no, confirm,* or *stop* to the warning prompt.

yes (or *y*)	Deletes the paths.
no (or *n*)	Does not delete the paths.
confirm	Prompts you to confirm the deletion of each path, as if *+confirm* had been specified.
stop	Does not delete the paths, and aborts the action of the command.

Confirmation Prompts

If you specify *+confirm,* you are prompted to confirm each deletion. These requests for confirmation follow any warning controlled by *+warn.* For example:

delete 15 paths? yes
delete <.scl.inven.post>?

Reply *yes, no, back, continue,* or *stop* to the confirmation prompt.

yes (or *y*)	Deletes the object and prompts you for the next path to be deleted.
no (or *n*)	Does not delete the object and prompts you for the next path to be deleted.
back	Repeats the previous prompt.

continue	Deletes this path and all following paths without further prompting.
stop	Aborts the deletion.

Deletions are not actually done until you respond to all warnings and confirmations.

Examples

```
ᵁ delete input
delete 1 path? yes
1 path and 3 versions deleted
```

Warns the user and deletes all versions of object *input*.

```
ᵁ delete inp?*
delete 4 paths? yes
4 paths and 10 versions deleted
```

Warns the user and deletes all versions of all objects whose pathnames begin with *inp*.

```
ᵁ delete in?*
delete 4 paths? confirm
delete <inp>? yes
delete <input>? no
delete <insight>? no
delete <into>? yes
2 paths deleted
```

Confirms and deletes all versions of two objects. Two objects are not deleted.

LOGOS Commands: *display*

The *display* command displays LOGOS objects or their attributes.

Syntax

d[*isplay*] **pathnames**
    [+*compile*[=**directives**]]
    [+*nopathname*]
    [+*surrogates*[=*b* | *c* | *i* | *l* | *n*]]

pathnames Is a list of objects to be displayed.

+*compile*[=**directives**]
Displays a compiled version of the object rather than the source. Optional values for +*compile* allow you to specify additional compilation directives. Without a value, the object is compiled according to the global compilation directives and to the directives saved in its [:*c*] attribute. If this modifier is omitted, the source is displayed without compilation.

+*nopathname* Refrains from displaying the pathname, type, version number, and attribute of displayed objects. If you omit +*nopathname*, this information displays as a header to each object's definition.

+*surrogates*[=*b* | *c* | *i* | *l* | *n*]
Specifies the nontypable characters in a variable that are to be displayed by *surrogates*. Following is a list of surrogates.

b	Backspace is represented by a surrogate.
c	Carriage return is represented by a surrogate.
i	Idle is represented by a surrogate.
l	Linefeed is represented by a surrogate.
n	Null is represented by a surrogate.

If you specify +*surrogates* with no value, Backspace, Idle, Linefeed, and Null (but not Carriage return) are displayed by surrogates. If you omit +*surrogates* altogether, no character is displayed by a surrogate.

Result

The command returns the display of the objects specified as the result.

Usage

Backspace, Carriage return, Idle, Linefeed, and Null cannot be unambiguously displayed in a source listing. Consequently, LOGOS enables you to show them through *surrogates*. A surrogate is formed by the character's identifying letter (for example, *b* for Backspace) overstruck with ¨ a dieresis (¨). Some terminals are not capable of displaying that combination of characters. On those terminals the display may consist of just the dieresis.

Examples

ᵁ *display checkitem*

Displays the source of the object *checkitem* exactly as stored.

ᵁ *display checkitem*[:*d*]

Displays the documentation for the object *checkitem*.

ᵁ *display report.?* +surrogates=cl*

Displays the source of all objects in directory *report.* Within variables, all occurrences of Carriage return and Linefeed will appear as surrogates.

ᵁ *display .john.utilformattilte +compile*

Displays the function *.john.utilformattitle* after any compilation directives in *.john.utilformattilte*[:*c*] or in the *environment compile* parameter have been applied to the source.

ᵁ *display .john.utilformattitle +compile=x,r*

Displays the version of *.john.utilformattitle* compiled after the *x* (decomment) and *r* (rename) directives have been merged into the list of active directives.

LOGOS Commands: *distribute*

The *distribute* command places LOGOS objects into a set of end environments which are known to use those objects according to a specified audit file.

Syntax

dist[*ribute*] **pathnames**
  +*audit*=**filename**
  [+*compile*[=**directives**]]
  [+*environments*=**envs**]
  [+*override*]
  [+*replacement*=**pathnames**]
  [+*show*]
  [+*task*[=**task**]]

pathnames	Is a list of objects to be distributed.
+*audit*=**filename**	Specifies the audit file. This modifier must be included unless an audit file is named in the *environment audit* parameter.

+*compile*[=**directives**]

Specifies compilation directives to be applied before the objects are distributed. If you specify +*compile* without a value, the objects are compiled the way they were last placed into their end environments. If you omit +*compile*, only compilation directives specified in the objects or in your environment are applied.

+*environment*=**envs**

Indicates the end environments to which you want to distribute. The default value is the value specified in the *environment update* parameter. The argument to the modifier contains three fields for type, name, and location, separated by spaces. The type of environment is represented by one of the following letters:

c	Cluster
f	File component
p	Paging file
w	Workspace

For further information on specifying environments, see the discussion of the *update* parameter under the *environment* command.

+*override* Overrides any existing registrations on pathnames. If you omit +*override*, the objects cannot be distributed into a pathname with registration set.

+replacement=**pathnames**

Specifies a series of replacement pathnames. These objects will be distributed instead of the objects named in the **pathnames** argument in each environment where any object in **pathnames** was distributed (according to the audit file). The used list in the audit file is updated to reflect the new names. If you omit *+replacement*, **pathnames** are distributed directly.

+show

Displays, while the command is running, the location in the end environments into which each object in **pathnames** (or in *+replacement*) is being distributed.

+task[=***task***]

Identifies the auxiliary task which will be used to distribute objects into workspaces, if any of the end environments is a workspace. If you specify *+task* without an argument, the default auxiliary task *aux* is used. This modifier must be specified if distributions to workspaces are going to be done. If this modifier is omitted, the task named in the *environment task* modifier, if any, is used.

Result

None.

Usage

This command is used to avoid the need to completely regenerate an end environment or set of end environments. Note that an audit file must have been used in the original generation of the end environments. The command also updates the audit file with reference information about the objects being distributed.

Using *distribute* on page files can sometimes cause problems because the calling tree is not recalculated by the *distribute* command. If the object being distributed has a different calling tree than it did originally, the *build* and *filesave* command with the *+update* modifier should be used instead. See the description of *filesave* in this guide for more information. Or, use the *calls* command to generate the list of items that are now in the root object's calling tree.

Examples

```
U distribute .public.logos.paging.Δlpagein +audit=1234567 mysys
1 object distributed to 1 end environment
```

Places the latest version of the object *.public.logos.paging.Δlpagein* in one end environment on the basis of the information in audit file *1234567 mysys*.

```
ᵁ distribute .public.logos.paging.Δlpagein +audit=1234567 mysys +show
.public.logos.paging.Δlpagein to file: 1234567 sysfile, 5; page: ★
1 object distributed to 1 end environment
```

The addition of the *+show* modifier to the above command causes the display of each end environment distribution.

```
ᵁ distribute .public.logos.paging.Δlpagein +audit=1234567 mysys
+replace=.john.logos.paging.Δlpagein | Δlpageinsub +show
pathnames selected for replacement:
.john.logos.paging.Δlpageub[f10]
.john.logos.paging.Δlpageinsub[f3]
.public.logos.paging.Δlpagein to file: 1234567 sysfile, 5; page: ★
2 objects distributed to 1 end environment
```

Places both *.john.logos.paging.Δlpagein* and *.john.logos.paging.Δlpageinsub* in the end environment locations where the audit file shows *.public.logos.paging.Δlpagein* to be used.

LOGOS Commands: *edit*

The *edit* command opens LOGOS objects for editing.

Syntax

e[*dit*] [=]**names**
    [+*browse*[=*off* | *on* | *lock*]]
    [+*command*=**command**]
    [+*disttask*=**task**]
    [+*override]*
    [+*register*]
    [+*status*=**text**]
    [+*task*[=**task**]]
    [+*workdir*=**pathnames**]

[=]**names** Is a list of one or more objects to be edited. If you specify more than one object, they are stacked to be edited in your specified order. If you precede any name with an equal sign (=), that object is taken from your active workspace. If any object in **names** is not found, a new object of that type is opened. The default type for new objects is a function.

+*browse*[=*off* | *on* | *lock*]
Prevents users from saving changes to objects. +*browse* accepts three arguments:

off	Allows edited objects to be saved if you have write access. This is the default setting.
on	Inhibits you from saving changes made to the object. This still allows full access to the display and inquiry features of the LOGOS editor.
lock	Prevents you from turning *browse* mode off. You can overwrite this by changing the name of the object. *setname* will automatically turn *browse* mode off if you use it to successfully change the name of the edited object.

If you specify +*browse* without an argument, +*browse=on* is assumed.

+*command*=**command**
Specifies a command to be executed immediately after opening each object listed in **names**.

+*disttask*=**task**	Specifies the name of the auxiliary task to be used by the Application Debugging Assistant to distribute changed objects to the active workspace.
+*override*	Overrides registration on any of the objects listed in **names**.
+*register*	Registers out all objects listed in names.
+*status*=**text**	Overrides the standard status line on the editor screen.
+*task*[=**task**]	Allows you to edit objects in the workspace of an auxiliary task. The syntax is:

edit =foo +task[*=**task***]

If *environment task* is set, that task is used unless overridden by a local *+task*. Whether *environment task* is set or not, if you do not specify **task**, the default value *aux* is used. The local task may be specified by using ⋆.

For example, if no *environment task* is set, you could use *+task* in the following ways:

ed =foo	Uses the current task.
ed =foo +task	Uses task *aux*.
ed =foo +task=aux3	Uses task *aux3*.
ed =foo +task=⋆	Uses the current task.

If *environment task* is set, for example, to *aux*2, you could use *+task* in the following ways:

ed =foo	Uses task *aux2*.
ed =foo +task	Uses task *aux*.
ed =foo +task=aux3	Uses task *aux3*.
ed =foo +task=⋆	Uses the current task.

+*workdir*=**pathnames**
Specifies alternative working directories for the duration of the command.

Result

The command returns the pathnames of any objects saved during the editing session.

Usage

The LOGOS *edit* command is based upon workspace *7 del*, which in turn is based on the VS APL Extended Editor and Full Screen Manager, used as the *)xedit on* command. See the *VS APL Extended Editor and Full Screen Manager Program Description/Operations Manual* [IBM publication H20-2341], supplemented by the notes about editor commands in this manual.

Examples

ᵘ *edit test.report*

Edits *test.report.*

ᵘ *edit test.report +command=/title←*

Edits *test.report*, positioning the reference at the first occurrence of *title←*.

ᵘ *edit =reportnew*

Edits the workspace object *reportnew*.

LOGOS Commands: *enroll*

The *enroll* command adds a new user to LOGOS, changes an existing one, or deletes a user. At some sites, the use of this command is restricted.

Each user in LOGOS has a primary enrollment entry, which specifies the preferred alias for his user number. Each user number may have more than one alias associated with it, in which case these other aliases are considered secondaries. Analogously, each alias has associated with it a primary user number -- namely, the number which ultimately claims ownership for its LOGOS files. Each alias may also have more than one user number associated with it, in which case these other user numbers are considered secondaries. In this manner, users may have multiple aliases, and different users may share the same alias.

Syntax

```
enr[oll] [alias] [user]
        [+alias=alias]
        [+cmddir=pathnames]
        [+delete[=yes]]
        [+flags=[,|/]i|p|s|m]
        [+groups=[,|/]groups]
        [+name=name]
        [+password=password]
        [+reset=yes]
        [+user=user]
        [+workdir=pathnames]
```

alias Specifies the new alias to enroll, or an existing alias for which some enrollment information is going to be changed; if you do not specify it, a prompt is issued.

user Specifies the user number for that alias, if it is a new enrollment or if it is a secondary entry. (If the user already exists, then its primary enrollment entry is assumed in the absence of this parameter.) A user's primary alias may be used in place of his account number as the second parameter to the command.

+alias=**alias** Changes the alias of an existing LOGOS member.

+cmddir=**pathnames**
Sets or changes the user's command directories.

+delete[=*yes*] Requests that the specified alias and user be deleted from LOGOS. If you specify an argument, it must be *yes*.

+flags=[,|/]*i*|*p*|*s*|*m*
Sets or changes the user's status flags. These are:

i	Internal access
p	Preferred access
s	Steward access
m	Maintenance access

Precede flags with a comma (,) to add them to the existing list, or with a slash (/) to delete them.

+groups=[,|/]**groups**
Sets or changes the user's groups membership. Precede group codes with a comma (,) to add them to the existing list in which the user is enrolled, or with a slash (/) to delete them.

+name=**name** Sets or changes the user's full name. Enter the full name last name first, followed by a comma, a blank, and then any familiar names and initials. This format ensures that the name can be properly collated in the LOGOS user directory.

+password=**password**
Sets or changes the user's password. If you do not supply a password, the existing password is removed.

+reset=yes Requests that the entry command line and terminal type for the user be restored to their LOGOS defaults. This allows a steward to help a user out of an awkward situation.

+user=**user** Changes the user number of an existing LOGOS member. The argument can be an alias, in which case the alias's primary account number is inferred.

+workdir=**pathnames**
Sets or changes the user's working directories.

Usage

For new members you must specify the alias, user number, name, and password. You are prompted for any that you do not specify.

Examples

---

u *enroll dick 2239945 +name=Thompson, Dick S. +password=yellow +group=test projdev logos*

Enrolls a new user with the alias *dick*, the user number *2239945*, the full name *Dick S. Thompson*, and the password *yellow*. Enrolls the user in the groups *test*, *projdev*, and *logos*.

---

u *enroll dick +alias=dsmith*

Changes the alias *dick* to *dsmith*.

---

u *enroll dsmith dick*

Estalishes the secondary alias *dsmith* for the alias *dick*.

---

u *enroll dick +group=/projdev logos*

Removes the alias *dick* from the groups *projdev* and *logos*.

---

u *enroll dick 2239945 +delete=yes*

Deletes the user *dick* from LOGOS.

---

u *enroll dick*

Displays enrollment information for existing alias *dick*.

LOGOS Commands: *environment*

The *environment* command changes or displays aspects of your LOGOS environment, such as the command separator character, working directories, and keyword definitions.

Syntax

env[*ironment*] [**parameter**] [**value**]
        [+*destack*]
        [+*profile*]
        [+*reset*]
        [+*stack*]

parameter Is the name of the environment parameter whose value is to be changed or displayed. If you omit **parameter,** the values of all parameters are displayed.

You can specify only one parameter at a time. The valid parameters are as follows:

audit	Default audit file
cmddir	Command directories
compile	Compilation directives
debug	Script debug setting
entry	Entry command line
exit	Exit command line
field	Screen field attributes
keyword	Keyword definitions
sepchar	Command separator character
status	Status line detail
task	Auxiliary task identity
terminal	Terminal type
track	Tracking table setting
update	Environments to be updated
workdir	Working directories

Each of these is described below.

value Is the new value of *parameter*. If you omit *value*, the previous value is displayed and not changed. The form of *value* depends on the parameter you've specified. See the description of each parameter, below.

+*destack* Restores the last stacked environment entry before any modifications made in the same command invocation.

+profile	Saves the environment setting for a specified parameter (or all of the parameters, if no particular one is specified) in your profile after the environment is changed according to the parameter and value you've specified.
+reset	Resets the specified parameter (or all parameters, if none is specified) to its saved value in your profile.
+stack	Stacks an environment entry, before any modifications made in the same command invocation.

Result

Without a parameter, the result is all environment settings currently in effect. With a parameter, the result is the setting of the specified environment parameter.

Usage

You may specify the *shortest unambiguous truncation* for any parameter. For example, *w* is sufficient to denote *workdir*, and *co* to denote *compile*.

You may specify only one parameter in a single *environment* command.

audit Parameter

The *audit* parameter establishes an audit file name. That file is used for audit records when the *build, distribute, filesave, shell, snap,* and *wssave* command are used without the *+audit* modifier. It is also used by various audit file scripts when an audit file is not specified.

Syntax

env audit **filename**

The argument **filename** is the audit file to use.

Example

ᵁ *env audit 1234567 prodaudit*

NOTE: The *audit* parameter is a session value that is not stored in your LOGOS profile.

cmddir Parameter

The *cmddir* parameter establishes or displays the directories in which LOGOS searches for scripts. You may also use the *cmddir* command for this purpose.

The command directory in effect when you are enrolled in LOGOS is *.public.logos.cmds.*

Syntax

env cmddir [**pathnames**]

pathnames is a list of pathnames specifying the command directories to be established.

Command directories are searched for scripts in the same order as they are specified.

compile Parameter

The *compile* parameter specifies global compilation directives that are to be applied to every object fetched from LOGOS into an end environment.

IMPORTANT: Use this parameter with care! Everything fetched from LOGOS, including script and user-defined compilation directive functions, are compiled when *environment compile* has a non-empty value. NEVER use the user-defined compilation directives (a and z) in an *environment compile* parameter. LOGOS will attempt to compile the user-defined directive function with a compiled version of itself, causing an endless loop.

Also, composite scripts that another user has shared with you may not be able to be compiled if you do not have the required access to paths localized in the scripts.

Specifying an environment compilation directive is a very useful technique, however, when used inside of a script that is building end environments.

Syntax

env compile [**directives**]

directives is a list of directives to be established. Separate the directives you specify with commas.

Example

ᵁ *environment compile x,d,r=0,p*

For more information about compilation and compilation directives, see the section *Compilation Directives* in this Reference Guide.

debug Parameter

The *debug* parameter selects LOGOS's reaction to errors occurring in a script.

Syntax *env debug* [*on* | *off*]

on	Causes a script error to result in interactive debugging.
off	Causes a script error to result in abandonment of execution and a return to the LOGOS command prompt.

entry Parameter

The *entry* parameter specifies an *entry command line,* a LOGOS command line to be executed upon each entry into LOGOS.

Syntax *env entry* [**command**]

command is any LOGOS command or series of commands. Since a command may be a script, arbitrarily complex actions are possible.

exit Parameter

The *exit* parameter specifies an *exit command line,* a LOGOS command to be executed upon each exit from LOGOS.

Syntax *env exit* [**command**]

command is any LOGOS command or series of commands.

field Parameter

The *field* parameter displays or establishes the colour and highlighting given to fields on the screen.

Colour and highlighting are used only for LOGOS full screens, not standard command prompt dialogue, and even then only for devices that support the features, such as the IBM 3179 and 3279 display stations.

Syntax

env field [**field**=[**colour**] [**highlight**]]

field — Is the field for which colour or highlighting are being specified. If you omit field, information about all fields is displayed. **field** may be any of:

status	Status information
message	Error messages
command	Command line
title	Title lines
frame	Screen borders
input	Input fields
output	All other output fields

colour — Is the field's colour. If you omit both **colour** and **highlight**, information about the specified field displays. **colour** may be any of:

neutral	turquoise
blue	white
green	yellow
pink	high (monochrome)
red	

highlight — Is the field's highlighting. If you omit both **colour** and **highlight**, information about the specified field displays. **highlight** may be any of:

neutral
blink
off
reverse
underline

Separate **colour** and **highlight** with a blank; separate specifications for fields with commas.

Example

ᵁ *environment field status=yellow reverse,title=blue*

keyword Parameter

The *keyword* parameter displays or maintains keywords. You can also use the *keyword* command for this purpose.

A *keyword* is a phrase you define which you can call at any time in response to a command prompt from LOGOS. Specify a keyword in a LOGOS command by preceding it with a backslash (\).

Syntax

env keyword [**name** [**definition**]]

name is name of the keyword to be defined or whose definition is to be displayed. If you omit **name** and **definition**, the names of all keywords currently defined are displayed.

definition is the definition to be given **name**. If you omit **definition**, the definition of **name** displays. If you specify two single APL quotes (''), the keyword is deleted.

sepchar Parameter

The *sepchar* parameter displays or establishes the global command separator character. This character separates commands in command lines that do not imply a local separator, and is also the LOGOS command prompt.

Syntax

env sepchar [**sepchar**]

sepchar is a single character that is neither an alphanumeric, a blank, nor a LOGOS metacharacter. The metacharacters which cannot be used are the following:

```
( ) + { } ' \ ∇ . [ ] ↑ ← ? ⍎ = ⍕ ⎕ α ⍵ ⍙ ∆ _
```

You can override the global *sepchar* character immediately upon entry to LOGOS, without altering the value saved in your profile, by specifying the desired separator as the first character in the argument to the *environment entry* parameter.

Example

∪ *env entry* '◇ *command1* ◇ *command2*'

This establishes the diamond (◇) as the separator character for this line.

∪ *env sepchar* ◇

This establishes the diamond as the separator character for this LOGOS session.

status Parameter

The *status* parameter selects the level of detail to be displayed on the status line at the top of the screen (this parameter applies only to certain device types).

Syntax

env status [**status**]

status is the status control. Valid settings are:

full	Selects a two-line status area.
half	Selects a one-line status area.
off	Selects a zero-line status area
none	Retains the size of the status area but preempts writes to it.

task Parameter

The *task* parameter displays or defines the task with which LOGOS workspace interactions occur by default. If there is a task specified, that is the task used by the commands *build, distribute, edit, get, save, send, shell, talk,* and *wssave* when a task is not specified.

Syntax

env task [**task**]

task is the names of the task with whose active workspace LOGOS interactions are to occur by default. The task identity may be that of any LOGOS task initiated via the *signon* command, or ⋆ to indicate the active workspace.

NOTE: The task parameter is a session value which is not stored in the LOGOS profile.

However, a command or script invoked as a part of the *env entry* parameter could set a particular task name to ensure that this environment parameter is always set appropriately.

terminal Parameter

The *terminal* parameter identifies the type of device you are using.

Syntax *env terminal* [**terminal type**]

terminal type is the type of device you are using. LOGOS uses this information to take advantage of your terminal's special features. Valid terminal types are:

unspecified	Specifies no terminal type.
ibm3270	Specifies a terminal in the IBM 3270 family of devices (including IBM 3279).
hds108	Specifies a Concept HDS/108.

The terminal type may be specified either as an absolute quantity, as in IBM3270, or as a quantity to be evaluated by LOGOS, as in:

```
∪ environment terminal
(⍕⍎(2 7⍴'hds108 ibm3270')[⎕io+2=⎕runs[⎕io;⎕io+2];])
```

track Parameter

The *track* parameter controls whether or not objects fetched into the workspace are tracked in the variable ⍙*LTRACK*. The tracking table is maintained by the *get* and *build* commands, for use primarily by *snap* and *wssave.*

Syntax *env track* [*on* | *off*]

on	Tracks objects fetched into the workspace in the variable ⍙*LTRACK*.
off	Does not track objects fetched into the workspace.

update Parameter

The *environment update* parameter tells LOGOS which end environments to update with the new version of an object when using the Application Debugging Assistant, or *distribute*. The *update* parameter is a session value that is not stored in your LOGOS profile.

The argument to the update parameter contains type, name, and location of each end environment, delimited by a separator character. The separator character is identified as the first non-alphanumeric in the argument. For example:

environment update /**type name location**

The / character is the separator character. The type of end environment can be any of the following letters:

c	Cluster
f	File component
p	Paging area
w	Workspace

The following rules apply to the way you specify the name and location of the end environment.

Rules Governing Name and Location Of The End Environment

If:	**Then:**
the type is *f*, *p*, or *w*,	**name** must be an account number and name. The account number is optional. If you don't specify it, LOGOS assumes your account number.
the type is *c*,	**name** must be a pathname.
the type is *f*,	**location** must be a component number within the file.
the type is *p*,	**location** must be the paging area's starting component number.

For example:

/f 1234567 myfile 550

Because the type of end environment is f (file), the name is an account number and name, and the location is a component number.

The update parameter takes the wild-card characters *?* and *?**. The comma (,) is not a valid character. For example:

*/c .myws.cmds.?**

Because the type of end environment is c (cluster), the name is a pathname.

NOTE: This parameter is a workspace session property and is not saved with your profile.

workdir Parameter

The *workdir* parameter establishes or displays your working directories. You may also use the *workdir* command for this purpose. A **working directory** is the directory which the LOGOS file system searches for pathnames not specified from the root. If you have more than one working directory, they are searched in the order specified.

The working directory in effect when you are enrolled in LOGOS is your alias.

Syntax

env workdir [**pathnames**]

pathnames is a list of pathnames specifying the working directories to be established.

Working directories are searched for objects in the same order as they are specified.

LOGOS Commands: *exit*

The *exit* command ends a LOGOS session and returns you to the environment from which you called LOGOS. The *exit* command takes effect immediately; any commands to its right on the line are ignored.

Syntax

ex[*it*] [**expression**]
        [*+reset*]

expression	Is the APL expression to be evaluated just before LOGOS ends.
+reset	Unties all LOGOS files and resets the tables which LOGOS maintains in the workspace to the state they were in before LOGOS was run.

Result

None.

Usage

A typical use of **expression** is to load a new workspace.

Use of *+reset* commonly results in an increase in working storage (reflected in the result of ⎕*wa*), but in no other effect. *+reset* is useful for returning a workspace to pristine form before saving it for general use.

Examples

```
∪ exit
```

Ends LOGOS.

```
∪ exit '⎕load ''666 box'''
saved 1988-01-07 23:50:04
```

Ends LOGOS and loads workspace *666 box*.

```
∪ exit +reset
    )erase logos
    )save
1988-09-25 23:35:40 payables
```

Resets the workspace, ends LOGOS, erases the *logos* function, and saves the workspace with its current name, *payables*.

```
∪ exit ⎕ts +reset
1988 9 25 23 45 62 439
```

Displays the system time, resets the workspace, and ends LOGOS.

LOGOS Commands: *export*

The *export* command copies objects from LOGOS paths to a file intended for export to another system or for storage outside LOGOS's domain (such as in a private backup). *export* differs from *copy* in the following ways:

- The destination path must be the pathname of an export file that was created by the *export* command with *+makedir* specified.
- The names of copied paths are identical to the original full pathnames, with the addition of the export file name as a prefix. For example, the objects *.john.reports.?** copied into the export file *.vp.transfer* are called *.vp.transfer.john.reports.?**.

Syntax

exp[*ort*] **pathnames**
[*+makedir*]
[*+override*]
[*+protect*]
[*+versions*[=**n**]]

pathnames	Is a list of paths to be exported.
+makedir	Creates directories not extant in the destination path.
+override	Overrides the registration of paths being exported.
+protect	Refrains from overwriting extant paths in the export directory. If you omit *+protect*, extant paths can be overwritten.
+versions[=**n**]	Provides control over which versions of objects are exported. By default, only the latest version of each specified object is exported. If you specify *+versions* without a value, or with a value of *all* or 0, all versions are exported. If **n** is a negative integer, the most recent **n** versions are exported; if positive, the oldest **n** versions are exported.

Result

The command returns the pathnames of the saved objects, including type and version number, as the result.

Usage

See the script *.public.logos.cmds.install* for an example of how to install an application from an export file.

When **pathnames** is a hierarchy, the entire specified hierarchy is exported under the destination pathname.

If you specify the [*d*] object type in **pathnames**, none of a directory's descendants are exported.

If you want to export specific, individual, noncurrent versions of an object, specify the version numbers in the pathname.

The command *copy .john.archive .* at the receiving end performs the inverse of all *export* operations made to the export file *.john.archive*. Using the *+protect* modifier to *copy* prevents overwriting existing objects.

Examples

```
u export dir.?* .john.archive +makedir
```

Exports all paths under *dir* and creates export directory *.john.archive* if necessary.

```
u export report1 report2 .john.archive
```

Exports two paths to the existing export file *.john.archive*.

LOGOS Commands: *filemaint*

The *filemaint* command reclaims file space consumed by deleted objects and generates reports describing the use of space within files.

Syntax

```
file[maint] pathname
        [+compress]
        [+extended]
```

pathname	Is the pathname of a LOGOS file and must contain two segments (for example, *.sys.tools*).
+compress	Reclaims the space used by deleted components within a file. The effects of this action are not visible until after the next fulldump and restore of the APL file system.
+extended	Provides additional information in the usage report (see Result).

Result

filemaint returns a basic report including:

- file size in bytes
- file reservation
- difference between size and reservation
- number of deleted objects

An extended report includes more information, including:

- total space consumed by deleted objects
- total number of objects in the file (excluding those deleted)
- number of objects with non-empty note attributes
- number of objects which are registered out
- individual counts of each object type (clusters, directories, functions, links, scripts, and variables)

Usage

When you use the *delete* command to delete an object from a LOGOS file, LOGOS removes the object from the hierarchy and marks the space the object occupied as reusable but does not actually reclaim the space. The next time data is written to the file, this space is reused.

In certain situations (for example, the deletion of a large number of objects followed by a period where no new data is written to the file) a file can grow to be much larger than the sum of the sizes of the objects in it. *filemaint* can reclaim this space and inquire upon the amount of orphaned space in a file.

Examples

---

u *filemaint .dkol.appll*

Generates a basic report for *.dkol.appll*. For example:

```
2,599,856   total bytes
3,681,764   bytes reserved
1,081,908   bytes free
      533   deleted objects
```

---

u *filemaint .dkol.appll +extended*

Generates an extended report for *.dkol.appll*. For example:

```
2,599,856   total bytes
3,681,764   bytes reserved
1,081,908   bytes free
      533   deleted objects
  729,660   byes consumed by deleted objects
    2,334   total objects
        7   objects have note set
        2   objects are registered out
       37   clusters
      149   directories
    1,975   functions
        2   links
       51   scripts
      120   variables
```

LOGOS Commands: *filesave*

The *filesave* command generates a LOGOS paging area. The *filesave* command uses information supplied by *build* and *shell* commands issued in this session since the last *filesave* command.

Syntax

```
filesave [filename[cn]]
        [+audit=filename]
        [+end=n]
        [+lock=passnumber]
        [+note=text]
        [+overwrite[=audit|,|buffer|,|dest]]
        [+size=n]
        [+update[=nodes]]
```

filename[**cn**]	Is the SHARP APL file that is to contain the paging area. LOGOS creates the specified destination file for you if it does not exist. If you omit filename, the file identifier is taken from the *+file* modifier of a buffered *build* command. cn identifies the starting component number of the paging area to be built. If you omit **cn**, LOGOS builds a new paging area starting at the first available component.
+audit=**filename**	Identifies an audit file to contain information about this generation. *+audit* is relevant only when you're generating a paging area or a workspace. If the specified audit file does not exist, LOGOS creates it for you. If you omit *+audit*, the value of *+audit* in a buffered *build* command (if any) is used.
+end=**n**	Specifies the upper component number limit for this paging area. 0 (zero) indicates no limit.
+lock=**passnumber**	Specifies the paging file's passnumber. If you omit *+lock*, the value of this modifier in a buffered *build* command, or 0, is used.
+note=**text**	Specifies text to be placed in the header component of the associated paging area, providing documentation about the area.

+overwrite[=*audit*|,|*buffer*|,|*dest*]

Specifies one or more working areas to be cleared.

+overwrite=audit

Generates a new audit record.

+overwrite=buffer

Overwrites the internal "build buffer" (which contains the *build* statements accumulated previous to a *filesave*).

+overwrite=dest

Overwrites the destination node.

+overwrite or *+overwrite=audit,buffer,dest*

Overwrites all working areas.

If you omit *+overwrite*, no working areas are overwritten.

+size=**n** Specifies the maximum size, in bytes, of any node whose size was not specified by its *build* statement. **n** may be any positive integer. When a node's size exceeds the value of this modifier, LOGOS automatically splits its contents into two or more separate nodes. If you omit *+size*, node size is not limited.

+update[=**nodes**]

Specifies nodes to be updated. Separate nodenames in **nodes** with blanks. If you omit *+update*, the entire paging area is regenerated. This is the default setting.

Result None.

Usage *filesave* is used solely for the generation of LOGOS paging areas. Each paging area consists of two or more consecutive components of a file, and is used as a repository for *named packages* called nodes. Each paging area has a primary node which is called the *base node* and is represented by ⋆. LOGOS contains a standard set of paging utilities, found in the LOGOS directory *.public.logos.paging*, which are available for your use. For more information on the use of paging areas, see Chapter 8 of the *LOGOS User's Guide*.

Use the *+note* modifier to help distinguish among paging areas or files, in the event that you are perusing them manually.

filesave will run better, faster, and with reduced chance of workspace storage problems if an audit file is specified. This is due to the amount of information which must be maintained when tree analysis is being performed. When an audit file is specified, this information is kept in that file. Audit files are also desirable because they allow you to use *distribute* to make incremental modifications to end environments you have generated using them.

Example

```
∪ build +depth +workdir=.john.logos.util .public.logos.util +overwrite
∪ build ⋆ .public.logos.paging.Δlpagein | Δlpageout | Δlpwsfull | Δlpcmprs ⎕trap
∪ build 'calculate edit model report'
∪ shell 'calculate edit model report'
∪ build 'ask output print getdata putdata openfiles'
∪ filesave 1234567 model 10 +audit=1234567 audmodel
generating 10 nodes to paging area 1234567 model, 10
10 nodes (including 4 shells) generated using 147 objects
generation 8 of paging area 1234567 model, 10 saved 03mar86 13:54 by john
```

The first *build* statement defines a new set of default modifiers to apply when generating the nodes specified in the statements which follow. The second *build* statement establishes the base node, a set of objects to be brought into the workspace when the paging area is opened. The third *build* statement specifies a set of primary nodes, which in this case are commands. The *shell* statement which follows specifies that shells are to be built around the primary nodes. The last *build* statement specifies a set of ancillary nodes to be paged in wherever they are needed. Finally, the *filesave* command specifies the file which will contain the paging area, and the component in the file at which the area is to start. This command also specifies the audit file to be used to keep track of object placement within nodes in the paging area.

LOGOS Commands: *get*

The *get* command fetches objects from the LOGOS file system and deposits them in your execution environment.

Syntax

g[*et*] **pathnames**
[+*compile*=**directives**]
[+*protect*]
[+*recursive*[=*1* | 2 | *all*]]
[+*task*=[**task**]]
[+*workdir*=**pathnames**]

pathnames Is a list of objects to be fetched.

+*compile*=**directives**
Specifies the compilation directives to be applied to the source form. If you omit +*compile*, the objects are compiled according to directives saved with them and directives global to your LOGOS environment.

+*protect* Does not overwrite the existing workspace objects. If you omit +*protect*, workspace objects may be overwritten.

+*recursive*[=*1* | 2 | *all*]
Indicates which objects subordinate to a specified directory are to be retrieved. If you specify +*recursive* without a value, +*recursive=all*, or +*recursive=0*, all subordinate objects are retrieved. If you specify +*recursive=1*, only the named level is retrieved. If you specify +*recursive=2*, only the named level's direct descendants and not the named level itself are retrieved. This is the default setting.

+*task*[=**task**] Specifies the auxiliary task into whose active workspace the retrieved objects are to be placed. If you omit +*task*, the objects are retrieved in the workspace of the task names in the *environment task* parameter. If no *environment task* is named, the objects are retrieved into your local active workspace. If you specify +*task* without a value, the objects are retrieved into the workspace of the auxiliary task with the default name *aux*.

+*workdir*=**pathnames**
Specifies working directories to be in effect for the duration of the command. If you omit +*workdir*, your global working directories are used.

Result

The command returns the pathnames and version numbers of the retrieved objects.

Usage

By default, the object form of the requested pathnames is retrieved. You may have other attributes, such as source, documentation, or note, retrieved by specifying the attribute in **pathnames**.

The attribute being defined is put, by default, in your current execution environment. This is either the active workspace of the task in which you're running LOGOS, or the active workspace of the auxiliary task specified by *environment task*. Selection of the execution environment may be overridden by the *+task* modifier.

Examples

```
u get util.vtom
```

Fetches the object form of *vtom* into the workspace.

```
u get replace vtom +compile=x +workdir=util
```

Fetches the object form of *util.replace* and *util.vtom*, with comments removed.

```
u get util[:d] +task=taska
```

Fetches the documentation from *util* into the active workspace of task *taska*.

LOGOS Commands: *group*

The *group* command allows you to add new groups to LOGOS, change information about existing groups, and delete groups.

Syntax

gro[*up*] [**alias**]
[+*alias*=**alias**]
[+*delete*[=*yes*]]
[+*enrollment*=[, | /]**aliases**]
[+*flags*=[, | /]*i* | *c*]
[+*mentor*=**alias**]
[+*name*=**name**]

alias	Is the alias of the group to be enrolled or changed. An alias can be a maximum of twelve characters or numerals in length, starting with a letter.
+*alias*=**alias**	Specifies the new alias for an existing group. If you omit +*alias*, the group's alias is not changed.
+*delete*[=*yes*]	Deletes the specified group. If you supply an argument, it must be *yes*.

+*enrollment*=[, | /]**aliases**

Specifies or changes a group's membership. To add aliases to a group, preceed the aliases with a comma (,). To delete aliases from a group, preceed the aliases with a slash (/).

+*flags*=[, | /]*i* | *c*

Sets or resets status flags for the group. The flags are:

i	Sets the group's internal flag.
c	Sets the group's closed flag.

Currently, LOGOS does not use either of these flags and setting them has no effect.

+*mentor*=**alias**	Specifies the alias to be the group's mentor. The mentor of a group is the user responsible for establishing and controlling its composition. A mentor is similar to a project leader or a meeting coordinator.
+*name*=**name**	Specifies the group's full name. The name can be a maximum of 45 characters from the ASCII character set. This name is used to collate the group in the LOGOS user directory, so it is effective to begin the name with the group's central theme.

Result

The command returns the new enrollment information for the group, or its enrollment information just prior to deletion.

Usage

To enroll a new group, you must specify **alias**, *+name*, *+mentor*, and *+enrollment*. You are prompted for missing information.

If you omit the value to any modifier, you are prompted for the missing information.

The mentor of a group need not be a member of it.

If the full name you want to use contains parentheses, as in *inventory control (dev)*, and you are specifying it on the command line, enclose the name in quotes. Without quotes LOGOS treats the parenthetical expression as a command, and an error occurs.

To any prompt issued by *group*, you can enter the keyword *stop* or *back*. *stop* aborts the enrollment operation without change to the group, and *back* returns you to the previous prompt issued. To enter the words *stop* and *back*, preceed them with a slash (/).

Any LOGOS user may create a group. Mentors of a group can change group membership by adding or deleting aliases. Remember that group membership automatically grants the access that has been given to the group to new members.

Examples

U *group inventdev +name=inventory control group +mentor=invent +enrollment=invent invbase dick*

Enrolls the group *inventdev* with *invent* as the mentor and *invent*, *invbase*, and *dick* as members.

U *group invent +enrollment=, bob jstanley*

Adds the aliases *bob* and *jstanley* to the group *invent*.

U *group invent +delete=yes*

Deletes the group *invent* from LOGOS.

LOGOS Commands: *help*

The *help* command obtains information about another LOGOS command.

Syntax

h[*elp*] [**command**]

command is the command whose description is to be displayed. If you omit **command**, information on the various kinds of help available is displayed.

Result

The command returns the information requested as the result.

Usage

help followed by a command name is equivalent to *??* followed by the same name.

Examples

ᴜ *help*

Displays the various kinds of help you can request.

ᴜ *help list*

Displays general, syntax, and usage information about the *list* command.

LOGOS Commands: *import*

The *import* command attaches a LOGOS file to the LOGOS system. Typically, the file will have been generated via an *export* operation performed on another system, followed by a retrieval of the file to the current system. But the file might also have been created on the current system at some point in the past, and retrieved from a backup. Without performing the *import* operation, a retrieved file is not accessible to LOGOS.

Syntax

imp[*ort*] **pathname [newpathname [oldpathname]]**

pathname	Is the LOGOS pathname or APL file identifier of the file as retrieved.
newpathname	Is the LOGOS pathname or APL file identifier which the imported file is to assume. If you omit **newpathname** by specifying '' instead of a pathname, the imported file retains its present name.
oldpathname	Is the pathname that the file possessed when it was first generated, if different from **pathname**.

Result

The command returns the pathname of the imported file as the result. By default, this result is not displayed (although a message including it is). To display or capture the result, use assignment on the command line.

Usage

import does not actually move any objects. Rather, it simply attaches a file to the LOGOS file system. To move the objects, use the *copy* command.

Since *export* moves objects and *import* doesn't, these two commands are not inverses. The inverse of an *export* command is a *copy* command with a target path of a dot (.). See the description of *export* for more details.

Examples

```
∪ import '1234567 archive'
.john.archive imported
```

Imports the file *1234567 archive* (*john* is the primary alias for user *1234567*).

```
u import archive
.john.archive imported
```

Imports the file *1234567 archive*. LOGOS assumes that the first argument is a file identifier if it contains no dots.

```
u import .john.archive
.john.archive imported
```

Imports the file *1234567 archive*.

```
u import '1234567 archive' .john.temp
.john.archive imported as .john.temp
```

Imports the file *1234567 archive* as *.john.temp.*

```
u import '1234567 retrieved' .john.temp .john.archive
.john.retrieved imported as .john.temp
```

Imports the file *1234567 retrieved* as *.john.temp.* Its original name was *.john.archive.*

LOGOS Commands: *keyword*

The *keyword* command displays or maintains your keyword table.

A *keyword* is a phrase you define which you can call at any time in response to a command prompt from LOGOS. A keyword entered in a LOGOS command must be preceded by a backslash (\).

Syntax

k[*eyword*] [**name** [**definition**]]

name	Is the keyword to be defined or displayed. If you omit **name** and **definition**, the names of all keywords currently defined are displayed.
definition	Is the definition to be given **name**. If you omit **definition**, the definition of **name** is displayed. If you specify `''`, the keyword is deleted. The keyword's definition cannot exceed 500 characters in length.

Result

If you use *keyword* to define or delete a keyword, the command returns an empty result. If you use *keyword* without an argument, it returns the names of all of your defined keywords. If you use *keyword* to inquire upon one or more keywords, it returns the definitions of these keywords (see the example below).

Usage

You may supply an argument of *keyword* enclosed in parentheses (LOGOS evaluates the expression) to display all your keywords and their definitions.

Use *environment +profile* to save your keywords in your profile.

Examples

```
u keyword
```

Lists the names of the keywords you have defined.

```
u keyword compare .mde.logos.myscripts.compare
u workdir \compare
.mde.logos.myscripts.compare
```

Defines the keyword *compare* and uses it to establish a new working directory.

```
u keyword box '±□load ''666 box'''
u keyword
compare box
u keyword box
keyword box '±□load ''666 box'''
```

Defines the keyword *box* to load workspace *666 box*, displays the names of existing keywords, and displays the definition of *box*.

```
u keyword (keyword)
keyword compare '.mde.logos.myscripts.compare'
keyword box '±□load ''666 box'''
```

Displays the definitions of all keywords.

```
u keyword compare ''
u keyword
box
```

Deletes the keyword *compare* and displays the names of all existing keywords.

LOGOS Commands: *link*

The *link* command establishes a surrogate pathname by which you can access other LOGOS paths. When a link appears as a pathname or a portion of a pathname, LOGOS replaces the link with its value and then searches for the newly formed name.

Syntax

link **newpathname oldpathname**

newpathname	Is the name of the link and must be a name that doesn't exist. After a link is created, its pathname can be used in place of the **oldpathname**.
oldpathname	Can be any LOGOS object or directory.

Result

The command returns the pathname of the newly created link as the result.

Usage

Some commands -- such as *copy, delete*, and *list* -- operate on the link itself and do not resolve the link. For example, invoking *delete* on a link deletes the link, but not the path to which it resolves.

Most commands do resolve links to their ultimate pathnames; this can be disabled by specifying a type of [*l*] in the pathname argument to the command.

Use *list* with *+summary* to display the path to which a path is linked. Note that when links themselves are linked to other paths, you can traverse the intermediate paths and see the final path in the chain by using the *+ultimate* modifier with *list*.

A link can refer to another link, but cannot refer to itself. To change the value of a link, you must first delete the link and then use *link* to reinstate it.

Examples

```
u link util .public.util
u get util.b?*
.public.util.badinp[f2]
.public.util.bs[vl]
```

Establishes a link to a directory, and fetches those objects beginning with *b*.

```
u link myvtom .public.util.vtom
u get myvtom
.public.util.vtom[f2]
```

Establishes a link to an object, and then fetches that object.

LOGOS Commands: *list*

The *list* command displays information about objects and directories in the LOGOS file system. To display a LOGOS object, use the *display* command. To display the APL attributes of an object, use the *summarize* command.

Syntax

```
l[ist] [pathnames]
        [+column]
        [+data[=pn,type,...]
        [+extended]
        [+full]
        [+headings]
        [+long]
        [+overhead]
        [+recursive[=1|2|all]]
        [+summary]
        [+type]
        [+ultimate]
        [+versions[=n]]
```

pathnames	Is a list of objects or directories to be listed. If you omit **pathnames**, the path specified by the primary current working directory is used.
+column	Formats the report in as many columns as will fit across the display. This modifier is ignored for data, long, summary, and version reports. If you omit *+column*, the names display in a single column.

+data[=*pn,type,...*]

Displays information in a delimited data format, suitable for analysis by a program. Lines of the report consist of fields identified by a leading delimiter. 16 fields are defined, and any unique subset may be selected. Fields are:

pn	Pathname
type	Object type
size	Object size or directory descendant count
perm	Permission (*c,w,r,x*)
cts	Create timestamp (YYYMMDDHHMM)
cwho	Creator
ts	Last write timestamp (YYYMMDDHHMM)
who	Author
rts	Registration timestamp (YYYYMMDDHHMM; empty if not registered)
rwho	Registerer (empty if not registered)
vers	Versions available
ret	Retention (0 if all)
ver	Version number
cc	True change count
attrs	Attributes (*c,d,j,n,t,r,p*)
lpn	Link pathname (empty if not link)

Neither the field delimiter nor the maximum number of fields should be assumed. If you specify *+data* with no argument, the fields are returned in the order shown above. If you specify *+data* with an argument, fields are returned in the order specified.

Specifying *+data* overrides other modifiers that control the result format - column, long, summary, and version. The content of the result is affected by the presence or absence of the *+full*, *+overhead*, *+version*, or *+ultimate* modifiers. For example, if *pn* is one of the items requested and *+full* is specified, full pathnames will always be returned.

+extended Displays extended information on pathnames, including: the type, the version number, the attribute (if not source).

+full Displays rooted pathnames. If you omit *+full*, partial pathnames display when they are unambiguous.

+headings Displays report headings. This modifier applies only to data, summary, and long reports. If you omit *+headings*, report headings do not display.

+long	Displays a *long report.* A long report contains all the information in the summary report, plus the timestamp, the alias of the user who created the object, change count, number of versions in LOGOS, registration information, and broadcast note. If you omit both *+long* and *+summary*, only the object names display.
+overhead	Includes the total size of all attributes of the object in the *size* column in data, summary, and long reports. If you omit *+overhead,* the size of the specified attribute is reported. If you don't specify an attribute, the size of the source is reported.

+recursive[=*1*|*2*|*all*]

Controls the levels of directories to be processed. *+recursive* (without a value), *+recursive=all,* or *+recursive=0* specify that the named level and all subordinate directories are to be listed. *+recursive=1* lists only the named level. *+recursive=2* lists only the direct descendants of the named level, excluding the named level itself. If you omit this modifier, *+recursive=2* is assumed.

+summary Displays a *summary report.* For each object (if not a link), a summary report gives:

- pathname
- object type
- size
- your permission
- the timestamp when last modified
- the alias of the user who last modified the object
- the most recent version number
- retention period
- flags which indicate if the object is registered
- what attributes it has set.

If the object is a link, the summary report gives:

- pathname of the linked object
- object type
- size
- your permission
- the timestamp when last modified
- the alias of the user who last modified the object
- the pathname of the object to which the first object is linked

If you omit both *+summary* and *+long,* only the object names display.

+type	Displays the object type. This modifier is not necessary if you also specify *+summary* or *+long*.
+ultimate	Resolves links to ultimate paths, and displays information about these paths rather than the links themselves.
+versions[=**n**]	Displays information about specified versions of selected objects. If you do not specify a value, all versions of the objects display. If you specify **n**, only the first or last **n** versions are listed, depending on whether **n** is positive or negative, respectively. Use of this modifier implies a summary report if *+long* is not selected.

Result

The command returns the requested object or directory information as the result.

Usage

If you omit all modifiers, only names are displayed, in a single column.

+type is useful with *+column* to list the names and types of objects. *+summary* displays a convenient one- or two-line summary of each object. *+long* supplies the most detail.

You can list all objects of a particular type with the command *list ?**[*f*]

list **pathname**[*d*] *+full +recursive* where **pathname** is a directory, lists the full pathnames of all directories at or below a particular node in the file system.

list **pathname** *+recursive=1 +summary* where **pathname** is a directory, lists the entry for the directory itself, rather than its descendants. The count of descendants in the directory is reported as its size.

+full is useful when you want to pass the result of *list* to another LOGOS command because it identifies the pathname unambiguously without reference to the working directory. For example:

```
ᵁ display (list .dick.util.a?* +full)
```

Examples

ᵁ *list inventory*

Lists pathname *inventory*, or if it is a directory, all objects immediately subordinate to it.

ᵁ *list inventory +column +full +type*

Lists the full pathname of *inventory*, or if it is a directory, all objects immediately subordinate to it, in columnar format with object type.

ᵁ *list inv?⋆[f] +column*

Lists the pathnames of all functions beginning with *inv*, in columnar format.

ᵁ *list report chart +summary*

Lists a summary of the named objects, or if they are directories, the objects immediately subordinate to them.

ᵁ *list report chart +versions=¯3*

As above, but reports information on the last three versions of each object.

ᵁ *list .dick.util +recursive=1 +summary*

Lists a summary of the directory *.dick.util*, with the count of descendants in it reported as its size.

ᵘ *list chart*[:*d*] +*summary*

Lists a summary of the object *chart*, or if it is a directory, the objects immediately subordinate to it. The size of the object included in the report is the size of the documentation attribute.

ᵘ *list inv?**[*f*] +*d=pn,perm,size,who* +*headings* +*full*

Returns a delimited list containing the full pathnames, your permission, and the size and author of all functions beginning with *inv*. Delimited headings are included.

LOGOS Commands: *locate*

The *locate* command returns the locations of one or more strings or of a regular expression in LOGOS objects.

Syntax

loc[*ate*] **string** [**pathnames**]
[*+display*]
[*+flags=c* | *l* | *n* | *o* | *q* | *s* | *x*]
[*+lines*]
[*+multiple*]
[*+recursive*[*=1* | *2* | *all*]]
[*+show*]

string	Is the text or expression to be sought. Enclose it in quotes if it contains blanks or the command separator. Enclose it in curly braces ({ }) if it is a regular expression.
pathnames	Is a list of objects to be searched. This argument is optional and, if unspecified, defaults to your primary working directory.
+display	Displays the entire line on which matches to **string** occur. If you specify none of *+display*, *+lines*, or *+show*, the command displays only extended pathnames, indicating the object type and version number, of objects containing the string.

+flags=c|l|n|o|q|s|x

Specifies the parts of objects to be included in the search. If you omit *+flags*, the entire object is searched. This search is nonsyntactic and searches the object as an undifferentiated character vector. A function's comments and character constants are included in the search. Note: the flags *c*, *l*, *q*, and *x* are effective only when you specify them with the flags *s* or *o*.

c	Searches function comments.
l	Searches LOGOS comments only. (LOGOS comments begin with ⍝∇.)
n	Searches the composite elements of numeric vectors; see the usage note below.
o	When specified with *c*, *l*, *q*, or *x*, searches those types of lines only.
q	Searches quoted strings (character constants).
s	Syntactic search. Searches for occurrences of **string** as a syntactic unit within the non-comment lines of the objects being searched. If any of the flags *c*, *l*, *q*, or *x* are also specified, those types of lines are included. If flag *o* is also specified, restricts search to only those types of lines.
x	Searches executed quoted strings (that is, character constants that are arguments to ⍎ functions).

+lines Displays the line numbers of lines in which **string** is found. If you specify none of *+display*, *+lines*, or *+show*, the command displays only extended pathnames, indicating the type of object and the version number, of objects containing the string.

+multiple Searches for multiple strings. See the usage note below. If you omit *+multiple*, **string** is sought as one unit.

+recursive[=1|2|all]

Controls the recursive processing of directories. *+recursive* (without a value), *+recursive=all* or *+recursive=0* indicate that the named level in **pathnames** and all subordinate objects are to be searched. *+recursive=1* indicates that only the named level is to be searched. *+recursive=2* indicates that only the direct descendants of the named level, excluding the named level itself, are to be searched. This last specification is the default behaviour if you omit the modifier altogether.

+show Displays the entire line on which matches occurred, with a caret (^) pointing to each match. If you specify none of *+display*, *+lines*, or *+show*, only pathnames, types, and version numbers of objects containing the string display.

Result

The command returns the display of all matches as the result, in the format you requested.

Usage

Use *+multiple* to search for several different strings simultaneously. When you specify *+multiple*, the first character of the string argument delimits the substrings to be sought.

You can specify only a single regular expression to *locate* even with *+multiple*. However, you can unite several regular expressions using the alternation construct between each expression. For example:

```
{α:|[lL]ω:}
```

To search functions for names of objects, specify *+flags=s*. Names in comments and character constants are ignored unless *c* and *q* are also specified.

To include comments and character constants in a syntactic search, specify *+flags=cqs*.

In a syntactic search, a single number such as 1 is not found if it appears in a vector. For example:

```
1 0 /j
```

Use *+flags=ns* to match constants that are embedded in vectors.

Examples

```
∪ locate dig
```

Searches for *dig* in your primary working directory.

```
∪ locate dig (workdir)
```

Searches for *dig* in your working directories.

---

u *locate dig compfn*

Searches for *dig* in *compfn*.

---

u *locate z dir +flags=s*

Searches for the name *z* in *dir*.

---

u *locate 1 compfn +lines*

Searches for the digit *1* in *compfn*, and returns the line numbers of each object line in which it is found.

---

u *locate 1 compfn +flags=s +lines*

Searches for the numeric scalar *1*.

---

u *locate* {α: | [*IL*]ω: } *dir*

Searches for the regular expressions α: and [*IL*]ω: in *dir*.

---

u *locate /pgm←/pgs← dir dirx +flags=s +multiple +show*

Searches for the syntactic constructs *pgm←* and *pgs←* in *dir* and *dirx*, and displays the context of each occurrence. The slash (/) is the delimiter for the string argument.

LOGOS Commands: *output*

The *output* command generates various classes of LOGOS output from within a script.

Syntax

```
out[put] text
        [+error]
        [+message]
        [+quadprime]
        [+result]
        [+status]
```

text	Is a character vector to be displayed.
+error	Defines **text** as an error message, signals errors to the LOGOS command processor, and aborts execution of the script.
+message	Defines **text** as LOGOS message class output. If you do not specify a modifier, this is the default class.
+quadprime	Displays or assigns **text** as if it were quadprime (⍞) output.
+result	Defines **text** as LOGOS result class output, the result of the script calling the *output* command.
+status	Writes **text** to the LOGOS status line.

Result

The result is the argument. By default, this result does not display. *output* also has the ability to pass its result on as the result of the script which calls it.

Usage

See the *LOGOS User's Guide* for a description of how *output* may be used in scripts.

Example

```
ᵘ display .dick cmds.divide
.dick.cmds.divide[s2]:
[1]   quotient←divide +Dividend= +Divisor=
[2]   Dividend←⎕fi Dividend ⋄ Divisor←⎕fi Divisor
[3]   → (Divisor≠0)ρl0
[4]   )output division by zero +error
[5]   l0:)output dividend is (⍕Dividend) +message
[6]   )output divisor is (⍕Divisor) +message
[7]   quotient←Dividend÷Divisor

ᵘ divide 22 7
dividend is 22
divisor is 7
3.142857143
ᵘ divide 3 0
division by zero
  divide 3 0
  ^
```

Note that *output +result* cannot be used in a script who's header defines an explicit result, as in the example above. To use *output +result*, we would have to rewrite the examples as shown below:

```
ᵘ display .dick cmds.divide
.dick.cmds.divide[s2]:
[1]   divide +Dividend= +Divisor=
[2]   Dividend←⎕fi Dividend ⋄ Divisor←⎕fi Divisor
[3]   → (Divisor≠0)ρl0
[4]   )output division by zero +error
[5]   l0:)output dividend is (⍕Dividend) +message
[6]   )output divisor is (⍕Divisor) +message
[11]  )output (⍕Dividend÷Divisor) +result

ᵘ divide 22 7
dividend is 22
divisor is 7
3.142857143
ᵘ divide 3 0
division by zero
  divide 3 0
  ^
```

LOGOS Commands: *references*

The *references* command displays a list of end environments which contain copies of the objects specified in the pathname arguments.

Syntax

```
ref[erences] [pathnames]
        [+audit]
        [+delete]
        [+environments=envs]
        [+headings]
        [+pathname]
        [+recursive[=1|2|all]]
        [+unreferenced]
```

pathnames Is a list of pathnames to be searched for in end environments. **pathnames** may be elided when *references* is used to delete environments specified in the *+environments* modifier.

+audit Provides complete information about the exact location of objects in the various end environments. This modifier does not take a value. LOGOS keeps a record of the audit file which was used when the paths were created. If *+audit* is not selected, a report containing just the end environments and their types displays.

+delete Deletes the environments selected with *+environments* from the used list.

+environments=**envs**
Specifies the end environments to be searched for references (see the Usage notes, below).

+headings Prints a heading, labelling the various columns of output, at the top of the output.

+pathnames Specifies that the result is to contain full pathnames only. This modifier is used in conjunction with the *+environments* and *+unreferenced* modifiers.

+recursive[=*1*|2|*all*]
Controls iteration through directory levels encountered. *+recursive* (with no value), *+recursive=0*, and *+recursive=all* signify that the named level and all its descendants are to be searched. *+recursive=1* signifies only the named level. *+recursive=2* signifies direct descendants of the named level, excluding the named level. If this modifier is not specified, *+recursive=2* is assumed.

+unreferenced Identifies objects which are not referenced by any applications.

Result

All of the locations in which the specified objects are found are listed in the result returned by this command. Varying levels of detail can be controlled with the use of the modifiers.

Usage

The *references* command is particularly useful when you are considering making any potentially harmful changes to a system, such as deleting an object. It enables you to see all end environments in which the object is used before making the change.

The *references* command resolves links in its pathname arguments. Include `[l]` at the end of the pathname to disable link resolution. If *util.vtom* is a link, *references util.vtom* lists the references to the object to which the link refers, and *references util.vtom*`[l]` lists the references to the link itself.

The argument to the *+environments* modifier can contain one or more environment selection templates. Multiple templates are delimited by a separator character. The separator character is identified as the first non-alphanumeric in the modifier's argument.

An environment selection template is divided into two parts, separated by a comma. The first part is the end environment specification consisting of type, name, and location; the second part is an optional audit file name.

Type is a single letter indicating the end environment type. This can be one of:

c	Cluster
f	File component
l	Link
p	Paging area
s	Script
w	Workspace

Name and location of the end environment depend on the type of end environment, as shown in the following table.

Rules Governing Name and Location of End Environment

If:	Then:
the type is f, p, or w,	the name must consist of an account number and name.
the type is c, l, or s,	the name must be a pathname.
the type is f,	the location must be the component number within the file.
the type is p,	the location must be the paging area's starting component number.

The audit file name contains a single file name field. For example:

+environments=w 1234567 mysys, 1234567 audit

This specifies the workspace *1234567 mysys* which is a part of the audit file *1234567 audit.*

Limited regular expressions can be applied to the environment specification. The following example selects all data files and paging areas on account *1234567.*

*+environments=f|p 1234567 ?**

If the comma and audit file name are elided, end environments are selected whether or not they are kept track of by an audit file.

If the comma is present and the audit file name elided, then only environments that are not kept track of by an audit file are selected.

Examples

```
u references .john.source.utils.vtom
f 1234567 fnsfile 10
p 1234567 paging 25 1234567 auditpg
s .john.source.cmds.print
```

Displays the three end environments which are known to use *.john.source.utils.vtom.* Shows:

- the reference type (*f*, *p*, and *s* above)
- the name of the end environment in which the object was found
- the component number if it's a file or the starting component number of a paging area
- the audit file used in the generation of the paging file

The end environments listed are component 10 of *1234567 fnsfile,* one or more nodes in the paging area *1234567 paging 25,* which is being tracked by the audit file *1234567 auditpg,* and the script *.john.source.cmds.print.*

```
u references .john.source.utils.vtom +headings
--------pathname-------- type ---end environment--- -loc- -----audit file-----
.john.source.utils.vtom       f        1234567 fnsfile      10
                              p        1234567 paging       25       1234567 auditpg
                              s        .john.source.cmds.print
```

Produces the same report as above with column headings.

```
u references .john.source.utils.vtom +headings +audit
--------pathname-------- type ---end environment--- -loc- -----audit file-----
.john.source.utils.vtom       f        1234567 fnsfile      10
                              p        1234567 paging       25       1234567 auditpg
version=23       references=5    saved=26apr89 23:10
pages=calculate edit    erase   model   print
                              s        .john.source.cmds.print
```

Expands upon the information being tracked in the audit file. Shows the version number used in the generation, the number of references to the object, and the saved timestamp. Prints a list of the individual nodes which contain the object.

LOGOS Commands: *register*

The *register* command registers an object out or in, or alters its registration potential.

When you *register out* an object, any other user who tries to get the object is warned that you're working on it. Another user can't change the object unless he overrides its registration. If another user overrides your registration, you are warned the next time you access the object.

When you *register in* an object, it is available to any user with sufficient permission.

The registration potential of an object is *on* or *off*. This refers to its propensity to be registered out automatically whenever it is opened for modification with the *edit* command.

Syntax

```
reg[ister] out|in|on|off pathnames
        [+conditional]
        [+override]
        [+recursive[=1|2|all]]
```

out or *in*	Indicates whether the named objects are to be registered out or in.
on or *off*	Indicates whether the registration potential of an object is to be on or off.
pathnames	Is a list of objects to be registered.
+conditional	Conditionally alters an object's registration state, depending on whether or not it has registration potential set. Objects without registration potential are ignored. This modifier has no effect when the first argument to the *register* command is *on* or *off*.
+override	Overrides any existing out-registration of the specified objects by another LOGOS user. When registration is overridden, the original registrant is notified the next time he accesses the object. If you omit *+override*, existing registration is not overridden, and any attempt to register an object already registered is rejected.

+recursive[=*1* | *2* | *all*]
Controls the level of recursion to be applied to the named level's subordinate directories. *+recursive* (without a value), *+recursive=all*, or *+recursive=0* indicate that the named level and all subordinate directories are to be registered. *+recursive=1* registers just the named level. *+recursive=2* registers just the direct descendants of the named level, excluding the named level itself. If you omit *+recursive* altogether, *+recursive=1* is assumed.

Result

The command returns the names of the paths whose registration was altered as the result. By default, this result is not displayed.

Usage

An object's registration out is not overridden unless you have authority to do so and you specify *+override.*

If **pathnames** specifies or implies a directory, only that directory is registered, unless you specify *+recursive.*

Registration potential set on a directory is automatically passed on to any objects subsequently created beneath that directory.

Examples

```
ᵁ register out util.vtom
```

Registers out *util.vtom.*

```
ᵁ register out util.Δrcat util.Δvtom
already registered: util.Δvtom
```

Registers out *util.Δrcat.*

```
ᵁ register in util.Δvtom
```

Registers in *util.Δvtom.*

ᵁ *register on util*

Sets on registration potential of directory *util*. New objects created under it will inherit this property and, whenever edited, will be registered automatically.

```
[1]  myed +Path=;source;type;.proj.tools.edit.med
[2]  )type←list \Path +data=type ⍝ get object type
[3]  →(1=⍴type←1↓type)⍴l0 ⍝ extract type specifier
[4]  )output only one path can be edited at a time! +error
[5]  l0:)register out \+Path +conditional ⍝conditionally register object out
[6]  )source←display \Path +nopathname ⍝ capture source
[7]  source←med source ⍝ edit source with private editor
[8]  )save \Path[(⍎type)] +value=⍎source ⍝ save edited source
[9]  )register in \+Path +conditional ⍝ conditionally register it back in
```

You can use *register +conditional* to write a script that allows you to use your own personal editor to edit LOGOS objects, while mimicking the LOGOS editor's handling of registration.

LOGOS Commands: *replace*

The *replace* command changes all instances of one or more strings (or regular expressions) in a set of LOGOS objects with another string or strings.

Syntax

rep[*lace*] **oldstring newstring pathnames**
  [*+display*]
  [*+flags=c* | *l* | *n* | *o* | *q* | *s* | *x*]
  [*+lines*]
  [*+multiple*]
  [*+override*]
  [*+recursive*[*=1* | *2* | *all*]]
  [*+show*]

oldstring Is the string to be replaced. Enclose it in quotes if it contains blanks or the command separator. Enclose it in braces({ }) if it is a regular expression.

newstring Is the string to replace **oldstring**. Enclose **newstring** in quotes if it contains blanks or the command separator. Enclose it in braces({ }) if it is a regular expression.

pathnames Specifies the paths in which **oldstring** is to be replaced.

+display Displays the entire line on which matches and replacements occurred. If none of *+display*, *+lines*, or *+show* is specified, *replace* returns only the pathnames, types, and version numbers of objects which were modified.

+flags=c | l | n | o | q | s | x

Specifies the parts of objects to be included in the search. If you omit *+flags*, the entire object is searched. This search is *nonsyntactic*. The command searches the object as an undifferentiated character vector. A function's comments and character constants are included in the search. Note: the flags *c, l, q,* and *x* are effective only when you specify them with the flags *s* or *o*.

c	Searches function comments.
l	Searches LOGOS comments (LOGOS comments begin with ⍝∇).
p	Searches the composite elements of numeric vectors. See the usage note below.
o	When specified with the flags *c, l, q,* or *x*, restricts the search to those types of lines.
q	Searches quoted strings (character constants).
s	Syntactic search. Searches for occurrences of **oldstring** as a syntactic unit within the non-comment lines of the objects being searched. If any of the flags *c, l, q,* or *x* are also specified, those types of lines are included. If flag *o* is also specified, restricts search to only those types of lines.
x	Searches executed quoted strings (character constant that are arguments to ⍎ functions).

+lines

Displays line numbers of lines in which **oldstring** is replaced. If you specify none of *+display, +lines,* or *+show, replace* returns only the extended pathnames, indicating object type and version number, of objects which were modified.

+multiple

Searches for or replaces multiple strings. If you specify *+multiple*, the first character of both **oldstring** and **newstring** is taken to delimit the multiple strings. If you omit *+multiple*, both **oldstring** and **newstring** are taken to specify one string each.

+override

Overrides registration of modified objects so that the modified version can be saved.

+recursive[=*1* | *2* | *all*]

Indicates the level of directories subordinate to the named level which are to be searched. *+recursive* (without a value), *+recursive=all*, or *+recursive=0* indicate that the named level and all subordinate directories are to be searched. *+recursive=1* searches only the named level. *+recursive=2* searches only the direct descendants of the named level, excluding the named level itself. This latter specification is the default behaviour if you omit this modifier altogether.

+show Displays the entire line on which **oldstring** was replaced, with a caret (∧) indicating where the replacement was done. If you specify none of *+display*, *+lines*, or *+show*, *replace* returns only the pathnames, types, and version numbers of objects which were modified.

Result

The command returns the display of all replacements, in the format you requested.

Usage

If you specify *+multiple* and one string in **newstring**, several strings in **oldstring** are replaced by the same string. Also, **oldstring** is assumed to specify several strings to be sought, and the first character of **oldstring** is taken to delimit these multiple strings.

If you also specify *+multiple*, **newstring** is assumed to specify several strings, and the first character of **newstring** is taken to delimit these multiple strings.

Examples

```
∪ replace ALF ¯10↓ALF comp
```

Replaces all occurrences of *ALF* in object *comp* with ¯10↓*ALF*.

```
∪ replace /¯1/0/1    /QN1/Q0/Q1 comp +multiple +flags=s +lines
```

Replaces syntactic occurrences in *comp* of ¯*1* with *QN1*, *0* with *Q0*, and *1* with *Q1*. Also displays the line numbers.

```
∪ replace {((ρα),1)ρα←} ⍕ dir dirx +recursive +show
```

Replaces all occurrences of the regular expression {((ρα),1)ρα←} with ⍕ in all objects in paths *dir* and *dirx*. The context of each replacement is displayed.

LOGOS Commands: *retain*

The *retain* command specifies the maximum number of versions of a path to be stored in LOGOS.

Syntax

```
ret[ain] count pathnames
        [+recursive[=1|2|all]]
```

count Is the maximum number of versions to be retained. The largest number you can specify is 255. *all* or *0* indicates that all versions should be retained. If the specified retention count is less than the number of versions currently stored, the oldest versions of the paths are deleted. The number of versions stored never exceeds the active retention count.

pathnames Is a list of pathnames to which the retention count is to be applied.

+recursive[=*1*|2|*all*]
Indicates the level of recursion through directories subordinate to the named level to which the retention count applies. *+recursive* (without a value), *+recursive=all*, or *+recursive=0* include the named level and all subordinate directories in the retention count specification. *+recursive=1* includes only the named level of **pathnames**. This is the default behaviour if you omit the modifier altogether. *+recursive=2* include only the direct descendants of the named level, excluding the named level itself.

Result

The command returns the pathnames on which retention was changed as the result. These are not displayed unless requested by assignment.

Usage

A retention count of 1 avoids saving back versions of an object.

A path's retention is copied from its parent directory at the time the path is created.

The *retain* command may also be applied to an alias-level path. The retention for an alias-level path, and hence all paths created under it, is set to 10 by default.

Examples

---

u *retain 20 .dba +recursive*

Establishes a retention count of 20 for all paths under *.dba*.

---

u *retain all .dick.modules.util.?**

Establishes boundless retention for paths under *.dick.modules.util.*

LOGOS Commands: *save*

The *save* command saves directories, objects, and their attributes in the LOGOS file system.

Syntax

s[*ave*] **pathnames**
  [+*in*]
  [+*makedir*]
  [+*override*]
  [+*protect*]
  [+*task*[=**task**]]
  [+*value*=[±]**value**]
  [+*workdir*=**pathname**]

pathnames	Is a list of pathnames the saved objects are to have.
+*in*	Cancels the object's out-registration if you set it. If you omit +*in* and the object is registered out, the object's registration remains in effect.
+*makedir*	Allows the creation of intermediate directories implied in **pathnames**. If you omit +*makedir*, intermediate directories are not created, and the save is aborted.
+*override*	Overrides the object's registration by another user. If you omit +*override* and the object is registered to another user, you cannot save it.
+*protect*	Indicates that existing paths are not to be saved over. If you omit +*protect*, the paths specified in **pathnames** can be saved over existing paths.
+*task*[=**task**]	Specifies the auxiliary task in whose active workspace the object to be saved is found. If you specify +*task* without an argument, the default auxiliary task *aux* is assumed. If you omit +*task*, the object is found in the *current execution environment*, unless +*value* is specified, in which case the object is defined in-line. The current execution environment is specified by *environment task*, and may be the active workspace or an auxiliary task.

+value=[⍎]**value**

Specifies the value of the object being saved. If the argument begins with ⍎, it is evaluated in the active workspace, and its result becomes the value; otherwise, the argument directly becomes the value. Note that use of ⍎ in this context differs from the LOGOS ⍎ command, in that the latter always returns a character vector result, while <*save +value*=⍎**value**> saves the result of the expression without further manipulation.

+workdir=**pathname**

Specifies the directory into which pathnames not specified from the root are to be stored. If you omit *+workdir*, the primary global working directory is used.

Result

The command returns the extended pathnames, indicating object type and version number, of the saved objects as the result.

Usage

The *terminal segment* of the **pathname** argument normally specifies the workspace object to be saved.

If you don't specify an object type (for example, *modules*[*d*]), objects at the second level (right below an alias) are assumed to be directories. Objects at all lower levels are assumed to be functions or variables. To save an empty directory, enter a command of the form:

```
∪ save test.alpha[d]
```

Version numbers are incremented, by default, only when the source attribute is saved. You can always explicitly specify the version to be saved.

Use *+value* when you're saving an object not found in any workspace, or when you're saving an object attribute other than its source. *+value* is evaluated once for every name being saved.

Examples

```
∪ save test.alpha.vtom
```

Saves the object *vtom* from the current execution environment.

```
∪ save test.alpha.⎕io +value=⍎0
```

Saves a scalar numeric 0 value for ⎕*io*.

```
∪ save test.alpha.⎕io[:n] +value=nondefault origin required
```

Saves a broadcast note on the nondefault ⎕*io*.

```
∪ save test.alpha.text[:n]
```

Saves the variable *text* from the current execution environment as the broadcast note attribute for *test.alpha.text*.

```
∪ ⍎cnt←⎕io−1
∪ save cm1 cm2 cm3 cm4 +value='⍎>(a⊃b⊃c⊃d)[cnt← cnt+1]'
```

Saves four paths, each with a different value taken from the workspace objects *a*, *b*, *c*, and *d*, respectively.

LOGOS Commands: *send*

The *send* command transmits input to and receives output from an S-task signed on with the *signon* command.

Syntax

```
se[nd] [line]
        [+asynch]
        [+break]
        [+immex]
        [+retract[=on|off]]
        [+suppress]
        [+task[=task]]
```

line	Is a line of input to be sent to the S-task. If you omit **line**, no input is sent to the S-task. *send* ' ' sends an empty line to the S-task.
+asynch	Sends the **line** and does not wait for output. The command returns immediately with an empty result. If you omit *+asynch*, the command waits for the S-task's response to **line**.
+break	Sends a break signal to the S-task.
+immex	Checks that the S-task is in immediate execution mode, and, if it isn't, sends a break or *input interrupt* signal to the task to force it into immediate execution mode. If the task is not in immediate execution mode and the attempts at forcing it fail, the command fails. If you omit *+immex*, line is sent regardless of the S-task's mode.
+retract [=*on* | *off*]	Requests or cancels permission to retract the shared variable used to interface with an auxiliary S-task. The shared variable is global to your workspace, so retract permission is required only if you plan to clear the workspace, load another workspace, or explicitly retract or expunge the variable. If you specify *+retract* without an argument, retract permission is requested.
+suppress	Specifies that an error signalled as a result of execution of line in the S-task is not to cause the *send* command to fail. If you omit *+suppress*, signalling of an error as a result of execution of the input line by the auxiliary task causes *send* to fail, and halts execution of the command line or script.
+task[=**task**]	Provides the name of the task to which the line is to be sent. If you specify *+task* without an argument, the default task name *aux* is used. If you omit *+task*, the task named in *environment task*, if any, is used.

Result

Usually, the result of *send* is all of the output generated by the execution of **line** in the S-task. If you omit **line**, no input is sent to the S-task but any pending output from the task is passed back as the command result. (Output might be pending if you've used *send* earlier with *+asynch.*)

Usage

Two types of event in the S-task are considered errors by the *send* command. If **line** is a system command, then messages such as *incorrect command* or *ws locked* are considered errors. If **line** is not a system command, any output that resembles a canonical error display is considered an error. The *+immex* modifier is useful when **line** is a system command.

For example, if the command *)send)clear +immex* appears in a script, the caller of the script can be certain that the *)clear* command will be successfully issued or that the script will halt.

Regardless of retract permission, an S-task that is in immediate execution mode will end if the shares to it are broken. Retract permission meaningfully applies only to S-tasks that are doing work or are in *)keyboard lock* state.

Examples

ᵁ *send)load state +immex*

Loads workspace *state* in the S-task, and forces immediate execution mode.

ᵁ *send ⎕pdef names +suppress*

Defines the contents of package *names* in the S-task, suppressing errors in the *⎕pdef.*

```
∪ send '''a'' ⎕nl 3'
```

or

```
∪ send (⍕ 'a' ⎕nl 3)
```

Finds all functions in the S-task which begin with the character *a*.

LOGOS Commands: *share*

The *share* command extends or revokes access to LOGOS paths, to specific users or groups of users enrolled in LOGOS. This command also returns information about current permission on paths.

Syntax

sh[*are*] **aliases pathnames**
  [*+delete*]
  [*+permission=c* | *w* | *r* | *x*]
  [*+recursive*[*=1* | *2* | *all*]]

aliases Is a list of LOGOS users to whom access is to be extended or revoked. An alias of ⎕*all* indicates all LOGOS users. Enclose multiple aliases in quotes.

pathnames Is a list of paths to which access is to be granted or revoked.

+delete Revokes the permission designated in the *+permission* modifier from the aliases and paths named in the arguments. If you do not specify *+permission,* all access is revoked from the specified aliases. If you do not specify *+delete,* access is granted to the specified aliases.

+permission=c | *w* | *r* | *x*
Specifies the permission to be extended to (or revoked from, if *+delete* is selected) each alias. The argument to this modifier may be one or more of the following:

c	Control permission
w	Write permission
r	Read permission
x	Execute permission

If you omit *+permission,* the command by default grants no permission. If you specify *+delete,* it revokes all permission from aliases.

+recursive[*=1* | *2* | *all*]
Controls access to only certain levels of paths.

+recursive=all	Grants or revokes access to the named level and all subordinate paths, as do *+recursive* (with no value) and *+recursive=0.*
+recursive=1	Grants or revokes access to the named level only. This is the default behaviour if you omit *+recursive.*
+recursive=2	Grants or revokes access to the named level's direct descendants only, excluding the named level itself.

Result The command returns a list of pathnames, aliases, and permissions as the result.

Usage The following are some useful forms of this command.

Selective inquiry:

share **aliases pathnames**

Full inquiry:

share ' ' **pathnames**

Selective grant:

share **aliases pathnames** *+perm=rx*

Selective revoke (all permission):

share **aliases pathnames** *+delete*

More selective revoke:

share **aliases pathnames** *+delete +perm=rx*

Full revoke:

share ' ' **pathnames** *+delete*

Without the specification of modifiers, the command acts in an inquiry capacity only, and returns information on the permission of the named aliases to the named paths. If aliases is empty (' '), the permission of all users to the named paths is returned.

Specifying the *+permission* modifier requests a grant operation. Additionally or exclusively specifying *+delete* requests a revoke operation.

The permission of all aliases to a set of pathnames can be revoked in a single operation by specifying aliases of ' ' and selecting the *+delete* modifier.

The following table summarizes the effect that a user sees for each type of access for objects and directories:

Permission level	**Effect for objects**	**Effect for directories**
x	use object form	match specific directory entries
r	access source form and attributes	list directory entries
w	save source and attributes	save new objects, delete objects
c	set and show permission to object	set and show permission to object and subdirectories

Examples

```
u share ' ' .scl.inven.post
.scl.inven.post dba      rx
                klh      wrx
                mde      rx
```

Displays aliases with access to *.scl.inven.post* with their access levels.

```
u share dba .scl.inven.post
.scl.inven.post dba rx
```

Displays *dba*'s access to *.scl.inven.post.*

```
u share □all .scl.inven.post|find +permission=x
.scl.inven.find □all     x
.scl.inven.post □all     x
                dba     rx
                klh    wrx
                mde     rx
```

Grants execute access to *.scl.inven.find* and *.scl.inven.post* to all users.

```
u share '□all mde' .scl.inven.post +delete +permission=rx
.scl.inven.post dba     rx
                klh    wrx
```

Revokes read and execute access to *.scl.inven.post* from all users and from *mde.*

```
u share klh .scl.inven.post +delete +permission=r
.scl.inven.post dba     rx
                klh    w x
```

Revokes *klh*'s read permission to *.scl.inven.post.*

LOGOS Commands: *shell*

The *shell* command generates a shell. A *shell* is an APL function which "covers" a node in a paging file or a component in an ordinary APL file. Localized in its header are all the functions and variables which the root function of that node or file component might need to call. Once the shell is invoked, it pages in the node or reads the component from file, and executes the root function.

Syntax

```
she[ll] [destination] [source]
        [+audit=filename]
        [+compile=directives]
        [+exclude=[,|/]names]
        [+file=filename]
        [+header=,|/names]
        [+lock=passnumber]
        [+name=name]
        [+qlx=expression]
        [+skeleton=pathname]
        [+task[=task]]
        [+variant=a|e|l|r|s|t]
```

source	The package or cluster that is to be the source of the shell may be taken from either a **node** of a LOGOS paging file, or a component (N) of an APL file that contains a package.
destination	The valid destinations depend on the source, as shown in the following table:

Source	Destination	
NODE	NODE	The source and destination node names must be the same. In fact, only the destination need be specified; the source will be assumed. Use *+name* if you want to give the shell function a different name from that of the node's root function. This combination must be used in conjunction with the *build* and *filesave* commands that create NODE.
N	<>	The shell function will be placed into the workspace - either your active workspace, or in an auxiliary task. Requires *+name* to provide the name of the shell function, unless *+skeleton* is specified. Requires *+file* to specify the name of the source file.
	CN	When the destination is another component number, the shell function will be placed into that component. N and CN must be different. Requires *+name* to provide the name of the shell function, unless *+skeleton* is specified. Requires *+file* to specify the name of the file that is both the source of the package and the destination for the shell.
	PATHNAME.	If the destination is a LOGOS pathname, the shell function is stored in that path. Requires *+name* to provide the name of the shell function, and this name must match the terminal segment of the pathname. Requires *+file* to specify the source file.

+audit=**filename** Identifies an audit file to contain information about this generation. If you omit *+audit*, no audit file is used.

+compile=**directives**
Specifies compilation directives for the pathname named in the *+skeleton* modifier. If you omit *+compile*, only compilation directives specified in the object or latent in your environment are used.

+*exclude*=[, | /]**names**

Indicates node names which are not to be included in the analysis of the shell or in its header. If you omit +*exclude*, all the names included in the package are included in the function header. If you omit both , and /, **names** forms the complete exclusion list.

,	Adds the objects specified in **names** to any exclusion list established by a global *shell* command.
/	Removes objects specified in **names** from any exclusion list established by a global *shell* command.
names	Is a list of objects to form the exclusion list, or to be added to or removed from the global exclusion list. Separate names in **names** with blanks.

+*file*=**fileid** Identifies the destination and source file.

+*header*=, | /**names**

Specifies local names to be added to or deleted from the shell function's header.

,	Adds the names specified in **names** to the shell header.
/	Removes the names specified in **names** from the header.
names	Is a list of names to be added to or removed from the locals list. Separate names by blanks.

+*lock*=**passnumber**

passnumber specifies the destination and source file's passnumber. If you omit +*lock*, no passnumber is used.

+*name*=**name** Specifies the shell's name. If you omit +*name*, the name of the primary root function of the source is used. Required if the source is a component from an APL file.

+*qlx*=**expression** Specifies an alternate APL expression to be used to start the contents of the shell. The default is based on the shell's name.

+*skeleton*=**pathname**

Specifies a function that is to be used as the frame for the shell. See the usage note below.

+*task*[=**task**] Identifies the task whose active workspace will receive the resultant shell if <> is specified as the destination. If you specify +*task* without an argument, the default auxiliary task *aux* is assumed.

+variant=a | e | l | r | s | t

Specifies shell generation controls for shells constructed from a node of a LOGOS paging file.

a	Supports ambivalency. If the function's header is dyadic, the left argument may be elided. If you omit *a*, all arguments must be supplied.
e	Includes ⎕*ec* in the header. If you omit *e*, the shell's header includes ⎕*ec* only if the root function's header does.
l	Locks the shell.
r	Traps *result error*. If you omit *r*, *result error* is not trapped.
s	Causes paging to be self-contained. The shell is to contain all the code to page in the package. If you omit *s*, the shell calls the LOGOS paging utilities.
t	Causes the shell's stack level to be transparent to existing signalled exits from within the shell. If you omit *t*, signalled exits are not passed upward.

Result

None.

Usage

If the header of the root function is *z←x fn1 y;a;b;c*, and it calls functions *fn2* and *fn3*, the shell function's header is built as *z←x fn1 y;fn1;fn2;fn3*. If you specify *+skeleton=fn1*, so that the stored version of function *fn1* is used as a frame, the shell function's header becomes *z←x fn1 y;a;b;c;fn1;fn2;fn3*.

It is important to note that shell functions built around packages from an ordinary APL file assume by default that the root function in that package (your argument to *+name*) does not take any arguments and does not return a result. If you want to create a shell function that returns a result, or accepts arguments, then use *+skeleton* to provide the pathname of a function with the appropriate syntax and definition to which *shell* can add the locals list.

Examples

```
∪ shell <> 30 +header=,ask Δqi /start +file=termdrivers +name=input
```

Builds a shell around component 30 of the file *termdrivers* and deposits it in the workspace. The shell's header includes *ask* and Δ*qi* in its locals list and removes *start*.

```
∪ shell 10 30 +file=termdrivers +name=input +qlx=getinput
```

Creates a shell called *input* from component 30 of the file *termdrivers*, and writes it to component 10 of the same file. The default start up expression is replaced by a call to the function *getinput*.

```
∪ display .proj.cmds.buildshell[s] +n
[1] buildshell
.
.
.
[5] )build +depth=all +file=projpage 40
[6] )build start
[7] )shell start +name=input +v=als
[7] )filesave
```

This script creates a shell around the node *start* of the LOGOS paging file *projpage*. The shell will have the name *input*. The variants indicate that the shell will be constructed so that a left argument will be optional; the shell will be locked; the code to page in the shell objects will be self-contained in the shell - no utilities will be called.

LOGOS Commands: *si*

The *si* command displays the LOGOS execution stack.

Syntax *si*

Result The command returns a display similar to that generated by the APL system command)*si*, with scripts being denoted by their pathnames.

Usage This command is most likely to be used in debugging mode, where you would type)*si*.

Example

```
∪ lrep
invalid script parameter: \Pathname
.mde.logos.mycmds.listc[4] )t←list \Pathname
                                   ∧
*debug*
    )si
.mde.logos.mycmds.listc[4] *
.mde.logos.mycmds.lrep[5]
logos[20]
logos[2]
```

LOGOS Commands: *signon*

The *signon* command signs on an auxiliary S-task. An S-task started with this command begins with a clear workspace. You communicate with an auxiliary S-task with the *send, talk,* and *transfer* commands.

NOTE: The maximum number of S-tasks that can be signed on to any given account is an APL startup parameter. The value varies by site and over time. The only indication that you've reached the limit is the message *number in user* in the S-task processor, or a result of *3 0* from ⎕*run.*

Syntax

```
sig[non] [usernumber|alias[:password]]
        [+retract]
        [+task[=task]]
```

usernumber	Is the user of the S-task. If you omit **usernumber** or **alias**, the S-task is signed on with your user number.
alias	Is the alias. This alias' primary user number identifies the user of the task. If you omit **usernumber** or **alias**, the S-task is signed on with your user number.
:	Indicates that you are going to specify a password. If you specify **user number** or **alias** but omit the colon (:), you are prompted for a password, unless the **usernumber** you specify or the user number represented by **alias** is yours.
password	Is the user's password. If you omit **password** but specify the colon (:), the password is assumed to be empty.
+retract	Requests permission to retract the shared variable used to interface with an auxiliary S-task. The shared variable is global to your workspace, so retract permission is required only if you plan to clear the workspace, load another workspace, or explicitly retract or expunge the variable.
+task[=**task**]	Specifies the name by which the auxiliary task may be referenced. If you specify *+task* without an argument, or omit the modifier altogether, the default auxiliary task *aux* is assumed.

Result

The command returns the name of the task signed on as the result. By default, this result is not displayed.

Examples

```
u signon
```

Signs on an S-task with your user number and with name *aux*.

```
u signon 1234567:sesame
```

Signs on an S-task for user *1234567*, whose password is *sesame*.

```
u signon devel:poppy
```

Signs on an S-task for the user identified by LOGOS alias *devel*'s primary user number. That user's password is *poppy*.

```
u signon devel
```

Signs on an S-task with the same alias as above, but in this case, you will be prompted for the password.

```
u signon +task=build
```

Signs on an S-task with your user number with the name *build*.

LOGOS Commands: *snap*

The *snap* command stores workspace data in the LOGOS file system. With the use of certain modifiers (*+audit* and *+script*), *snap* can optionally build images of a workspace in the form of audits and scripts.

snap analyzes your workspace, and saves those objects which:

- were fetched from LOGOS and altered in the workspace
- have never been stored in LOGOS
- are stored in LOGOS, but available tracking information is insufficient to determine their location.

Objects in the first case are stored in the path where they originated, and objects in other two cases are stored in the primary working directory.

Syntax

snap [**namelist**]
    [*+audit*=**filename**]
    [*+confirm*]
    [*+exclude*=**names**]
    [*+makedir*]
    [*+overwrite*]
    [*+script*=**pathname**]
    [*+view*]
    [*+workdir*=**pathnames**]

namelist	Is a list of workspace objects which are to be analyzed. namelist may be a list of objects or limited regular expressions. If the argument is omitted, all workspace objects are analyzed.
+audit=**filename**	Identifies an audit file to contain information about the location of objects analyzed. If you omit *+audit*, no audit file is used. *snap* does not use the filename stored in *environment audit*.
+confirm	Requests confirmation of each object to be saved. Confirmation is described in a usage note below. If you omit *+confirm*, objects are saved without confirmation.
+exclude=**names**	Excludes the objects in **names** from analysis. **names** may include limited regular expressions.
+makedir	Allows the creation of intermediate directories implied by *+workdir*.

+overwrite	Generates a new audit record. If you omit *+overwrite*, the existing audit record is used.

*+script=***pathname**

Specifies the pathname in which to save a script describing the objects snapped. If the pathname already exists, new objects are added to the existing script.

+view	Returns a list of pathnames to allow you to preview the objects which have changed in your workspace, without actually moving any of the objects into LOGOS.

*+workdir=***pathnames**

Specifies new working directories for the duration of the command. If you omit *+workdir*, the global working directories are used.

Result

The command returns pathnames of the saved objects, including type and version number, as the result.

Usage

LOGOS counterparts to workspace objects are located through parent pathname comments, the workspace's tracking table, and the current working directories. Only objects which have been changed or are new are saved by *snap.*

System variables other than ⎕*er*, ⎕*ec*, ⎕*ht*, and ⎕*sp* which are set to non-default values are saved by *snap.*

Confirmation Prompts

If you specify *+confirm*, you are prompted to confirm each object to be saved. For example:

save <.scl.inven.post>?

Reply *yes, no, back, continue*, or *stop* to the confirmation prompt.

yes (or *y*)	Saves the object and prompts you for the next path.
no (or *n*)	Does not save the object and prompts you for the next path to be saved.
back	Repeats the previous prompt.
continue	Saves this path and all following paths without further prompting.
stop	Aborts *snap.*

Paths are not actually saved until all confirmations are complete.

Examples

```
∪ snap Δ?⋆
```

Analyzes workspace objects beginning with the character Δ, and saves those which are new or changed.

```
∪ snap +script=.dick.util.gen
```

Analyzes all workspace objects, saving those which are new or changed, and builds a script in *.dick.util.gen* which can be used to generate the workspace.

```
∪ names←snap +view
```

Assigns to *names* the full pathnames of objects which have changed or are not yet stored in LOGOS.

LOGOS Commands: *summarize*

The *summarize* command displays summary information about LOGOS objects.

The summary includes: the object's pathname, type, version number, size, number of lines (if a function or script), rank and shape (if a variable), and number of members (if a package or cluster). The leading or trailing comment of a function or script, and summary information about objects in packages or clusters can also be displayed.

Syntax

sum[*marize*] **pathnames**
    [*+comments*]
    [*+expand*]
    [*+full*]
    [*+headings*]
    [*+recursive*[=*1* | *2* | *all*]]
    [*+versions*[=**n**]]

pathnames	Is a list of objects for which information is to be displayed.
+comments	Displays the first or last line of a function or script if the line contains only a comment. If you omit *+comments*, comment lines do not display.
+expand	Computes and displays summary information for the contents of packages and clusters.
+full	Displays full pathnames. If you omit *+full*, partial pathnames are displayed.
+headings	Displays report headings. If you omit *+headings*, report headings are not displayed.
+recursive[=1 | 2 | *all*]	Indicates the level to which subordinate directories are to be processed. *+recursive* (without a value), *+recursive=all*, or *+recursive=0* specify that the named level and all subordinate directories are to be summarized. *+recursive=1* processes only the named level. *+recursive=2* processes only the named level's direct descendants, excluding the named level itself. This behaviour is the default if the modifier is omitted altogether.
+versions[=**n**]	Displays information about only certain versions of selected objects. If you specify *+versions* without a value, all versions are displayed. If you provide **n**, only the first or last **n** versions are reported, depending upon whether **n** is positive or negative, respectively.

Result

The command returns the requested summary information as the result.

Usage

summarize's result gives each object's LOGOS type, as well as its APL type where appropriate. The following types are displayed:

c (*pk*)	Cluster
f (*ed*)	Function--explicit, dyadic
f (*em*)	Function--explicit, monadic
f (*en*)	Function--explicit, niladic
f (*nd*)	Function--nonexplicit, dyadic
f (*nm*)	Function--nonexplicit, monadic
f (*nn*)	Function--nonexplicit, niladic
f (⍫)	Function--locked
l (-)	Link
s (*en*)	Script--explicit, no parameters
s (*ep*)	Script--explicit, parameters
s (*nn*)	Script--nonexplicit, no parameters
s (*np*)	Script--nonexplicit, parameters
v (*ar*)	Variable--nested array
v (*bl*)	Variable--boolean
v (*ch*)	Variable--character
v (*cm*)	Variable--complex
v (*fl*)	Variable--floating-point
v (*in*)	Variable--integer
v (*pk*)	Variable--package
v (-)	Variable--undefined

summarize also displays for each object information that depends on the object's type:

- If the object is a function, its number of lines is shown in square brackets [].
- If the object is a variable, its rank is shown in parentheses ().
- If the object is a package or a cluster, the number of objects in it is shown in angle brackets <>.

summarize normally resolves links, but through use of the [*l*] notation, it can be induced not to resolve a link and instead to report on the link itself.

The name of a packaged object is formed by extending the pathname, using a jot (∘) as the separator. For example, the object *report* in package *sys.modules* is displayed as *sys.modules∘report*. If *report* is itself a package, the object *print* within it is displayed as *sys.modules∘report∘print*.

Examples

ᵘ *summarize chart report*

Displays a summary of the named objects, or if they are directories, the objects immediately subordinate to them.

ᵘ *summarize chart report +expand*

As above, but expands packages and clusters into their components.

ᵘ *summarize inv?⋆*

Displays a summary of all objects beginning with *inv*.

ᵘ *summarize chart +comments +versions=¯2*

Displays a summary of *chart*, or its descendants, reporting on the last two versions of each object and including leading or trailing function comments.

LOGOS Commands: *syntax*

The *syntax* command computes a report describing static errors within a program. It tests conditions such as illegal characters, symbol juxtaposition problems, mismatched parentheses, brackets, or quotes, and suspicious use of names. *syntax* does not actually execute the program; consequently, a tool such as this command should be used to supplement but not replace careful program and system testing.

Syntax

syn[*tax*] **pathnames**
    [*+all*]
    [*+display*]
    [*+lines*]
    [*+quotes*]
    [*+recursive*[*=1* | *2* | *all*]]
    [*+show*]

pathnames	Is a list of objects on which you want to compute reports.
+all	Computes all errors, including suspicious name references which may not be erroneous.
+display	Displays the entire line on which errors occurred.
+lines	Displays the line numbers on which errors occurred, followed by a symbol denoting the type of error. See the note on result.
+quotes	Causes quoted strings logically appearing after executes (⍎) within the program to be examined as if they were not quoted.
+recursive [*=1* | *2* | *all*]	Iterates through directory levels encountered. The argument to this modifier may be *1*, *2*, or *all*, signifying the named level, its direct descendants but excluding the named level, or the named level and all descendants, respectively. If you omit this modifier, *+recursive=2* is assumed. If you specify *+recursive* without a value, *+recursive=all* is assumed.
+show	Displays the entire line on which errors occurred, with a symbol pointing to each error. The symbol denotes the type of error at that location. Because errors relating to parentheses, brackets, and quotes are obvious, the caret (^) is substituted as the pointer.

Result

Errors are classified into a number of categories. If the *+lines* modifier is selected, the category is represented by a symbol following the line number on which the error was detected. If *+show* is selected, the category is represented by the symbol under the location where the error was detected (errors relating to parentheses, brackets, and quotes are indicated by a caret). The notational symbols and their corresponding error categories are:

^	Generic syntax error
(	Parenthesis error
[	Bracket error
'	Quote error
÷	Domain error
.	Constant error
?	Suspicious reference

Generic syntax errors include most incorrect uses of symbols. For example, a dyadic symbol used monadically; an improper outer product; an improperly labelled line; use of branch not as the root function of a statement; or redundant use of a diamond, all constitute syntax errors.

Parentheses, bracket, and *quote errors* refer to mismatched instances of the paired delimiters `(...)`, `[...]`, and `'...'`, respectively.

Domain errors arise from apparent use of a character argument where a numeric one was expected. As the *syntax* command does not execute the program, only a limited number of such cases is detected.

Constant errors refer to illegal formation of numeric constants. For example, *4..1* and *8je4* are illegal constants, whereas *4.1* and *8j8e4* are legal ones.

Suspicious references denote the use of names which are unusual but may or may not be erroneous in the running application. For example, a local variable which is not assigned a value, or a name which is used to define a line-label more than once, is considered suspicious.

Usage

If none of the modifiers *+display*, *+lines*, or *+show* are selected, *syntax* returns the pathnames and version numbers of those objects in which any error was found.

Examples

```
∪ syntax .mde.logos.test +r
```

Reports all objects in which an object occurred. For example, this command returns the result:

```
.mde.logos.test.aplterm.cov[f1]
.mde.logos.test.aplterm.genchars[f1]
.mde.logos.test.foo[f13]
.mde.logos.test.genreport[f13]
.mde.logos.test.hoo[f13]
```

```
∪ syntax .mde.logos.test +r +lines
```

Reports all the objects in which an error occurred, and the line numbers on which errors occurred, followed by a symbol denoting the type of error. For example, this command returns the result:

```
.mde.logos.test.aplterm.cov[f1]     7^
.mde.logos.test.aplterm.genchars[f1]     11^
.mde.logos.test.foo[f13]     2(   3^
.mde.logos.test.genreport[f1]     54(    65^
.mde.logos.test.hoo[f1]     2(   3^
```

```
∪ syntax .mde.logos.test.genreport +show
```

Displays the entire line on which errors occurred, with a symbol pointing to each error. The symbol denotes the type of error at that location. For example, this command returns the result:

```
.mde.logos.test.genreport[f1]  (2 errors)
[54] pv←((i←∨\mmsk))<u>Q1↓u,Q1) ∨Q1↓mmsk,Q0 ⍝ mask marking last blank in each
field
    ^

[65] L7:pv→pv∧mmsk∨t←na≥+\pv ⋄ →L9 ⋄ v←⊂v∨t ⋄ →L9 ⍝ enforce long right scope rule
     ^
```

LOGOS Commands: *talk*

The *talk* command carries on an interactive session with an auxiliary S-task.

Syntax

tal[*k*] [**task**]
        [+*prompt*=**prompt**]

task	Is the name of the task with which you wish to interact. If the argument is elided, the task named in the *environment task* parameter is used, unless this name is ⋆. In the latter case, the default name *aux* is used.
+*prompt*=**prompt**	Specifies a value to be used as a prompt for your input to the auxiliary S-task.

Result

None.

Usage

A session controlled by the *talk* command is almost completely indistinguishable from normal sessions. Following are the differences.

1 *talk* issues an immediate execution prompt (carriage return followed by six blanks) when the auxiliary task is awaiting input. The +*prompt* modifier can be used to alter the prompt, so that sessions using *talk* can look different from ordinary APL sessions.

2 If you are signed on using a terminal which supports a status line, and you have the full status line enabled, the second line displays a message reminding you that you are communicating with an auxiliary task.

3 Break or Attention causes the prompt *abandon, resume or break:* to be issued. Valid responses are:

abandon	Exits the *talk* command.
resume	Causes the session to resume just after the point at which you signalled the interrupt.
break	causes a break signal to be transmitted to the S-task

4 The pseudo-system command)*disconnect* causes the *talk* command to terminate. In addition,)*logos* invokes a LOGOS command, as in)*logos list* +*column*.)*logos* can be abbreviated if the character after the right parenthesis is a valid LOGOS command separator. For example, the previous command is equivalent to)×*list* +*column*.

5 While in a *talk* session, entering) recalls the last APL expression; entering)) recalls the last LOGOS command entered.

The *talk* and *send* commands may be intermixed during a LOGOS session.

Termination of the *talk* command by way of either)*disconnect* or a response of *abandon* to the interrupt prompt does not result in the termination of the auxiliary task itself. To terminate the auxiliary task, send)*off* to it.

Examples

```
∪ talk
```

Enters into an interactive session with the auxiliary task implied by the *environment task* parameter.

```
∪ talk gen +prompt=' : '
```

Enters into an interactive session with the auxiliary task *gen*, changing the immediate execution prompt for the session.

LOGOS Commands: *tasks*

The *tasks* command displays a report which describes the auxiliary tasks started by the inquiring task. This command can be used to obtain a list of task names, or detailed information regarding specific tasks.

Syntax

```
t[asks] [task]
        [+headings]
        [+users=users]
```

task	Is a list of task names on which to report detailed information. If you enter more than one task name, separate the names with spaces.
+headings	Displays column headings for the report.
+users=**users**	Restricts the command so that it reports only on tasks which are running on accounts listed in **users**. **users** may contain aliases, user numbers, or a mixture of both.

Result

Without an argument, the command returns the names of all auxiliary tasks started by you. With an argument, information is returned for each specified task, including the task number, sign-on time, CPU usage, and an indication of whether or not retract permission was requested.

Examples

```
ᵘ tasks
aux gen logg
```

Displays the names of auxilary tasks started by this user.

```
ᵘ task logg
logg  2023   mde          20apr89 15:29       60       115       yes
```

Displays details on the auxilary task *logg*.

```
ᵁ tasks (tasks) +headings
name--id----account-------signon time---------cpu-----connect--retract
aux   2015   mde          20apr89 15:27         1        286        no
gen   2016   mde          20apr89 15:28         1        270       yes
logg  2023   mde          20apr89 15:29        62        151       yes
```

Displays details on all auxilary tasks along with column headings.

LOGOS Commands: *transfer*

The *transfer* command transfers objects between two tasks.

Syntax

```
tr[ansfer] namelist
        [+from [=task]]
        [+protect]
        [+suppress]
        [+to [=task]]
```

namelist	Is a list of objects to be transferred.
+from[=**task**]	Specifies the task from which the object is to be transferred. If you omit =**task**, the default auxiliary task is assumed. If you omit *+from* altogether, the object is transferred from the local active workspace.
+protect	Does not overwrite objects in the target workspace. This is analagous to the operation of the APL *)pcopy* system command.
+suppress	Indicates that errors in the source or target task are not to cause the *transfer* command to end with an error (thereby interrupting a script that might have issued the command). An error of this type would occur if an object with the same name as one being transferred existed in the destination workspace, and *+protect* was specified.
+to[=**task**]	Specifies the task to which the object is to be transferred. If you omit =**task**, the default auxiliary task is assumed. If you omit *+to* altogether, the object is transferred to the local active workspace.

Result

The command returns the names of any objects which LOGOS was unable to transfer as the result.

Usage

The *transfer* command can also be used to transfer objects between two auxiliary task workspaces. Note that *transfer* only deals with workspace-resident objects. To move objects between the LOGOS filesystem and a workspace, use the *get* and *save* commands, which also work with auxiliary tasks.

Examples

The following examples illustrate the way the *+from* and *+to* modifiers can be used separately or in combination to construct a transfer path between any two LOGOS tasks. The examples here assume that three auxiliary tasks are signed on, one named *aux* (the default name), one named *a*, and one named *b*.

```
u transfer x +to
```

x is transferred from the active workspace to the *aux* task (default auxiliary task).

```
u transfer x +from
```

x is transferred from the *aux* task to the active workspace.

```
u transfer x +to=a
```

x is transferred from the active workspace to the *a* task.

```
u transfer x +from=a
```

x is transferred from the *a* task to the active workspace.

```
u transfer x +from +to=a
```

x is transferred from the *aux* task to the *a* task.

```
u transfer x +from=a +to=b
```

x is transferred from the *a* task to the *b* task.

LOGOS Commands: *version*

The *version* command displays information about the version of LOGOS you are using.

Syntax

ver[sion]

Result

The command returns information about the current version of LOGOS as the result.

Example

```
∪ version
logos version 2.0 (19oct89 20:24)
copyright (c) 1990 Reuter:file Ltd.
```

Indicates that you are using LOGOS version 2.0, which was generated at 20:24 on 19 October 1989.

LOGOS Commands: *whois*

The *whois* command prints the identity of a user or a group in LOGOS. The search performed by *whois* may be upon an alias, a user number, both of these, or a full name.

Syntax

```
who[is] id
        [+all]
        [+extended]
        [+name]
        [+summary]
```

id	Is interpreted as an alias (as in *whois testnum*), as a user number (as in *whois 1234567*), or as an alias and user number pair (as in *whois testnum 1234567*). Searches on user numbers require preferred LOGOS access.
+all	Displays entries for secondary as well as primary alias and user number matches. Primary entries are indicated by an asterisk (⋆) beside the alias (and beside the user number, if *+extended* is selected).
+extended	Displays the enrollment date for each matching user or group, as well as the user number for matching users. This modifier requires preferred LOGOS access.
+name	Treats the argument as a name rather than an alias or group code. A prefix search is assumed, so that an argument of **name** is equivalent to **name***?⋆*. To search for a string anywhere in a name, use *?⋆***name** (equivalent to *?⋆***name***?⋆*).
+summary	Displays the groups for each matching user, as well as the membership for each matching group.

Result

The command returns the requested user or group information as the result.

Usage

Any inquiry on, or reference to, user numbers by this command requires preferred LOGOS access.

+summary reports group membership, which is not displayed by default.

+all is useful if you are performing an inquiry on a user or alias that has secondary entries.

Examples

ᵁ *whois john*

Displays the full name of the person or group enrolled with alias *John.*

ᵁ *whois richards +name*

Displays the full name of any enrolled users or groups whose name begins with *richards.*

ᵁ *whois ?*apl +name*

Displays the full name of any enrolled users or groups whose name contains the sequence *apl.*

ᵁ *whois lhg +summary*

Displays the full name and group membership of the person or group enrolled with alias *lhg.*

LOGOS Commands: *with*

The *with* command applies a series of commands to a list of arguments itemwise. (*with* is a *metacommand.*)

Syntax

wi[*th*] **expression arguments**
  [+*surrogate*=**character**]

expression Is a single LOGOS command, or a series of LOGOS commands separated from each other by the separator character (for example, ᵘ). **expression** must be enclosed in quotes if it includes any blanks or separator characters. **expression** may also contain the argument placeholder metacharacter, α.

arguments Is a list of arguments delimited by blanks or carriage returns.

+*surrogate*=**character**
Uses the character you specify as the argument placeholder instead of α. The symbol used as a surrogate must not be alphanumeric, and may not be one of the following:

```
{ } [ ] ( ) ∆ ⍙ ∇ \ ← + = ? . ' ⎕ ⍎ ⍕
```

Result

The command returns the concatenation of the results of each command executed, if the result would have been displayed. A command whose output is assigned (*x←command*), discarded (*←command*), or which does not normally display a result (for example, *delete*), does not contribute to the composite result returned by *with*. Note that you can always use the ⎕← construct to force a command to display its result -- and hence be included in the result of *with.*

Usage

with extracts the first argument from **arguments**, interpolates it into the expression wherever the placeholder appears, and executes the entire expression. *with* then extracts the next argument from the list and repeats the process. This continues until the argument list is exhausted.

Examples

```
∪ with 'locate ⎕appendr reports.yearend[α]' 4 5 6 7
```

is equivalent to:

```
∪ locate ⎕appendr reports.yearend[4] reports.yearend[5] reports.yearend[6]
reports.yearend[7]
```

```
∪ with 'locate α reports.yearend[ρ]' 4 5 6 7 +surrogate=ρ
```

is equivalent to:

```
∪ locate α reports.yearend[4] reports.yearend[5]
reports.yearend[6] reports.yearend[7]
```

The APL character α is sought in the specified paths, and ρ is made the placeholder for the command.

LOGOS Commands: *workdir*

The *workdir* command establishes or displays your working directories.

The working directory in effect when you are first enrolled in LOGOS is your alias.

Syntax

```
w[orkdir ] [pathnames]
        [+reset]
```

pathnames	Is a list of pathnames specifying the working directories to be established. If you omit **pathnames**, your working directories do not change.
+reset	Resets the working directory to the value saved in your profile.

Result

The command returns the working directories in effect after the operation completes as the result.

Usage

A *working directory* is where the LOGOS file system looks for pathnames not specified from the root.

If you specify more than one working directory, they are searched in the order specified.

Use *environment +profile* to save your working directories in your profile.

Examples

```
u workdir
.dick
```

Displays the current working directory's pathname.

```
u workdir comp
.dick.comp
```

Extends the working directory's pathname.

ᵘ *workdir .lhg.gen.temp.wsfns* (*workdir*)
.lhg.gen.temp.wsfns .dick.comp

Establishes two working directories by introducing one in front of the previous setting.

ᵘ *workdir +reset*
.dick

Resets the working directory from the value saved in your profile.

LOGOS Commands: *wssave*

The *wssave* command saves the active workspace of an execution environment.

Syntax

```
wssave [wsid]
        [+audit=filename]
        [+information]
        [+overwrite]
        [+task[=task]]
        [+user=user]
```

wsid	Is the name under which the workspace is to be saved. If you omit this argument, it defaults to the current workspace name.
+*audit*=**filename**	Identifies an audit file to contain information about this saved workspace generation. If you omit *+audit*, the file named in *environment audit*, if any, is used.
+*information*	Causes the inclusion in the saved workspace of a variable called *∆LINFO*, containing the name of the script (if any) that called *wssave*.
+*overwrite*	Uses a new audit record.
+*task*[=**task**]	Specifies the name of the task whose active workspace is to be saved. If you specify *+task* without an argument, the default auxiliary task *aux* is assumed. If you omit *+task* altogether, the current execution environment, as specified in *environment task*, is saved.
+*user*=**user**	Specifies the user name (alias) or number under which the workspace is to be saved, if other than the account on which you are currently running. If you do not provide an account password, it will be requested with a protected prompt.

Result

The command returns the system save timestamp of the workspace as the result.

Usage

This command is the preferred method of saving a workspace. Only through it will objects reflect their use in a workspace via the *references* command.

Examples

ᵁ *wssave message +user=dick*

Saves the active workspace as message on *dick*'s account.

ᵁ *wssave mymail +task=aux*

Saves the active workspace of auxiliary task *aux* as *mymail.*

LOGOS Commands: *wstofile*

The *wstofile* command builds a source file suitable for input to the SHARP APL Workspace Documentation Facility (WSDOC). This command also allows you to document other LOGOS attributes, such as *compilation directives, documentation,* and *note.*

Syntax

wst[*ofile*] **pathnames**
  [+*attributes*=*c* | *d* | *j* | *n* | *t*]
  [+*file*=**filename**]
  [+*pathnames*]
  [+*recursive*[=*1* | *2* | *all*]]
  [+*state*=**pathname**]
  [+*wsid*=**wsid**]

pathnames Is a list of objects to be incorporated in the WSDOC source file.

+*attributes*=*c* | *d* | j | *n* | *t*

Specifies the non-default LOGOS attributes which are to be documented in addition to the source (which is always documented). Valid attributes are:

c	Compilation directives
d	Documentation
j	Journal
n	note
t	Tag

If you specify +*attributes* with no argument, all object attributes are assumed. This modifier also causes a LOGOS header line, including the pathname, type, version, timestamp, and writer, to appear before each documented object.

+*file*=**filename** Specifies the name of the file to be built. If you omit +*file*, the value of +*wsid* is used. If you omit both +*file* and +*wsid, clear ws* is used.

+*pathnames* Replaces object names by full LOGOS pathnames in the *wsdoc* report.

+*recursive*[=1 | 2 | *all*]

Indicates the level to which subordinate directories are processed. +*recursive* (without a value), +*recursive=all*, or +*recursive=0* process the named level and all directories subordinate to it. +*recursive=1* processes only the named level. +*recursive*=*2* processes only the named level's direct descendants, excluding the named level itself. This latter behaviour is the default if you omit +*recursive* altogether.

+state=**pathname**
Selects the *wsdoc* state to be used in the source file. The pathname specified must be a package that represents valid *wsdoc* state settings, such as would have been set by the *wsdoc* state program..

+wsid=**wsid** Specifies the name of the workspace appearing in the WSDOC report. If you omit *+wsid, clear ws* is used.

Result

The command returns the tie number of the source file built as the result.

Usage

If the specified file already exists and is a valid WSDOC source file, the pathnames to be documented are added to the file. If the file does not exist, it is created.

+wsid sets the file name as well as the workspace name if *+file* is not specified.

Scripts appear as variables in the WSDOC report.

Non-default attributes of directories as well as objects are included in the report, if both *+attributes* and *+recursive* are selected.

Examples

```
u wstofile test.wsfns +attributes +pathnames +wsid=wsfns
srcfile 20 - 1234567 wsfns
38 functions
20 variables
```

A WSDOC source file is created. When processed by *wsdoc*, the summary and definition reports will contain the full LOGOS pathnames, a LOGOS header line, and any attributes associated with the objects.

```
u wstofile test +attributes=dj +recursive
srcfile 21 - 1234567 clearws
145 functions
42 variables (including 11 scripts)
```

A WSDOC source file is created from all objects below *test*. 11 scripts found were included as variables. When processed by WSDOC, the definition reports will contain a LOGOS header line and the documentation and journal attributes of the objects, if set.

```
∪ wstofile doc.∆?*[f] util[f] +file=output +wsid=misc
srcfile 22 - 1234567 output
52 functions
0 variables
```

A single WSDOC source file is created for all functions in two specified paths. The file is given name *output*, but the workspace is identified as *misc*.

LOGOS Commands: *xref*

The *xref* command computes and displays a cross-reference table for functions stored in LOGOS. The cross-reference table gives the location and type of each reference to each identifier used in each named program.

Syntax

xr[*ef*] **pathnames**
    [+*quotes*]
    [+*symbols*=**symbols**]

pathnames	Is a list of objects for which cross-reference tables are to be computed and displayed.
+*quotes*	Examines character constants (quoted strings) that are arguments to the execute (⍎) function as if they are not quoted. If you omit +*quotes*, such character constants are not examined.
+*symbols*=**symbols**	Specifies APL characters to be cross-referenced along with ordinary APL identifiers. If you omit +*symbols*, the APL characters →⍎:⍞⎕ are cross-referenced. *symbols* may be any APL characters.

Result

The command returns the requested cross-reference tables as the result.

Usage

For each identifier found in a program, *xref* computes the identifier's local type, and an indication of whether or not the name occurs "suspiciously". For example, a local variable that is not assigned a value, or a name that is used to define a line label more than once is considered "suspicious".

An identifier's type is one of the following:

la	Left argument
ll	Line label
lv	Local variable
qf	System function ("quad function")
ra	Right argument
rs	Result
⋆	Indeterminate

A suspicious identifier is marked with a query (?) after its type.

For each reference to an identifier in a program, *xref* computes the line number of the reference, and its type.

The reference type is one of the following:

Blank	Simple reference
←	Simple assignment
[	Indexed reference
[←	Indexed assignment
:	Line label definition

A name enclosed in quotes and otherwise not detected as an identifier can be cross-referenced if the ⍝∇∪ codetag is included on the program line. For example:

```
[3] i←124 ⎕svc 'ctl' ⋄ i←1 ⎕svc 'ctl' ⍝∇∪ ctl
```

Examples

```
∪ xref report cmdtop +symbols=→⍎⌶
```

Computes and displays cross-reference tables for paths *report* and *cmdtop*. The tables include references to APL symbols →⍎⌶.

```
∪ xref wkstation +quotes
```

Computes and displays a cross-reference table for *wkstation*. Quoted strings that are arguments to ⍎ are included in the analysis.

COMPILATION DIRECTIVES

About Compilation Directives

Compilation directives, which are described fully in Chapter 10 of the *LOGOS User's Guide*, specify in a general way the actions you want the LOGOS compiler to take on objects in your system. They can be applied to a single object, or to arbitrary groups of objects.

A list of compilation directives may appear in any of three places:

- Saved with the object itself (as the object's `[:c]` attribute)
- As a modifier to a command (for example, *get .mde.utils.vtom +compile=d,R=2,y*)
- In your session profile (as specified by the *environment* command)

Syntax

In general, a compilation directive list has the syntax:

d1[=**val1**],**d2**[=**val2**],...

For example:

a=prolog13,d,x,l,z=.mabra.tools.tscom

Summary of Compilation Directives

The directives are described in the following sections. The following letters indicate the object types with which the directive can be used:

f	function
s	script
v	variable

a=**pathname**	The **User-defined Prologue** directive (f, s, v) executes a user-defined function or cluster immediately before compilation of each object begins.
c=**pathname**	The **Context** directive (f, s, v) defines a context for evaluation. The argument **pathname** must be a cluster. Objects in **pathname** are to be available for evaluation during compilation.
d	The **Diamondize** directive (f) diamondizes the object. Each line is to be merged with the next, separated by a diamond (⋄). Does not merge a line with its successor if it contains a comment, if the following line is labelled, or if either line contains the ⍝∇⋄ code tag.)
e	The **Evaluate** directive (f, s, v) evaluates expressions commented by ⍝∇± code tags.

f

The **Format** directive (v) formats the object by the SHARP APL Text Editor (from workspace *4 edit*).

i=**list**

The **Inclusion** directive (f) includes the tagged segments specified in **list** in the compiled object. The value **list** is one or more numbers, names, or expressions separated by blanks, as in *i=mon test* or *i=1*. Expressions can contain identifiers as well as symbols from the set:

```
( ) ∨ ∧ ⍱ ⍲ ~ = ≠ < ≤ ≥ >
```

and must yield a Boolean result.

Identifiers in the expression are assigned a value of 1 if they correspond to names appearing in the argument to the active *i* directive. All other identifiers have a value of 0.

Expressions are evalutated using standard APL (right-to-left) precedence rules. If the result of the expression is a 1, or if no *i* directive is in force, the line is retained. If an *i* directive is in force and the expression yields a 0, then the line is removed.

For example, a line containing the code tag ⍝∇∊ *unix∨mvs∧test* would be retained if the argument to the *i* compilation directive contained the name *unix*, or the names *mvs* and *test*. Each line of the source can potentially have an inclusion tag indicated by ⍝∇∊. The *i* directive controls which of these tagged segments appear in the object, by selecting any lines whose ⍝∆∊ statements include any values in common with the specified inclusion list.

l

The **Lock** directive (f, s) locks a function or script.

p[=*0*|*1*]

The **Parent Pathname Tag** directive (f) includes the parent directory name.

p=0 (or *p*) appends a comment beginning with ⍝⋆ to the last line of a function. The comment contains the parent directory name from the root, and the object's version number in brackets.

p=1 adds a comment beginning with ⍝⋆ as the last (separate) line of the function. For example, in ⍝⋆*.public.logos.paging.*[*3*]-----, the path *.public.logos.paging* identifies the directory (note the terminal dot, implying extension of the path); the phrase [*3*] is the source's version number; and the ----- is a checksum for the *snap* command.

q

The **Implant Tracking Statement** directive (f) includes the least-recently-used (LRU) statement. This statement generates a record of a function's use for page-out control.

r[=[_]*0*|*1*|2[_]]

The **Rename** locals directive (f) renames local variables and line labels. If you specify _ at the beginning or end of the value, names beginning with characters in the second alphabet are not changed.

*r=*0 renames locals and labels *⍙99*, *⍙98*, *⍙97*, ... , *⍙00*, *⍙199*, ... This is also the scheme used if you specify *r* without an argument.

r=1 renames locals and labels *a, b, c, ... , aa, ab, ...* .

r=2 gives locals and labels random seven-character names.

w=**workdirs**

The **Working Directory** directive (f, s) specifies a set of working directories. These directories are used when searching for objects to be included in a composite script, and when evaluating pathnames in code tags.

x[=∇|*0*|*1*]

The **Excise Commands** directive (f) decomments the object.

x=0 (or *x*) removes all comments from compiled functions. This is the default behaviour if you do not specify *x*.

x=1 retains the opening ⍝ of whole-line comments, thus preventing the renumbering of function lines. Removes end-of-line comments in their entirety.

x=∇ removes both whole-line and trailing LOGOS comments, and retains regular comments.

y[=_]

The **Remove Labels** directive (f) removes line labels.

y removes line labels and replaces all references to line labels with constants.

y=_ removes line labels except for those beginning with letters in the second alphabet and replaces references to line labels with constants.

z=**pathname**

The **User-defined Epilogue** directive (f, s, v) executes a user-defined function or cluster at the end of compilation.

Turning Off Directives

Any directive can be forcibly turned off by using a ~ before its letter. For example, to turn off decommenting, use ~*x*. Expressions of this form provide a convenient way to negate the effect of a directive saved in the [:*c*] attribute of the object.

You may also specify any directive in the second alphabet. For example, *A* or I=*list*. A directive in the second alphabet takes precedence over its simple variant and cannot be overridden. This is discussed below.

Precedence of Directives

The order in which you specify directives is irrelevant. LOGOS computes the set of directives to take effect using the following precedence rules (listed in order of most dominant to least):

- a directive in the second alphabet saved with the object
- a directive in the second alphabet in the user's profile
- a directive in the second alphabet passed as a command parameter
- an ordinary directive (a directive in the first alphabet) saved with the object
- an ordinary directive in the user's profile
- an ordinary directive passed as a command parameter

CODE TAGS

About Code Tags

Code tags, which are described fully in Chapter 10 of the *LOGOS User's Guide*, are special comments which appear in the body of an object. They qualify the actions specified by the compilation directives. Because code tags appear in the source of an object, they are processed object by object. Code tags remain a part of the object after compilation, unless you specifically remove them using the *x* (decomment) compilation directive.

A code tag is distinguished by its opening ⍝∇. ⍝ marks it as an APL comment, and ∇ as a code tag. A symbol immediately following ⍝∇ identifies the code tag's type.

Summary of Code Tags

The code tags are listed below:

⍝∇∪ **namelist**	Union: treats names in **namelist** as if they were explicitly referenced in this line. This is useful for informing LOGOS of 'hidden' references to names. For example, references in quoted arguments to execute or ⎕*trap* definitions. This makes it possible for commands such as *build* and *calls* to process these names. It also enables the WSDOC program to build accurate WSGRAPHS and *xref* listings.
⍝∇; **namelist**	Local: treats names in **namelist** as local to this function and does not perform calling-tree analysis on them. This is used with the *build* and *calls* command. With the *build* command you can specify that an unlocalized name is not to be treated as a global and not to be searched for during analysis.
⍝∇~ **namelist**	Exclude: excludes names in **namelist** from the actions of the *r* (rename) and *y* (delabel) directives.
⍝∇∊ **list**	Inclusion set: includes the entire line only if **list** has any values in common with the set defined by the *i* directive. If the *i* directive is not set, this tag is ignored and all lines are included.

⍝∇⍎ **dlm**	Evaluate expressions: **dlm** is the evaluated expression delimiter for the line. Expressions appearing between pairs of delimiters are evaluated and their results replace the original expressions in the source. Objects referenced within the delimiters must exist either in the workspace or in a path specified by the *c* compilation directive.
⍝∇⋄	Nondiamondize: does not diamondize this line. Does not merge it with the preceding or succeeding line.
⍝∇⍺*text* ⍝∇⍵*text*	User-defined: these are used to include information in an object which is of interest to a user-defined (*a* or *z*) directive. The compiler does not actively process these tags.

Evaluation Environment

The environment in which names in a code tag or in delimited expressions in a line are evaluated depends on how the tag is represented. The following general forms illustrate:

⍝∇ᵁ **name**	Takes **name** literally.
⍝∇ᵁ **name.**	Treats **name.** (any name that includes dot) as a LOGOS path. This pathname can include limited regular expressions. The effective list of names is generated by extracting the terminal segment of every path that matches the pathname. For example, to create a reference to every name in the directory *titles* with *t*, you would type: ⍝∇ᵁ *.proj.source.rep.titles.t?⋆*
⍝∇ᵁ **⍎name**	Fetches the object **name** which must be a character variable in the active workspace. The value of the variable is scanned for identifiers, to which the ⍝∇ᵁ creates references.
⍝∇ᵁ **⍎name.**	Fetches the object **name.** which must be a character variable stored in LOGOS. The value of the variable is scanned for identifiers, to which the ⍝∇ᵁ creates references.

REGULAR EXPRESSIONS

About Regular Expressions

Limited regular expressions are used in pathnames, and full regular expressions are used to express search and replacement strings for commands such as *locate* and *replace*.

Full regular expressions consist of two templates, a locator template and an action template. The locator template is the pattern of strings for which you are searching. The action template is a list of actions to take when a locator template finds a match of the pattern.

Notation

rx	Any regular expression
rrx	Restricted regular expression: any regular expression not containing a closure component
cp	Any single-character pattern

Limited Regular Expressions

Limited regular expression characters are always recognized in pathnames. No braces are required around them.

?	Any character
cp*	Zero or more occurrences (closure)
rx1 | **rx2**	Either of the regular expressions **rx1** or **rx2** (alternation)

Full Regular Expressions

Full regular expression characters must be marked using braces ({ }) or dieresis (¨) in order for them to be recognized.

Escape mechanisms include:

{	Beginning of regular expression
}	Ending of regular expression
¨	Reverse special or literal treatment of next character

Single-character components include:

c	The character specified by **c**
?	Any character except Carriage return (or statement end if searching function text)
[c1c2c3]	Any of the characters **c1 c2 c3**
[c1-c2]	Any letter of the alphabet or digit between and including **c1** and **c2** (character range)
[cs1~cs2]	All characters in set **cs1** which do not appear in set **cs2** (**cs1** and **cs2** may include a character range)
[~cs2]	All characters in the global character set which do not appear in character set **cs2**

Alternation, grouping, elision, and closure include:

rx1 | rx2	Either of the regular expressions **rx1** or **rx2** (alternation)
(rx)	Groups the regular expression
cp−	Zero or one occurrences (elision)
(rx)−	Zero or one occurrences (elision)
cp⋆	Zero or more occurrences (closure)
(rrx)⋆	Zero or more occurrences (closure)
cp+	One or more occurrences (positive closure)
(rrx)+	One or more occurrences (positive closure)

Miscellaneous components include:

◇	Statement delimiter
⊣	Line delimiter
_	Context delimiter
α	Any identifier
ω	Any number
⊂**rx**⊃	Pattern tag

Action template components include:

⊣	Carriage return
⊂**n**⊃	String matching pattern tag **n** in locator template
∇	Evaluated string delimiter
←	Replace string with evaluated expression (recognized only when appearing immediately after the first of a pair of ∇'s)

AUDIT SCRIPTS

Audit Scripts: *afdelpn*

The *afdelpn* script removes a pathname reference from within an audit file. You can remove a specific pathname reference either entirely or from a specific end environment by using the *+environments* modifier.

Syntax

afdelpn **pathnames**

pathnames	is mandatory and indicates the pathnames you want to remove from the audit file.
+*audit*=**name**	Specifies the name of the audit file from which the report will be produced. If you do not specify this modifier, the value in *environment audit* is used.
+*environments* =**envs**	Limits the search to a specific set of environments. The argument contains up to three fields for type, name, and location, separated by spaces. The first character can be a non-alphanumeric to allow multiple entries. The type of environment is represented by one of the following letters:

c	Cluster
f	File component
p	Paging area
w	Workspace

For more information on the types of end environments, see the description of the *afpaths* script in this chapter, where they are described in detail.

Examples

```
ᵁ build 10 vtom rcat cr lf +audit=myaudit +file=myfile +workdir=.public.util
```

This builds a package containing *vtom*, *rcat*, *lf*, and *cr* from *.public.util* into component 10 of *myfile*, and tracks this environment using the *myaudit* audit file.

You could then use *afdelpn* to remove the reference to *.public.util.cr* from the audit file:

```
ᵁ afdelpn .public.util.cr +audit=myaudit +environment=f myfile 10
1 object removed from 1 environment within audit file <1234567 myaudit>
```

Audit Scripts: *afenvs*

The *afenvs* script returns the names of all of the environments tracked by an audit file. The nodes within any paging area can also be listed.

Syntax

afenvs **audit**

audit	Is the audit file from which the environments are to be listed. If elided, the setting of *environment audit* is used.
+headings	Displays report headings.
+nodes	Includes the node names within the paging areas.

Examples

```
U build <10> .public.util.cr | lf +file=myfile +audit=myaudit +overwrite
2 clustered objects saved in component 10 of file 1234567 myfile
```

LOGOS builds a package into component 10 of *myfile* and tracks this environment in the audit file *myaudit.*

You could then use *afenvs* in the following way:

```
U afenvs myaudit +headings
t---------name-------- -loc-
f   1234567 myfile    10
```

This shows that the latest audit record in *myfile* is tracking 1 end environment of type *f* called *1234567 myfile* in component 10.

Audit Scripts: *aflist*

The *aflist* script lists pathname information from within an audit file.

Syntax

aflist **pathnames**

pathnames is mandatory and indicates the names of the paths you want to display. Like the *list* command, pathnames can either be rooted or based on the current working directory set, and can contain limited regular expressions. In addition, if an object name is prefixed by a ⊤, the audit file is searched for all paths with a terminal segment (or object name) matching the specified name.

+*audit*=**name** Specifies the name of the audit file from which the report will be produced. If you do not specify this modifier, the value in *environment audit* is used.

+*column* Formats the table portion of the report into as many columns as will fit on the display.

+*extended* Includes version number, type, and compilation directives for each pathname displayed.

+*symbols* Includes each function's symbol table in the report. Each symbol is preceded by a character identifying the symbol's relationship to the function. See the following list.

Character	**Meaning**
←	Result
⊣	Right argument
⊢	Left argument
∘	Global
;	Local
:	Label
∪	⍝∇∪

+*version* Includes version number and type for each pathname displayed.

Examples

```
∪ build 10 vtom rcat cr lf +audit=myaudit +file=myfile +workdir=.public.util
```

This builds a package containing *vtom, rcat, lf,* and *cr* from *.public.util* into component 10 of *myfile,* and tracks this environment using the audit file *myaudit.*

You could then use *aflist* to return all pathnames tracked by *myaudit* which have a terminal segment name of *vtom*:

```
∪ aflist ⊤vtom +audit=myaudit
.public.util.vtom
```

To include the version number and type as part of the pathname, use the *+version* modifier:

```
∪ aflist ⊤vtom +audit=myaudit +version
.public.util.vtom[f2]
```

To include extended object information (including version number, type, and compilation directives) use the *+extended* modifier:

```
∪ aflist ⊤vtom +audit=myaudit +extended
.public.util.vtom[f2] +compile=x
```

To include the function's or script's tree analysis symbol table, use the *+symbols* modifier:

```
∪ aflist .public.util.vtom +audit=myaudit +extended +symbols +column
.public.util.vtom[f2] +compile=x
  ⊣a    ;b    ;c    ⊢dlm    ←z
```

This indicates that *z* is the result, *a* is the right argument, *dlm* is the left, and *b* and *c* are locals.

Audit Scripts: *afpaths*

afpaths returns the names of paths stored in one or more end environments or nodes within paging areas.

Syntax

afpaths **environments**

environments specifies the end environments and nodes for which you want to list the contents. An end environment specification contains up to three fields for type, name, and location, separated by spaces. The type of environment is represented by one of the following letters:

c	Cluster
f	File component
p	Paging area
w	Workspace

If the type is *c*, the name is a root LOGOS pathname. For example:

/c .myws.cmds

If the type is *f*, *p*, or *w*, the name is assumed to be a standard workspace or file name (for example, if you omit the account number, LOGOS assumes the account number of the alias). If the type is *f* or *p*, the location indicates a component number within the file. If the type is *p*, you can also specify one or more node names (if you omit this, all nodes are assumed). For example:

/w 1234567 myws /f myfile 10

+*audit*=**name** Specifies the name of the audit file from which the report will be produced. If you do not specify this modifier, the setting of *environment audit* is used.

+*column* Formats the report in as many columns as will fit across the display.

+*strip*=**pathname**
Specifies the directory names to omit from the pathnames in the result.

Examples

```
υ build <10> .public.util.lf | cr +audit=myaudit +overwrite
2 clustered objects saved into component 10 of file 1234567 myfile
```

LOGOS builds a package into component 10 of *myfile* and tracks it using the *myaudit* audit file. You could then use the *afpaths* script to display the pathnames used to generate the above package:

```
υ afpaths f myfile 10 +audit=myaudit
.public.util.cr
.public.util.lf
```

You can also use the *+strip* modifier to remove a portion of the front of each pathname:

```
υ afpaths f myfile 10 +audit=myaudit +strip=.public.util
cr
lf
```

Audit Scripts: *afrecs*

The *afrecs* script returns a summary of the audit records stored in an audit file.

Syntax

afrecs **audit**

audit	is the audit file from which the audit records are to be listed. If elided, the setting of *environment audit* is used.
+column	Formats the report in as many columns as will fit on the display.
+headings	Displays report headings.

Examples

```
u build <10> .public.util.cr | lf +audit=myaudit +file=myfile
logos audit file created: 22jun89 17:38 by klh
appending 9 pad components to file <1234567 myfile> starting at component 1
2 clustered objects saved in component 10 of file 1234567 myfile
```

The *build* command generates a package containing two objects, *lf* and *cr*, into component 10 of *1234567 myfile*. You could use the *afrecs* command to examine the audit file's records.

```
u afrecs myaudit +headings
-rec- -ver- ----saved---- -alias
   1     1  22jun89 17:38  klh
```

In the following example, the same component is generated again, this time with three objects:

```
u build <10> .public.util.cr | lf | vtom +audit=myfile +file=myfile
3 clustered objects saved in component 10 of file 1234567 myfile
```

The *afrecs* script shows the following:

```
ᵁ afrecs myaudit +extended +headings
-rec- -ver- ----saved---- -ali -envs -nodes -dirs- -symbs
 1     2 22jun89 17:43 klh       1       0       1         82
```

Notice the version number has changed from 1 to 2.

In the following example, the component is rebuilt with its original two objects and a new audit record is requested:

```
ᵁ build <10> .public.util.cr | lf +audit=myaudit +file=myfile +overwrite
2 clustered objects saved in component 10 of file 1234567 myfile
```

The *afrecs* script now shows the following:

```
afrecs myaudit +headings
-rec- -ver- ----saved---- -alias
  1     2 22jun89 17:43 klh
  2     1 22jun89 17:48 klh
```

Audit Scripts: *afshare*

The *afshare* script controls the sharing of LOGOS audit files between accounts.

Syntax

afshare **user audit**

user	indicates for whom to extend or delete access. It is mandatory and must be an account number. If you specify an alias, the primary account number of the alias is assumed. **audit** is the name of the audit file whose permission is to be changed.

If you do not specify a modifier, this script simply returns a summary of the current access settings. If you omit **user** by specifying `''`, the summary contains all of the entries which currently have access to share the audit file.

+delete	Deletes the account from the access matrix.
+permission =r | w | c	Specifies the permission level to be granted. These are:

r	Read access
w	Write access
c	Control access

If control access is granted, the user also has read access without a passnumber.

Audit Scripts: *afwhere*

The *afwhere* script lists the environments where particular pathnames are used.

Syntax

afwhere **pathnames**

pathnames	Specifies the pathnames for which you want to list the host environments. It is a mandatory argument. Normally, the **pathnames** argument is rooted. However, if you prefix the names with ⊤, *afwhere* searches for any path with a terminal segment matching the name that follows the ⊤.
+*audit*=**name**	Specifies the name of the audit file from which the report will be produced. If you do not specify this modifier, the value in *environment audit* is used.
+*column*	Formats the report in as many columns as will fit across the display.
+*environments* =**envs**	Limits the search to a specific set of environments. The argument contains up to three fields for type, name, and location, separated by spaces. The first character can be a non-alphanumeric delimiter to allow multiple entries. The type of environment is represented by one of the following letters:

c	Cluster
f	File component
p	Paging area
w	Workspace

For more information on the types of end environments, see the description of the *afpaths* script in this chapter, where they are described in detail.

+*nodes*	Changes the result so that it returns node names within paging areas.

Examples

```
ᵁ build 10 .public.util.vtom | rcat +audit myaudit +file=myfile
```

This builds a package containing *vtom* and *rcat* from the *.public.util* directory into component 10. You could then use *afwhere* to display the names of all environments that contain an object whose terminal segment name is *vtom*:

```
ᵁ afwhere ᵀvtom +audit=myaudit
f   1234567 myfile    10
```

Audit Scripts: *afxref*

The *afxref* script is used to produce a cross reference of pathnames by end environment and nodes.

Syntax

afxref **audit**

audit	is the audit file on which the cross reference is to be based. If elided, the setting of *environment audit* is used.
+clearout	Clears the outfile specified using *+outfile* of all components before starting.
+environments **=envs**	Limits the search to a specific set of environments. The argument contains up to three fields for type, name, and location, separated by spaces. The first character may be a non-alphanumeric delimiter to allow for multiple entries.

The type of environment is represented by one of the following letters:

c	Cluster
f	File component
p	Paging area
w	Workspace

For more information on the types of end environments, see the description of the *afpaths* script in this chapter, where they are descibed in detail.

+extended	Adds the user who saved the file and the save date to each environment's report.
+length=**n**	Indicates page length (for pagination).
+outfile=**filename**	Indicates the output file name to which the report is appended.
+strip=**pathname**	Specifies the directory names to omit from the pathnames in the result.
+width=**n**	Indicates line width.

Examples

```
ᵘ build <10> .public.util.cr | lf +audit=myaudit +file=myfile +overwrite
2 clustered object save to component 10 of file 1234567 myfile
```

You could then generate a cross reference for *myaudit*:

```
ᵘ afxref myaudit +extended +width=80
pathname cross reference by environment according to audit file <1234567 myaudit>
file: 1234567 myfile 10 (2 objects; last saved 30jun89 18:22 by klh)
.public.util.cr .public.util.lf
xref completed
```

In this example, the cross reference is displayed on the terminal using a width of 80. It also includes the extended information for each environment.

You could also spool the output to a file using the *+outfile* modifier:

```
ᵘ afxref myaudit +clearout +extended +outfile=outfile
xref completed
```

Or you could trim *.public.util* from the beginning of pathames:

```
ᵘ afxref myaudit +extended +clearout +outfile=spoolfile +strip=.public.util
```

Audit Scripts: *pfdelobj*

The *pfdelobj* script removes objects from specific nodes within a paging area.

Syntax

pfdelobj **names**

names	is mandatory and indicates which objects are to be removed from the specified nodes.
+*area*=**name**	Specifies the name of the affected paging area. You must specify this modifier.
+*lock*=**number**	Specifies an optional paging area file passnumber.
+*nodes*=**names**	Specifies the names of the nodes from which the objects are to be removed. You must specify this modifier.

Examples

```
ᵁ build +depth=all +workdir=.public.util
ᵁ build vtom
ᵁ filesave 1234567 myfile
generating 1 node in to paging area 1234567 myfile, 20
1 node generated using 1 object
generation 1 of paging area 1234567 myfile, 20 saved 30jun89 14:28 by klh
```

This generates the *vtom* function into a node called *vtom* within the paging area *1234567 myfile, 20.*

You could then use *pfdelobj* with the *+area* and *+node* modifiers to remove the *vtom* object:

```
ᵁ pfdelobj vtom +area=1234567 myfile 20 +node=vtom
1 object removed from 1 node
```

EDITOR COMMANDS

Editor Commands: ⍎

The ⍎ command can be used in two ways:

- with an argument as an alternate form of the *apl* command. See the section on the *apl* command for details.

- without an argument as an alternate form of the *aplw* command. See the section on the *aplw* command for details.

Editor Commands: ∇

The ∇ command can be used in two ways:

- with an argument (an object name) as an alternate form of the *edit* command. See the section on the *edit* command for details.

- without an argument, as an alternate form of the *end* command. See the section on the *end* command for details.

Editor Commands:)

The) command can be used in two ways:

- with an argument (a LOGOS command) as an alternate form of the *logos* command. See the section on the *logos* command for details.
- without an argument to redisplay the last editor command.

Editor Commands: *add*

The *add* command adds one or more lines to an object.

Syntax

ad[*d*] [*up*] [**integer**] [**string**]

up	Inserts the new lines above the reference line, as opposed to below.
integer	Is the number of lines to insert.
string	Is the string of text to insert on the lines added.

Usage

If you do not provide an argument, *add* inserts one line. If you provide the first argument, *add* inserts that number of lines. If you provide the first and second argument, *add* puts the string of text on the added lines.

Examples

---

add 2

Inserts two empty lines following the reference line.

---

add 5 A

Adds five empty comment lines after the reference line.

---

addup 2

Adds two empty lines above the reference line.

---

upadd 2

Adds two empty lines above the reference line.

Editor Commands: *again*

The *again* command executes the last editor command entered.

Syntax

a[*gain*] [**integer** | ⋆]

integer	Is the number of times to execute the command.
⋆	Executes the command until it fails.

Usage

If you do not enter an argument, *again* executes the last command once.

Examples

---

again ⋆

If the last command invoked was *locate*, repeat *locate* until the argument string can no longer be found.

The reference line becomes the last line containing the occurrence of the argument string to *locate*.

Editor Commands: *apl*

The *apl* command allows you to execute APL expressions from within the editor.

Syntax

apl **expression**

expression	Is the expression to execute as an APL expression.

Result

In full screen mode, the result of the expression displays in the editor message area. That is the one-line field just above the command input field.

Editor Commands: *aplw*

The *aplw* command allows you to execute APL expressions from within the editor, and then waits for you to press Enter before exiting the command.

Syntax

aplw [**expression**]

expression	Is the expression to execute as an APL expression.

Result

When you are in full screen mode, you are returned temporarily to the standard screen to view the result of the expression.

Usage

After *aplw* executes the expression and returns the result, you must press Enter to terminate the command.

If you do not supply an argument, *aplw* invokes the ⎕ version of the command, and issues a ⍎⎕ prompt. Here you can enter an expression. To terminate the command, press the Space Bar, then Enter.

Editor Commands: *bottom*

The *bottom* command moves the reference line to the last line of the object.

Syntax *b*[*ottom*]

Editor Commands: *browse*

The *browse* command allows you to place constraints on yourself so that you can browse through an object, but you cannot save any changes, regardless of whether you have write access or not.

Syntax

br[*owse*] [*off* | *on* | *lock*]

off	Allows edited objects to be saved. This is the default setting.
on	Inhibits you from saving changes made to the object.
lock	Prevents you from turning browse mode off. The only way to override this is to change the name of the object. *setname* will automatically turn browse mode off.

Result

If you are editing in full screen mode, a message appears in the header of the screen to remind you that you are editing an object but will not be able to save the changes.

Usage

If you specify *browse* without an argument and browse mode is on, it is switched off. If you specify *browse* without an argument and browse mode is off, it is switched on.

Editor Commands: *change*

The *change* command locates occurrences of a specified string or pattern and replaces them with another string or pattern.

Syntax

c[*hange*][*up*] [_]/**string1** [/ [**string2** [/**integer1**|★ [**integer2**|★]]]]

up	Reverses the direction in which the object is searched.
_	Performs a syntactic replacement.
string1	Is the string to change. Treats strings containing curly braces or the dieresis character as a regular expression pattern.
string2	Is the replacement string. Treats strings containing curly braces or the dieresis character as a regular expression pattern.
integer1|★	Is the number of lines to be searched.

integer1	Searches that number of lines from the reference line down.
−integer1	Searches that number of lines above the reference line.
★	Searches all lines from the reference line to the end.
−★	Searches all lines above the reference line.

integer2|★	Is the occurrence within a line to change.

integer2	Changes that occurrence of the string, counting occurrences from the left end of the line.
−integer2	Changes that occurrence of the string, counting from the right end of the line.
★	Changes all occurrences on a line.

Usage

If **string1** is empty, *change* treats it as a search argument and matches the empty strings on either side of every character on a line.

After the command is executed, the last line on which a change was made becomes the reference line.

If **string2** is empty, change removes occurrences of the **string1**.

Examples

```
change/lead/gold/* *
```

Replaces all occurrences of the string *lead* to the string *gold.*

```
change_/a/b
```

Changes all occurrences of *a* to *b* without affecting occurrences of *a* that are part of a word (for example, *alpha*).

```
change//-
```

Replaces the empty strings on either side of every character of a line with –. For example, the line *abc* would become *–a–b–c–*.

```
change/cat/dog/1 *
```

Changes all occurrences of *cat* to *dog* on the current line.

```
change//[/* 1
```

Prefixes the reference line and every line after it with [.

```
top ᵁ change_/rcat/rowcat/* *
```

Changes every occurrence of the identifier *rcat* with the string *rowcat.*

Editor Commands: *copy*

The *copy* command copies a line or group of lines from one location to another.

Syntax

co[*py*][*up*] [**integer1**|*] [**integer2**|*]

up	Reverses the direction of the command.
integer1|*	Is the number of lines to copy, including the reference line. The default value is 1.

integer1	Copies the specified number of lines, counting from the reference line forward.
–integer1	Copies the specified number of lines, counting from the reference line backwards.
*	Copies all lines from the reference line to the end of the object.
–*	Copies all lines from the reference line to the beginning of the object.

integer2|*	Is the number of lines before or after the reference line to put the copied block. The default value is 0, which puts the copied block immediately after the reference line.

integer2	Puts the copied block the specified number of lines after the reference line.
–integer2	Puts the copied block the specified number of lines before the reference line.
*	Puts the copied block at the end of the object.
–*	Puts the copied block at the beginning of the object.

Usage

If you do not specify an argument, *copy* duplicates the reference line immediately below itself.

Examples

copy

Duplicates the reference line immediately below itself.

---

*copy 3 **

Appends a copy of the reference line and the two lines following it to the end of the object.

---

copy 1 2

Places a copy of the reference line 2 lines below itself (skips 2 lines and inserts the copy).

Editor Commands: *delete*

The *delete* command removes lines from an object.

Syntax

del[*ete*] [*up*] [**integer** | *]

up	Reverses the direction of the command.
integer | *	Is the number of lines to delete.

integer	Deletes the specified number of lines starting at the reference line and counting down.
–integer	Deletes the specified number of lines starting at the reference line and counting up.
*	Deletes all lines from the reference line to the end.
–*	Deletes all lines from the reference line to the top.

Usage

If you do not specify an argument, a default value of 1 is assumed.

Examples

delete –3

Deletes the reference line and the two lines above it.

deleteup 3

Deletes the reference line and the two lines above it.

delete –★

Deletes the reference line and all lines above it.

deleteup ★

Deletes the reference line and all lines below it.

Editor Commands: *diamond*

The *diamond* command specifies a surrogate for the diamond character on an IBM 3XXX display station which avoids the need to use the overstrike character to create the diamond.

Syntax

dia[*mond*] [**character**] [*on*|*off*]

character	Is the surrogate character to be used. The default is the cap ∩ character.
on	Enables the recognition and substitution of the diamond surrogate.
off	Disables the recognition and substitution of the diamond surrogate.

Usage

diamond takes two optional arguments. The first (**character**) is the surrogate character to be used. The default is the cap (∩) character. The second argument is one of the keywords *on* or *off* (the default is *on*). It enables or disables the recognition and substitution of the diamond surrogate.

Characters that are valid to use for surrogates are the same as those that may be used for separator characters: non-alphanumeric, and not one of the reserved characters:

```
{ } [ ] ( ) / \ . ↑ ⍎ ⍕ ∇ ⍙ ∆ + ' ? ← ⎕ _ =
```

Editor Commands: *display*

The *display* command displays objects in your workspace or any LOGOS path to which you have read access.

Syntax

```
display [[=]name]
        [+nolines]
```

[=]**name**	Is the name of the object.	
	name	Is an object in LOGOS.
	=name	Is a workspace resident object.
+nolines	Omits opening and closing delta characters and bracketed line numbers from the display if the object is a function or script.	

Result

If you are editing in full screen mode, the object displays in the editor output window.

Usage

If you do not provide an argument, *display* displays the contents of the editor clipboard.

If you enclose the argument in brackets, *display* treats it as a reference to the pathname of the object currently being edited. For example, the argument [*1*] is equal to **pathname**[*1*] where **pathname** is the pathname of the current object.

Examples

display =mat

Displays the workspace-resident object *mat.*

display .public.util.vtom

Displays the *vtom* function in the public utility library.

display [*3*:*j*]

Displays the journal for version 3 of the object currently being edited.

Editor Commands: *down*

The *down* command moves the reference line pointer down a specified number of lines.

Syntax

d[*own*] [**integer** | ⋆]

integer | ⋆	Is the number of lines from the current reference line to make the new reference line.	
	integer	Makes the line *the specified number of lines from the current reference line* the new reference.
	⋆	Makes the last line of the object the reference line (equivalent of the *bottom* command).

Examples

---

down 3

Makes the line three lines below the current reference line the new reference line.

Editor Commands: *edit*

The *edit* command opens objects for editing.

Syntax

```
e[dit] [=]name
        [+browse[=off|on|lock]]
        [+disttask=task]
        [+makedir]
        [+override]
        [+register]
        [+task[=task]]
```

[=]**name**	Is one or more object names to be opened for editing.

name	Is an object in LOGOS.
=name	Is a workspace-resident object.

+browse[=*off*|*on*|*lock*]

Allows the viewing and manipulation of an object without risk of corrupting the source.

+browse	Toggles browse mode from on to off, or from off to on.
+browse=off	Allows changes to objects to be saved.
+browse=on	Does not allow changes to objects to be saved.
+browse=lock	Does not allow browse mode to be turned off.

+disttask=**task**

Specifies the name of the auxiliary task to be used by the Application Debugging Assistant to distribute changed objects to the active workspace.

+makedir	Allows the creation of intermediate directories specified in the argument to the *edit* command. This is identical in function to the *+makedir* modifier supported by the LOGOS *save* command.
+override	Overrides existing registration (must be specified with *+register*).
+register	Causes the editor to register out each of the objects named in the argument before opening it for editing.

+*task*[=**task**]	Allows editing of objects in the workspace of an auxiliary task started from within LOGOS. If *+task* is specified without an argument, the default task name *aux* is assumed. If *+task* is omitted, the task named in the *environment task* parameter, if any, is used.

Usage

If you include the name in brackets, it is treated as a reference to the pathname of the object being edited.

Editor Commands: *end*

The *end* command saves changes to an object, closes the object and leaves the editor.

Syntax

```
end [name]
        [+makedir]
        [+override]
        [+register]
```

name	Is the pathname under which to save the current object.
+makedir	Allows the creation of intemediate directories specified in the argument. This is identical in function ot the *+makedir* modifier to the LOGOS *save* command.
+override	Overrides existing registration (must be specified with *+register*).
+register	Registers in the object you are closing.

Usage

If you specify the **name** argument, it must be a pathname whose terminal segment matches the object's current name. *end* **name** allows you to change the directory in which the object is saved, but does not change the object's name. Use *setname* if you want to change the name.

end[*0*] will save the object but will not change the version.

If ∇ is used instead of *end* to save an object, it does not support the **name** argument.

Editor Commands: *format*

The *format* command places the object being edited in canonical format.

Syntax

fo[*rmat*]

Result

For functions and scripts, the visible effects of this are realignment of labels and comments, removal of redundant blanks etc.

In character and numeric variables, *format* removes trailing blanks from lines of character data and reformats numbers and aligns columns of numeric data.

format displays an error message if the object cannot be properly formatted (for example, because of a damaged function header, invalid numeric data, etc.).

Editor Commands: *get*

The *get* command retrieves text from an object or the screen and inserts it in the current object, either immediately after the reference line, or if you specify the *up* variant, above it.

Syntax

ge[*t*] [*up*] [**name**|* [**integer**]]
  [+*task*[=**task**]]

up	Reverses the direction of the command.
name|*	Is the object from which to extract the text.
	name — Fetches text from the specified object in the workspace, or if name contains a non-alphanumeric pathname symbol (for example, a dot (.) or bracket ([), from an object in LOGOS.
	* — Fetches text from the screen display.
integer	When used with *, is the number of screen lines to copy from the location of the cursor in the editor display to the object.
+*task*[=**task**]	Fetches text from variables in the workspace of an S-task started from within LOGOS. If *+task* is specified without an argument, the default task name *aux* is assumed. If *+task* is omitted, the task named in the *environment task* parameter, if any, is used.

Usage

If you enclose name in brackets, it is treated as a reference to the pathname of the object being edited. For example, the argument [*1*] is equal to **pathname**[*1*] where pathname is the pathname of the current object. This makes if convenient to reference alternate versions and attributes of an opened object.

If you copy text from the screen, the text to copy and the destination within the object must be visible on the screen simultaneously. The copy can occur from the display window or from one location in the object to another.

If you do not provide an argument, *get* fetches text from its own internal buffer. The only way to write something to this buffer is to use the *put* command without a variable name argument. This makes if convenient to use *put* and *get* together.

Examples

get buf

Inserts the contents of the variable *buf* after the reference line.

top ᵘ *delete* ⋆ ᵘ *get* []

Discards all changes made to the currently open LOGOS object and restores the original source.

getup ⋆ *3*

Copies three lines of text from the cursor position on the screen to just before the reference line.

Editor Commands: *header*

The *header* command assists in the maintenance of function and script headers. It identifies two suspicious categories of names:

- Global identifiers, which are assigned within the body of the object but do not appear in the header.
- Unreferenced identifiers, which are localized in the header but do not appear anywhere in the body of the object.

Optionally, the *header* command can also take corrective action on based on the user's input.

Syntax

hea[*der*] [*g*|*u*] [∇|∆|_]

g|*u*	Restricts *header* to report only one class of names.		
	g	Identifies global identifiers.	
	u	Identifies unreferenced identifiers.	
∇|∆|_	Conditions *header* to ignore certain references.		
	∇	Ignores names if they correspond to functions present in the active workspace.	
	∆	Ignores names beginning with ∆.	
	_	Ignores names beginning with a character in the second alphabet.	

Result

header reports suspicious references in the form of a menu.

If you are editing in full screen mode, this is a full screen menu. Each name is followed by an input field beside it and a single character (*g* or *u*).

If you are editing in line mode, suspicious references display one by one, and you are prompted for input for each.

Usage

In full screen mode, place any character in the input field beside the identifier and corrective action will be taken.

In line mode, enter *y* to take corrective action.

When you take corrective action, global identifiers are localized, and unreferenced identifiers are removed from the header. In addition, the locals list is sorted alphabetically.

Editor Commands: *help*

The help command obtains information about another command.

Syntax

h[*elp*] [**command**]

command	Is the name of the command for which you want more information.

Result

If you omit the argument, a summary of all commands is displayed.

If you provide an argument, the command lists general, syntax, and usage information about that command.

Usage

On an asynchronous terminal, the results will be printed at the terminal. On a 3X7X device, the results will appear in the editor display window.

In addition, on a 3X7X device, you may use PF1 to display a menu of command names and other topics. You may select individual items on which you want to receive detailed help information by tabbing to the desired item, typing any character in the input field next to the item, then pressing the Enter key. Only one item can be selected at a time.

You may also display help on any of the following topics by entering the topic name as the argument to the *help* command:

Topic name	Information provided
command summary	List of all editor commands and a brief summary of their action.
pf keys	PF key definitions.
keywords	Summary of how to use keywords from within the editor.
separator characters	Using separator characters in the command line.
utility interface	Details on how to use the ∆*ledutil* function.

Editor Commands: *highlight*

The *highlight* command identifies all occurrences of a string or pattern within a specified range.

Syntax

hi[*ghlight*] [*up*] [_]**/string/attribute/** [**integer1** | ★ [**integer2** | ★]]

_ Performs a syntactic search.

string Is the string or pattern to be located. Patterns must be enclosed in curly braces ({ }) or each metacharacter in the pattern must be prefixed with a dieresis (¨).

attribute Indicates how you want to highlight matching strings or patterns (for example, colour, flashing, underlined, etc.) You can select one colour attribute and one character attribute. They are:

Colour Attributes		Character Attributes	
b	Blue	*d*	Default
d	Default	*f*	Flash
g	Green	*i*	Inverse
p	Pink	*u*	Underline
r	Red		
t	Turquoise		
w	White		
y	Yellow		

integer1 | ★ Specifies the number of lines to search.

integer1	Searches that number of lines from the reference line down.
−integer1	Searches that number of lines before the reference line.
★	Searches all lines from the reference line to the end.
−★	Searches all lines before the reference line to the top.

integer2 | ⋆	Specifies the number of occurrences on a line to highlight.	
	integer2	Highlights the specified occurrence counting from the left end of the line.
	–integer2	Highlights the specified occurrence counting from the right end of the line.
	⋆	Highlights all occurrences on a line.

Result

If you are editing in full screen mode, *highlight* alters the screen attributes of these occurrences.

If you are editing in line mode, *highlight* displays the lines containing occurrences.

Usage

If you do not provide the range of text to be searched (**integer1** and **integer2**), all occurrences within the object, both before and after the reference line, are highlighted.

Examples

---

highlight/if the/p/1

Makes all occurrences of the string *if the* on the reference line appear pink.

---

highlight_/tmp/rf

Makes all references to the variable *tmp* anywhere in the object flash red.

---

highlight /

Turns off all highlighting.

Editor Commands: *input*

The *input* command allows entry of multiple lines into an object.

Syntax

inp[*ut*] **string**

string is a string to be placed on a new line immediately below the reference line.

Result

After placing the string on a new line, the editor then goes into input mode, where it awaits input.

Usage

If you do not supply an argument, the editor inserts a blank line directly below the reference line and goes into input mode.

In input mode, you can enter more text. To add another line, press Enter. To terminate input mode, press Enter twice.

Editor Commands: *insert*

The *insert* command allows you to insert text into an object.

Syntax

in[*sert*] [*up*] **string**

up	Reverses the direction of the command.
string	Is the string to be added.

Result

insert adds the new line after the reference line, or if you use the *up* variant, above it.

Usage

If you do not specify an argument, the editor goes into input mode where you can enter text. To terminate input mode, press Enter twice.

Editor Commands: *join*

The *join* command merges two lines of an object into a single line.

Syntax

j[*oin*][*up*] [/**string**[/]]

up	Reverses the direction of the command.
/**string**[/]	Is a string to be inserted between the merging lines. You must specify a delimiter character at the beginning of the string. To enter an argument containing trailing blanks, specify a delimiter character at the end of the string as well.

Result

The reference line joins to the line beneath it, or if you specify the *up* variant, to the line above it.

Examples

---

join/◇

Joins the reference line to the line following it with a diamond between the merged lines.

Editor Commands: *lastline*

The *lastline* command recalls the last command entered and allows its re-execution.

Syntax

last[*line*] [**string**]

string is a string to be inserted before the recalled command, and can contain several commands delimited by the separator character.

Result

The recalled command appears on the command line.

Usage

If **string** is specified, that text will be inserted in front of the recalled command, and the entire command line will be immediately executed. If **string** has not been specified, press Enter to execute the recalled command.

You can also use the short forms *ll* or). You cannot provide an argument when you specify), or the editor interprets it as a LOGOS command.

Editor Commands: *locate*

The *locate* command searches the object for a string or pattern and moves the reference line to the next occurrence of that string or pattern.

Syntax

l[*ocate*][*up*] [_][/[**string**[/*inc*]]]

up	Reverses the direction of the search.
_	Causes the command to perform a syntactic search. For example, the command *locate_/and* will not recognize a match in the string *command* because the substring is part of a word.
string	Is the string to search for. If this string contains curly braces or dieresis characters, it is treated as a regular expression pattern.
inc	Includes the reference line in the search.

Usage

If you do not specify a string argument, the last *locate* command is repeated.

After you use the *locate* command once, you can abbreviate it to / to find the next occurrence of the string.

Examples

---

locate_/t

Moves the reference line pointer to the next line containing a reference to the variable *t*.

/

Locates the next occurrence of the last string searched for.

Editor Commands: *logos*

The *logos* command allows execution of LOGOS commands from within the editor.

Syntax

log[*os*] **command**

command is the LOGOS command to be executed.

Result

If you are editing in full screen mode, the output of the *logos* command displays in the editor display window.

Usage

You can also abbreviate the logos command to `)`.

You cannot invoke the LOGOS *edit* or *exit* commands from within the editor.

Examples

)*list*

Invokes the LOGOS *list* command from within the editor.

Editor Commands: *move*

The *move* command moves a line or group of lines from one location to another.

Syntax

m[*ove*] [*up*] [**integer1**|* [**integer2**|*]]

up	Reverses the direction of the command.
integer1|*	Specifies the number of lines to move.

integer1	Moves lines from the reference line forward.
−integer1	Moves lines from the reference line backward.
*	Moves the reference and all lines forwards from it.
−*	Moves the reference line and all lines backwards from it.

integer2|*	Specifies the destination of the lines relative to the reference line.

integer2	Puts the moved block the specified number of lines after the reference line.
−integer2	Puts the moved block the specified number of lines before the reference line.
*	Puts the moved block at the end of the object.
−*	Puts the moved block at the beginning of the object.

Usage

If you do not specify **integer1**, the default value 1 is used. This moves one line below the reference line.

If you do not specify **integer2**, the default value 0 is used. This moves the block immediately below, or if you use the *up* variant, above itself.

Examples

*move 3 **

Moves the reference line and the two lines following it to the end of the object.

move

Swaps the reference line and the line immediately below it.

Editor Commands: *names*

The *names* command reports the names of all objects open for editing.

Syntax

na[*mes*] [*def*|*window*]

def	Displays the list of keywords. On a 3XXX device, this is displayed in the editor message line.
window	Displays names in a scrollable window area if you are editing in full screen mode.

Result

If you are editing in full screen mode, the names display in the message area.

Usage

When you are editing in full screen mode and there are more names than will fit in the message area, you can use the *window* keyword to display them in a scrollable window area.

Editor Commands: *next*

The *next* command moves the reference line pointer.

Syntax

n[*ext*] **integer** | ⋆

integer | ⋆ Specifies the number of lines to move the reference line.

integer	Moves the reference line forward the specified number of lines.
–integer	Moves the reference line back the specified number of lines.
⋆	Moves the reference line to the last line of the object.
–⋆	Moves the reference line to the first line of the object.

Editor Commands: *put*

The *put* command extracts text from the object being edited and places it in an APL variable or the editor buffer.

Syntax

pu[*t*] [**integer** | ⋆ **variable**]
[+*task*[=**task**]]

integer | ⋆	Indicates the number of lines to extract, including the reference line.
	integer — Extracts the specified number of lines.
	⋆ — Extracts all lines.
variable	Is the name of the variable to contain the extracted text.
+*task*[=**task**]	Allows the user to put text into a variable in the workspace of an s-task which was started from within LOGOS. If +*task* is specified without an argument, the default task name *aux* is assumed. If +*task* is omitted, the task named in the *environment task* parameter, if any, is used.

Usage

If you do not specify the first argument, the default value of 1 is used.

If you do not specify the second argument, the text is placed into the editor's internal buffer. The only way to read this buffer is to use the *get* command without an argument.

Examples

---

top ᵁ put ⋆ mat

Puts every line of text in the object being edited into the character matrix *mat.*

---

top ᵁ put 3 ᵁ bottom ᵁ get

Puts a copy of the first three lines of the object at the end of the object.

Editor Commands: *putnum*

The *putnum* command extracts the line numbers from an object an writes them to a variable.

Syntax

putn[*um*] [**integer** | ⋆ **variable**]
        [+*task*[=**task**]]

integer | ⋆	Specifies the number of line numbers to extract.	
	integer	Extracts the specified number of line numbers starting with the reference line.
	⋆	Extracts all line numbers from the reference line to the end of the object.
	–⋆	Extracts all line numbers above the reference line to the top of the object.
variable	Is the name of the variable to contain the line numbers.	
+*task*[=**task**]	Allows the user to put the line numbers into a variable in the workspace of an s-task which was started from within LOGOS. If *+task* is specified without an argument, the default task name *aux* is assumed. If *+task* is omitted, the task named in the *environment task* parameter, if any, is used.	

Result

The variable specified will contain the line numbers in floating point format.

Usage

If you do not specify the first argument, the default value of 1 is used.

If you do not specify the second argument, the text is placed into the editor's internal buffer. The only way to read this buffer is to use the *get* command without an argument.

Editor Commands: *quit*

The *quit* command terminates the editing of one or all objects without saving any changes.

Syntax

q[*uit*] [*imm*|*all*]

imm	Closes the current object open for editing without saving any changes.
all	Closes all objects opened for editing without saving any changes.

Usage

The only way you can use *quit* without an argument is if you have not made any changes to the object. If you have modified the object in any way, the editor insists that you provide an argument to the *quit* command (*all* or *imm*) before closing it.

Editor Commands: *renum*

The *renum* command renumbers the lines of an object.

Syntax

ren[*um*] [**integer1** [**integer2**]]

integer1	Specifies the first line number to appear in the new series.
integer2	Specifies the increment between line numbers.

Usage

The default value for both arguments is 1. If you do not specify the first argument and the object is a function or a variable with an index origin of 1, *renum* uses a value of 0.

Editor Commands: *replace*

The *replace* command deletes the reference line and substitutes a new line in its place.

Syntax

r[*eplace*] [**string**]

string is the string to use as the new line.

Usage

If you do not provide an argument, the editor goes into input mode. To terminate input mode, press Enter twice.

Editor Commands: *resequence*

The *resequence* command is useful when editing functions or scripts which use a name followed by a number for labels (for example, *l1*). *resequence* provides a simple means of putting labels back in numeric order when you have introduced additional labels into the object during an editing session.

Syntax

res[*equence*] [**integer1** [**integer2**]]
        [+*prefix*=**label**]

integer1	Is the number to use as the suffix of the first label.
integer2	Is the increment between labels.
+*prefix*=**label**	Specifies the prefix string searched for when looking for labels to renumber. If omitted, the default label *l* is used.

Usage

If you do not specify **integer1**, the default value of 0 is used. If you do not specify **integer2**, the default value of 1 is used.

resequence only affects labels which fit the pattern of a name followed by a number. Both the labels and references to them will change.

resequence does not examine the contents of quoted strings, such as arguments to execute (⍎) or assignments to ⎕*trap*. If *resequence* is used on a function with references to labels in quoted strings, the function may be modified incorrectly. To prevent this, avoid using labels which fit the pattern of a name followed by a number in these contexts, or change them so the reference to a line label occurs outside of quotation marks. For example:

```
⎕trap←'∇1 e →l7'
```

could be rewritten as:

```
⎕trap←'∇1 e →',⍕l7
```

Examples

resequence 1 5 +prefix=error

Resequences all labels of the form *error0, error1*, so that the first label is *error1*, the next is *error5* and so on.

Editor Commands: *sepchar*

The *sepchar* command defines the editor command separator character and enables or disables its recognition.

Syntax

sep[*character*] [**character**] [*on* | *off* | *imm*]

character	Is the character to be used as the command separator. This character must not be alphabetic, numeric, or reserved: { } [ ] ( ) / \ . ↑ ↓ ⍋ ⍒ ∇ ∆ ⍙ + ' _ ? ← ⎕ =
on	Enables the recognition of the command separator character.
off	Disables the recognition of the command separator character.
imm	Causes all commands, including those with deferred separation, to recognize the separator character.

Examples

---

sep imm ∪ *apl v←10 30*⍴'⋆∘0' ∪ *get v* ∪ *sep on*

Makes the *apl* command recognize the ∪ as a separator character.

Editor Commands: *setname*

The *setname* command allows you to change the name of the object you are editing.

Syntax

setn[*ame*] [=]**name**
[+*task*[=**task**]]

name Is the new pathname for the object.

=name Treats the name as a reference to an object in the workspace and conditions the editor to save the object in the workspace, not back in the LOGOS hierarchy.

+*task*[=**task**] Allows the user to specify that an object is to become ws-resident in the workspace of an S-task which was started from within LOGOS. This is valid only with a workspace resident object, or one that you want to make resident in the workspace by using *setname*=**name**. If +*task* is specified without a value, the default task name *aux* is assumed. If this modifier is omitted, the task named in the *environment task* parameter is used, if any.

Note: If the object was originally opened from the workspace of an suiliary task, *setname*=**name** without +*task* will result in the changed object being saved back in the task from which it was opened, regardless of the *environment task* setting. In this case, to alter the workspace which receives the changed object, you must specify the +*task* modifier with the appropriate value.

Result

If the object is a function or an object, the header is adjusted to reflect the new name.

Usage

If you enclose name in brackets, it is treated as a reference to the pathname of the object currently being edited.

Examples

---

setname scan

Changes the name of the opened object to *scan*.

---

setname[:*d*]

Changes the name of the current object so that it becomes the documentation attribute of the object.

Editor Commands: *settype*

The *settype* command allows the user to change the data type of the object being edited.

Syntax

sett[*ype*] [*f* | *s* | *vc* | *vn*] [*0* | *1* | *2*]

[*f* | *s* | *vc* | *vn*]	Specifies the type of argument. Types of arguments are:

f	Function
s	Script
cv	Character variable
nv	Numeric variable

[*0* | *1* | *2*]	Indicates the rank of the type (must be provided if you specify a type of *cv* or *nv*).

Usage

If you do not specify an argument, settype reports the object's current type using the following codes:

f	Function
s	Script
cv **n**	Character variable of rank n
nv **n**	Numeric variable of rank n

n represents rank.

Invalid type conversions (for example, setting a numeric variable to a function) are reported when you try to save the object.

Examples

---

settype f

Converts the object into a function.

---

settype vc 2

Converts the object to a character vector.

Editor Commands: *sort*

The *sort* command sorts the list of local identifiers in the header of a function or script. This makes it easier to locate a given identifier within the list.

Syntax

so[*rt*]
        [+*lexographic*]

+*lexographic*	Sorts names using their length as a primary key and their alphabetic ranking as a secondary key.

Result

Sorts names in strict alphabetical order unless you use +*lexographic*.

If a name appears more than once, *sort* removes redundant entries.

Editor Commands: *split*

The *split* command breaks a line into two separate lines.

Syntax

spl[*it*] [*up*] * | **/string** [**/integer**]

up	Reverses the direction of the command.
*	If you are editing in full screen mode, the line splits at the cursor position.
/string	Splits the reference line before the specified string.
/integer	Specifies at which occurrence of **string** to split the line.

integer	Counts from the beginning of the line.
–integer	Counts from the end of the line.

Result

The first part of the line remains the reference line, unless you use the *up* variant, causing the second part of the line to become the reference line.

Examples

---

split *

Splits the line at the cursor. This is equivalent to F10.

---

split/hold

Splits the line at the first occurrence of the string *hold.*

---

split/hold/−1

Splits the line at the last occurrence of the string *hold*.

Editor Commands: *super*

The *super* command allows you to edit a line in a manner similar to the IBM APL/SV ∇ editor's 'superedit' form of dot-and-comma editing.

Syntax

`s[uper] [integer]`

integer	Indicates the number of the character at which to place the cursor.

Result

If you are editing in full screen mode, the reference line is placed into the input area with the cursor over the specified character.

If you are editing in line mode, the line displays and the keyboard unlocks for dot/comma editing, similar to the `[n⎕m]` command.

Editor Commands: *switch*

The *switch* command puts the current object aside temporarily and moves the editor to another object previously opened for editing.

Syntax

sw[*itch*]

Editor Commands: *syntax*

The *syntax* command computes a report describing static errors within a program. It tests the following conditions:

- illegal characters
- symbol juxtaposition problems
- mismatched parentheses, brackets, or quotes
- suspicious use of names (optional)

Syntax

```
sy[ntax] [attributes]
         [+all]
         [+display]
         [+lines]
         [+quotes]
         [+show]
```

attributes	Is a screen attribute (one character) or a pair of screen attributes (two characters) to be used to highlight errors. For a list of attributes, see the *highlight* comannd.
+all	Computes all errors, including suspicious name references which may not be erroneous.
+display	Displays the entire line on which errors occurred.
+lines	Displays the line numbers on which errors occurred, followed by a symbol denoting the type of error (See below for a list of symbols).
+quotes	Causes quoted strings logically appearing after executes within the program to be examined as if they were unquoted.
+show	Displays the entire line on which errors occurred, with a symbol pointing to each error. The symbol denotes the type of error at that location. Because errors relating to parentheses, brackets, and quotes are obvious, the caret (^) is substituted as the pointer.

Symbols indicating errors are:

^	*Generic syntax error.* These include most incorrect uses of symbols. For example: a dyadic symbol used monadically an improper outer product an improperly labelled line use of branch not as the root function of a statement redundant use of diamond
(	*Parenthesis error.*
[	*Bracket error.*
'	*Quote error.*
+	*Domain error.* These arise from apparent use of a character argument where a numeric one was expected. As the syntax command does not execute the program, only a limited number of these are detected.
.	*Constant error.* These refer to illegal formation of numeric constants. For example, *4..1* and *8je4* are illegal constants, while *4.1* and *8j8e4* are legal ones.
?	*Suspicious reference.* These denote use of names which are unusual but may or may not be erroneous in the running application. For example, a local variable which is not assigned a value, or a name which is used to define a line-label more than once, is considered suspicious.

Result

If you are editing in full screen mode, errors are highlighted with the attributes specified. If no attributes were specified, but one of *+display, +lines* or *+show* were included, the report appears in the editor output window.

Usage

The *syntax* command does not actually execute the program. This command should be used to supplement but not replace careful program and system testing.

If *syntax* is used without arguments or modifiers, the pathname of the object is returned if any errors are detected. If no errors are detected, the result is empty.

NOTE: This command has the effect of using the *format* command on the function, so the modified flag is set to *on* even if no changes have been made.

Editor Commands: *top*

The *top* command moves the reference line pointer to the first line of the object.

Syntax *to*[*p*]

Editor Commands: *type*

The *type* command displays a range of object lines on line mode terminals.

Syntax

t[*ype*] [**integer** | *]

integer | *	Is the number of lines to display, including the reference line.	
	integer	Displays the specified number of lines, including the reference line.
	*	Displays all lines from the reference line to the end of the object.

Usage

If integer is not specified, it defaults to 1.

Editor Commands: *up*

The *up* command moves the reference line up the specified number of lines.

Syntax

u[*p*] [**integer** | *]

integer | *	Specifies the number of lines to move the reference line.	
	integer	Moves the reference line the specified number of lines.
	*	Moves the reference line to the first line of the object (equivalent to the *top* command).

Editor Commands: *use*

The *use* command allows the user to build a series of editor commands into a variable and then interpolate that variable into the editor command line. This command provides a rudimentary macro capability in the editor.

Syntax

us[*e*] [**variable**]

variable	Is the name of the variable.

Usage

If you do not provide an argument, the internal editor buffer is used (see the *get* and *put* command for more information on the buffer).

Examples

The function *alcomments* takes the canonical representation of a function as its argument and returns as a result the canonical representation with all comments aligned. If you define the variable *align* to contain the string:

```
'top ∪ put ⋆ temp ∪ delete ⋆ ∪ sep imm ∪ temp←alcomments temp ∪ get temp'
```

you can invoke the comment alignment function with the command:

```
use align
```

Editor Commands: *verify*

The *verify* command sets or clears editor state parameters.

Syntax

v[*erify*] [*on* | *off*] [*num* | *nonum*] [*log* | *nolog*] [*prompt* | *noprompt*]

on | *off* Control verification mode.

on	Displays the new reference line after commands which change it. Default for line mode terminals.
off	Suppresses the display of the new reference line after commands which change it. Default for 3X7X type terminals.

num | *nonum* Controls the display of line numbers.

num	Displays line numbers to the left of each line.
nonum	Suppresses the display of line numbers to the left of each line.

log | *nolog* Controls the logging of LOGOS command output to the standard screen during sessions in full screen mode.

log	Writes output and commands entered to the standard session log, providing an audit trail of LOGOS commands executed during an editor session.
nolog	Display command output in the editor window (in full screen mode) but does not enter it in the session log.

prompt | *noprompt*
Controls whether the editor prompts for command input.

prompt	Prompts for input.
noprompt	Does not prompt for input.

Usage

verify with no arguments returns the current setting for each option.

Editor Commands: *vput*

The *vput* command extracts selected text from the edited object and places it in an APL variable or the editor buffer. This is similar to the *put* command. However, *vput* creates character vector variables, where *put* writes data in character matrix format.

Syntax

```
vp[ut] [integer|*] [variable]
       [+task[=task]]
```

integer | *	Indicates the number of lines to extract, including the reference line.
	integer — Extracts the specified number of lines.
	* — Extracts all lines from the reference line to the end of the object.
	−* — Extracts all lines from the reference line to the beginning of the object.
variable	Is the name of the variable to which the extracted text is to be assigned.
+task[=**task**]	Allows editing of objects in the workspace of an auxiliary task started from within LOGOS. If *+task* is specified without an argument, the default task name *aux* is assumed. If *+task* is omitted, the task named in the *environment task* parameter, if any, is used.

Usage

If you do not specify the first argument, a default value of 1 is used.

If you do not specify the second argument, the text is placed into the editor's internal buffer. The only way to read this buffer is to use the *get* command without an argument.

Examples

top ᵘ *vput* ⋆ *vec*

Puts every line of text in the edited object in the character vector *vec*.

top ᵘ *vput 3* ᵘ *bottom* ᵘ *get*

Places a copy of the first three lines of the object at the end of the object.

Editor Commands: *window*

The *window* command configures and manipulates the editor's output window. Windows are created only during full screen editing session.

Syntax

wi[*ndow*] [*up*] [**integer** | **+integer** | **−integer** | **/string**]

up	Reverses the direction of the command.
integer	Sets the maximum number of lines in a window.
+integer	Scrolls the window forward that number of lines.
−integer	Scrolls the window backwards that number of lines.
/string	Puts the line containing the string on the first line of the window.

Result

When a command creates output, the window displays that output. For example, if the window is set to 12 lines and the *xref* command creates 20 lines of output, the window displays 12 lines of output. On the other hand, if the command creates 10 lines of output, the window is 10 lines.

Usage

If you specify a string, *window* searches forwards from the first line currently displayed, unless you specify the *up* variant, in which case *window* searches backwards from the first line currently displayed.

The largest window size you can set is 12 less than the number of lines the terminal displays. On a 24 line screen, a window of 12 lines is the largest size you can set.

Editor Commands: *xref*

The *xref* command produces a cross-reference listing of the function or script being edited.

Syntax

```
xr[ef]
        [+quotes]
        [+symbols=symbols]
```

+quotes	Examines quoted strings logically appearing after executes within the function or script as if they were unquoted.
+symbols=**symbols**	Cross-references arbitrary, additional APL symbols along with ordinary APL identifiers. If this is omitted, the symbols → ⍎ : ⎕ ⍞ are cross-referenced by default.

Result

For each identifier located within a program, *xref* computes the identifier's local type, and an indication of whether or not the name occurs suspiciously (for example, a local variable which is not assigned a value, or a name which is used to define a line-label more than once, is considered suspicious). The identifiers are:

la	left argument
ll	line label
lv	local variable
qf	quad (system) function
ra	right argument
rs	result
⋆	indeterminate

xref marks suspicious identifiers with a query (?) after its type.

For each identifier reference within a function or script, *xref* computes the line number of the reference, and its type. The reference type can be:

(blank)	simple reference
←	simple assignment
[	indexed reference
[←	Indexed assignment
:	Line-label definition

In full screen mode, the listing displays in the editor window.

Examples

---

xref +*s*=+→←ρ

Produces a cross reference of objects and includes any use of the symbols +, →, ←, and ρ.

---

xref +*q* +*s*=⎕⍝⍎

Produces a cross reference listing that includes references to objects inside quoted strings. The result also includes uses of the symbols ⎕, ⍝, and ⍎.

EDITOR FUNCTION KEY SUMMARY

Function Key	Description
F1	Help.
F2	Toggle the line number protection on and off.
F3	End (save the object), or close the display window.
F4	Enter input mode on the line after the cursor.
F5	Switch to the next object on the edit stack.
F6	If the cursor is in the edit area, make the line the cursor is on the reference line. If the cursor is in the command area, move the cursor to the end of the reference line.
F7	Scroll backward.
F8	Scroll forward.
F9	Recall the last command line.
F10	Split the line the cursor is on at the cursor location.
F11	Move the cursor to the end of the line it is currently positioned on.
F12	Switch to journal, documentation, or source attribute.

THE EDITOR UTILITY INTERFACE

About the Editor Utility Interface

The function *Δledutil* provides an interface between the editor and scripts invoked from within the editor. This makes it possible to extend the editor by writing scripts that use *Δledutil* to manipulate the editor buffer, execute editor commands under program control, etc.

Summary of Right Arguments

The right argument to *Δledutil* is a character vector command name. The following table summarizes the commands and what they do:

Command	Description
cmd	execute editor command in left argument
readbuf	read edit buffer
writebuf	write left argument to edit buffer
readref	read location of reference line
writeref	set refernce line to index in left argument
readnos	read line numbers
writenos	set line numbers to left argument
readhi	read highlighting information
writehi	write highlighting information in left argument
readwin	read window contents
writewin	write left argument to window
readwref	read window reference line position
writewref	set window reference line position
writemsg	write left argument to message area
status	read object status

Detailed Command Descriptions

cmd: execute editor command

The left argument is passed to the editor for evaluation as a command line. The result is a return code from the command. Currently, this is always 0. Future versions of the interface will return other values.

readbuf: read edit buffer

The contents of the edit buffer are read and returned as the function result. This result will always be a character vector, regardless of the type of the object being edited. Every line, including the last one, will be terminated with a carriage return.

writebuf: write edit buffer

The left argument (which must be a character vector) is written to the edit buffer, replacing the previous contents.

readref: read location of reference line

The result is the index of the reference line within the edit buffer. This index is ⎕*io*-relative.

writeref: set reference line

The reference line is set to the line whose index appears in the integer left argument. The value of this argument is ⎕*io*-sensative.

readnos: read line numbers

The result is an integer vector of line numbers. The numbers returned are generated by multiplying the line numbers displayed by the editor by 10,000, guaranteeing an integer result.

writenos: set linenumbers

The line numbers displayed by the editor are replaced by those in the integer vector left argument. The values in the left argument must be 10,000 times the values you expect the editor to display.

readhi: read highlighting information

The result returned is a two-element vector of enclosures representing the current highlighting attributes. The first element is an n×2 character matrix of attributes (e.g. 'pi' for 'pink' and 'inverse'). See the description of the editor ***highlight*** command for a complete list of attributes. The second element is an enclosed n×m boolean matrix. Each row corresponds to a row of the attribute descriptions in the first enclosed element. Each column corresponds to a position in the edit buffer. If an element is 1, then the corresponding position in the edit buffer has the indicated attributes set.

writehi: write highlighting information

The left argument must conform to the structure returned by the 'readhi' operation. This command writes the attributes indicated to the screen.

readwin: read window contents

The current contents of the window are returned as a character vector.

writewin: write to window

The character vector left argument is written to the editor window. Optionally, a two element vector of enclosed character vectors may be passed as the left argument. The first enclosed vector is written to the 'title line' of the window. The second is written to the window proper.

readwref: read window reference line position

The result is the index of the window reference line within the window buffer. This index is ⎕*io*-relative. If no window exists, ¯*1* is returned.

writewref: set window reference line position

The window reference line is set to the line whose index appears in the integer left argument. The value of this argument is ⎕*io*-sensitive.

writemsg: write message

The character vector left argument is written to the editor message area.

status: return status information

The result of this operation is a two element vector of enclosures:

[0] – n×m+2 matrix of type/attribute/pathname information

[;0]	[;1]	0 2↓⍵
object type	*attribute*	*pathname*

[1] – n×6 matrix of integers:

[;0]	[;1]	[;2]	[;3]
version number	*variable type* *0 = char* *1 = num*	*variable rank*	*modified flag* *1 = modified;* *0 = not modified*

[;4]	[;5]
browse state *0 = read/write access* *1 = browse* *2 = browse lock* *3 = read-only access*	*registration* *0 = not registered out* *1 = registered out*

Using Δ*ledutil*

The Δ*ledutil* function must be used from within a script that is called during your editor session. For example, suppose you have created a script with the following definition that uses Δ*ledutil*.

```
[1] copyit +Start= +Count= +Dest=;x;lines; .public.logos.util.Δledutil;tmp;⎕trap
[2] ⎕trap ←'∇ 0 c →e2'
[3] lines←0.0001×Δledutil'readnos'
[4] →(~(⎕fi Dest)∊lines)/e1
[5] x←'⊃[',Start,']'
[6] x←x,'⊃put ',Count,'  tmp  '
[7] x←x,'⊃[',Dest,']'
[8] x←x,'⊃get tmp'
[9] ⍎x Δledutil 'cmd'  ⋄ →0
[10] e1:'invalid destination'Δledutil 'writemsg'  ⋄ →0
[11] e2:('script error ',⎕er[⎕io;]) Δledutil 'writemsg'
```

During your editor session, you could use your script instead of the editor's *copy* command to copy the 5 lines starting at line 8 and inserting them after line 20 of the object in the editor:

```
∇u )copyit 8 5 20
```

LOGOS COMMAND CROSS REFERENCE

Cross Reference of Arguments

Argument	Command
[=]names	∇ *edit*
in | out | on | off	*register*
[alias]	*alias enroll group*
aliases	*share*
arguments	*with*
[command]	*? ?? help*
count	*retain*
[definition]	*keyword*
[destination]	*build shell*
[expression]	⍎ *exit*
expression	⍕ *with*
[filename]	*filesave*
id	*whois*
[line]	*send*
[namelist]	*snap*
namelist	*transfer*
[name]	*keyword*
newpathname	*link*
[newpathname]	*import*
newstring	*replace*
[oldpathname]	*import*

Cross Reference of Arguments, continued

Argument	**Command**
oldpathname	*link*
oldstring	*replace*
[**parameter**]	*environment*
[**password**]	*alias*
[**pathnames**]	*build cmddir list locate references workdir*
pathnames	*calls copy delete display distribute export get register replace retain save share summarize syntax wstofile xref*
pathname	*filemaint import*
primaries	*compare*
[**secondaries**]	*compare*
[**source**]	*shell*
string	*locate*
[**task**]	*talk tasks*
text	*output*
[**user**]	*enroll*
[**usernumber | alias**[**:password**]]	*signon*
[**value**]	*environment*
[**wsid**]	*wssave*

Cross Reference of Modifiers

Modifier	Commands
+*alias*=**alias**	*enroll group*
+*all*	*syntax whois*
+*asynch*	*send*
+*attributes*=*c* | *d* | *j* | *n* | *t*	*wstofile*
+*attributes*[=*c* | *d* | *j* | *n* | *s* | *t*]	*compare*
+*audit*=**filename**	*build distribute filesave shell snap wssave*
+*audit*	*references*
+*break*	*send*
+*browse*[=*off* | *on* | *lock*]	∇ *edit*
+*cmddir*=**pathnames**	*enroll*
+*column*	*list*
+*command*=**command**	∇ *edit*
+*comments*	*summarize*
+*compile*=**cd**	*build calls get shell*
+*compile*[=**cd**]	*compare display distribute*
+*compress*	*filemaint*
+*conditional*	*register*
+*confirm*	*delete snap*
+*data*[=*pn,type,...*]	*list*
+*delete*[=*yes*]	*enroll group*

Cross Reference of Modifiers, continued

Modifier	**Commands**
+delete	*references share*
+depth[=*all* | **n**]	*build calls*
+destack	*environment*
+display	*compare locate replace syntax*
+disttask=**task**	∇ *edit*
+end=**n**	*filesave*
+enrollment=[, | /]**aliases**	*group*
+environments=**envs**	*distribute references*
+error	*output*
+exclude=**names**	*calls snap*
+exclude=[, | /]**names**	*build shell*
+expand	*summarize*
+extended	*compare filemaint list whois*
+file=**filename**	*build shell wstofile*
+flags[=*b* | *l* | *s*]	*compare*
+flags=c | *l* | *n* | *o* | *q* | *s* | *x*	*locate replace*
+flags=[, | /]*i* | *c*	*group*
+flags=[, | /]*i* | *p* | *s* | *m*	*enroll*
+from[=**task**]	*transfer*
+full	*list summarize*

Cross Reference of Modifiers, continued

Modifier	**Commands**
+*groups*=[,|/]**groups**	*enroll*
+*header*=,|/**names**	*shell*
+*headings*	*list references summarize tasks*
+*immex*	*send*
+*inference=yes|no*	*build*
+*information*	*wssave*
+*interleave*	*compare*
+*in*	*copy save*
+*keepin*=**n**	*build*
+*lines*	*compare locate replace syntax*
+*lock*=**n**	*buildfilesave shell*
+*long*	*list*
+*makedir*	*copy export save snap*
+*match*	*compare*
+*mentor*=**alias**	*group*
+*message*	*output*
+*multiple*	*locate replace*
+*name*=**name**	*enroll group shell*
+*name*	*whois*
+*newpass*[=**password**]	*alias*
+*noncurrent*	*delete*

Cross Reference of Modifiers, continued

Modifier	Commands
+*nopathname*	*display*
+*note*=**text**	*filesave*
+*notfound*	*calls*
+*overhead*	*list*
+*override*	∇ *copy delete distribute edit export register replace save*
+*overwrite*	*snap wssave*
+*overwrite*[=*audit* | , | *buffer* | , | *dest*]	*buildfilesave*
+*password*=**password**	*enroll*
+*pathnames*	*calls wstofile*
+*pathname*	*references*
+*permission*=*c* | *w* | *r* | *x*	*share*
+*profile*	*environment*
+*prompt*=**prompt**	*talk*
+*protect*	*build copy export get save transfer*
+*qlx*=**expression**	*shell*
+*quadprime*	*output*
+*quotes*	*syntax xref*

Cross Reference of Modifiers, continued

Modifier	Commands
+recursive[*=1* | *2* | *all*]	*build compare get list locate references register replace retain share summarize syntax wstofile*
+register	∇ *edit*
+replacement=**pathnames**	*distribute*
+reset=yes	*enroll*
+reset	*cmddir environment exit workdir*
+result	*output*
+retract[*=on* | *off*]	*send*
+retract	*signon*
+script=**pathname**	*snap*
+show	*compare distribute locate replace syntax*
+size=**n**	*buildfilesave*
+skeleton=**pathname**	*shell*
+stack	*environment*
+state=**pathname**	*wstofile*
+status=**text**	∇ *edit*
+status	*output*
+summary	*list whois*
+suppress	*send transfer*
+surrogate=**character**	*with*

Cross Reference of Modifiers, continued

Modifier	**Commands**
+*surrogates*[=*b* | *c* | *i* | *l* | *n*]	*display*
+*symbols*=**symbols**	*xref*
+*task*[=**task**]	∇ *build distribute edit get save send shell signon wssave*
+*to*[=**task**]	*transfer*
+*type*	*list*
+*ultimate*	*list*
+*unreferenced*	*references*
+*unused*	*delete*
+*update*[=**nodes**]	*build filesave*
+*user*=**user**	*enroll wssave*
+*users*=**users**	*tasks*
+*value*=[±]**value**	*save*
+*variant*=*a* | *e* | *l* | *r* | *s* | *t*	*shell*
+*versions*[=**n**]	*copy export list summarize*
+*view*	*snap*
+*warn*=**n**	*delete*
+*workdir*=**pathnames**	∇ *build calls edit enroll get snap*
+*workdir*=**pathname**	*save*
+*wsid*=**wsid**	*wstofile*