Native Files

`{R} ← {X} ⎕MKDIR Y`	Creates new fully-qualified files/directories Y along with intermediate directories if required, as specified by X. X can be any summation of the following values (maximum value = 3): • 0 : directory Y is only created if its path already exists and its base name does not exist. This is the default. • 1 : no action is taken if directory Y already exists. • 2 : directory Y and any part of its path that does not already exist is created. Returns 1 for each file/directory specified in Y that is created successfully, 0 otherwise (if X is 0 or 2 then an error message is also generated if directory Y cannot be created successfully).
`{R} ← X ⎕NAPPEND Y`	Appends the ravel of array X to the end of the native file that has tie number Y[1]; optionally, Y[2] can specify the conversion code to use to convert array X (by default, 80 is assumed when using the Unicode version – see *Conversion Codes*).
`{R} ← X ⎕NCOPY Y`	Copies native files/directories from one or more sources specified by Y to destination X. If X specifies an existent directory then each source in Y is copied into that directory, otherwise X specifies the name of the copy. X must specify an existent directory if Y contains multiple names or if the *Wildcard* variant option (Boolean) is set. For files, variant options can preserve attributes when copying (*PreserveAttributes*, Boolean) and define behaviour if the target file already exists (*IfExists* can be 'Error', 'Skip', 'Replace' or 'ReplaceIfNewer'). The *ProgressCallBack* variant option invokes an APL callback function before, during, and after copying each file/directory.
`{R} ← X ⎕NCREATE Y`	Creates a new native file with name X and file tie number Y; a tie number of 0 allocates the next available tie number to the file. Optionally (requires variant) allows semi-automatic naming of files and for existing files to be overwritten.
`{R} ← {X} ⎕NDELETE Y`	Deletes each fully-qualified file/empty directory specified in Y, returning a shy result of the count of top-level entities successfully deleted. A wildcard option (requires variant) can be used. If a file/directory specified in Y does not exist then behaviour is determined by X. X can be any summation of the following values (maximum value = 3): • 0 : each file/directory specified in Y must exist. This is the default. • 1 : no action is taken if a file/directory specified in Y does not exist. • 2 : any non-empty directory specified in Y and its contents are deleted.
`{R} ← X ⎕NERASE Y`	Erases the tied native file that has name X and file tie number Y.
`R ← ⎕NEXISTS Y`	Returns 1 for each of the fully-qualified files/directories specified in Y that exists (can be accessed), 0 otherwise.
`R ← {X} ⎕NGET Y`	Reads a text file and returns a 3-element vector in which: • [1] is the character data in the file. • [2] is a character vector specifying the file encoding (see *File Encoding*). • [3] is numeric and is either ⍬ (if no line separator was found in the file) or a vector specifying the first newline separator found in the file (see *Line Separators*). Y is either a character vector/scalar containing the name of the file to be read or a 2-item vector in which [1] is the filename and [2] is an integer scalar specifying `flags`: • if `flags` is 0 then R[1] is a character vector. This is the default. • if `flags` is 1 then R[1] is a nested array of character vectors. X is a character vector specifying the decoding format to use if the specified file does not start with a recognised BOM (see *File Encoding*). If no BOM is present and no decoding format is specified, then the file is examined and its encoding format is deduced.
`R ← {X} ⎕NINFO Y`	Returns an array of the information specified by X about Y (file/directory names or native file tie numbers). X can specify any of the following values, in any order: • 0 : Name of Y. This is the default. • 1 : Type of Y. This can be 0 (not known), 1 (directory), 2 (file), 3 (character device) or 4 (symbolic link). On UNIX and Mac OS, can also be 5 (block device), 6 (FIFO) or 7 (socket). • 2 : Size of Y in bytes. • 3 : The time Y was last modified as a timestamp in ⎕TS format (local time). • 4 : The user ID of the owner of Y. • 5 : The name of the owner of Y.

	• 6 : Whether Y is hidden. • 7 : Target of symbolic link – only applies when Type=4. • 8 : Current file position (offset) – only applies when the source in Y is a native file tie number. • 9 : The time Y was last accessed as a timestamp in ⎕TS format if available (local time). • 10 : The time Y was created as a timestamp in ⎕TS format if available (local time). • 11 : Whether Y can be read. • 12 : Whether Y can be written to. • 13: The time Y was last modified as a Dyalog Date Number (UTC). • 14: The time Y was last accessed as a Dyalog Date Number (UTC). • 15: The time Y was created (if available, otherwise the time of the last status change) as a Dyalog Date Number (UTC). The file/directory name can include wildcard options, recursion through sub-directories and property reporting for symbolic links (require variant); in these cases, the result R remains the same shape as X but its depth increases.
`{R} ← X ⎕NLOCK Y`	Changes the lock status (as defined by X) of part of the native file that has file tie number `Y[1]`; optionally, `Y[2]` can define the offset from 0 of the first byte to apply the lock change to (defaults to 0) and `Y[3]` can specify the number of bytes impacted by the lock change (defaults to the maximum possible file size) (see *File Encoding*).
`R ← ⎕NNAMES`	Lists the names of all tied native files.
`{R} ← X ⎕NMOVE Y`	Moves native files/directories from one or more sources specified by Y to destination X. If X specifies an existent directory then each source in Y is moved into that directory. X must specify an existent directory if Y contains multiple names or if the *Wildcard* variant option (Boolean) is set. For files, variant options can define behaviour if moving to a different file system (*RenameOnly*, Boolean) and or if the target file already exists (*IfExists* can be 'Error' or 'Skip'). The *ProgressCallBack* variant option invokes an APL callback function before, during, and after moving each file/directory.
`R ← ⎕NNUMS`	Lists the tie numbers of all tied native files.
`R ← {X} ⎕NPARTS Y`	Splits the fully-qualified files/directories specified in Y into constituent parts, returning a 3-element vector (or vector of 3-element vectors if Y comprises multiple files/directories) in which: • `R[1]` is the specified path. If X is 1 then the fully-qualified path is derived and returned. • `R[2]` is the base name, that is, the filename stripped of its path and extension. • `R[3]` is the file extension, including the leading `.` character.
`{R} ← X ⎕NPUT Y`	Writes character data X to a text file Y and returns (shy) the number of bytes successfully written to the file. Y is either a character vector/scalar containing the name of the file to be written or a 2-item vector in which `[1]` is the filename and `[2]` is an integer scalar specifying `flags`: • if `flags` is 0 then, if the file already exists, it will not be overwritten and an error is generated. This is the default. • if `flags` is 1 then the file will be written irrespective of whether it already exists. • if `flags` is 2 then X will be appended to Y. X can comprise up to three elements: • `[1]` is a vector of character vectors, each of which represents a line in the file to be written, or a simple character vector. • `[2]` is a character vector specifying the encoding to use (see *File Encoding*). Optional – the default is UTF-8-NOBOM. • `[3]` is numeric and is either ⍬ (same as the default) or a scalar/vector specifying the newline separator to use (see *Line Separators*). Optional – the default is (13 10) on Microsoft Windows and 10 on other platforms.
`R ← ⎕NREAD Y`	Reads the content of the native file identified by file tie number `Y[1]`; `Y[2]` specifies the conversion code to use (see *Conversion Codes*), `Y[3]` specifies the count (see *Conversion Codes*) and, optionally, `Y[4]` can define the offset from 0 of the first byte to read.
`{R} ← X ⎕NRENAME Y`	Renames the tied native file that has file tie number Y to have name X.

`{R ← X ⎕NREPLACE Y`	Replaces content in a native file identified by file tie number `Y[1]` with `X`; `Y[2]` defines the offset from 0 of the first byte to replace and, optionally, `Y[3]` specifies the conversion code to use (by default, 80 is assumed when using the Unicode version) (see *Conversion Codes*).
`{R} ← X ⎕NRESIZE Y`	Changes the size of the native file that has file tie number `Y` to size `X` (either by truncating the file or by extending it with undefined additional bytes).
`R ← ⎕NSIZE Y`	Returns the size in bytes of the native file that has file tie number `Y`.
`{R} ← X ⎕NTIE Y`	Ties the native file that has name `X` using file tie number `Y[1]`; optionally, `Y[2]` can specify the type of access needed by other users (see *Access Codes*).
`{R} ← ⎕NUNTIE Y`	Unties all native files that have a tie number in vector `Y` (`⎕NUNTIE 0` does not untie any files but flushes all file caches to disk) and returns the tie numbers of native files that have been untied.
`{R} ←{X} ⎕NXLATE Y`	Associates the native file that has tie number `Y` with character translation vector `X`. Note that: • if `X` is not specified then the currently-associated translation vector is returned. • if `X` has the value `(⍳256)-⎕IO` then the translation process is bypassed and raw input/output is provided. • if `Y` is set to 0, then the translate vector used by `⎕DR` is used. Unicode version only: This is only relevant when processing native files that contain characters expressed as indices into `⎕AV`.

Access Codes

The access codes used by `⎕NTIE` are integer values calculated as the sum of:

- the type of access needed from users who have already tied the native file
- the type of access to grant to users who subsequently try to open the file while you have it open

Needed from existing users	
0	read access
1	write access
2	read and write access

Granted to subsequent users	
0	compatibility mode
16	no access (exclusive)
32	read access
48	write access
64	read and write access

Conversion Codes

The conversion codes used by `⎕NAPPEND`, `⎕NREAD` and `⎕NREPLACE` vary according to the installation of Dyalog used to read the native file; the following tables show the conversion codes for the Unicode and Classic version respectively.

Value	Number of Bytes	Result Type	Result Shape
11	count	1 bit Boolean	8 x count
80	count	8 bit character	count
82*	count	8 bit character	count
83	count	8 bit integer	count
160	2 x count	16 bit character	count
163	2 x count	16 bit integer	count
320	4 x count	32 bit character	count
323	4 x count	32 bit integer	count
645	8 x count	64 bit floating	count

Value	Number of Bytes	Result Type	Result Shape
11	count	1 bit Boolean	8 x count
-	-	-	-
82	count	8 bit character	count
83	count	8 bit integer	count
-	-	-	-
163	2 x count	16 bit integer	count
-	-	-	-
323	4 x count	32 bit integer	count
645	8 x count	64 bit floating	count

* Conversion code 82 is permitted in the Unicode Edition for compatibility and causes 1-byte data on file to be *translated* (according to `⎕NXLATE`) from `⎕AV` indices into normal (Unicode) characters of type 80, 160 or 320.

File Encoding

The file encoding used by ⎕NGET and ⎕NPUT. The UTF formats can be qualified with -BOM (for example, UTF-8-BOM) or –NOBOM (for example, UTF-16LE-NOBOM) to specify whether a BOM is/should be present; this qualification is always present when returned by ⎕NGET.

Encoding	Description
UTF-8	Data is encoded into UTF-8 format. If -BOM or -NOBOM is not appended, the default is -NOBOM.
UTF-16 UTF-16BE UTF-16LE	Data is encoded into UTF-16 format with either big or little endianness. The default for UTF-16 is the endianness of the host system (BE on AIX platforms, LE on others). If -BOM or -NOBOM is not appended, the default is -BOM.
UTF-32 UTF-32BE UTF-32LE	Data is encoded into UTF-32 format with either big or little endianness. The default for UTF-32 is the endianness of the host system (BE on AIX platforms, LE on others). If -BOM or -NOBOM is not appended, the default is -BOM.
ASCII	Data is encoded into 7-bit ASCII format
Windows-1252	Data is encoded into 8-bit Windows-1252 format
ANSI	ANSI is a synonym of Windows-1252

Line Endings

The line terminators recognised by ⎕NGET and ⎕NPUT.

For ⎕NGET:

- if R[1] is simple, then it comprises the contents of the file with all line separators normalised to ⎕UCS 10.
- if R[1] is nested, then it comprises the contents of the file split on the occurence of any of the line separators.

Value	Char	Description	Notes
13	CR	Carriage Return (U+000D)	Newline separators recognised by ⎕NGET (R[3]) and ⎕NPUT (X[3]).
10	LF	Line Feed (U+000A)	
13 10	CRLF	Carriage Return followed by Line Feed	
133	NEL	New Line (U+0085)	
11	VT	Vertical Tab (U+000B)	
12	FF	Form Feed (U+000C)	
2028	LS	Line Separator (U+2028)	
2029	PS	Paragraph Separator (U+2029)	

In addition, ⎕NPUT's NEOL variant specifies how embedded line endings are treated (0 = embedded line ending characters are preserved as is, 1 = every embedded LF is replaced by the host-specific line terminator (the default), and 2 = every embedded line ending character is replaced by the host-specific line terminator).

File Locking

Unlike component files, which can be tied with an exclusive tie or a share tie, native files cannot be tied in different ways. Instead, ⎕NLOCK is used to lock byte ranges within files, thereby managing access between users. There are three possible lock statuses:

- 1 means unlock
- 2 means read (share) lock – multiple read locks can exist over the same byte-range. The presence of a read lock prevents a write lock being obtained
- 3 means write lock – only one write lock can exist for a specific byte-range of a native file. The presence of a write lock prevents a read lock being obtained

The lock status can also, optionally, define a timeout period in seconds; if this period is exceeded before the lock status change has occurred, then a TIMEOUT error is displayed (defaults to no timeout limit).

Different file servers can follow different locking standards – ⎕NLOCK does not standardise this.