

# Native Files

<p><code>{R} ← {X} □MKDIR Y</code></p>	<p>Creates new fully-qualified files/directories <b>Y</b> along with intermediate directories if required, as specified by <b>X</b>. <b>X</b> can be any summation of the following values (maximum value = 3):</p> <ul style="list-style-type: none"> <li>• 0 : directory <b>Y</b> is only created if its path already exists and its base name does not exist. This is the default.</li> <li>• 1 : no action is taken if directory <b>Y</b> already exists.</li> <li>• 2 : directory <b>Y</b> and any part of its path that does not already exist is created.</li> </ul> <p>Returns 1 for each file/directory specified in <b>Y</b> that is created successfully, 0 otherwise (if <b>X</b> is 0 or 2 then an error message is also generated if directory <b>Y</b> cannot be created successfully).</p>
<p><code>{R} ← X □NAPPEND Y</code></p>	<p>Appends the ravel of array <b>X</b> to the end of the native file that has tie number <b>Y</b>[ 1 ]; optionally, <b>Y</b>[ 2 ] can specify the conversion code to use to convert array <b>X</b> (by default, 80 is assumed when using the Unicode version – see <i>Conversion Codes</i>).</p>
<p><code>{R} ← X □NCOPY Y</code></p>	<p>Copies native files/directories from one or more sources specified by <b>Y</b> to destination <b>X</b>. If <b>X</b> specifies an existent directory then each source in <b>Y</b> is copied into that directory, otherwise <b>X</b> specifies the name of the copy. <b>X</b> must specify an existent directory if <b>Y</b> contains multiple names or if the <i>Wildcard</i> variant option (Boolean) is set. For files, variant options can preserve attributes when copying (<i>PreserveAttributes</i>, Boolean) and define behaviour if the target file already exists (<i>IfExists</i> can be 'Error', 'Skip', 'Replace' or 'ReplaceIfNewer'). The <i>ProgressCallback</i> variant option invokes an APL callback function before, during, and after copying each file/directory.</p>
<p><code>{R} ← X □NCREATE Y</code></p>	<p>Creates a new native file with name <b>X</b> and file tie number <b>Y</b>; 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.</p>
<p><code>{R} ← {X} □NDELETE Y</code></p>	<p>Deletes each fully-qualified file/empty directory specified in <b>Y</b>, 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 <b>Y</b> does not exist then behaviour is determined by <b>X</b>. <b>X</b> can be any summation of the following values (maximum value = 3):</p> <ul style="list-style-type: none"> <li>• 0 : each file/directory specified in <b>Y</b> must exist. This is the default.</li> <li>• 1 : no action is taken if a file/directory specified in <b>Y</b> does not exist.</li> <li>• 2 : any non-empty directory specified in <b>Y</b> and its contents are deleted.</li> </ul>
<p><code>{R} ← X □NERASE Y</code></p>	<p>Erases the tied native file that has name <b>X</b> and file tie number <b>Y</b>.</p>
<p><code>R ← □NEXISTS Y</code></p>	<p>Returns 1 for each of the fully-qualified files/directories specified in <b>Y</b> that exists (can be accessed), 0 otherwise.</p>
<p><code>R ← {X} □NGET Y</code></p>	<p>Reads a text file and returns a 3-element vector in which:</p> <ul style="list-style-type: none"> <li>• [ 1 ] is the character data in the file.</li> <li>• [ 2 ] is a character vector specifying the file encoding (see <i>File Encoding</i>).</li> <li>• [ 3 ] is numeric and is either <math>\emptyset</math> (if no line separator was found in the file) or a vector specifying the first newline separator found in the file (see <i>Line Separators</i>).</li> </ul> <p><b>Y</b> 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 <b>f l a g s</b>:</p> <ul style="list-style-type: none"> <li>• if <b>f l a g s</b> is 0 then <b>R</b>[ 1 ] is a character vector. This is the default.</li> <li>• if <b>f l a g s</b> is 1 then <b>R</b>[ 1 ] is a nested array of character vectors.</li> </ul> <p><b>X</b> is a character vector specifying the decoding format to use if the specified file does not start with a recognised BOM (see <i>File Encoding</i>). If no BOM is present and no decoding format is specified, then the file is examined and its encoding format is deduced.</p>
<p><code>R ← {X} □NINFO Y</code></p>	<p>Returns an array of the information specified by <b>X</b> about <b>Y</b> (file/directory names or native file tie numbers). <b>X</b> can specify any of the following values, in any order:</p> <ul style="list-style-type: none"> <li>• 0 : Name of <b>Y</b>. This is the default.</li> <li>• 1 : Type of <b>Y</b>. 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).</li> <li>• 2 : Size of <b>Y</b> in bytes.</li> <li>• 3 : The time <b>Y</b> was last modified as a timestamp in <math>\square</math>TS format (local time).</li> <li>• 4 : The user ID of the owner of <b>Y</b>.</li> <li>• 5 : The name of the owner of <b>Y</b>.</li> </ul>

	<ul style="list-style-type: none"> <li>• 6 : Whether <b>Y</b> is hidden.</li> <li>• 7 : Target of symbolic link – only applies when Type=4.</li> <li>• 8 : Current file position (offset) – only applies when the source in <b>Y</b> is a native file tie number.</li> <li>• 9 : The time <b>Y</b> was last accessed as a timestamp in <code>□TS</code> format if available (local time).</li> <li>• 10 : The time <b>Y</b> was created as a timestamp in <code>□TS</code> format if available (local time).</li> <li>• 11 : Whether <b>Y</b> can be read.</li> <li>• 12 : Whether <b>Y</b> can be written to.</li> <li>• 13: The time <b>Y</b> was last modified as a Dyalog Date Number (UTC).</li> <li>• 14: The time <b>Y</b> was last accessed as a Dyalog Date Number (UTC).</li> <li>• 15: The time <b>Y</b> was created (if available, otherwise the time of the last status change) as a Dyalog Date Number (UTC).</li> </ul> <p>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 <b>R</b> remains the same shape as <b>X</b> but its depth increases.</p>
<code>{R} ← X □NLOCK Y</code>	Changes the lock status (as defined by <b>X</b> ) of part of the native file that has file tie number <b>Y</b> [ 1 ]; optionally, <b>Y</b> [ 2 ] can define the offset from 0 of the first byte to apply the lock change to (defaults to 0) and <b>Y</b> [ 3 ] can specify the number of bytes impacted by the lock change (defaults to the maximum possible file size) (see <i>File Encoding</i> ).
<code>R ← □NNAMES</code>	Lists the names of all tied native files.
<code>{R} ← X □NMOVE Y</code>	Moves native files/directories from one or more sources specified by <b>Y</b> to destination <b>X</b> . If <b>X</b> specifies an existent directory then each source in <b>Y</b> is moved into that directory. <b>X</b> must specify an existent directory if <b>Y</b> contains multiple names or if the <i>Wildcard</i> variant option (Boolean) is set. For files, variant options can define behaviour if moving to a different file system ( <i>RenameOnly</i> , Boolean) and or if the target file already exists ( <i>IfExists</i> can be 'Error' or 'Skip'). The <i>ProgressCallBack</i> variant option invokes an APL callback function before, during, and after moving each file/directory.
<code>R ← □NNUMS</code>	Lists the tie numbers of all tied native files.
<code>R ← {X} □NPARTS Y</code>	Splits the fully-qualified files/directories specified in <b>Y</b> into constituent parts, returning a 3-element vector (or vector of 3-element vectors if <b>Y</b> comprises multiple files/directories) in which: <ul style="list-style-type: none"> <li>• <b>R</b> [ 1 ] is the specified path. If <b>X</b> is 1 then the fully-qualified path is derived and returned.</li> <li>• <b>R</b> [ 2 ] is the base name, that is, the filename stripped of its path and extension.</li> <li>• <b>R</b> [ 3 ] is the file extension, including the leading <code>.</code> character.</li> </ul>
<code>{R} ← X □NPUT Y</code>	Writes character data <b>X</b> to a text file <b>Y</b> and returns (shy) the number of bytes successfully written to the file. <b>Y</b> 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 <b>f l a g s</b> : <ul style="list-style-type: none"> <li>• if <b>f l a g s</b> is 0 then, if the file already exists, it will not be overwritten and an error is generated. This is the default.</li> <li>• if <b>f l a g s</b> is 1 then the file will be written irrespective of whether it already exists.</li> <li>• if <b>f l a g s</b> is 2 then <b>X</b> will be appended to <b>Y</b>.</li> </ul> <b>X</b> can comprise up to three elements: <ul style="list-style-type: none"> <li>• [ 1 ] is a vector of character vectors, each of which represents a line in the file to be written, or a simple character vector.</li> <li>• [ 2 ] is a character vector specifying the encoding to use (see <i>File Encoding</i>). Optional – the default is UTF-8-NOBOM.</li> <li>• [ 3 ] is numeric and is either <math>\emptyset</math> (same as the default) or a scalar/vector specifying the newline separator to use (see <i>Line Separators</i>). Optional – the default is (13 10) on Microsoft Windows and 10 on other platforms.</li> </ul>
<code>R ← □NREAD Y</code>	Reads the content of the native file identified by file tie number <b>Y</b> [ 1 ]; <b>Y</b> [ 2 ] specifies the conversion code to use (see <i>Conversion Codes</i> ), <b>Y</b> [ 3 ] specifies the count (see <i>Conversion Codes</i> ) and, optionally, <b>Y</b> [ 4 ] can define the offset from 0 of the first byte to read.
<code>{R} ← X □NRENAME Y</code>	Renames the tied native file that has file tie number <b>Y</b> to have name <b>X</b> .

<code>{R} ← X □NREPLACE Y</code>	Replaces content in a native file identified by file tie number <code>Y[1]</code> with <code>X</code> ; <code>Y[2]</code> defines the offset from 0 of the first byte to replace and, optionally, <code>Y[3]</code> specifies the conversion code to use (by default, 80 is assumed when using the Unicode version) (see <i>Conversion Codes</i> ).
<code>{R} ← X □NRESIZE Y</code>	Changes the size of the native file that has file tie number <code>Y</code> to size <code>X</code> (either by truncating the file or by extending it with undefined additional bytes).
<code>R ← □NSIZE Y</code>	Returns the size in bytes of the native file that has file tie number <code>Y</code> .
<code>{R} ← X □NTIE Y</code>	Ties the native file that has name <code>X</code> using file tie number <code>Y[1]</code> ; optionally, <code>Y[2]</code> can specify the type of access needed by other users (see <i>Access Codes</i> ).
<code>{R} ← □NUNTIE Y</code>	Unties all native files that have a tie number in vector <code>Y</code> ( <code>□NUNTIE 0</code> does not untie any files but flushes all file caches to disk) and returns the tie numbers of native files that have been untied.
<code>{R} ← {X} □NXLATE Y</code>	Associates the native file that has tie number <code>Y</code> with character translation vector <code>X</code> . Note that: <ul style="list-style-type: none"> <li>if <code>X</code> is not specified then the currently-associated translation vector is returned.</li> <li>if <code>X</code> has the value <code>(⊔256) - □IO</code> then the translation process is bypassed and raw input/output is provided.</li> <li>if <code>Y</code> is set to 0, then the translate vector used by <code>□DR</code> is used.</li> </ul> Unicode version only: This is only relevant when processing native files that contain characters expressed as indices into <code>□AV</code> .

## 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		Granted to subsequent users			
0	read access	0	compatibility mode	48	write access
1	write access	16	no access (exclusive)	64	read and write access
2	read and write access	32	read 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
1287	16 x count	128 bit decimal	count
1289	16 x count	128 bit complex	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
1287	16 x count	128 bit decimal	count
1289	16 x count	128 bit complex	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 `OPEN` and `WRITE`. 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 `OPEN`.

Encoding	Description
UTF-8	Data is encoded into UTF-8 format. If <code>-BOM</code> or <code>-NOBOM</code> is not appended, the default is <code>-NOBOM</code> .
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 <code>-BOM</code> or <code>-NOBOM</code> is not appended, the default is <code>-BOM</code> .
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 <code>-BOM</code> or <code>-NOBOM</code> is not appended, the default is <code>-BOM</code> .
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 `OPEN` and `WRITE`.

For `OPEN`:

- 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 occurrence of any of the line separators.

Value	Char	Description	Notes
13	CR	Carriage Return (U+000D)	Newline separators recognised by <code>OPEN (R[3])</code> and <code>WRITE (X[3])</code> .
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, `WRITE`'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, `LOCK` 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 – `LOCK` does not standardise this.