Predefined native! values

all

```
USAGE:
     ALL conds

DESCRIPTION:
     Evaluates and returns the last value if all are truthy; else NONE.
     ALL is a native! value.

ARGUMENTS:
     conds        [block!]

>> help all
USAGE:
     ALL conds

DESCRIPTION:
     Evaluates and returns the last value if all are truthy; else NONE.
     ALL is a native! value.

ARGUMENTS:
     conds        [block!]
```

any

```
USAGE:
     ANY conds

DESCRIPTION:
     Evaluates and returns the first truthy value, if any; else NONE.
     ANY is a native! value.

ARGUMENTS:
     conds        [block!]
```

arccosine

```
USAGE:
     ARCCOSINE cosine

DESCRIPTION:
     Returns the trigonometric arccosine (in degrees by default in range [0,180]).
     ARCCOSINE is a native! value.
```

```
ARGUMENTS:
     cosine       [number!] "in range [-1,1]."

REFINEMENTS:
     /radians     => Angle is returned in radians [0,pi].

RETURNS:
     [float!]
```

arcsine

```
USAGE:
     ARCSINE sine

DESCRIPTION:
     Returns the trigonometric arcsine (in degrees by default in range [-90,90]).
     ARCSINE is a native! value.

ARGUMENTS:
     sine         [number!] "in range [-1,1]."

REFINEMENTS:
     /radians     => Angle is returned in radians [-pi/2,pi/2].

RETURNS:
     [float!]
```

arctangent

```
USAGE:
     ARCTANGENT tangent

DESCRIPTION:
     Returns the trigonometric arctangent (in degrees by default in range [-90,90]).
     ARCTANGENT is a native! value.

ARGUMENTS:
     tangent      [number!] "in range [-inf,+inf]."

REFINEMENTS:
     /radians     => Angle is returned in radians [-pi/2,pi/2].

RETURNS:
     [float!]
```

arctangent2

```
USAGE:
     ARCTANGENT2 y x

DESCRIPTION:
     Returns the smallest angle between the vectors (1,0) and (x,y) in degrees by
default (-180,180].
     ARCTANGENT2 is a native! value.

ARGUMENTS:
     y            [number!]
     x            [number!]

REFINEMENTS:
     /radians     => Angle is returned in radians (-pi,pi].

RETURNS:
     [float!]
```

as

```
USAGE:
     AS type spec

DESCRIPTION:
     Coerce a series into a compatible datatype without copying it.
     AS is a native! value.

ARGUMENTS:
     type         [datatype! block! paren! any-path! any-string!] "The datatype or
example value."
     spec         [block! paren! any-path! any-string!] "The series to coerce."
```

as-money

```
USAGE:
     AS-MONEY currency amount

DESCRIPTION:
     Combine currency code and amount into a monetary value.
     AS-MONEY is a native! value.

ARGUMENTS:
     currency     [word!]
     amount       [integer! float!]
```

```
RETURNS:
     [money!]
```

as-pair

```
USAGE:
     AS-PAIR x y

DESCRIPTION:
     Combine X and Y values into a pair.
     AS-PAIR is a native! value.

ARGUMENTS:
     x            [integer! float!]
     y            [integer! float!]
```

bind

```
USAGE:
     BIND word context

DESCRIPTION:
     Bind words to a context; returns rebound words.
     BIND is a native! value.

ARGUMENTS:
     word         [block! any-word!]
     context      [any-word! any-object! function!]

REFINEMENTS:
     /copy        => Deep copy blocks before binding.

RETURNS:
     [block! any-word!]
```

break

```
USAGE:
     BREAK

DESCRIPTION:
     Breaks out of a loop, while, until, repeat, foreach, etc.
     BREAK is a native! value.
```

```
REFINEMENTS:
     /return      => Forces the loop function to return a value.
        value        [any-type!]
```

browse

```
USAGE:
     BROWSE url

DESCRIPTION:
     Open web browser to a URL or file mananger to a local file.
     BROWSE is a native! value.

ARGUMENTS:
     url          [url! file!]
```

call

```
USAGE:
     CALL cmd

DESCRIPTION:
     Executes a shell command to run another process.
     CALL is a native! value.

ARGUMENTS:
     cmd          [string! file!] "A shell command or an executable file."

REFINEMENTS:
     /wait        => Runs command and waits for exit.
     /show        => Force the display of system's shell window (Windows only).
     /console     => Runs command with I/O redirected to console (CLI console only at
present).
     /shell       => Forces command to be run from shell.
     /input       =>
        in           [string! file! binary!] "Redirects in to stdin."
     /output      =>
        out          [string! file! binary!] "Redirects stdout to out."
     /error       =>
        err          [string! file! binary!] "Redirects stderr to err."

RETURNS:
     0 if success, -1 if error, or a process ID.
     [integer!]
```

case

```
USAGE:
     CASE cases

DESCRIPTION:
     Evaluates the block following the first truthy condition.
     CASE is a native! value.

ARGUMENTS:
     cases        [block!] "Block of condition-block pairs."

REFINEMENTS:
     /all         => Test all conditions, evaluating the block following each truthy
condition.
```

catch

```
USAGE:
     CATCH block

DESCRIPTION:
     Catches a throw from a block and returns its value.
     CATCH is a native! value.

ARGUMENTS:
     block        [block!] "Block to evaluate."

REFINEMENTS:
     /name        => Catches a named throw.
        word         [word! block!] "One or more names."
```

checksum

```
USAGE:
     CHECKSUM data method

DESCRIPTION:
     Computes a checksum, CRC, hash, or HMAC.
     CHECKSUM is a native! value.

ARGUMENTS:
     data         [binary! string! file!]
     method       [word!] {MD5 SHA1 SHA256 SHA384 SHA512 CRC32 TCP ADLER32 hash.}

REFINEMENTS:
```

```
     /with        => Extra value for HMAC key or hash table size; not compatible with
TCP/CRC32/ADLER32 methods.
        spec         [any-string! binary! integer!] {String or binary for MD5/SHA*
HMAC key, integer for hash table size.}

RETURNS:
     [integer! binary!]
```

compliment?

```
USAGE:
     COMPLEMENT? bits

DESCRIPTION:
     Returns TRUE if the bitset is complemented.
     COMPLEMENT? is a native! value.

ARGUMENTS:
     bits         [bitset!]
```

compose

```
USAGE:
     COMPOSE value

DESCRIPTION:
     Returns a copy of a block, evaluating only parens.
     COMPOSE is a native! value.

ARGUMENTS:
     value        [block!]

REFINEMENTS:
     /deep        => Compose nested blocks.
     /only        => Compose nested blocks as blocks containing their values.
     /into        => Put results in out block, instead of creating a new block.
        out          [any-block!] "Target block for results, when /into is used."
```

compress

```
USAGE:
     COMPRESS data

DESCRIPTION:
     compresses data. return GZIP format (RFC 1952) by default.
```

```
     COMPRESS is a native! value.

ARGUMENTS:
     data         [any-string! binary!]

REFINEMENTS:
     /zlib        => Return ZLIB format (RFC 1950).
     /deflate     => Return DEFLATE format (RFC 1951).
```

construct

```
USAGE:
     CONSTRUCT block

DESCRIPTION:
     Makes a new object from an unevaluated spec; standard logic words are evaluated.
     CONSTRUCT is a native! value.

ARGUMENTS:
     block        [block!]

REFINEMENTS:
     /with        => Use a prototype object.
        object       [object!] "Prototype object."
     /only        => Don't evaluate standard logic words.
```

context?

```
USAGE:
     CONTEXT? word

DESCRIPTION:
     Returns the context to which a word is bound.
     CONTEXT? is a native! value.

ARGUMENTS:
     word         [any-word!] "Word to check."

RETURNS:
     [object! function! none!]
```

continue

```
USAGE:
     CONTINUE
```

```
DESCRIPTION:
     Throws control back to top of loop.
     CONTINUE is a native! value.
```

cosine

```
USAGE:
     COSINE angle

DESCRIPTION:
     Returns the trigonometric cosine.
     COSINE is a native! value.

ARGUMENTS:
     angle        [number!]

REFINEMENTS:
     /radians     => Angle is specified in radians.

RETURNS:
     [float!]
```

debase

```
USAGE:
     DEBASE value

DESCRIPTION:
     Decodes binary-coded string (BASE-64 default) to binary value.
     DEBASE is a native! value.

ARGUMENTS:
     value        [string!] "The string to decode."

REFINEMENTS:
     /base        => Binary base to use.
        base-value   [integer!] "The base to convert from: 64, 58, 16, or 2."
```

decompress

```
USAGE:
     DECOMPRESS data

DESCRIPTION:
```

```
     Decompresses data. Data in GZIP format (RFC 1952) by default.
     DECOMPRESS is a native! value.

ARGUMENTS:
     data         [binary!]

REFINEMENTS:
     /zlib        => Data in ZLIB format (RFC 1950).
        size         [integer!] "Uncompressed data size. Use 0 if don't know."
     /deflate     => Data in DEFLATE format (RFC 1951).
        size         [integer!] "Uncompressed data size. Use 0 if don't know."
```

dehex

```
USAGE:
     DEHEX value

DESCRIPTION:
     Converts URL-style hex encoded (%xx) strings.
     DEHEX is a native! value.

ARGUMENTS:
     value        [any-string!]

RETURNS:
     Always return a string.
     [string!]
```

difference

```
USAGE:
     DIFFERENCE set1 set2

DESCRIPTION:
     Returns the special difference of two data sets.
     DIFFERENCE is a native! value.

ARGUMENTS:
     set1         [block! hash! string! bitset! typeset! date!]
     set2         [block! hash! string! bitset! typeset! date!]

REFINEMENTS:
     /case        => Use case-sensitive comparison.
     /skip        => Treat the series as fixed size records.
        size         [integer!]

RETURNS:
```

```
     [block! hash! string! bitset! typeset! time!]
```

do

```
USAGE:
     DO value

DESCRIPTION:
     Evaluates a value, returning the last evaluation result.
     DO is a native! value.

ARGUMENTS:
     value        [any-type!]

REFINEMENTS:
     /expand      => Expand directives before evaluation.
     /args        => If value is a script, this will set its system/script/args.
        arg           "Args passed to a script (normally a string)."
     /next        => Do next expression only, return it, update block word.
        position     [word!] "Word updated with new block position."
```

does

```
USAGE:
     DOES body

DESCRIPTION:
     Defines a function with no arguments or local variables.
     DOES is a native! value.

ARGUMENTS:
     body         [block!]
```

either

```
USAGE:
     EITHER cond true-blk false-blk

DESCRIPTION:
     If conditional expression is truthy, evaluate the first branch; else evaluate the
alternative.
     EITHER is a native! value.

ARGUMENTS:
     cond         [any-type!]
```

```
     true-blk     [block!]
     false-blk    [block!]
```

enbase

```
USAGE:
     ENBASE value

DESCRIPTION:
     Encodes a string into a binary-coded string (BASE-64 default).
     ENBASE is a native! value.

ARGUMENTS:
     value        [binary! string!] "If string, will be UTF8 encoded."

REFINEMENTS:
     /base        => Binary base to use.
        base-value   [integer!] "The base to convert from: 64, 58, 16, or 2."
```

enhex

```
USAGE:
     ENHEX value

DESCRIPTION:
     Encode URL-style hex encoded (%xx) strings.
     ENHEX is a native! value.

ARGUMENTS:
     value        [any-string!]

RETURNS:
     Always return a string.
     [string!]
```

equal?

```
USAGE:
     EQUAL? value1 value2

DESCRIPTION:
     Returns TRUE if two values are equal.
     EQUAL? is a native! value.

ARGUMENTS:
```

```
     value1       [any-type!]
     value2       [any-type!]
```

exclude

```
USAGE:
     EXCLUDE set1 set2

DESCRIPTION:
     Returns the first data set less the second data set.
     EXCLUDE is a native! value.

ARGUMENTS:
     set1         [block! hash! string! bitset! typeset!]
     set2         [block! hash! string! bitset! typeset!]

REFINEMENTS:
     /case        => Use case-sensitive comparison.
     /skip        => Treat the series as fixed size records.
        size         [integer!]

RETURNS:
     [block! hash! string! bitset! typeset!]
```

exit

```
USAGE:
     EXIT

DESCRIPTION:
     Exits a function, returning no value.
     EXIT is a native! value.
```

exp

```
USAGE:
     EXP value

DESCRIPTION:
     Raises E (the base of natural logarithm) to the power specified.
     EXP is a native! value.

ARGUMENTS:
     value        [number!]
```

```
RETURNS:
     [float!]
```

extend

```
USAGE:
     EXTEND obj spec

DESCRIPTION:
     Extend an object or map value with list of key and value pairs.
     EXTEND is a native! value.

ARGUMENTS:
     obj          [object! map!]
     spec         [block! hash! map!]

REFINEMENTS:
     /case        => Use case-sensitive comparison.
```

forall

```
USAGE:
     FORALL 'word body

DESCRIPTION:
     Evaluates body for all values in a series.
     FORALL is a native! value.

ARGUMENTS:
     'word        [word!] "Word referring to series to iterate over."
     body         [block!]
```

foreach

```
USAGE:
     FOREACH 'word series body

DESCRIPTION:
     Evaluates body for each value in a series.
     FOREACH is a native! value.

ARGUMENTS:
     'word        [word! block!] "Word, or words, to set on each iteration."
     series       [series! map!]
```

```
     body         [block!]
```

forever

```
USAGE:
     FOREVER body

DESCRIPTION:
     Evaluates body repeatedly forever.
     FOREVER is a native! value.

ARGUMENTS:
     body         [block!]
```

func

```
USAGE:
     FUNC spec body

DESCRIPTION:
     Defines a function with a given spec and body.
     FUNC is a native! value.

ARGUMENTS:
     spec         [block!]
     body         [block!]
```

function

```
USAGE:
     FUNCTION spec body

DESCRIPTION:
     Defines a function, making all set-words found in body, local.
     FUNCTION is a native! value.

ARGUMENTS:
     spec         [block!]
     body         [block!]

REFINEMENTS:
     /extern      => Exclude words that follow this refinement.
```

get

```
USAGE:
     GET word

DESCRIPTION:
     Returns the value a word refers to.
     GET is a native! value.

ARGUMENTS:
     word         [any-word! any-path! object!]

REFINEMENTS:
     /any         => If word has no value, return UNSET rather than causing an error.
     /case        => Use case-sensitive comparison (path only).

RETURNS:
     [any-type!]
```

get-env

```
USAGE:
     GET-ENV var

DESCRIPTION:
     Returns the value of an OS environment variable (for current process).
     GET-ENV is a native! value.

ARGUMENTS:
     var          [any-string! any-word!] "Variable to get."

RETURNS:
     [string! none!]
```

greater-or-equal?

```
USAGE:
     GREATER-OR-EQUAL? value1 value2

DESCRIPTION:
     Returns TRUE if the first value is greater than or equal to the second.
     GREATER-OR-EQUAL? is a native! value.

ARGUMENTS:
     value1       [any-type!]
```

```
     value2       [any-type!]
```

greater

```
USAGE:
     GREATER? value1 value2

DESCRIPTION:
     Returns TRUE if the first value is greater than the second.
     GREATER? is a native! value.

ARGUMENTS:
     value1       [any-type!]
     value2       [any-type!]
```

has

```
USAGE:
     HAS vars body

DESCRIPTION:
     Defines a function with local variables, but no arguments.
     HAS is a native! value.

ARGUMENTS:
     vars         [block!]
     body         [block!]
```

if

```
USAGE:
     IF cond then-blk

DESCRIPTION:
     If conditional expression is truthy, evaluate block; else return NONE.
     IF is a native! value.

ARGUMENTS:
     cond         [any-type!]
     then-blk     [block!]
```

in

```
USAGE:
     IN object word

DESCRIPTION:
     Returns the given word bound to the object's context.
     IN is a native! value.

ARGUMENTS:
     object       [any-object!]
     word         [any-word!]
```

intersect

```
USAGE:
     INTERSECT set1 set2

DESCRIPTION:
     Returns the intersection of two data sets.
     INTERSECT is a native! value.

ARGUMENTS:
     set1         [block! hash! string! bitset! typeset!]
     set2         [block! hash! string! bitset! typeset!]

REFINEMENTS:
     /case        => Use case-sensitive comparison.
     /skip        => Treat the series as fixed size records.
        size         [integer!]

RETURNS:
     [block! hash! string! bitset! typeset!]
```

lesser-or-equal?

```
USAGE:
     LESSER-OR-EQUAL? value1 value2

DESCRIPTION:
     Returns TRUE if the first value is less than or equal to the second.
     LESSER-OR-EQUAL? is a native! value.

ARGUMENTS:
     value1       [any-type!]
```

```
     value2       [any-type!]
```

lesser?

```
USAGE:
     LESSER? value1 value2

DESCRIPTION:
     Returns TRUE if the first value is less than the second.
     LESSER? is a native! value.

ARGUMENTS:
     value1       [any-type!]
     value2       [any-type!]
```

list-env

```
USAGE:
     LIST-ENV

DESCRIPTION:
     Returns a map of OS environment variables (for current process).
     LIST-ENV is a native! value.

RETURNS:
     [map!]
```

log-10

```
USAGE:
     LOG-10 value

DESCRIPTION:
     Returns the base-10 logarithm.
     LOG-10 is a native! value.

ARGUMENTS:
     value        [number!]

RETURNS:
     [float!]
```

log-2

```
USAGE:
     LOG-2 value

DESCRIPTION:
     Return the base-2 logarithm.
     LOG-2 is a native! value.

ARGUMENTS:
     value        [number!]

RETURNS:
     [float!]
```

log-e

```
USAGE:
     LOG-E value

DESCRIPTION:
     Returns the natural (base-E) logarithm of the given value.
     LOG-E is a native! value.

ARGUMENTS:
     value        [number!]

RETURNS:
     [float!]
```

loop

```
USAGE:
     LOOP count body

DESCRIPTION:
     Evaluates body a number of times.
     LOOP is a native! value.

ARGUMENTS:
     count        [integer! float!]
     body         [block!]
```

lowercase

```
USAGE:
     LOWERCASE string

DESCRIPTION:
     Converts string of characters to lowercase.
     LOWERCASE is a native! value.

ARGUMENTS:
     string       [any-string! char!] "Value to convert (modified when series)."

REFINEMENTS:
     /part        => Limits to a given length or position.
        limit        [number! any-string!]

RETURNS:
     [any-string! char!]
```

max

```
USAGE:
     MAX value1 value2

DESCRIPTION:
     Returns the greater of the two values.
     MAX is a native! value.

ARGUMENTS:
     value1       [scalar! series!]
     value2       [scalar! series!]
```

min

```
USAGE:
     MIN value1 value2

DESCRIPTION:
     Returns the lesser of the two values.
     MIN is a native! value.

ARGUMENTS:
     value1       [scalar! series!]
     value2       [scalar! series!]
```

NaN?

```
USAGE:
     NAN? value

DESCRIPTION:
     Returns TRUE if the number is Not-a-Number.
     NAN? is a native! value.

ARGUMENTS:
     value        [number!]

RETURNS:
     [logic!]
```

negative?

```
USAGE:
     NEGATIVE? number

DESCRIPTION:
     Returns TRUE if the number is negative.
     NEGATIVE? is a native! value.

ARGUMENTS:
     number       [number! money! time!]

RETURNS:
     [logic!]
```

new-line

```
USAGE:
     NEW-LINE position value

DESCRIPTION:
     Sets or clears the new-line marker within a list series.
     NEW-LINE is a native! value.

ARGUMENTS:
     position     [any-list!] "Position to change marker (modified)."
     value        [logic!] "Set TRUE for newline."

REFINEMENTS:
     /all         => Set/clear marker to end of series.
     /skip        => Set/clear marker periodically to the end of the series.
```

```
        size         [integer!]

RETURNS:
     [any-list!]
```

new-line?

```
USAGE:
     NEW-LINE? position

DESCRIPTION:
     Returns the state of the new-line marker within a list series.
     NEW-LINE? is a native! value.

ARGUMENTS:
     position     [any-list!] "Position to change marker."

RETURNS:
     [any-list!]
```

not

```
USAGE:
     NOT value

DESCRIPTION:
     Returns the logical complement of a value (truthy or falsy).
     NOT is a native! value.

ARGUMENTS:
     value        [any-type!]
```

not-equal?

```
USAGE:
     NOT-EQUAL? value1 value2

DESCRIPTION:
     Returns TRUE if two values are not equal.
     NOT-EQUAL? is a native! value.

ARGUMENTS:
     value1       [any-type!]
     value2       [any-type!]
```

now

```
USAGE:
     NOW

DESCRIPTION:
     Returns date and time.
     NOW is a native! value.

REFINEMENTS:
     /year        => Returns year only.
     /month       => Returns month only.
     /day         => Returns day of the month only.
     /time        => Returns time only.
     /zone        => Returns time zone offset from UTC (GMT) only.
     /date        => Returns date only.
     /weekday     => Returns day of the week as integer (Monday is day 1).
     /yearday     => Returns day of the year (Julian).
     /precise     => High precision time.
     /utc         => Universal time (no zone).

RETURNS:
     [date! time! integer!]
```

parse

```
USAGE:
     PARSE input rules

DESCRIPTION:
     Process a series using dialected grammar rules.
     PARSE is a native! value.

ARGUMENTS:
     input        [binary! any-block! any-string!]
     rules        [block!]

REFINEMENTS:
     /case        => Uses case-sensitive comparison.
     /part        => Limit to a length or position.
        length       [number! series!]
     /trace       =>
        callback     [function! [event [word!] match? [logic!] rule [block!] input
[series!] stack [block!] return: [logic!]]]

RETURNS:
     [logic! block!]
```

positive?

```
USAGE:
     POSITIVE? number

DESCRIPTION:
     Returns TRUE if the number is positive.
     POSITIVE? is a native! value.

ARGUMENTS:
     number       [number! money! time!]

RETURNS:
     [logic!]
```

prin

```
USAGE:
     PRIN value

DESCRIPTION:
     Outputs a value.
     PRIN is a native! value.

ARGUMENTS:
     value        [any-type!]
```

print

```
USAGE:
     PRINT value

DESCRIPTION:
     Outputs a value followed by a newline.
     PRINT is a native! value.

ARGUMENTS:
     value        [any-type!]
```

recycle

```
USAGE:
     RECYCLE
```

```
DESCRIPTION:
     Recycles unused memory.
     RECYCLE is a native! value.

REFINEMENTS:
     /on          => Turns on garbage collector.
     /off         => Turns off garbage collector.
```

reduce

```
USAGE:
     REDUCE value

DESCRIPTION:
     Returns a copy of a block, evaluating all expressions.
     REDUCE is a native! value.

ARGUMENTS:
     value        [any-type!]

REFINEMENTS:
     /into        => Put results in out block, instead of creating a new block.
        out          [any-block!] "Target block for results, when /into is used."
```

remove-each

```
USAGE:
     REMOVE-EACH 'word data body

DESCRIPTION:
     Removes values for each block that returns truthy value.
     REMOVE-EACH is a native! value.

ARGUMENTS:
     'word        [word! block!] "Word or block of words to set each time."
     data         [series!] "The series to traverse (modified)."
     body         [block!] {Block to evaluate (return truthy value to remove).}
```

repeat

```
USAGE:
     REPEAT 'word value body

DESCRIPTION:
     Evaluates body a number of times, tracking iteration count.
```

```
     REPEAT is a native! value.

ARGUMENTS:
     'word        [word!] "Iteration counter; not local to loop."
     value        [integer! float!] "Number of times to evaluate body."
     body         [block!]
```

return

```
USAGE:
     RETURN value

DESCRIPTION:
     Returns a value from a function.
     RETURN is a native! value.

ARGUMENTS:
     value        [any-type!]
```

same?

```
USAGE:
     SAME? value1 value2

DESCRIPTION:
     Returns TRUE if two values have the same identity.
     SAME? is a native! value.

ARGUMENTS:
     value1       [any-type!]
     value2       [any-type!]
```

set

```
USAGE:
     SET word value

DESCRIPTION:
     Sets the value(s) one or more words refer to.
     SET is a native! value.

ARGUMENTS:
     word         [any-word! block! object! any-path!] "Word, object, map path or
block of words to set."
     value        [any-type!] "Value or block of values to assign to words."
```

```
REFINEMENTS:
     /any          => Allow UNSET as a value rather than causing an error.
     /case         => Use case-sensitive comparison (path only).
     /only         => Block or object value argument is set as a single value.
     /some         => None values in a block or object value argument, are not set.

RETURNS:
     [any-type!]
```

set-env

```
USAGE:
     SET-ENV var value

DESCRIPTION:
     Sets the value of an operating system environment variable (for current process).
     SET-ENV is a native! value.

ARGUMENTS:
     var           [any-string! any-word!] "Variable to set."
     value         [string! none!] "Value to set, or NONE to unset it."
```

shift

```
USAGE:
     SHIFT data bits

DESCRIPTION:
     Perform a bit shift operation. Right shift (decreasing) by default.
     SHIFT is a native! value.

ARGUMENTS:
     data          [integer!]
     bits          [integer!]

REFINEMENTS:
     /left         => Shift bits to the left (increasing).
     /logical      => Use logical shift (unsigned, fill with zero).

RETURNS:
     [integer!]
```

sign?

```
USAGE:
     SIGN? number

DESCRIPTION:
     Returns sign of N as 1, 0, or -1 (to use as a multiplier).
     SIGN? is a native! value.

ARGUMENTS:
     number       [number! money! time!]

RETURNS:
     [integer!]
```

sine

```
USAGE:
     SINE angle

DESCRIPTION:
     Returns the trigonometric sine.
     SINE is a native! value.

ARGUMENTS:
     angle        [number!]

REFINEMENTS:
     /radians     => Angle is specified in radians.

RETURNS:
     [float!]
```

size?

```
USAGE:
     SIZE? file

DESCRIPTION:
     Returns the size of a file content.
     SIZE? is a native! value.

ARGUMENTS:
     file         [file!]

RETURNS:
```

```
     [integer! none!]
```

square-root

```
USAGE:
     SQUARE-ROOT value

DESCRIPTION:
     Returns the square root of a number.
     SQUARE-ROOT is a native! value.

ARGUMENTS:
     value        [number!]

RETURNS:
     [float!]
```

stats

```
USAGE:
     STATS

DESCRIPTION:
     Returns interpreter statistics.
     STATS is a native! value.

REFINEMENTS:
     /show        => TBD:.
     /info        => Output formatted results.

RETURNS:
     [integer! block!]
```

strict-equal?

```
USAGE:
     STRICT-EQUAL? value1 value2

DESCRIPTION:
     Returns TRUE if two values are equal, and also the same datatype.
     STRICT-EQUAL? is a native! value.

ARGUMENTS:
     value1       [any-type!]
```

```
     value2       [any-type!]
```

switch

```
USAGE:
     SWITCH value cases

DESCRIPTION:
     Evaluates the first block following the value found in cases.
     SWITCH is a native! value.

ARGUMENTS:
     value        [any-type!] "The value to match."
     cases        [block!]

REFINEMENTS:
     /default     => Specify a default block, if value is not found in cases.
        case         [block!] "Default block to evaluate."
```

tangent

```
USAGE:
     TANGENT angle

DESCRIPTION:
     Returns the trigonometric tangent.
     TANGENT is a native! value.

ARGUMENTS:
     angle        [number!]

REFINEMENTS:
     /radians     => Angle is specified in radians.

RETURNS:
     [float!]
```

throw

```
USAGE:
     THROW value

DESCRIPTION:
     Throws control back to a previous catch.
     THROW is a native! value.
```

```
ARGUMENTS:
     value        [any-type!] "Value returned from catch."

REFINEMENTS:
     /name        => Throws to a named catch.
        word         [word!]
```

to-hex

```
USAGE:
     TO-HEX value

DESCRIPTION:
     Converts numeric value to a hex issue! datatype (with leading # and 0's).
     TO-HEX is a native! value.

ARGUMENTS:
     value        [integer!]

REFINEMENTS:
     /size        => Specify number of hex digits in result.
        length       [integer!]

RETURNS:
     [issue!]
```

to-local-file

```
USAGE:
     TO-LOCAL-FILE path

DESCRIPTION:
     Converts a Red file path to the local system file path.
     TO-LOCAL-FILE is a native! value.

ARGUMENTS:
     path         [file! string!]

REFINEMENTS:
     /full        => Prepends current dir for full path (for relative paths only).

RETURNS:
     [string!]
```

transcode

```
USAGE:
     TRANSCODE src

DESCRIPTION:
     Translates UTF-8 binary source to values. Returns one or several values in a
block.
     TRANSCODE is a native! value.

ARGUMENTS:
     src          [binary! string!] {UTF-8 input buffer; string argument will be UTF-8
encoded.}

REFINEMENTS:
     /next        => Translate next complete value (blocks as single value).
     /one         => Translate next complete value, returns the value only.
     /prescan     => Prescans only, do not load values. Returns guessed type.
     /scan        => Scans only, do not load values. Returns recognized type.
     /part        => Translates only part of the input buffer.
        length       [integer! binary!] "Length in bytes or tail position."
     /into        => Optionally provides an output block.
        dst          [block!]
     /trace       =>
        callback     [function! [event [word!] input [binary! string!] type [word!
datatype!] line [integer!] token return: [logic!]]]

RETURNS:
     [block!]
```

try

```
USAGE:
     TRY block

DESCRIPTION:
     Tries to DO a block and returns its value or an error.
     TRY is a native! value.

ARGUMENTS:
     block        [block!]

REFINEMENTS:
     /all         => Catch also BREAK, CONTINUE, RETURN, EXIT and THROW exceptions.
```

type?

```
USAGE:
     TYPE? value

DESCRIPTION:
     Returns the datatype of a value.
     TYPE? is a native! value.

ARGUMENTS:
     value        [any-type!]

REFINEMENTS:
     /word        => Return a word value, rather than a datatype value.
```

union

```
USAGE:
     UNION set1 set2

DESCRIPTION:
     Returns the union of two data sets.
     UNION is a native! value.

ARGUMENTS:
     set1         [block! hash! string! bitset! typeset!]
     set2         [block! hash! string! bitset! typeset!]

REFINEMENTS:
     /case        => Use case-sensitive comparison.
     /skip        => Treat the series as fixed size records.
        size         [integer!]

RETURNS:
     [block! hash! string! bitset! typeset!]
```

unique

```
USAGE:
     UNIQUE set

DESCRIPTION:
     Returns the data set with duplicates removed.
     UNIQUE is a native! value.

ARGUMENTS:
```

```
     set          [block! hash! string!]

REFINEMENTS:
     /case        => Use case-sensitive comparison.
     /skip        => Treat the series as fixed size records.
        size         [integer!]

RETURNS:
     [block! hash! string!]
```

unless

```
USAGE:
     UNLESS cond then-blk

DESCRIPTION:
     If conditional expression is falsy, evaluate block; else return NONE.
     UNLESS is a native! value.

ARGUMENTS:
     cond         [any-type!]
     then-blk     [block!]
```

unset

```
USAGE:
     UNSET word

DESCRIPTION:
     Unsets the value of a word in its current context.
     UNSET is a native! value.

ARGUMENTS:
     word         [word! block!] "Word or block of words."
```

until

```
USAGE:
     UNTIL body

DESCRIPTION:
     Evaluates body until it is truthy.
     UNTIL is a native! value.

ARGUMENTS:
```

```
     body         [block!]
```

uppercase

```
USAGE:
     UPPERCASE string

DESCRIPTION:
     Converts string of characters to uppercase.
     UPPERCASE is a native! value.

ARGUMENTS:
     string       [any-string! char!] "Value to convert (modified when series)."

REFINEMENTS:
     /part        => Limits to a given length or position.
        limit        [number! any-string!]

RETURNS:
     [any-string! char!]
```

value?

```
USAGE:
     VALUE? value

DESCRIPTION:
     Returns TRUE if the word has a value.
     VALUE? is a native! value.

ARGUMENTS:
     value

RETURNS:
     [logic!]
```

wait

```
USAGE:
     WAIT value

DESCRIPTION:
     Waits for a duration in seconds or specified time.
     WAIT is a native! value.
```

```
ARGUMENTS:
     value        [number! time! block! none!]

REFINEMENTS:
     /all         => Returns all events in a block.
```

while

```
USAGE:
     WHILE cond body

DESCRIPTION:
     Evaluates body as long as condition block evaluates to truthy value.
     WHILE is a native! value.

ARGUMENTS:
     cond         [block!] "Condition block to evaluate on each iteration."
     body         [block!] "Block to evaluate on each iteration."
```

while

```
USAGE:
     WHILE cond body

DESCRIPTION:
     Evaluates body as long as condition block evaluates to truthy value.
     WHILE is a native! value.

ARGUMENTS:
     cond         [block!] "Condition block to evaluate on each iteration."
     body         [block!] "Block to evaluate on each iteration."
```

zero?

```
USAGE:
     ZERO? value

DESCRIPTION:
     Returns TRUE if the value is zero.
     ZERO? is a native! value.

ARGUMENTS:
     value        [number! money! pair! time! char! tuple!]

RETURNS:
```

```
    [logic!]
```