SHELL

December 11, 2025

Canine-Table

POSIX Nexus serves as a comprehensive cross-language reference hub that explores the implementation and behavior of POSIX-compliant functionality across a diverse set of programming environments. Built atop the foundational IEEE Portable Operating System Interface (POSIX) standards, this project emphasizes compatibility, portability, and interoperability between operating systems.

Abstract

Contents

I IP Module

I Network Interface Grouper

nx_ip_g_net()

```
nx_ip_g_net()
{
    ls --color=never -l '/sys/class/net/' | ${AWK:-$(nx_cmd_awk)} \
        -v ex="$1" \
        -F '/' \
    "
        $(nx_data_include -i "${NEXUS_LIB}/awk/nex-misc-extras.awk")
    "'
        BEGIN {
            if (ex == "-e")
                ex = "export "
            else
                ex = ""
            virt = ""
            phy = ""
        }
        /devices\/pci/{
            phy = phy " " $NF
        }
        /devices\/virtual/{
            virt = virt " " $NF
        } END {
            printf("%s%s\n%s%s\n",
                ex, __nx_stringify_var("G_NEX_NET_VIRT", substr(virt, 2)),
                ex, __nx_stringify_var("G_NEX_NET_PHY", substr(phy, 2)))
        }
    '
}
```

Network Interface Grouper — The Net Glyph

- **Purpose** ⇝ Groups system network interfaces into virtual and physical sets
- **Input** ⇝ Reads from `/sys/class/net`; optional flag `-e` prepends `export`
- **Mechanism** ⇝ AWK scans symlink targets: `/devices/pci` → physical, `/devices/virtual` → virtual
- **Output** ⇝ Two environment assignments: G_NEX_NET_VIRT and G_NEX_NET_PHY
- **Cleanup** ⇝ Whitespace trimmed via `substr`
- **Use Case** ⇝ Used to audit and export interface groups for higher-level networking overlays

Export grouped interfaces

- **Input** ⇝ Run `nx_ip_g_net -e`
- **Expected** ⇝ Outputs `export G_NEX_NET_VIRT="lo vnet0"` and `export G_NEX_NET_PHY="eth0 wlan0"`

```
nx_ip_g_net -e
```

I NetNS Args Resolver

__nx_ip_a_netns()

```
__nx_ip_a_netns()
{
    NEX_S="$(__nx_ip_i_netns "$NEX_k_n" && printf '%s' "$1" || printf '%s' "n:$1")"
    shift
    nx_data_optargs "$NEX_S" "$@"
    ! __nx_ip_i_netns "$NEX_k_n" && test -n "$NEX_k_n" && __nx_ip_n_netns "$NEX_k_n"
}
```

NetNS Args Resolver — The Arg Glyph

- **Purpose** ⇝ Resolves leftover input string for network namespace operations
- **Input** ⇝ First token is mode flag (e.g. `e`, `n`); subsequent tokens passed to `nx_data_optargs`
- **Mechanism** ⇝ `nx_data_optargs` gobbles matched options like TeX macros, not left-to-right; NEX_S holds the stringified remainder
- **Equivalence** ⇝ NEX_S ≈ shell `$*` (all args as a single string); NEX_R ≈ shell `$@` (args preserved as list)
- **Creation** ⇝ If namespace doesn't exist but key is set, calls `__nx_ip_n_netns` to create
- **Return** ⇝ Populates NEX_S with stringified input not consumed by option parsing
- **Use Case** ⇝ Provides normalized argument string for downstream exec/remove/-move glyphs

Stringify leftover args

- **Input** ⇝ Namespace key `NEX_k_n="demo"`; call `__nx_ip_a_netns e -f -g`
- **Optarg** ⇝ `nx_data_optargs` consumes `-f`
- **Expected Result** ⇝ `NEX_S="e -g"` (stringified remainder)

```
NEX_k_n="demo"
__nx_ip_a_netns e -f -g
```

II Data Module

II Command-Line Parser

nx_data_optargs()

```
nx_data_optargs()
{
    eval "$(
```

```
        ${AWK:-$(nx_cmd_awk)} \
            -v inpt="$(nx_str_chain "$@")" \
            "
                $(nx_data_include -i "${NEXUS_LIB}/awk/nex-sh-extras.awk")
            "'
                BEGIN {
                    print nx_sh_optargs(inpt)
                }
            '
    )"
}
```

Command-Line Parser — The Optargs Glyph

- **Purpose** ⇝ Parses shell arguments into structured variables using AWK macros
- **Input** ⇝ Raw argument list $@, chained into a single string via `nx_str_chain`
- **Mechanism** ⇝ Delegates parsing to `nx_sh_optargs` defined in `nex-sh-extras.awk`
- **Output** ⇝ Populates `NEX_k_*` and `NEX_f_*` variables with parsed options
- **Use Case** ⇝ Acts as the TeX-style gobbler: consumes matching options, leaves remainder stringified in NEX_S

Parse options into variables

- **Input** ⇝ `nx_data_optargs -f -g value`
- **Expected** ⇝ Sets `NEX_f_f="<nx:true/>"`, `NEX_f_g="value"`, remainder in NEX_S

```
nx_data_optargs -f -g value
```

II Option Matcher

nx_data_match()

```
nx_data_match()
(
    nx_data_optargs 'o@v:bl' "$@"
    ${AWK:-$(nx_cmd_awk)} \
        -v str="${NEX_k_v:-$NEX_S}" \
        -v opt="$NEX_K_o" \
        -v bnd="${NEX_f_b:-'<nx:false/>'}" \
        -v ln="${NEX_f_l:-'<nx:false/>'}" \
    "
        $(nx_data_include -i "${NEXUS_LIB}/awk/nex-struct-extras.awk")
    "'
        BEGIN {
            nx_trim_split(opt, opts, "<nx:null/>")
            if ((str = nx_option(str, opts, res, bnd == "<nx:true/>", ln == "<nx:true/>")) != -1) {
                print str
                str = 0
            } else {
                str = 1
            }
            delete res
            delete opts
            exit str
        }
    '
)
```

II Flag Semantics

Boolean Flag Behavior

- **-f** ⇝ First occurrence sets NEX_f_f="<nx:true/>"
- **-f -f** ⇝ Second occurrence negates, toggling to <nx:false/>
- **-fval** ⇝ Explicit value disables toggle; sets NEX_f_f="val"
- **-f=val** ⇝ Same as above; assigns literal string "val"
- **-f=val for f** ⇝ Escaped assignment; assigns full string "val for f" without toggle

Flag Toggle Example

Input ⇝ `nx_data_optargs 'f' -f -f`

Expected ⇝ First `-f` sets `NEX_f_f="<nx:true/>"`, second `-f` negates to `"<nx:false/>"`

```
nx_data_optargs 'f' -f -f
```

Flag Reactivation Example

Input ⇝ `nx_data_optargs 'f' -f=<nx:true/> -f`

Expected ⇝ Special assignment reactivates toggle; first sets true, second negates to <nx:false/>

```
nx_data_optargs 'f' -f=<nx:true/> -f
```

Flag Reactivation from False

Input ⇝ `nx_data_optargs 'f' -f<nx:false/> -f`

Expected ⇝ Starts from false, next `-f` flips back to <nx:true/>

```
nx_data_optargs 'f' -f<nx:false/> -f
```

II Flag Gobbling Logic

Flag Branch — Behavior

- **fs** ⇝ If next char matches flag separator (=), assign explicit value: `-f=val`
- **fa** ⇝ If next char matches append separator (+), append new segment: `-f+'val'`
- **fr** ⇝ If next char matches replace/pop separator (-), replace or pop last segment: `-f-'val'`
- **ln > 2** ⇝ If flag token has extra chars, gobble them as inline value: `-fval`
- **else** ⇝ Default toggle: first `-f` true, second `-f` false
- **debug** ⇝ If verbosity N > 3, print debug info and final flag value

Inline Value

- **Input** ⇝ `-fval`
- **Expected** ⇝ Sets `NEX_f_f="val"`

```
nx_data_optargs 'f' -fval
```

Explicit Assignment

- **Input** ⇝ `-f=foo`
- **Expected** ⇝ `NEX_f_f="foo"`; toggle disabled

```
nx_data_optargs 'f' -f=foo
```

Append Segment

- **Input** ⇝ `-f+'extra'`
- **Expected** ⇝ Appends `"extra"` to current flag value

```
nx_data_optargs 'f' -f+'extra'
```

Pop/Replace Segment

- **Input** ⇝ `-f-'new'`
- **Expected** ⇝ Replaces last appended segment with `"new"`

```
nx_data_optargs 'f' -f+'old' -f-'new'
```

Required Keyword (k:)

- **Input** ⇝ `nx_data_optargs 'k:' -k foo`
- **Expected** ⇝ `NEX_k_k="foo"`; next token gobbled
- **Input** ⇝ `nx_data_optargs 'k:' -k`
- **Expected** ⇝ No explicit error; parser attempts to gobble, finds nothing, leaves `NEX_k_k` unset

```
nx_data_optargs 'k:' -k foo
```

Keyword with Required Argument (k:)

- **Spec** ⇝ `'k:'` ⟶ keyword requires a value
- **Input** ⇝ `nx_data_optargs 'k:' -k foo`
- **Expected** ⇝ `NEX_k_k="foo"`; next token gobbled
- **Input** ⇝ `nx_data_optargs 'k:' -k`
- **Expected** ⇝ No error; parser attempts to gobble, finds nothing, leaves `NEX_k_k` unset

```
nx_data_optargs 'k:' -k foo
```

Keyword Array (k@)

- **Spec** ⇝ `'k@'` ⟶ keyword collects multiple values
- **Input** ⇝ `nx_data_optargs 'k@' -k foo -k bar -k baz`
- **Expected** ⇝ `NEX_K_k="foo<nx:null/>bar<nx:null/>baz"`; each `-k` gobbles the next token
- **Optional** ⇝ If no value follows, array remains empty

```
nx_data_optargs 'k@' -k foo -k bar -k baz
```

Option Matcher — The Match Glyph

- **Purpose** ⇝ Matches a candidate string against parsed options
- **Input** ⇝ Option spec `'o@v:bl'` defines expected flags; arguments passed through
- **Mechanism** ⇝ Parses options via `nx_data_optargs`, then calls `nx_option` to resolve
- **Comparison** ⇝ Supports boundary toggle (`bnd`) and length toggle (`ln`)
- **Return** ⇝ Prints matched string if found, exits 0; exits 1 if no match
- **Use Case** ⇝ Used to validate and select options in higher-level command wrappers

Match option against candidates

- **Input** ⇝ `nx_data_match -o "optA optB" -v "optA"`
- **Expected** ⇝ Prints `"optA"` and exits 0
- **Failure** ⇝ If no match, exits 1

```
nx_data_match -o "optA optB" -v "optA"
```