SHELL

SHELL

November 14, 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 Interface Classifier — The Net Glyph

nx_ip_net() — Classify Physical and Virtual Interfaces

```
nx_ip_net()
{
    ls --color=never -l '/sys/class/net/' | ${AWK:-$(nx_cmd_awk)} -v ex="$1" -F '/' '
        BEGIN {
            if (ex == "-e")
                ex = "export "
            else
                ex = ""
            virt = ""
            phy = ""
        }
        /devices\/pci/{
            phy = phy " " $NF
        }
        /devices\/virtual/{
            virt = virt " " $NF
        }
        END {
            printf("%sG_NEX_NET_VIRT\x22=%s\x22\n%sG_NEX_NET_PHY=\x22%s\x22\n", ex, substr(virt, 2), ex, substr(phy, 2));
        }
    '
}
```

Interface Classifier — The Net Glyph

- **Purpose** ⇝ Classifies network interfaces into virtual and physical sets using symlink lineage from `/sys/class/net`
- **Input** ⇝ Optional flag `-e` to prefix output with `export`
- **Mechanism** ⇝ AWK parses symlink targets for `/devices/pci` (physical) and `/devices/virtual` (virtual)
- **Output** ⇝ Defines G_NEX_NET_VIRT and G_NEX_NET_PHY with space-separated interface names
- **Use Case** ⇝ Used in overlay scripts, interface audits, routing logic, or symbolic network containment

Classify interfaces and export results

- **Initial State** ⇝ System has `lo`, `eth0`, and `virbr0`
- **Operation** ⇝ Run `nx_ip_net -e` to classify and export
- **Expected Result** ⇝ Outputs `export G_NEX_NET_VIRT="lo virbr0"` and `export G_NEX_NET_PHY="eth0"`

```
nx_ip_net -e
```

Classify interfaces without export prefix

- **Initial State** ⇝ System has `wlan0` and `docker0`
- **Operation** ⇝ Run `nx_ip_net` without arguments
- **Expected Result** ⇝ Outputs `G_NEX_NET_VIRT="docker0"` and `G_NEX_NET_PHY="wlan0"`

```
nx_ip_net
```

I Layer 2 Address Generator — The L2 Glyph

nx_ip_l2() — Generate Unique MAC-like Address

```
nx_ip_l2()
(
    eval "$(nx_str_optarg ':n:' "$@")"
    test -n "$n" && n="-n $n"
    while :; do
        tmpa="$(${AWK:-$(nx_cmd_awk)} -v addr="$(nx_str_rand 12 xdigit)" '
            BEGIN {
                l = split(tolower(addr), hex, "")
                do {
                    s = s ":" hex[l] hex[l-1]
                } while ((l-=2) > 0)
                delete hex
                print substr(s, 2)
            }
        ')"
        g_nx_ip_l2 $n -a | grep -q "$tmpa" || break
    done
    printf '%s\n' "$tmpa"
)
```

Layer 2 Address Generator — The L2 Glyph

- **Purpose** ⇝ Generates a unique MAC-like address not currently in use
- **Input** ⇝ Optional flag `-n` to namespace the lookup via `g_nx_ip_l2`
- **Mechanism** ⇝ Generates 12 random hex digits, formats into colon-separated MAC form, checks for collision
- **Collision Check** ⇝ Uses `g_nx_ip_l2 -a` to verify uniqueness before emitting
- **Use Case** ⇝ Used in virtual interface creation, container overlays, or symbolic L2 address staging

Generate a unique MAC-like address

- **Initial State** ⇝ No arguments passed; default namespace lookup
- **Operation** ⇝ Generates and emits a unique address like `fa:3c:9e:12:ab:77`
- **Expected Result** ⇝ Returns a MAC-like string not found in `g_nx_ip_l2 -a`

```
nx_ip_l2
```

Generate address with namespace constraint

- **Initial State** ⇝ Namespace `veth` passed via `-n veth`
- **Operation** ⇝ Ensures uniqueness within `g_nx_ip_l2 -n veth -a`
- **Expected Result** ⇝ Returns a MAC-like string not found in the `veth` namespace

```
nx_ip_l2 -n veth
```

I Layer 2 Address Inspector — The L2 Audit Glyph

g_nx_ip_l2() — Inspect MAC Addresses Across Namespaces

```
g_nx_ip_l2()
(
    eval "$(nx_str_optarg ':n:a' "$@")"
    test -n "$n" && n="ip netns exec $n "
    test -n "$a" && $n ip neighbor | ${AWK:-$(nx_cmd_awk)} '{ print $(NF - 1) }'
    tmpa="$($n ip -json address show $NEX_OPT_RMDR 2> /dev/null)" && nx_data_jdump "$tmpa" | ${AWK:-$(nx_cmd_awk)} '/\.nx

\[[0-9]+\]

\.address =/{print $NF}'
)
```

Layer 2 Address Inspector — The L2 Audit Glyph

- **Purpose** ⇝ Inspects and emits MAC-like addresses from interfaces and neighbors, optionally scoped to a namespace
- **Input** ⇝ `-n` for namespace; `-a` to emit neighbor MACs
- **Mechanism** ⇝ Uses `ip -json address show` and `ip neighbor` to extract Layer 2 addresses
- **Output** ⇝ Emits one MAC per line from either interface JSON or neighbor table
- **Use Case** ⇝ Used in collision checks, symbolic L2 audits, or overlay address validation

Emit MAC addresses from current namespace

- **Initial State** ⇝ System has interfaces with MACs like `fa:3c:9e:12:ab:77`
- **Operation** ⇝ Run `g_nx_ip_l2` with no arguments
- **Expected Result** ⇝ Emits MAC-like addresses from `ip -json address show`

```
g_nx_ip_l2
```

Emit neighbor MACs from namespace

- **Initial State** ⇝ Namespace `veth` has active neighbors
- **Operation** ⇝ Run `g_nx_ip_l2 -n veth -a`
- **Expected Result** ⇝ Emits MACs from `ip netns exec veth ip neighbor`

```
g_nx_ip_l2 -n veth -a
```

I ARP Table Parser — The ARP Glyph

nx_ip_arp() — Parse and Emit ARP Table as JSON

```
nx_ip_arp()
{
        ${AWK:-$(nx_cmd_awk)} '
                {
                        if (! header) {
                                header = 1
                                next
                        }
                        iface[$NF] = iface[$NF] "{\x22ip\x22:\x22" $1 "\x22,\x22type\x22:\x22" $2 "\x22,\x22flags\x22:\x22" $3 "\x22,\x22hw\x22:\x22" $4 "\x22,\x22mask\x22:\x22" $5 "\x22},"
                } END {
                        for (face in iface) {
                                sub(/,$/,"]},", iface[face])
                                s = s "{\x22" face "\x22:[" iface[face]
                        }
                        sub(/,$/, "]", s)
                        print "[" s
                        delete iface
                }
        ' $(
              test -z "$1" && printf '%s' '/proc/self/net/arp' || {
                        test -f "$1" && printf '%s' "/proc/$1/net/arp" || printf '%s' '/proc/net/arp'
                }
        )
}
```

ARP Table Parser — The ARP Glyph

- **Purpose** ⇝ Parses the system ARP table and emits structured JSON grouped by interface
- **Input** ⇝ Optional PID or file path to target a specific ARP table
- **Mechanism** ⇝ AWK parses fields from `/proc/*/net/arp` and groups entries by interface
- **Output** ⇝ Emits a JSON array of interface-keyed ARP entries with fields: `ip`, `type`, `flags`, hw, `mask`
- **Use Case** ⇝ Used in symbolic network audits, container overlays, or L2/L3 mapping rituals

Parse current process ARP table

- **Initial State** ⇝ System has ARP entries for `eth0` and `docker0`
- **Operation** ⇝ Run `nx_ip_arp` with no arguments
- **Expected Result** ⇝ Emits JSON with keys `"eth0"` and `"docker0"` containing ARP entries

```
nx_ip_arp
```

Parse ARP table for specific PID

- **Initial State** ⇝ PID `1234` has a net namespace with ARP entries
- **Operation** ⇝ Run `nx_ip_arp 1234`
- **Expected Result** ⇝ Emits JSON from `/proc/1234/net/arp` grouped by interface

```
nx_ip_arp 1234
```

I Interface Name Resolver — The Name Glyph

nx_ip_name() — Resolve Unique Interface Name

```
nx_ip_name()
{
        tmpa="$(
                eval "$(nx_str_optarg ':n:b:v' "$@")"
                test -n "$NEX_OPT_RMDR" || exit
               test -n "$n" && tmpd="ip netns exec $n " || tmpd=""
                tmpa="$(${AWK:-$(nx_cmd_awk)} -v name="$NEX_OPTSTR_RMDR" -v base="$b" 'BEGIN {
                        if (name !~ /^[0-9A-Za-z_-]{1,15}$/)
                                exit 1
                        if (match(name, /[0-9]+$/)) {
                               if ((cur = substr(name, 1, RSTART - 1)) == base)
                                        exit 3
                                printf("tmpa=%s tmpb=\x22%s\x22", substr(name, RSTART), cur)
                        } else {
                            printf("tmpa=0 tmpb=\x22%s\x22", name)
                        }
                }')" || {
                        test $? -eq 3 && exit 3
                        nx_io_printf -E "$NEX_OPT_STR_RMDR is an invalid name, names must be 1 to 15 character of '0-9,a-z,A-Z,-._'" 1>&2
                        unset tmpa
                        exit 1
                }
                eval "$tmpa"
                while $tmpd ip link show "$tmpb$tmpa" 2>/dev/null 1>&2 || test "$tmpb$tmpa" = "$b"; do
                        tmpa=$((tmpa+1))
                      test "$(nx_str_len "$tmpb$tmpa")" -le 15 || {
                                nx_io_printf -E "interface name '$tmpb$tmpa' is to[o] long, the maximum length is 15." 1>&2
                                exit 2
                        }
                done
                printf 'tmpa=\x22%s\x22 tmpc=\x22%s\x22\n' "$tmpb$tmpa" "$v"
        )" || return
        eval "$tmpa"
        test -n "$tmpc" && printf '%s\n' "$tmpa"
        unset tmpc
}
```

Interface Name Resolver — The Name Glyph

- **Purpose** ⇝ Resolves a unique interface name by incrementing suffixes and validating namespace collisions
- **Input** ⇝ `-n` for namespace, `-b` for base prefix, `-v` for optional tag
- **Validation** ⇝ Rejects names longer than 15 characters or invalid characters outside `[0-9A-Za-z_-]`
- **Mechanism** ⇝ Splits numeric suffix, checks for collisions via `ip link show`, increments until unique
- **Output** ⇝ Emits `tmpa` as resolved name and `tmpc` as optional tag
- **Use Case** ⇝ Used in veth pair creation, container overlays, or symbolic interface staging

Resolve unique name with base prefix

- **Initial State** ⇝ Base prefix `veth` and desired name `veth0`
- **Operation** ⇝ Run `nx_ip_name -b veth veth0`
- **Expected Result** ⇝ Emits `tmpa="veth1"` if `veth0` exists

```
nx_ip_name -b veth veth0
```

Resolve name within namespace

- **Initial State** ⇝ Namespace `ns1` contains `eth0`
- **Operation** ⇝ Run `nx_ip_name -n ns1 eth0`
- **Expected Result** ⇝ Emits `tmpa="eth1"` if `eth0` exists in `ns1`

```
nx_ip_name -n ns1 eth0
```

Resolve name with optional tag via `-v`

- Initial State ⇝ Base prefix `veth`, desired name `veth0`, and tag `uplink`
- Operation ⇝ Run `nx_ip_name -b veth -v uplink veth0`
- Expected Result ⇝ Emits `tmpa="veth1"` and `tmpc="uplink"` if `veth0` exists

```
nx_ip_name -b veth -v uplink veth0
```

I Namespace Lifecycle Manager — The Netns Glyph

nx_ip_netns() — Create or Remove Network Namespace

```
nx_ip_netns()
(
        eval "$(nx_str_optarg 'r' "$@")"
        eval "$(
                test -n "$r" && {
                        nx_data_repeat '
                                ip netns | grep -q "$NEX_ARG" && {
                                        kill "$(cat /var/run/nex-$NEX_ARG.pid)" 2> /dev/null
                                        ip netns delete "$NEX_ARG"
                                        rm -f "/var/run/netns/nex-$NEX_ARG.pid"
                                }
                        ' "$NEX_OPT_RMDR"
                } || {
                        nx_data_repeat '
                                ip netns | grep -q "$NEX_ARG" || {
                                       ip netns add "$NEX_ARG" && {
                                                ip netns exec "$NEX_ARG" sysctl --system 1> /dev/null 2>&1
                                                ip netns exec "$NEX_ARG" ip link set lo up
                                               nohup setsid nsenter --net="/var/run/netns/$NEX_ARG" sleep infinity 1> /dev/null 2>&1 &
                                                printf $! > "/var/run/nex-$NEX_ARG.pid"
                                        }
                                }
                        ' "$NEX_OPT_RMDR"
                }
        )"
)
```

Namespace Lifecycle Manager — The Netns Glyph

- **Purpose** ⇝ Creates or removes a Linux network namespace with optional persistent process
- **Input** ⇝ Namespace name via trailing argument; `-r` flag triggers removal
- **Mechanism** ⇝ Uses `ip netns` to create/delete; spawns persistent `sleep` via `nsenter` and stores PID
- **Output** ⇝ Creates or removes `/var/run/nex-$name.pid` and namespace entry in `/var/run/netns`
- **Use Case** ⇝ Used in container overlays, symbolic network isolation, or namespace lifecycle rituals

Create a new network namespace

- **Initial State** ⇝ No namespace named `ns1` exists
- **Operation** ⇝ Run `nx_ip_netns ns1`
- **Expected Result** ⇝ Creates `ns1`, brings up `lo`, spawns persistent `sleep`, stores PID

```
nx_ip_netns ns1
```

Remove an existing network namespace

- **Initial State** ⇝ Namespace `ns1` exists with PID file
- **Operation** ⇝ Run `nx_ip_netns -r ns1`
- **Expected Result** ⇝ Kills persistent process, deletes namespace, removes PID file

```
nx_ip_netns -r ns1
```

I Namespace Executor — The Exec Glyph

__nx_ip_exec() — Emit Namespace Execution Prefix

```
__nx_ip_exec()
{
        test -n "$1" && {
                nx_ip_netns "$1"
                printf '%s ' "${1:+ip netns exec $1}"
        }
}
```

Namespace Executor — The Exec Glyph

- **Purpose** ⇝ Emits a namespace-prefixed command string for use in subshells or command substitution
- **Input** ⇝ Namespace name as first argument
- **Mechanism** ⇝ Ensures namespace exists via `nx_ip_netns`, then emits `ip netns exec $name`
- **Output** ⇝ Prints `ip netns exec $name` if namespace is provided; empty string otherwise
- **Use Case** ⇝ Used to prefix commands with namespace context in symbolic overlays or interface rituals

Emit namespace exec prefix for a given namespace

- **Initial State** ⇝ Namespace `ns1` may or may not exist
- **Operation** ⇝ Run `__nx_ip_exec ns1`
- **Expected Result** ⇝ Ensures `ns1` exists, emits `ip netns exec ns1`

```
__nx_ip_exec ns1
```

I Altname Remover — The Alt Glyph

d_nx_ip_alt() — Remove Alternate Interface Name

```
d_nx_ip_alt()
{
        g_nx_ip_alt $2 | grep -q '^'"$1"'$' || return 1
        $(__nx_ip_exec "$2") ip link property del dev $(g_nx_ip_ifname "$1" "$2") altname "$1"
}
```

Altname Remover — The Alt Glyph

- **Purpose** ⇝ Removes an alternate name from a network interface, scoped optionally to a namespace
- **Input** ⇝ $1: altname to remove; $2: optional namespace
- **Validation** ⇝ Checks if altname exists via `g_nx_ip_alt`; aborts if not present
- **Mechanism** ⇝ Uses `ip link property del` to remove altname from resolved interface name
- **Use Case** ⇝ Used in symbolic interface cleanup, altname mutation, or namespace-scoped interface audits

Remove altname from interface in namespace

- **Initial State** ⇝ Interface `eth0` in namespace `ns1` has altname `uplink`
- **Operation** ⇝ Run `d_nx_ip_alt uplink ns1`
- **Expected Result** ⇝ Removes altname `uplink` from `eth0` in `ns1`

```
d_nx_ip_alt uplink ns1
```

I Layer 2 Address Inspector — The L2 Glyph

g_nx_ip_l2() — Emit MAC-like Addresses from Namespace

```
g_nx_ip_l2()
(
        eval "$(nx_str_optarg ':n:a' "$@")"
        test -n "$n" && n="ip netns exec $n "
        test -n "$a" && $n ip neighbor | ${AWK:-$(nx_cmd_awk)} '{
↪print $(NF - 1) }'
        tmpa="$($n ip -json address show $NEX_OPT_RMDR 2>
↪/dev/null)" && nx_data_jdump "$tmpa" | ${AWK:-$(nx_cmd_awk)}
↪'/\.nx

\[[0-9]+\]

\.address =/{print $NF}'
)
```

Layer 2 Address Inspector — The L2 Glyph

- **Purpose** ⇝ Emits MAC-like addresses from interfaces and neighbors, optionally scoped to a namespace
- **Input** ⇝ `-n` for namespace; `-a` to emit neighbor MACs
- **Mechanism** ⇝ Uses `ip -json address show` and `ip neighbor` to extract Layer 2 addresses
- **Output** ⇝ Emits one MAC per line from either interface JSON or neighbor table
- **Use Case** ⇝ Used in collision checks, symbolic L2 audits, or overlay address validation

Emit MAC addresses from current namespace

- **Initial State** ⇝ System has interfaces with MACs like `fa:3c:9e:12:ab:77`
- **Operation** ⇝ Run `g_nx_ip_l2` with no arguments
- **Expected Result** ⇝ Emits MAC-like addresses from `ip -json address show`

```
g_nx_ip_l2
```

Emit neighbor MACs from namespace

- </> Initial State ⇝ Namespace `ns1` has active neighbors
- </> Operation ⇝ Run `g_nx_ip_l2 -n ns1 -a`
- </> Expected Result ⇝ Emits MACs from `ip netns exec ns1 ip neighbor`

```
g_nx_ip_l2 -n ns1 -a
```

I Interface Name Resolver — The Ifname Glyph

g_nx_ip_ifname() — Resolve Kernel Interface Name

```
g_nx_ip_ifname()
(
        tmpa="$($(__nx_ip_exec "$2") ip -json link show $1 2>/dev/null)" && nx_data_jdump "$tmpa" | ${AWK:-$(nx_cmd_awk)} '/.nx

\[[0-9]+\]

\.ifname/{print $NF}
'
)
```

Interface Name Resolver — The Ifname Glyph

- Purpose ⇝ Resolves the kernel-assigned interface name for a given link index or identifier
- Input ⇝ $1: link index or name; $2: optional namespace
- Mechanism ⇝ Uses `ip -json link show` and parses `ifname` field via `nx_data_jdump`
- Output ⇝ Emits the resolved interface name as seen by the kernel
- Use Case ⇝ Used in altname mutation, symbolic interface mapping, or namespace-scoped link audits

Resolve interface name from link index in namespace

- **Initial State** ⇝ Link index 3 exists in namespace `ns1`
- **Operation** ⇝ Run `g_nx_ip_ifname 3 ns1`
- **Expected Result** ⇝ Emits interface name like `eth0` or `veth3`

```
g_nx_ip_ifname 3 ns1
```

I Altname Auditor — The Alt Audit Glyph

g_nx_ip_alt() — Emit Alternate Interface Names

```
g_nx_ip_alt()
(
        eval "$(nx_str_optarg ':n:' "$@")"
        test -n "$n" && n="ip netns exec $n "
        $n ip -json link show $NEX_OPT_RMDR 2> /dev/null | nx_data_jdump | ${AWK:-$(nx_cmd_awk)} '/\.altname/{print $NF}'
)
```

Altname Auditor — The Alt Audit Glyph

- **Purpose** ⇝ Emits alternate names assigned to a given interface, optionally scoped to a namespace
- **Input** ⇝ `-n` for namespace; trailing argument is interface name or index
- **Mechanism** ⇝ Uses `ip -json link show` and parses `altname` fields via `nx_data_jdump`
- **Output** ⇝ Emits one altname per line
- **Use Case** ⇝ Used in symbolic interface audits, altname mutation, or namespace-scoped link overlays

Emit altnames from interface in current namespace

- </> Initial State ⇝ Interface `eth0` has altnames `uplink` and `primary`
- </> Operation ⇝ Run `g_nx_ip_alt eth0`
- </> Expected Result ⇝ Emits `uplink` and `primary` on separate lines

```
g_nx_ip_alt eth0
```

Emit altnames from interface in namespace

- </> Initial State ⇝ Interface `veth0` in namespace `ns1` has altname `peer`
- </> Operation ⇝ Run `g_nx_ip_alt -n ns1 veth0`
- </> Expected Result ⇝ Emits `peer`

```
g_nx_ip_alt -n ns1 veth0
```