Network Working Group R. Quattlebaum
Internet-Draft J. Woodyatt
Intended status: Informational Nest Labs, Inc.
Expires: November 10, 2017 May 9, 2017

# Spinel Host-Controller Protocol
## draft-rquattle-spinel-unified-00

## Abstract

This document describes the Spinel protocol, which facilitates the control and management of IPv6 network interfaces on devices where general purpose application processors offload network functions at their interfaces to network co-processors (NCP) connected by simple communication links like serial data channels. While initially developed to support Thread(R), Spinel's layered design allows it to be easily adapted to other similar network technologies.

This document also describes various Spinel specializations, including support for the Thread(R) low-power mesh network technology.

## Status of This Memo

This Internet-Draft will expire on November 10, 2017.

## Copyright Notice

Table of Contents

1. Introduction . . . . . . . . . . . . . . . . . . . . . . . . 6
  1.1. About this Draft . . . . . . . . . . . . . . . . . . . . 7
    1.1.1. Scope . . . . . . . . . . . . . . . . . . . . . . . . 7
    1.1.2. Renumbering . . . . . . . . . . . . . . . . . . . . . 7
2. Frame Format . . . . . . . . . . . . . . . . . . . . . . . . 8
  2.1. Header Format . . . . . . . . . . . . . . . . . . . . . . 8
    2.1.1. FLG: Flag . . . . . . . . . . . . . . . . . . . . . . 8
    2.1.2. NLI: Network Link Identifier . . . . . . . . . . . . 9
    2.1.3. TID: Transaction Identifier . . . . . . . . . . . . . 9
    2.1.4. Command Identifier (CMD) . . . . . . . . . . . . . . 9
    2.1.5. Command Payload (Optional) . . . . . . . . . . . . . 10
3. Data Packing . . . . . . . . . . . . . . . . . . . . . . . . 10
  3.1. Primitive Types . . . . . . . . . . . . . . . . . . . . . 10
  3.2. Packed Unsigned Integer . . . . . . . . . . . . . . . . . 11
  3.3. Data Blobs . . . . . . . . . . . . . . . . . . . . . . . 12
  3.4. Structured Data . . . . . . . . . . . . . . . . . . . . . 13
  3.5. Arrays . . . . . . . . . . . . . . . . . . . . . . . . . 13
4. Commands . . . . . . . . . . . . . . . . . . . . . . . . . . 14
  4.1. CMD 0: (Host->NCP) CMD_NOOP . . . . . . . . . . . . . . . 14
  4.2. CMD 1: (Host->NCP) CMD_RESET . . . . . . . . . . . . . . 14
  4.3. CMD 2: (Host->NCP) CMD_PROP_VALUE_GET . . . . . . . . . . 14
  4.4. CMD 3: (Host->NCP) CMD_PROP_VALUE_SET . . . . . . . . . . 15
  4.5. CMD 4: (Host->NCP) CMD_PROP_VALUE_INSERT . . . . . . . . 15
  4.6. CMD 5: (Host->NCP) CMD_PROP_VALUE_REMOVE . . . . . . . . 16
  4.7. CMD 6: (NCP->Host) CMD_PROP_VALUE_IS . . . . . . . . . . 17
  4.8. CMD 7: (NCP->Host) CMD_PROP_VALUE_INSERTED . . . . . . . 17
  4.9. CMD 8: (NCP->Host) CMD_PROP_VALUE_REMOVED . . . . . . . . 18
  4.10. CMD 18: (Host->NCP) CMD_PEEK . . . . . . . . . . . . . . 18
  4.11. CMD 19: (NCP->Host) CMD_PEEK_RET . . . . . . . . . . . . 19
  4.12. CMD 20: (Host->NCP) CMD_POKE . . . . . . . . . . . . . . 19
  4.13. CMD 21: (Host->NCP) CMD_PROP_VALUE_MULTI_GET . . . . . . 19
  4.14. CMD 22: (Host->NCP) CMD_PROP_VALUE_MULTI_SET . . . . . . 20
  4.15. CMD 23: (NCP->Host) CMD_PROP_VALUES_ARE . . . . . . . . . 21
5. Properties . . . . . . . . . . . . . . . . . . . . . . . . . 21
  5.1. Property Methods . . . . . . . . . . . . . . . . . . . . 22
  5.2. Property Types . . . . . . . . . . . . . . . . . . . . . 22

5.2.1. Single-Value Properties . . . . . . . . . . . . . . . . 22
5.2.2. Multiple-Value Properties . . . . . . . . . . . . . . . 23
5.2.3. Stream Properties . . . . . . . . . . . . . . . . . . . 23
5.3. Property Numbering . . . . . . . . . . . . . . . . . . . . 24
5.4. Property Sections . . . . . . . . . . . . . . . . . . . . 24
5.5. Core Properties . . . . . . . . . . . . . . . . . . . . . 25
5.5.1. PROP 0: PROP_LAST_STATUS . . . . . . . . . . . . . . . 25
5.5.2. PROP 1: PROP_PROTOCOL_VERSION . . . . . . . . . . . . . 25
5.5.3. PROP 2: PROP_NCP_VERSION . . . . . . . . . . . . . . . 26
5.5.4. PROP 3: PROP_INTERFACE_TYPE . . . . . . . . . . . . . 26
5.5.5. PROP 4: PROP_INTERFACE_VENDOR_ID . . . . . . . . . . . 27
5.5.6. PROP 5: PROP_CAPS . . . . . . . . . . . . . . . . . . . 27
5.5.7. PROP 6: PROP_INTERFACE_COUNT . . . . . . . . . . . . . 29
5.5.8. PROP 7: PROP_POWER_STATE . . . . . . . . . . . . . . . 29
5.5.9. PROP 8: PROP_HWADDR . . . . . . . . . . . . . . . . . . 30
5.5.10. PROP 9: PROP_LOCK . . . . . . . . . . . . . . . . . . 30
5.6. Stream Properties . . . . . . . . . . . . . . . . . . . . 30
5.6.1. PROP 112: PROP_STREAM_DEBUG . . . . . . . . . . . . . 30
5.6.2. PROP 113: PROP_STREAM_RAW . . . . . . . . . . . . . . 31
5.6.3. PROP 114: PROP_STREAM_NET . . . . . . . . . . . . . . 33
5.6.4. PROP 114: PROP_STREAM_NET_INSECURE . . . . . . . . . . 34
5.7. PHY Properties . . . . . . . . . . . . . . . . . . . . . . 34
5.7.1. PROP 32: PROP_PHY_ENABLED . . . . . . . . . . . . . . 34
5.7.2. PROP 33: PROP_PHY_CHAN . . . . . . . . . . . . . . . . 34
5.7.3. PROP 34: PROP_PHY_CHAN_SUPPORTED . . . . . . . . . . . 35
5.7.4. PROP 35: PROP_PHY_FREQ . . . . . . . . . . . . . . . . 35
5.7.5. PROP 36: PROP_PHY_CCA_THRESHOLD . . . . . . . . . . . 35
5.7.6. PROP 37: PROP_PHY_TX_POWER . . . . . . . . . . . . . . 35
5.7.7. PROP 38: PROP_PHY_RSSI . . . . . . . . . . . . . . . . 35
5.7.8. PROP 39: PROP_PHY_RX_SENSITIVITY . . . . . . . . . . . 36
5.8. MAC Properties . . . . . . . . . . . . . . . . . . . . . . 36
5.8.1. PROP 48: PROP_MAC_SCAN_STATE . . . . . . . . . . . . . 36
5.8.2. PROP 49: PROP_MAC_SCAN_MASK . . . . . . . . . . . . . 36
5.8.3. PROP 50: PROP_MAC_SCAN_PERIOD . . . . . . . . . . . . 36
5.8.4. PROP 51: PROP_MAC_SCAN_BEACON . . . . . . . . . . . . 37
5.8.5. PROP 52: PROP_MAC_15_4_LADDR . . . . . . . . . . . . . 37
5.8.6. PROP 53: PROP_MAC_15_4_SADDR . . . . . . . . . . . . . 38
5.8.7. PROP 54: PROP_MAC_15_4_PANID . . . . . . . . . . . . . 38
5.8.8. PROP 55: PROP_MAC_RAW_STREAM_ENABLED . . . . . . . . . 38
5.8.9. PROP 56: PROP_MAC_PROMISCUOUS_MODE . . . . . . . . . . 38
5.8.10. PROP 57: PROP_MAC_ENERGY_SCAN_RESULT . . . . . . . . 39
5.8.11. PROP 4864: PROP_MAC_WHITELIST . . . . . . . . . . . . 39
5.8.12. PROP 4865: PROP_MAC_WHITELIST_ENABLED . . . . . . . . 39
5.9. NET Properties . . . . . . . . . . . . . . . . . . . . . . 39
5.9.1. PROP 64: PROP_NET_SAVED . . . . . . . . . . . . . . . 39
5.9.2. PROP 65: PROP_NET_IF_UP . . . . . . . . . . . . . . . 39
5.9.3. PROP 66: PROP_NET_STACK_UP . . . . . . . . . . . . . . 40
5.9.4. PROP 67: PROP_NET_ROLE . . . . . . . . . . . . . . . . 40

5.9.5. PROP 68: PROP_NET_NETWORK_NAME . . . . . . . . . . . . 40
5.9.6. PROP 69: PROP_NET_XPANID . . . . . . . . . . . . . . . 40
5.9.7. PROP 70: PROP_NET_MASTER_KEY . . . . . . . . . . . . . 40
5.9.8. PROP 71: PROP_NET_KEY_SEQUENCE_COUNTER . . . . . . . . 40
5.9.9. PROP 72: PROP_NET_PARTITION_ID . . . . . . . . . . . . 40
5.9.10. PROP 73: PROP_NET_REQUIRE_JOIN_EXISTING . . . . . . . . 41
5.9.11. PROP 74: PROP_NET_KEY_SWITCH_GUARDTIME . . . . . . . . 41
5.9.12. PROP 75: PROP_NET_PSKC . . . . . . . . . . . . . . . . 41
5.10. IPv6 Properties . . . . . . . . . . . . . . . . . . . . . . 41
5.10.1. PROP 96: PROP_IPV6_LL_ADDR . . . . . . . . . . . . . 41
5.10.2. PROP 97: PROP_IPV6_ML_ADDR . . . . . . . . . . . . . 41
5.10.3. PROP 98: PROP_IPV6_ML_PREFIX . . . . . . . . . . . . 41
5.10.4. PROP 99: PROP_IPV6_ADDRESS_TABLE . . . . . . . . . . 41
5.10.5. PROP 101: PROP_IPv6_ICMP_PING_OFFLOAD . . . . . . . . 42
5.11. Debug Properties . . . . . . . . . . . . . . . . . . . . . 42
5.11.1. PROP 16384: PROP_DEBUG_TEST_ASSERT . . . . . . . . . 42
5.11.2. PROP 16385: PROP_DEBUG_NCP_LOG_LEVEL . . . . . . . . 42
6. Status Codes . . . . . . . . . . . . . . . . . . . . . . . . 43
7. Technology: Thread(R) . . . . . . . . . . . . . . . . . . . . 44
7.1. Capabilities . . . . . . . . . . . . . . . . . . . . . . . 44
7.2. Properties . . . . . . . . . . . . . . . . . . . . . . . . 45
7.2.1. PROP 80: PROP_THREAD_LEADER_ADDR . . . . . . . . . . . 45
7.2.2. PROP 81: PROP_THREAD_PARENT . . . . . . . . . . . . . . 45
7.2.3. PROP 82: PROP_THREAD_CHILD_TABLE . . . . . . . . . . . 45
7.2.4. PROP 83: PROP_THREAD_LEADER_RID . . . . . . . . . . . . 45
7.2.5. PROP 84: PROP_THREAD_LEADER_WEIGHT . . . . . . . . . . 45
7.2.6. PROP 85: PROP_THREAD_LOCAL_LEADER_WEIGHT . . . . . . . 46
7.2.7. PROP 86: PROP_THREAD_NETWORK_DATA . . . . . . . . . . . 46
7.2.8. PROP 87: PROP_THREAD_NETWORK_DATA_VERSION . . . . . . . 46
7.2.9. PROP 88: PROP_THREAD_STABLE_NETWORK_DATA . . . . . . . 46
7.2.10. PROP 89: PROP_THREAD_STABLE_NETWORK_DATA_VERSION . . 46
7.2.11. PROP 90: PROP_THREAD_ON_MESH_NETS . . . . . . . . . . 46
7.2.12. PROP 91: PROP_THREAD_LOCAL_ROUTES . . . . . . . . . . 47
7.2.13. PROP 92: PROP_THREAD_ASSISTING_PORTS . . . . . . . . 47
7.2.14. PROP 93: PROP_THREAD_ALLOW_LOCAL_NET_DATA_CHANGE . . 47
7.2.15. PROP 94: PROP_THREAD_MODE . . . . . . . . . . . . . . 47
7.2.16. PROP 5376: PROP_THREAD_CHILD_TIMEOUT . . . . . . . . 47
7.2.17. PROP 5377: PROP_THREAD_RLOC16 . . . . . . . . . . . . 47
7.2.18. PROP 5378: PROP_THREAD_ROUTER_UPGRADE_THRESHOLD . . . 48
7.2.19. PROP 5379: PROP_THREAD_CONTEXT_REUSE_DELAY . . . . . 48
7.2.20. PROP 5380: PROP_THREAD_NETWORK_ID_TIMEOUT . . . . . . 48
7.2.21. PROP 5381: PROP_THREAD_ACTIVE_ROUTER_IDS . . . . . . 48
7.2.22. PROP 5382: PROP_THREAD_RLOC16_DEBUG_PASSTHRU . . . . 48
7.2.23. PROP 5383: PROP_THREAD_ROUTER_ROLE_ENABLED . . . . . 48
7.2.24. PROP 5384: PROP_THREAD_ROUTER_DOWNGRADE_THRESHOLD . . 49
7.2.25. PROP 5385: PROP_THREAD_ROUTER_SELECTION_JITTER . . . 49
7.2.26. PROP 5386: PROP_THREAD_PREFERRED_ROUTER_ID . . . . . 49
7.2.27. PROP 5387: PROP_THREAD_NEIGHBOR_TABLE . . . . . . . . 49

```
         7.2.28. PROP 5388: PROP_THREAD_CHILD_COUNT_MAX  . . . . . . . .  49
         7.2.29. PROP 5389: PROP_THREAD_LEADER_NETWORK_DATA  . . . . .  50
         7.2.30. PROP 5390: PROP_THREAD_STABLE_LEADER_NETWORK_DATA . .  50
         7.2.31. PROP 5391: PROP_THREAD_JOINERS  . . . . . . . . . . .  50
         7.2.32. PROP 5392: PROP_THREAD_COMMISSIONER_ENABLED . . . . .  50
         7.2.33. PROP 5393: PROP_THREAD_BA_PROXY_ENABLED . . . . . . .  50
         7.2.34. PROP 5394: PROP_THREAD_BA_PROXY_STREAM  . . . . . . .  51
         7.2.35. PROP 5395: PROP_THREAD_DISOVERY_SCAN_JOINER_FLAG  . .  51
         7.2.36. PROP 5396:
                 PROP_THREAD_DISCOVERY_SCAN_ENABLE_FILTERING . . . . .  51
         7.2.37. PROP 5397: PROP_THREAD_DISCOVERY_SCAN_PANID . . . . .  51
         7.2.38. PROP 5398: PROP_THREAD_STEERING_DATA  . . . . . . . .  52
     8.  Feature: Network Save . . . . . . . . . . . . . . . . . . . .  52
       8.1.  Commands  . . . . . . . . . . . . . . . . . . . . . . . .  52
         8.1.1.  CMD 9: (Host->NCP) CMD_NET_SAVE . . . . . . . . . . .  52
         8.1.2.  CMD 10: (Host->NCP) CMD_NET_CLEAR . . . . . . . . . .  53
         8.1.3.  CMD 11: (Host->NCP) CMD_NET_RECALL  . . . . . . . . .  53
     9.  Feature: Host Buffer Offload  . . . . . . . . . . . . . . . .  54
       9.1.  Commands  . . . . . . . . . . . . . . . . . . . . . . . .  54
         9.1.1.  CMD 12: (NCP->Host) CMD_HBO_OFFLOAD . . . . . . . . .  54
         9.1.2.  CMD 13: (NCP->Host) CMD_HBO_RECLAIM . . . . . . . . .  54
         9.1.3.  CMD 14: (NCP->Host) CMD_HBO_DROP  . . . . . . . . . .  54
         9.1.4.  CMD 15: (Host->NCP) CMD_HBO_OFFLOADED . . . . . . . .  54
         9.1.5.  CMD 16: (Host->NCP) CMD_HBO_RECLAIMED . . . . . . . .  55
         9.1.6.  CMD 17: (Host->NCP) CMD_HBO_DROPPED . . . . . . . . .  55
       9.2.  Properties  . . . . . . . . . . . . . . . . . . . . . . .  55
         9.2.1.  PROP 10: PROP_HBO_MEM_MAX . . . . . . . . . . . . . .  55
         9.2.2.  PROP 11: PROP_HBO_BLOCK_MAX . . . . . . . . . . . . .  55
     10. Feature: Jam Detection  . . . . . . . . . . . . . . . . . . .  56
       10.1.  Properties . . . . . . . . . . . . . . . . . . . . . . .  56
         10.1.1.  PROP 4608: PROP_JAM_DETECT_ENABLE  . . . . . . . . .  56
         10.1.2.  PROP 4609: PROP_JAM_DETECTED . . . . . . . . . . . .  56
         10.1.3.  PROP 4610: PROP_JAM_DETECT_RSSI_THRESHOLD  . . . . .  57
         10.1.4.  PROP 4611: PROP_JAM_DETECT_WINDOW  . . . . . . . . .  57
         10.1.5.  PROP 4612: PROP_JAM_DETECT_BUSY  . . . . . . . . . .  57
         10.1.6.  PROP 4613: PROP_JAM_DETECT_HISTORY_BITMAP  . . . . .  58
     11. Feature: GPIO Access  . . . . . . . . . . . . . . . . . . . .  58
       11.1.  Properties . . . . . . . . . . . . . . . . . . . . . . .  58
         11.1.1.  PROP 4096: PROP_GPIO_CONFIG  . . . . . . . . . . . .  58
         11.1.2.  PROP 4098: PROP_GPIO_STATE . . . . . . . . . . . . .  59
         11.1.3.  PROP 4099: PROP_GPIO_STATE_SET . . . . . . . . . . .  60
         11.1.4.  PROP 4100: PROP_GPIO_STATE_CLEAR . . . . . . . . . .  60
     12. Feature: True Random Number Generation  . . . . . . . . . . .  61
       12.1.  Properties . . . . . . . . . . . . . . . . . . . . . . .  61
         12.1.1.  PROP 4101: PROP_TRNG_32  . . . . . . . . . . . . . .  61
         12.1.2.  PROP 4102: PROP_TRNG_128 . . . . . . . . . . . . . .  61
         12.1.3.  PROP 4103: PROP_TRNG_RAW_32  . . . . . . . . . . . .  62
     13. Security Considerations . . . . . . . . . . . . . . . . . . .  62
```

13.1. Raw Application Access . . . . . . . . . . . . . . . . . . . 62
14.1. URIs . . . . . . . . . . . . . . . . . . . . . . . . . . . . 63
Appendix A. Framing Protocol . . . . . . . . . . . . . . . . . . . 63
A.1. UART Recommendations . . . . . . . . . . . . . . . . . . . . 63
A.1.1. UART Bit Rate Detection . . . . . . . . . . . . . . . . . 64
A.1.2. HDLC-Lite . . . . . . . . . . . . . . . . . . . . . . . . 64
A.2. SPI Recommendations . . . . . . . . . . . . . . . . . . . . . 65
A.2.1. SPI Framing Protocol . . . . . . . . . . . . . . . . . . 66
A.3. I^2C Recommendations . . . . . . . . . . . . . . . . . . . . 67
A.4. Native USB Recommendations . . . . . . . . . . . . . . . . . 68
Appendix B. Test Vectors . . . . . . . . . . . . . . . . . . . . . 68
B.1. Test Vector: Packed Unsigned Integer . . . . . . . . . . . . 68
B.2. Test Vector: Reset Command . . . . . . . . . . . . . . . . . 68
B.3. Test Vector: Reset Notification . . . . . . . . . . . . . . . 68
B.4. Test Vector: Scan Beacon . . . . . . . . . . . . . . . . . . 69
B.5. Test Vector: Inbound IPv6 Packet . . . . . . . . . . . . . . 69
B.6. Test Vector: Outbound IPv6 Packet . . . . . . . . . . . . . . 69
B.7. Test Vector: Fetch list of on-mesh networks . . . . . . . . . 69
B.8. Test Vector: Returned list of on-mesh networks . . . . . . . 70
B.9. Test Vector: Adding an on-mesh network . . . . . . . . . . . 70
B.10. Test Vector: Insertion notification of an on-mesh network 71
B.11. Test Vector: Removing a local on-mesh network . . . . . . . 71
B.12. Test Vector: Removal notification of an on-mesh network . 71
Appendix C. Example Sessions . . . . . . . . . . . . . . . . . . . 72
C.1. NCP Initialization . . . . . . . . . . . . . . . . . . . . . 72
C.2. Attaching to a network . . . . . . . . . . . . . . . . . . . 72
C.3. Successfully joining a pre-existing network . . . . . . . . 73
C.4. Unsuccessfully joining a pre-existing network . . . . . . . 74
C.5. Detaching from a network . . . . . . . . . . . . . . . . . . 74
C.6. Attaching to a saved network . . . . . . . . . . . . . . . . 74
C.7. NCP Software Reset . . . . . . . . . . . . . . . . . . . . . 75
C.8. Adding an on-mesh prefix . . . . . . . . . . . . . . . . . . 75
C.9. Entering low-power modes . . . . . . . . . . . . . . . . . . 75
C.10. Sniffing raw packets . . . . . . . . . . . . . . . . . . . . 75
Appendix D. Glossary . . . . . . . . . . . . . . . . . . . . . . . 76
Appendix E. Acknowledgments . . . . . . . . . . . . . . . . . . . . 76
Authors' Addresses . . . . . . . . . . . . . . . . . . . . . . . . 78

## 1. Introduction

Spinel is host-controller protocol designed to enable interoperation over simple serial connections between general purpose device operating systems (OS) and network co-processors (NCP) for the purpose of controlling and managing their IPv6 network interfaces, achieving the following goals:

- Adopt a layered approach to the protocol design, allowing future support for other network protocols.

- Minimize the number of required commands/methods by providing a rich, property-based API.
- Support NCPs capable of being connected to more than one network at a time.
- Gracefully handle the addition of new features and capabilities without necessarily breaking backward compatibility.
- Be as minimal and light-weight as possible without unnecessarily sacrificing flexibility.

On top of this core framework, we define the properties and commands to enable various features and network protocols.

## 1.1. About this Draft

This document is currently in a draft status and is changing often. This section discusses some ideas for changes to the protocol that haven't yet been fully specified, as well as some of the impetus for the current design.

### 1.1.1. Scope

The eventual intent is to have two documents: A Spinel basis document which discusses the network-technology-agnostic mechanisms and a Thread(R) specialization document which describes all of the Thread(R)-specific implementation details. Currently, this document covers both.

### 1.1.2. Renumbering

Efforts are currently maintained to try to prevent overtly backward-incompatible changes to the existing protocol, but if you are implementing Spinel in your own products you should expect there to be at least one large renumbering event and major version number change before the standard is considered "baked". All changes will be clearly marked and documented to make such a transition as easy as possible.

To allow conclusive detection of protocol (in)compatibility between the host and the NCP, the following commands and properties are already considered to be "baked" and will not change:

- Command IDs zero through eight. (Reset, No-op, and Property-Value Commands)
- Property IDs zero through two. (Last status, Protocol Version, and NCP Version)

Renumbering would be undertaken in order to better organize the allocation of property IDs and capability IDs. One of the initial

goals of this protocol was for it to be possible for a host or NCP to only implement properties with values less than 127 and for the NCP to still be usable---relegating all larger property values for extra features or other capabilities that aren't strictly necessary.  This would allow simple implementations to avoid the need to implement support for PUIs (Section 3.2).

As time has gone by and the protocol has become more fleshed out, it has become clear that some of the initial allocations were inadequate and should be revisited if we want to try to achieve the original goal.

## 2.  Frame Format

A frame is defined simply as the concatenation of

- A header byte
- A command (up to three bytes, see Section 3.2 for format)
- An optional command payload

```
              +---------+--------+-----+-------------+
              | Octets: |   1    | 1-3 |      n      |
              +---------+--------+-----+-------------+
              | Fields: | HEADER | CMD | CMD_PAYLOAD |
              +---------+--------+-----+-------------+
```

### 2.1.  Header Format

The header byte is broken down as follows:

```
                  0   1   2   3   4   5   6   7
                +---+---+---+---+---+---+---+---+
                |  FLG  |  NLI  |      TID      |
                +---+---+---+---+---+---+---+---+
```

[CREF1]

#### 2.1.1.  FLG: Flag

The flag field of the header byte ("FLG") is always set to the value two (or "10" in binary).  Any frame received with these bits set to any other value else MUST NOT be considered a Spinel frame.

This convention allows Spinel to be line compatible with BTLE HCI. By defining the first two bit in this way we can disambiguate between Spinel frames and HCI frames (which always start with either "0x01" or "0x04") without any additional framing overhead.

### 2.1.2. NLI: Network Link Identifier

The Network Link Identifier (NLI) is a number between 0 and 3, which is associated by the OS with one of up to four IPv6 zone indices corresponding to conceptual IPv6 interfaces on the NCP. This allows the protocol to support IPv6 nodes connecting simultaneously to more than one IPv6 network link using a single NCP instance. The first Network Link Identifier (0) MUST refer to a distinguished conceptual interface provided by the NCP for its IPv6 link type. The other three Network Link Identifiers (1, 2 and 3) MAY be dissociated from any conceptual interface.

### 2.1.3. TID: Transaction Identifier

The least significant bits of the header represent the Transaction Identifier(TID). The TID is used for correlating responses to the commands which generated them.

When a command is sent from the host, any reply to that command sent by the NCP will use the same value for the TID. When the host receives a frame that matches the TID of the command it sent, it can easily recognize that frame as the actual response to that command.

The TID value of zero (0) is used for commands to which a correlated response is not expected or needed, such as for unsolicited update commands sent to the host from the NCP.

### 2.1.4. Command Identifier (CMD)

The command identifier is a 21-bit unsigned integer encoded in up to three bytes using the packed unsigned integer format described in Section 3.2. This encoding allows for up to 2,097,152 individual commands, with the first 127 commands represented as a single byte. Command identifiers larger than 2,097,151 are explicitly forbidden.

| CID Range | Description |
|---|---|
| 0 - 63 | Reserved for core commands |
| 64 - 15,359 | _UNALLOCATED_ |
| 15,360 - 16,383 | Vendor-specific |
| 16,384 - 1,999,999 | _UNALLOCATED_ |
| 2,000,000 - 2,097,151 | Experimental use only |

### 2.1.5. Command Payload (Optional)

Depending on the semantics of the command in question, a payload MAY be included in the frame. The exact composition and length of the payload is defined by the command identifier.

## 3. Data Packing

Data serialization for properties is performed using a light-weight data packing format which was loosely inspired by D-Bus. The format of a serialization is defined by a specially formatted string.

This packing format is used for notational convenience. While this string-based datatype format has been designed so that the strings may be directly used by a structured data parser, such a thing is not required to implement Spinel. Indeed, higly constrained applications may find such a thing to be too heavyweight.

Goals:

- Be lightweight and favor direct representation of values.
- Use an easily readable and memorable format string.
- Support lists and structures.
- Allow properties to be appended to structures while maintaining backward compatibility.

Each primitive datatype has an ASCII character associated with it. Structures can be represented as strings of these characters. For example:

- "C": A single unsigned byte.
- "C6U": A single unsigned byte, followed by a 128-bit IPv6 address, followed by a zero-terminated UTF8 string.
- "A(6)": An array of concatenated IPv6 addresses

In each case, the data is represented exactly as described. For example, an array of 10 IPv6 address is stored as 160 bytes.

### 3.1. Primitive Types

| Char | Name | Description |
|---|---|---|
| "." | DATATYPE_VOID | Empty data type. Used internally. |
| "b" | DATATYPE_BOOL | Boolean value. Encoded in 8-bits as either 0x00 or 0x01. All other values are illegal. |
| "C" | DATATYPE_UINT8 | Unsigned 8-bit integer. |
| "c" | DATATYPE_INT8 | Signed 8-bit integer. |
| "S" | DATATYPE_UINT16 | Unsigned 16-bit integer. |
| "s" | DATATYPE_INT16 | Signed 16-bit integer. |
| "L" | DATATYPE_UINT32 | Unsigned 32-bit integer. |
| "l" | DATATYPE_INT32 | Signed 32-bit integer. |
| "i" | DATATYPE_UINT_PACKED | Packed Unsigned Integer. See Section 3.2. |
| "6" | DATATYPE_IPv6ADDR | IPv6 Address. (Big-endian) |
| "E" | DATATYPE_EUI64 | EUI-64 Address. (Big-endian) |
| "e" | DATATYPE_EUI48 | EUI-48 Address. (Big-endian) |
| "D" | DATATYPE_DATA | Arbitrary data. See Section 3.3. |
| "d" | DATATYPE_DATA_WLEN | Arbitrary data with prepended length. See Section 3.3. |
| "U" | DATATYPE_UTF8 | Zero-terminated UTF8-encoded string. |
| "t(...)" | DATATYPE_STRUCT | Structured datatype with prepended length. See Section 3.4. |
| "A(...)" | DATATYPE_ARRAY | Array of datatypes. Compound type. See Section 3.5. |

All multi-byte values are little-endian unless explicitly stated otherwise.

### 3.2. Packed Unsigned Integer

For certain types of integers, such command or property identifiers, usually have a value on the wire that is less than 127. However, in order to not preclude the use of values larger than 255, we would need to add an extra byte. Doing this would add an extra byte to the majority of instances, which can add up in terms of bandwidth.

The packed unsigned integer format is based on the unsigned integer format in EXI [1], except that we limit the maximum value to the largest value that can be encoded into three bytes(2,097,151).

For all values less than 127, the packed form of the number is simply a single byte which directly represents the number. For values larger than 127, the following process is used to encode the value:

1. The unsigned integer is broken up into _n_ 7-bit chunks and placed into _n_ octets, leaving the most significant bit of each octet unused.
2. Order the octets from least-significant to most-significant. (Little-endian)
3. Clear the most significant bit of the most significant octet. Set the least significant bit on all other octets.

Where _n_ is the smallest number of 7-bit chunks you can use to represent the given value.

Take the value 1337, for example:

```
                           1337 => 0x0539
                                => [39 0A]
                                => [B9 0A]
```

To decode the value, you collect the 7-bit chunks until you find an octet with the most significant bit clear.

## 3.3. Data Blobs

There are two types for data blobs: "d" and "D".

- "d" has the length of the data (in bytes) prepended to the data (with the length encoded as type "S"). The size of the length field is not included in the length.
- "D" does not have a prepended length: the length of the data is implied by the bytes remaining to be parsed. It is an error for "D" to not be the last type in a type in a type signature.

This dichotomy allows for more efficient encoding by eliminating redundancy. If the rest of the buffer is a data blob, encoding the length would be redundant because we already know how many bytes are in the rest of the buffer.

In some cases we use "d" even if it is the last field in a type signature. We do this to allow for us to be able to append additional fields to the type signature if necessary in the future. This is usually the case with embedded structs, like in the scan results.

For example, let's say we have a buffer that is encoded with the datatype signature of "CLLD". In this case, it is pretty easy to

tell where the start and end of the data blob is: the start is 9 bytes from the start of the buffer, and its length is the length of the buffer minus 9. (9 is the number of bytes taken up by a byte and two longs)

The datatype signature "CLLDU" is illegal because we can't determine where the last field (a zero-terminated UTF8 string) starts. But the datatype "CLLdU" _is_ legal, because the parser can determine the exact length of the data blob-- allowing it to know where the start of the next field would be.

## 3.4. Structured Data

The structure data type ("t(...)") is a way of bundling together several fields into a single structure. It can be thought of as a "d" type except that instead of being opaque, the fields in the content are known. This is useful for things like scan results where you have substructures which are defined by different layers.

For example, consider the type signature "Lt(ES)t(6C)". In this hypothetical case, the first struct is defined by the MAC layer, and the second struct is defined by the PHY layer. Because of the use of structures, we know exactly what part comes from that layer. Additionally, we can add fields to each structure without introducing backward compatability problems: Data encoded as "Lt(ESU)t(6C)" (Notice the extra "U") will decode just fine as "Lt(ES)t(6C)". Additionally, if we don't care about the MAC layer and only care about the network layer, we could parse as "Lt()t(6C)".

Note that data encoded as "Lt(ES)t(6C)" will also parse as "Ldd", with the structures from both layers now being opaque data blobs.

## 3.5. Arrays

An array is simply a concatenated set of _n_ data encodings. For example, the type "A(6)" is simply a list of IPv6 addresses---one after the other. The type "A(6E)" likewise a concatenation of IPv6-address/EUI-64 pairs.

If an array contains many fields, the fields will often be surrounded by a structure ("t(...)"). This effectively prepends each item in the array with its length. This is useful for improving parsing performance or to allow additional fields to be added in the future in a backward compatible way. If there is a high certainty that additional fields will never be added, the struct may be omitted (saving two bytes per item).

This specification does not define a way to embed an array as a field alongside other fields.

# 4. Commands

## 4.1. CMD 0: (Host->NCP) CMD_NOOP

| | | |
|---|---|---|
| Octets: | 1 | 1 |
| Fields: | HEADER | CMD_NOOP |

No-Operation command. Induces the NCP to send a success status back to the host. This is primarily used for liveliness checks.

The command payload for this command SHOULD be empty. The receiver MUST ignore any non-empty command payload.

There is no error condition for this command.

## 4.2. CMD 1: (Host->NCP) CMD_RESET

| | | |
|---|---|---|
| Octets: | 1 | 1 |
| Fields: | HEADER | CMD_RESET |

Reset NCP command. Causes the NCP to perform a software reset. Due to the nature of this command, the TID is ignored. The host should instead wait for a "CMD_PROP_VALUE_IS" command from the NCP indicating "PROP_LAST_STATUS" has been set to "STATUS_RESET_SOFTWARE".

The command payload for this command SHOULD be empty. The receiver MUST ignore any non-empty command payload.

If an error occurs, the value of "PROP_LAST_STATUS" will be emitted instead with the value set to the generated status code for the error.

## 4.3. CMD 2: (Host->NCP) CMD_PROP_VALUE_GET

| | | | |
|---|---|---|---|
| Octets: | 1 | 1 | 1-3 |
| Fields: | HEADER | CMD_PROP_VALUE_GET | PROP_ID |

Get property value command. Causes the NCP to emit a "CMD_PROP_VALUE_IS" command for the given property identifier.

The payload for this command is the property identifier encoded in the packed unsigned integer format described in Section 3.2.

If an error occurs, the value of "PROP_LAST_STATUS" will be emitted instead with the value set to the generated status code for the error.

### 4.4. CMD 3: (Host->NCP) CMD_PROP_VALUE_SET

| | | | | |
|---|---|---|---|---|
| Octets: | 1 | 1 | 1-3 | n |
| Fields: | HEADER | CMD_PROP_VALUE_SET | PROP_ID | VALUE |

Set property value command. Instructs the NCP to set the given property to the specific given value, replacing any previous value.

The payload for this command is the property identifier encoded in the packed unsigned integer format described in Section 3.2, followed by the property value. The exact format of the property value is defined by the property.

If an error occurs, the value of "PROP_LAST_STATUS" will be emitted with the value set to the generated status code for the error.

### 4.5. CMD 4: (Host->NCP) CMD_PROP_VALUE_INSERT

| | | | | |
|---|---|---|---|---|
| Octets: | 1 | 1 | 1-3 | n |
| Fields: | HEADER | CMD_PROP_VALUE_INSERT | PROP_ID | VALUE |

Insert value into property command. Instructs the NCP to insert the given value into a list-oriented property, without removing other items in the list. The resulting order of items in the list is defined by the individual property being operated on.

The payload for this command is the property identifier encoded in the packed unsigned integer format described in Section 3.2, followed by the value to be inserted. The exact format of the value is defined by the property.

If the type signature of the property specified by "PROP_ID" consists of a single structure enclosed by an array ("A(t(...))"), then the contents of "VALUE" MUST contain the contents of the structure ("...") rather than the serialization of the whole item ("t(...)"). Specifically, the length of the structure MUST NOT be prepended to "VALUE". This helps to eliminate redundant data.

If an error occurs, the value of "PROP_LAST_STATUS" will be emitted with the value set to the generated status code for the error.

## 4.6. CMD 5: (Host->NCP) CMD_PROP_VALUE_REMOVE

| | | | | |
|---|---|---|---|---|
| Octets: | 1 | 1 | 1-3 | n |
| Fields: | HEADER | CMD_PROP_VALUE_REMOVE | PROP_ID | VALUE |

Remove value from property command. Instructs the NCP to remove the given value from a list-oriented property, without affecting other items in the list. The resulting order of items in the list is defined by the individual property being operated on.

Note that this command operates _by value_, not by index!

The payload for this command is the property identifier encoded in the packed unsigned integer format described in Section 3.2, followed by the value to be removed. The exact format of the value is defined by the property.

If the type signature of the property specified by "PROP_ID" consists of a single structure enclosed by an array ("A(t(...))"), then the contents of "VALUE" MUST contain the contents of the structure ("...") rather than the serialization of the whole item ("t(...)"). Specifically, the length of the structure MUST NOT be prepended to "VALUE". This helps to eliminate redundant data.

If an error occurs, the value of "PROP_LAST_STATUS" will be emitted with the value set to the generated status code for the error.

### 4.7. CMD 6: (NCP->Host) CMD_PROP_VALUE_IS

| Octets: | 1 | 1 | 1-3 | n |
|---|---|---|---|---|
| Fields: | HEADER | CMD_PROP_VALUE_IS | PROP_ID | VALUE |

Property value notification command. This command can be sent by the NCP in response to a previous command from the host, or it can be sent by the NCP in an unsolicited fashion to notify the host of various state changes asynchronously.

The payload for this command is the property identifier encoded in the packed unsigned integer format described in Section 3.2, followed by the current value of the given property.

### 4.8. CMD 7: (NCP->Host) CMD_PROP_VALUE_INSERTED

| Octets: | 1 | 1 | 1-3 | n |
|---|---|---|---|---|
| Fields: | HEADER | CMD_PROP_VALUE_INSERTED | PROP_ID | VALUE |

Property value insertion notification command. This command can be sent by the NCP in response to the "CMD_PROP_VALUE_INSERT" command, or it can be sent by the NCP in an unsolicited fashion to notify the host of various state changes asynchronously.

The payload for this command is the property identifier encoded in the packed unsigned integer format described in Section 3.2, followed by the value that was inserted into the given property.

If the type signature of the property specified by "PROP_ID" consists of a single structure enclosed by an array ("A(t(...))"), then the contents of "VALUE" MUST contain the contents of the structure ("...") rather than the serialization of the whole item ("t(...)"). Specifically, the length of the structure MUST NOT be prepended to "VALUE". This helps to eliminate redundant data.

The resulting order of items in the list is defined by the given property.

## 4.9. CMD 8: (NCP->Host) CMD_PROP_VALUE_REMOVED

| | | | | |
|---|---|---|---|---|
| Octets: | 1 | 1 | 1-3 | n |
| Fields: | HEADER | CMD_PROP_VALUE_REMOVED | PROP_ID | VALUE |

Property value removal notification command. This command can be sent by the NCP in response to the "CMD_PROP_VALUE_REMOVE" command, or it can be sent by the NCP in an unsolicited fashion to notify the host of various state changes asynchronously.

Note that this command operates _by value_, not by index!

The payload for this command is the property identifier encoded in the packed unsigned integer format described in Section 3.2, followed by the value that was removed from the given property.

If the type signature of the property specified by "PROP_ID" consists of a single structure enclosed by an array ("A(t(...))"), then the contents of "VALUE" MUST contain the contents of the structure ("...") rather than the serialization of the whole item ("t(...)"). Specifically, the length of the structure MUST NOT be prepended to "VALUE". This helps to eliminate redundant data.

The resulting order of items in the list is defined by the given property.

## 4.10. CMD 18: (Host->NCP) CMD_PEEK

| | | | | |
|---|---|---|---|---|
| Octets: | 1 | 1 | 4 | 2 |
| Fields: | HEADER | CMD_PEEK | ADDRESS | COUNT |

This command allows the NCP to fetch values from the RAM of the NCP for debugging purposes. Upon success, "CMD_PEEK_RET" is sent from the NCP to the host. Upon failure, "PROP_LAST_STATUS" is emitted with the appropriate error indication.

Due to the low-level nature of this command, certain error conditions may induce the NCP to reset.

The NCP MAY prevent certain regions of memory from being accessed.

The implementation of this command has security implications. See Section 13 for more information.

This command requires the capability "CAP_PEEK_POKE" to be present.

### 4.11. CMD 19: (NCP->Host) CMD_PEEK_RET

| | | | | | |
|---|---|---|---|---|---|
| Octets: | 1 | 1 | 4 | 2 | n |
| Fields: | HEADER | CMD_PEEK_RET | ADDRESS | COUNT | BYTES |

This command contains the contents of memory that was requested by a previous call to "CMD_PEEK".

This command requires the capability "CAP_PEEK_POKE" to be present.

### 4.12. CMD 20: (Host->NCP) CMD_POKE

| | | | | | |
|---|---|---|---|---|---|
| Octets: | 1 | 1 | 4 | 2 | n |
| Fields: | HEADER | CMD_POKE | ADDRESS | COUNT | BYTES |

This command writes the bytes to the specified memory address for debugging purposes.

Due to the low-level nature of this command, certain error conditions may induce the NCP to reset.

The implementation of this command has security implications. See Section 13 for more information.

This command requires the capability "CAP_PEEK_POKE" to be present.

### 4.13. CMD 21: (Host->NCP) CMD_PROP_VALUE_MULTI_GET

- Argument-Encoding: "A(i)"
- Required Capability: "CAP_CMD_MULTI"

Fetch the value of multiple properties in one command. Arguments are an array of property IDs. If all properties are fetched successfully, a "CMD_PROP_VALUES_ARE" command is sent back to the host containing the propertyid and value of each fetched property. The order of the results in "CMD_PROP_VALUES_ARE" match the order of properties given in "CMD_PROP_VALUE_GET".

Errors fetching individual properties are reflected as indicating a change to "PROP_LAST_STATUS" for that property's place.

Not all properties can be fetched using this method. As a general rule of thumb, any property that blocks when getting will fail for that individual property with "STATUS_INVALID_COMMAND_FOR_PROP".

### 4.14. CMD 22: (Host->NCP) CMD_PROP_VALUE_MULTI_SET

- o Argument-Encoding: "A(iD)"
- o Required Capability: "CAP_CMD_MULTI"

| | | | |
|---|---|---|---|
| Octets: | 1 | 1 | n |
| Fields: | HEADER | CMD_PROP_VALUE_MULTI_SET | Property/Value Pairs |

With each property/value pair being:

| | | | |
|---|---|---|---|
| Octets: | 2 | 1-3 | n |
| Fields: | LENGTH | PROP_ID | PROP_VALUE |

This command sets the value of several properties at once in the given order. The setting of properties stops at the first error, ignoring any later properties.

The result of this command is generally "CMD_PROP_VALUES_ARE" unless (for example) a parsing error has occured (in which case "CMD_PROP_VALUE_IS" for "PROP_LAST_STATUS" would be the result). The order of the results in "CMD_PROP_VALUES_ARE" match the order of properties given in "CMD_PROP_VALUE_MULTI_SET".

Since the processing of properties to set stops at the first error, the resulting "CMD_PROP_VALUES_ARE" can contain fewer items than the requested number of properties to set.

Not all properties can be set using this method. As a general rule of thumb, any property that blocks when setting will fail for that individual property with "STATUS_INVALID_COMMAND_FOR_PROP".

## 4.15. CMD 23: (NCP->Host) CMD_PROP_VALUES_ARE

- Argument-Encoding: "A(iD)"
- Required Capability: "CAP_CMD_MULTI"

| | | | |
|---|---|---|---|
| Octets: | 1 | 1 | n |
| Fields: | HEADER | CMD_PROP_VALUES_ARE | Property/Value Pairs |

With each property/value pair being:

| | | | |
|---|---|---|---|
| Octets: | 2 | 1-3 | n |
| Fields: | LENGTH | PROP_ID | PROP_VALUE |

This command is emitted by the NCP as the response to both the "CMD_PROP_VALUE_MULTI_GET" and "CMD_PROP_VALUE_MULTI_SET" commands. It is roughly analogous to "CMD_PROP_VALUE_IS", except that it contains more than one property.

This command SHOULD NOT be emitted asynchronously, or in response to any command other than "CMD_PROP_VALUE_MULTI_GET" or "CMD_PROP_VALUE_MULTI_SET".

The arguments are a list of structures containing the emitted property and the associated value. These are presented in the same order as given in the associated initiating command. In cases where getting or setting a specific property resulted in an error, the associated slot in this command will describe "PROP_LAST_STATUS".

## 5. Properties

Spinel is largely a property-based protocol, similar to representational state transfer (REST), with a property defined for every attribute that an OS needs to create, read, update or delete in the function of an IPv6 interface. The inspiration of this approach was memory-mapped hardware registers for peripherals. The goal is to avoid, as much as possible, the use of large complicated structures and/or method argument lists. The reason for avoiding these is because they have a tendency to change, especially early in development. Adding or removing a property from a structure can render the entire protocol incompatible. By using properties, you simply extend the protocol with an additional property.

Almost all features and capabilities are implemented using properties. Most new features that are initially proposed as commands can be adapted to be property-based instead. Notable exceptions include "Host Buffer Offload" (Section 9) and "Network Save" (Section 8).

In Spinel, properties are keyed by an unsigned integer between 0 and 2,097,151 (See Section 3.2).

## 5.1. Property Methods

Properties may support one or more of the following methods:

- "VALUE_GET" (Section 4.3)
- "VALUE_SET" (Section 4.4)
- "VALUE_INSERT" (Section 4.5)
- "VALUE_REMOVE" (Section 4.6)

Additionally, the NCP can send updates to the host (either synchronously or asynchronously) that inform the host about changes to specific properties:

- "VALUE_IS" (Section 4.7)
- "VALUE_INSERTED" (Section 4.8)
- "VALUE_REMOVED" (Section 4.9)

## 5.2. Property Types

Conceptually, there are three different types of properties:

- Single-value properties
- Multiple-value (Array) properties
- Stream properties

### 5.2.1. Single-Value Properties

Single-value properties are properties that have a simple representation of a single value. Examples would be:

- Current radio channel (Represented as a unsigned 8-bit integer)
- Network name (Represented as a UTF-8 encoded string)
- 802.15.4 PAN ID (Represented as a unsigned 16-bit integer)

The valid operations on these sorts of properties are "GET" and "SET".

### 5.2.2. Multiple-Value Properties

Multiple-Value Properties have more than one value associated with them. Examples would be:

- List of channels supported by the radio hardware.
- List of IPv6 addresses assigned to the interface.
- List of capabilities supported by the NCP.

The valid operations on these sorts of properties are "VALUE_GET", "VALUE_SET", "VALUE_INSERT", and "VALUE_REMOVE".

When the value is fetched using "VALUE_GET", the returned value is the concatenation of all of the individual values in the list. If the length of the value for an individual item in the list is not defined by the type then each item returned in the list is prepended with a length (See Section 3.5). The order of the returned items, unless explicitly defined for that specific property, is undefined.

"VALUE_SET" provides a way to completely replace all previous values. Calling "VALUE_SET" with an empty value effectively instructs the NCP to clear the value of that property.

"VALUE_INSERT" and "VALUE_REMOVE" provide mechanisms for the insertion or removal of individual items _by value_. The payload for these commands is a plain single value.

### 5.2.3. Stream Properties

Stream properties are special properties representing streams of data. Examples would be:

- Network packet stream (Section 5.6.3)
- Raw packet stream (Section 5.6.2)
- Debug message stream (Section 5.6.1)
- Network Beacon stream (Section 5.8.4)

All such properties emit changes asynchronously using the "VALUE_IS" command, sent from the NCP to the host. For example, as IPv6 traffic is received by the NCP, the IPv6 packets are sent to the host by way of asynchronous "VALUE_IS" notifications.

Some of these properties also support the host send data back to the NCP. For example, this is how the host sends IPv6 traffic to the NCP.

These types of properties generally do not support "VALUE_GET", as it is meaningless.

## 5.3. Property Numbering

While the majority of the properties that allow the configuration of network connectivity are network protocol specific, there are several properties that are required in all implementations.

Future property allocations SHALL be made from the following allocation plan:

| Property ID Range | Description |
|---|---|
| 0 - 127 | Reserved for frequently-used properties |
| 128 - 15,359 | Unallocated |
| 15,360 - 16,383 | Vendor-specific |
| 16,384 - 1,999,999 | Unallocated |
| 2,000,000 - 2,097,151 | Experimental use only |

For an explanation of the data format encoding shorthand used throughout this document, see Section 3.

## 5.4. Property Sections

The currently assigned properties are broken up into several sections, each with reserved ranges of property identifiers. These ranges are:

| Name | Range (Inclusive) | Documentation |
|---|---|---|
| Core | 0x00 - 0x1F, 0x1000 - 0x11FF | Section 5.5 |
| PHY | 0x20 - 0x2F, 0x1200 - 0x12FF | Section 5.7 |
| MAC | 0x30 - 0x3F, 0x1300 - 0x13FF | Section 5.8 |
| NET | 0x40 - 0x4F, 0x1400 - 0x14FF | Section 5.9 |
| Tech | 0x50 - 0x5F, 0x1500 - 0x15FF | Technology-specific |
| IPv6 | 0x60 - 0x6F, 0x1600 - 0x16FF | Section 5.10 |
| Stream | 0x70 - 0x7F, 0x1700 - 0x17FF | Section 5.5 |
| Debug | 0x4000 - 0x4400 | Section 5.11 |

Note that some of the property sections have two reserved ranges: a primary range (which is encoded as a single byte) and an extended range (which is encoded as two bytes). properties which are used more frequently are generally allocated from the former range.

## 5.5. Core Properties

### 5.5.1. PROP 0: PROP_LAST_STATUS

- Type: Read-Only
- Encoding: "i"

| | |
|---|---|
| Octets: | 1-3 |
| Fields: | LAST_STATUS |

Describes the status of the last operation. Encoded as a packed unsigned integer.

This property is emitted often to indicate the result status of pretty much any Host-to-NCP operation.

It is emitted automatically at NCP startup with a value indicating the reset reason.

See Section 6 for the complete list of status codes.

### 5.5.2. PROP 1: PROP_PROTOCOL_VERSION

- Type: Read-Only
- Encoding: "ii"

| | | |
|---|---|---|
| Octets: | 1-3 | 1-3 |
| Fields: | MAJOR_VERSION | MINOR_VERSION |

Describes the protocol version information. This property contains four fields, each encoded as a packed unsigned integer:

- Major Version Number
- Minor Version Number

This document describes major version 4, minor version 1 of this protocol.

### 5.5.2.1. Major Version Number

The major version number is used to identify large and incompatible differences between protocol versions.

The host MUST enter a FAULT state if it does not explicitly support the given major version number.

### 5.5.2.2. Minor Version Number

The minor version number is used to identify small but otherwise compatible differences between protocol versions. A mismatch between the advertised minor version number and the minor version that is supported by the host SHOULD NOT be fatal to the operation of the host.

### 5.5.3. PROP 2: PROP_NCP_VERSION

- Type: Read-Only
- Packed-Encoding: "U"

```
+---------+-------------------+
| Octets: |         n         |
+---------+-------------------+
| Fields: | NCP_VESION_STRING |
+---------+-------------------+
```

Contains a string which describes the firmware currently running on the NCP. Encoded as a zero-terminated UTF-8 string.

The format of the string is not strictly defined, but it is intended to present similarly to the "User-Agent" string from HTTP. The RECOMMENDED format of the string is as follows:

```
STACK-NAME/STACK-VERSION[BUILD_INFO][; OTHER_INFO]; BUILD_DATE_AND_TIME
```

Examples:

- "OpenThread/1.0d26-25-gb684c7f; DEBUG; May 9 2016 18:22:04"
- "ConnectIP/2.0b125 s1 ALPHA; Sept 24 2015 20:49:19"

### 5.5.4. PROP 3: PROP_INTERFACE_TYPE

- Type: Read-Only
- Encoding: "i"

| | |
|---|---|
| Octets: | 1-3 |
| Fields: | INTERFACE_TYPE |

This integer identifies what the network protocol for this NCP. Currently defined values are:

- 0: Bootloader
- 2: ZigBee IP(TM)
- 3: Thread(R)

The host MUST enter a FAULT state if it does not recognize the protocol given by the NCP.

### 5.5.5. PROP 4: PROP_INTERFACE_VENDOR_ID

- Type: Read-Only
- Encoding: "i"

| | |
|---|---|
| Octets: | 1-3 |
| Fields: | VENDOR_ID |

Vendor identifier.

### 5.5.6. PROP 5: PROP_CAPS

- Type: Read-Only
- Packed-Encoding: "A(i)"

| | | | |
|---|---|---|---|
| Octets: | 1-3 | 1-3 | ... |
| Fields: | CAP_1 | CAP_2 | ... |

Describes the supported capabilities of this NCP. Encoded as a list of packed unsigned integers.

A capability is defined as a 21-bit integer that describes a subset of functionality which is supported by the NCP.

Currently defined values are:

- 1: "CAP_LOCK"
- 2: "CAP_NET_SAVE"
- 3: "CAP_HBO": Host Buffer Offload. See Section 9.
- 4: "CAP_POWER_SAVE"
- 5: "CAP_COUNTERS"
- 6: "CAP_JAM_DETECT": Jamming detection. See Section 10
- 7: "CAP_PEEK_POKE": PEEK/POKE debugging commands.
- 8: "CAP_WRITABLE_RAW_STREAM": "PROP_STREAM_RAW" is writable.
- 9: "CAP_GPIO": Support for GPIO access. See Section 11.
- 10: "CAP_TRNG": Support for true random number generation. See Section 12.
- 11: "CAP_CMD_MULTI": Support for "CMD_PROP_VALUE_MULTI_GET" (Section 4.13), "CMD_PROP_VALUE_MULTI_SET" (Section 4.14, and "CMD_PROP_VALUES_ARE" (Section 4.15).
- 16: "CAP_802_15_4_2003"
- 17: "CAP_802_15_4_2006"
- 18: "CAP_802_15_4_2011"
- 21: "CAP_802_15_4_PIB"
- 24: "CAP_802_15_4_2450MHZ_OQPSK"
- 25: "CAP_802_15_4_915MHZ_OQPSK"
- 26: "CAP_802_15_4_868MHZ_OQPSK"
- 27: "CAP_802_15_4_915MHZ_BPSK"
- 28: "CAP_802_15_4_868MHZ_BPSK"
- 29: "CAP_802_15_4_915MHZ_ASK"
- 30: "CAP_802_15_4_868MHZ_ASK"
- 48: "CAP_ROLE_ROUTER"
- 49: "CAP_ROLE_SLEEPY"
- 52: "CAP_NET_THREAD_1_0"
- 512: "CAP_MAC_WHITELIST"
- 513: "CAP_MAC_RAW"
- 514: "CAP_OOB_STEERING_DATA"
- 1024: "CAP_THREAD_COMMISSIONER"
- 1025: "CAP_THREAD_BA_PROXY"

Additionally, future capability allocations SHALL be made from the following allocation plan:

| Capability Range | Description |
|---|---|
| 0 - 127 | Reserved for core capabilities |
| 128 - 15,359 | _UNALLOCATED_ |
| 15,360 - 16,383 | Vendor-specific |
| 16,384 - 1,999,999 | _UNALLOCATED_ |
| 2,000,000 - 2,097,151 | Experimental use only |

### 5.5.7. PROP 6: PROP_INTERFACE_COUNT

- Type: Read-Only
- Packed-Encoding: "C"

| | |
|---|---|
| Octets: | 1 |
| Fields: | "INTERFACE_COUNT" |

Describes the number of concurrent interfaces supported by this NCP. Since the concurrent interface mechanism is still TBD, this value MUST always be one.

This value is encoded as an unsigned 8-bit integer.

### 5.5.8. PROP 7: PROP_POWER_STATE

- Type: Read-Write
- Packed-Encoding: "C"

| | |
|---|---|
| Octets: | 1 |
| Fields: | POWER_STATE |

Describes the current power state of the NCP. By writing to this property you can manage the lower state of the NCP. Enumeration is encoded as a single unsigned byte.

Defined values are:

- 0: "POWER_STATE_OFFLINE": NCP is physically powered off. (Enumerated for completeness sake, not expected on the wire)
- 1: "POWER_STATE_DEEP_SLEEP": Almost everything on the NCP is shut down, but can still be resumed via a command or interrupt.
- 2: "POWER_STATE_STANDBY": NCP is in the lowest power state that can still be awoken by an event from the radio (e.g. waiting for alarm)
- 3: "POWER_STATE_LOW_POWER": NCP is responsive (and possibly connected), but using less power. (e.g. "Sleepy" child node)
- 4: "POWER_STATE_ONLINE": NCP is fully powered. (e.g. "Parent" node)

### 5.5.9. PROP 8: PROP_HWADDR

- Type: Read-Only*
- Packed-Encoding: "E"

| | |
|---|---|
| Octets: | 8 |
| Fields: | HWADDR |

The static EUI64 address of the device, used as a serial number. This value is read-only, but may be writable under certain vendor-defined circumstances.

### 5.5.10. PROP 9: PROP_LOCK

- Type: Read-Write
- Packed-Encoding: "b"

| | |
|---|---|
| Octets: | 1 |
| Fields: | LOCK |

Property lock. Used for grouping changes to several properties to take effect at once, or to temporarily prevent the automatic updating of property values. When this property is set, the execution of the NCP is effectively frozen until it is cleared.

This property is only supported if the "CAP_LOCK" capability is present.

Unlike most other properties, setting this property to true when the value of the property is already true MUST fail with a last status of "STATUS_ALREADY".

## 5.6. Stream Properties

### 5.6.1. PROP 112: PROP_STREAM_DEBUG

- Type: Read-Only-Stream
- Packed-Encoding: "D"

| | |
|---|---|
| Octets: | n |
| Fields: | UTF8_DATA |

This property is a streaming property, meaning that you cannot explicitly fetch the value of this property. The stream provides human-readable debugging output which may be displayed in the host logs.

The location of newline characters is not assumed by the host: it is the NCP's responsibility to insert newline characters where needed, just like with any other text stream.

To receive the debugging stream, you wait for "CMD_PROP_VALUE_IS" commands for this property from the NCP.

### 5.6.2. PROP 113: PROP_STREAM_RAW

- o Type: Read-Write-Stream
- o Packed-Encoding: "dD"

| | | | |
|---|---|---|---|
| Octets: | 2 | n | n |
| Fields: | FRAME_DATA_LEN | FRAME_DATA | FRAME_METADATA |

This stream provides the capability of sending and receiving raw packets to and from the radio. The exact format of the frame metadata and data is dependent on the MAC and PHY being used.

This property is a streaming property, meaning that you cannot explicitly fetch the value of this property. To receive traffic, you wait for "CMD_PROP_VALUE_IS" commands with this property id from the NCP.

Implementations may OPTIONALLY support the ability to transmit arbitrary raw packets. Support for this feature is indicated by the presence of the "CAP_WRITABLE_RAW_STREAM" capability.

If the capability "CAP_WRITABLE_RAW_STREAM" is set, then packets written to this stream with "CMD_PROP_VALUE_SET" will be sent out over the radio. This allows the caller to use the radio directly, with the stack being implemented on the host instead of the NCP.

#### 5.6.2.1. Frame Metadata Format

Any data past the end of "FRAME_DATA_LEN" is considered metadata and is OPTIONAL. Frame metadata MAY be empty or partially specified. Partially specified metadata MUST be accepted. Default values are used for all unspecified fields.

The same general format is used for "PROP_STREAM_RAW", "PROP_STREAM_NET", and "PROP_STREAM_NET_INSECURE". It can be used for frames sent from the NCP to the host as well as frames sent from the host to the NCP.

The frame metadata field consists of the following fields:

| Field | Description | Type | Len | Default |
|---|---|---|---|---|
| MD_POWER | (dBm) RSSI/TX-Power | "c" int8 | 1 | -128 |
| MD_NOISE | (dBm) Noise floor | "c" int8 | 1 | -128 |
| MD_FLAG | Flags (defined below) | "S" uint16 | 2 | |
| MD_PHY | PHY-specific data | "d" data | >=2 | |
| MD_VEND | Vendor-specific data | "d" data | >=2 | |

The following fields are ignored by the NCP for packets sent to it from the host:

- MD_NOISE
- MD_FLAG

When specifying "MD_POWER" for a packet to be transmitted, the actual transmit power is never larger than the current value of "PROP_PHY_TX_POWER" (Section 5.7.6). When left unspecified (or set to the value -128), an appropriate transmit power will be chosen by the NCP.

The bit values in "MD_FLAG" are defined as follows:

| Bit | Mask | Name | Description if set |
|---|---|---|---|
| 15 | 0x0001 | MD_FLAG_TX | Packet was transmitted, not received. |
| 13 | 0x0004 | MD_FLAG_BAD_FCS | Packet was received with bad FCS |
| 12 | 0x0008 | MD_FLAG_DUPE | Packet seems to be a duplicate |
| 0-11, 14 | 0xFFF2 | MD_FLAG_RESERVED | Flags reserved for future use. |

The format of "MD_PHY" is specified by the PHY layer currently in use, and may contain information such as the channel, LQI, antenna, or other pertainent information.

### 5.6.3. PROP 114: PROP_STREAM_NET

- Type: Read-Write-Stream
- Packed-Encoding: "dD"

| Octets: | 2 | n | n |
|---|---|---|---|
| Fields: | FRAME_DATA_LEN | FRAME_DATA | FRAME_METADATA |

This stream provides the capability of sending and receiving data packets to and from the currently attached network. The exact format of the frame metadata and data is dependent on the network protocol being used.

This property is a streaming property, meaning that you cannot explicitly fetch the value of this property. To receive traffic, you wait for "CMD_PROP_VALUE_IS" commands with this property id from the NCP.

To send network packets, you call "CMD_PROP_VALUE_SET" on this property with the value of the packet.

Any data past the end of "FRAME_DATA_LEN" is considered metadata, the format of which is described in Section 5.6.2.1.

### 5.6.4. PROP 114: PROP_STREAM_NET_INSECURE

- Type: Read-Write-Stream
- Packed-Encoding: "dD"

```
+---------+---------------+------------+----------------+
| Octets: |       2       |     n      |       n        |
+---------+---------------+------------+----------------+
| Fields: | FRAME_DATA_LEN | FRAME_DATA | FRAME_METADATA |
+---------+---------------+------------+----------------+
```

This stream provides the capability of sending and receiving unencrypted and unauthenticated data packets to and from nearby devices for the purposes of device commissioning. The exact format of the frame metadata and data is dependent on the network protocol being used.

This property is a streaming property, meaning that you cannot explicitly fetch the value of this property. To receive traffic, you wait for "CMD_PROP_VALUE_IS" commands with this property id from the NCP.

To send network packets, you call "CMD_PROP_VALUE_SET" on this property with the value of the packet.

Any data past the end of "FRAME_DATA_LEN" is considered metadata, the format of which is described in Section 5.6.2.1.

## 5.7. PHY Properties

### 5.7.1. PROP 32: PROP_PHY_ENABLED

- Type: Read-Write
- Packed-Encoding: "b" (bool8)

Set to 1 if the PHY is enabled, set to 0 otherwise. May be directly enabled to bypass higher-level packet processing in order to implement things like packet sniffers.

### 5.7.2. PROP 33: PROP_PHY_CHAN

- Type: Read-Write
- Packed-Encoding: "C" (uint8)

Value is the current channel. Must be set to one of the values contained in "PROP_PHY_CHAN_SUPPORTED".

### 5.7.3. PROP 34: PROP_PHY_CHAN_SUPPORTED

- Type: Read-Only
- Packed-Encoding: "A(C)" (array of uint8)
- Unit: List of channels

Value is a list of channel values that are supported by the hardware.

### 5.7.4. PROP 35: PROP_PHY_FREQ

- Type: Read-Only
- Packed-Encoding: "L" (uint32)
- Unit: Kilohertz

Value is the radio frequency (in kilohertz) of the current channel.

### 5.7.5. PROP 36: PROP_PHY_CCA_THRESHOLD

- Type: Read-Write
- Packed-Encoding: "c" (int8)
- Unit: dBm

Value is the CCA (clear-channel assessment) threshold. Set to -128 to disable.

When setting, the value will be rounded down to a value that is supported by the underlying radio hardware.

### 5.7.6. PROP 37: PROP_PHY_TX_POWER

- Type: Read-Write
- Packed-Encoding: "c" (int8)
- Unit: dBm

Value is the transmit power of the radio.

When setting, the value will be rounded down to a value that is supported by the underlying radio hardware.

### 5.7.7. PROP 38: PROP_PHY_RSSI

- Type: Read-Only
- Packed-Encoding: "c" (int8)
- Unit: dBm

Value is the current RSSI (Received signal strength indication) from the radio. This value can be used in energy scans and for determining the ambient noise floor for the operating environment.

### 5.7.8. PROP 39: PROP_PHY_RX_SENSITIVITY

- Type: Read-Only
- Packed-Encoding: "c" (int8)
- Unit: dBm

Value is the radio receive sensitivity. This value can be used as lower bound noise floor for link metrics computation.

## 5.8. MAC Properties

### 5.8.1. PROP 48: PROP_MAC_SCAN_STATE

- Type: Read-Write
- Packed-Encoding: "C"
- Unit: Enumeration

Possible Values:

- 0: "SCAN_STATE_IDLE"
- 1: "SCAN_STATE_BEACON"
- 2: "SCAN_STATE_ENERGY"
- 3: "SCAN_STATE_DISCOVER"

Set to "SCAN_STATE_BEACON" to start an active scan. Beacons will be emitted from "PROP_MAC_SCAN_BEACON".

Set to "SCAN_STATE_ENERGY" to start an energy scan. Channel energy result will be reported by emissions of "PROP_MAC_ENERGY_SCAN_RESULT" (per channel).

Set to "SCAN_STATE_DISOVER" to start a Thread MLE discovery scan operation. Discovery scan result will be emitted from "PROP_MAC_SCAN_BEACON".

Value switches to "SCAN_STATE_IDLE" when scan is complete.

### 5.8.2. PROP 49: PROP_MAC_SCAN_MASK

- Type: Read-Write
- Packed-Encoding: "A(C)"
- Unit: List of channels to scan

### 5.8.3. PROP 50: PROP_MAC_SCAN_PERIOD

- Type: Read-Write
- Packed-Encoding: "S" (uint16)
- Unit: milliseconds per channel

### 5.8.4. PROP 51: PROP_MAC_SCAN_BEACON

- Type: Read-Only-Stream
- Packed-Encoding: "Ccdd" (or "Cct(ESSc)t(iCUdd)")

| | | | | | | |
|---|---|---|---|---|---|---|
| Octets: | 1 | 1 | 2 | n | 2 | n |
| Fields: | CH | RSSI | MAC_LEN | MAC_DATA | NET_LEN | NET_DATA |

Scan beacons have two embedded structures which contain information about the MAC layer and the NET layer. Their format depends on the MAC and NET layer currently in use. The format below is for an 802.15.4 MAC with Thread:

- "C": Channel
- "c": RSSI of the beacon
- "t": MAC layer properties (802.15.4 layer shown below for convenience)

    - "E": Long address
    - "S": Short address
    - "S": PAN-ID
    - "c": LQI
- NET layer properties (Standard net layer shown below for convenience)

    - "i": Protocol Number
    - "C": Flags
    - "U": Network Name
    - "d": XPANID
    - "d": Steering data

Extra parameters may be added to each of the structures in the future, so care should be taken to read the length that prepends each structure.

### 5.8.5. PROP 52: PROP_MAC_15_4_LADDR

- Type: Read-Write
- Packed-Encoding: "E"

The 802.15.4 long address of this node.

This property is only present on NCPs which implement 802.15.4

### 5.8.6. PROP 53: PROP_MAC_15_4_SADDR

- Type: Read-Write
- Packed-Encoding: "S"

The 802.15.4 short address of this node.

This property is only present on NCPs which implement 802.15.4

### 5.8.7. PROP 54: PROP_MAC_15_4_PANID

- Type: Read-Write
- Packed-Encoding: "S"

The 802.15.4 PANID this node is associated with.

This property is only present on NCPs which implement 802.15.4

### 5.8.8. PROP 55: PROP_MAC_RAW_STREAM_ENABLED

- Type: Read-Write
- Packed-Encoding: "b"

Set to true to enable raw MAC frames to be emitted from "PROP_STREAM_RAW". See Section 5.6.2.

### 5.8.9. PROP 56: PROP_MAC_PROMISCUOUS_MODE

- Type: Read-Write
- Packed-Encoding: "C"

Possible Values:

| Id | Name | Description |
|---|---|---|
| 0 | "MAC_PROMISCUOUS_MODE_OFF" | Normal MAC filtering is in place. |
| 1 | "MAC_PROMISCUOUS_MODE_NETWORK" | All MAC packets matching network are passed up the stack. |
| 2 | "MAC_PROMISCUOUS_MODE_FULL" | All decoded MAC packets are passed up the stack. |

See Section 5.6.2.

### 5.8.10. PROP 57: PROP_MAC_ENERGY_SCAN_RESULT

- Type: Read-Only-Stream
- Packed-Encoding: "Cc"

This property is emitted during energy scan operation per scanned channel with following format:

- "C": Channel
- "c": RSSI (in dBm)

### 5.8.11. PROP 4864: PROP_MAC_WHITELIST

- Type: Read-Write
- Packed-Encoding: "A(T(Ec))"
- OPTIONAL

Structure Parameters:

- "E": EUI64 address of node
- "c": Optional RSSI-override value. The value 127 indicates that the RSSI-override feature is not enabled for this address. If this value is omitted when setting or inserting, it is assumed to be 127. This parameter is ignored when removing.

### 5.8.12. PROP 4865: PROP_MAC_WHITELIST_ENABLED

- Type: Read-Write
- Packed-Encoding: "b"

## 5.9. NET Properties

### 5.9.1. PROP 64: PROP_NET_SAVED

- Type: Read-Only
- Packed-Encoding: "b"

Returns true if there is a network state stored/saved.

### 5.9.2. PROP 65: PROP_NET_IF_UP

- Type: Read-Write
- Packed-Encoding: "b"

Network interface up/down status. Non-zero (set to 1) indicates up, zero indicates down.

### 5.9.3. PROP 66: PROP_NET_STACK_UP

- Type: Read-Write
- Packed-Encoding: "b"
- Unit: Enumeration

Thread stack operational status. Non-zero (set to 1) indicates up, zero indicates down.

### 5.9.4. PROP 67: PROP_NET_ROLE

- Type: Read-Write
- Packed-Encoding: "C"
- Unit: Enumeration

Values:

- 0: "NET_ROLE_DETACHED"
- 1: "NET_ROLE_CHILD"
- 2: "NET_ROLE_ROUTER"
- 3: "NET_ROLE_LEADER"

### 5.9.5. PROP 68: PROP_NET_NETWORK_NAME

- Type: Read-Write
- Packed-Encoding: "U"

### 5.9.6. PROP 69: PROP_NET_XPANID

- Type: Read-Write
- Packed-Encoding: "D"

### 5.9.7. PROP 70: PROP_NET_MASTER_KEY

- Type: Read-Write
- Packed-Encoding: "D"

### 5.9.8. PROP 71: PROP_NET_KEY_SEQUENCE_COUNTER

- Type: Read-Write
- Packed-Encoding: "L"

### 5.9.9. PROP 72: PROP_NET_PARTITION_ID

- Type: Read-Write
- Packed-Encoding: "L"

The partition ID of the partition that this node is a member of.

### 5.9.10. PROP 73: PROP_NET_REQUIRE_JOIN_EXISTING

- Type: Read-Write
- Packed-Encoding: "b"

### 5.9.11. PROP 74: PROP_NET_KEY_SWITCH_GUARDTIME

- Type: Read-Write
- Packed-Encoding: "L"

### 5.9.12. PROP 75: PROP_NET_PSKC

- Type: Read-Write
- Packed-Encoding: "D"

## 5.10. IPv6 Properties

### 5.10.1. PROP 96: PROP_IPV6_LL_ADDR

- Type: Read-Only
- Packed-Encoding: "6"

IPv6 Address

### 5.10.2. PROP 97: PROP_IPV6_ML_ADDR

- Type: Read-Only
- Packed-Encoding: "6"

IPv6 Address + Prefix Length

### 5.10.3. PROP 98: PROP_IPV6_ML_PREFIX

- Type: Read-Write
- Packed-Encoding: "6C"

IPv6 Prefix + Prefix Length

### 5.10.4. PROP 99: PROP_IPV6_ADDRESS_TABLE

- Type: Read-Write
- Packed-Encoding: "A(t(6CLLC))"

Array of structures containing:

- "6": IPv6 Address
- "C": Network Prefix Length
- "L": Valid Lifetime

- "L": Preferred Lifetime
- "C": Flags

### 5.10.5. PROP 101: PROP_IPv6_ICMP_PING_OFFLOAD

- Type: Read-Write
- Packed-Encoding: "b"

Allow the NCP to directly respond to ICMP ping requests. If this is turned on, ping request ICMP packets will not be passed to the host.

Default value is "false".

## 5.11. Debug Properties

### 5.11.1. PROP 16384: PROP_DEBUG_TEST_ASSERT

- Type: Read-Only
- Packed-Encoding: "b"

Reading this property will cause an assert on the NCP. This is intended for testing the assert functionality of underlying platform/NCP. Assert should ideally cause the NCP to reset, but if "assert" is not supported or disabled boolean value of "false" is returned in response.

### 5.11.2. PROP 16385: PROP_DEBUG_NCP_LOG_LEVEL

- Type: Read-Write
- Packed-Encoding: "C"

Provides access to the NCP log level. Currently defined values are (which follows the RFC 5424):

- 0: Emergency (emerg).
- 1: Alert (alert).
- 2: Critical (crit).
- 3: Error (err).
- 4: Warning (warn).
- 5: Notice (notice).
- 6: Information (info).
- 7: Debug (debug).

If the NCP supports dynamic log level control, setting this property changes the log level accordingly. Getting the value returns the current log level. If the dynamic log level control is not supported, setting this property returns a "PROP_LAST_STATUS" with "STATUS_INVALID_COMMAND_FOR_PROP".

## 6. Status Codes

Status codes are sent from the NCP to the host via "PROP_LAST_STATUS" using the "CMD_VALUE_IS" command to indicate the return status of a previous command. As with any response, the TID field of the FLAG byte is used to correlate the response with the request.

Note that most successfully executed commands do not indicate a last status of "STATUS_OK". The usual way the NCP indicates a successful command is to mirror the property change back to the host. For example, if you do a "CMD_VALUE_SET" on "PROP_PHY_ENABLED", the NCP would indicate success by responding with a "CMD_VALUE_IS" for "PROP_PHY_ENABLED". If the command failed, "PROP_LAST_STATUS" would be emitted instead.

See Section 5.5.1 for more information on "PROP_LAST_STATUS".

- 0: "STATUS_OK": Operation has completed successfully.
- 1: "STATUS_FAILURE": Operation has failed for some undefined reason.
- 2: "STATUS_UNIMPLEMENTED": The given operation has not been implemented.
- 3: "STATUS_INVALID_ARGUMENT": An argument to the given operation is invalid.
- 4: "STATUS_INVALID_STATE" : The given operation is invalid for the current state of the device.
- 5: "STATUS_INVALID_COMMAND": The given command is not recognized.
- 6: "STATUS_INVALID_INTERFACE": The given Spinel interface is not supported.
- 7: "STATUS_INTERNAL_ERROR": An internal runtime error has occurred.
- 8: "STATUS_SECURITY_ERROR": A security or authentication error has occurred.
- 9: "STATUS_PARSE_ERROR": An error has occurred while parsing the command.
- 10: "STATUS_IN_PROGRESS": The operation is in progress and will be completed asynchronously.
- 11: "STATUS_NOMEM": The operation has been prevented due to memory pressure.
- 12: "STATUS_BUSY": The device is currently performing a mutually exclusive operation.
- 13: "STATUS_PROP_NOT_FOUND": The given property is not recognized.
- 14: "STATUS_PACKET_DROPPED": The packet was dropped.
- 15: "STATUS_EMPTY": The result of the operation is empty.
- 16: "STATUS_CMD_TOO_BIG": The command was too large to fit in the internal buffer.
- 17: "STATUS_NO_ACK": The packet was not acknowledged.

- 18: "STATUS_CCA_FAILURE": The packet was not sent due to a CCA failure.
- 19: "STATUS_ALREADY": The operation is already in progress or the property was already set to the given value.
- 20: "STATUS_ITEM_NOT_FOUND": The given item could not be found in the property.
- 21: "STATUS_INVALID_COMMAND_FOR_PROP": The given command cannot be performed on this property.
- 22-111: RESERVED
- 112-127: Reset Causes
  - 112: "STATUS_RESET_POWER_ON"
  - 113: "STATUS_RESET_EXTERNAL"
  - 114: "STATUS_RESET_SOFTWARE"
  - 115: "STATUS_RESET_FAULT"
  - 116: "STATUS_RESET_CRASH"
  - 117: "STATUS_RESET_ASSERT"
  - 118: "STATUS_RESET_OTHER"
  - 119: "STATUS_RESET_UNKNOWN"
  - 120: "STATUS_RESET_WATCHDOG"
  - 121-127: RESERVED-RESET-CODES
- 128 - 15,359: UNALLOCATED
- 15,360 - 16,383: Vendor-specific
- 16,384 - 1,999,999: UNALLOCATED
- 2,000,000 - 2,097,151: Experimental Use Only (MUST NEVER be used in production!)

## 7. Technology: Thread(R)

This section describes all of the properties and semantics required for managing a Thread(R) NCP.

Thread(R) NCPs have the following requirements:

- The property "PROP_INTERFACE_TYPE" must be 3.
- The non-optional properties in the following sections MUST be implemented: CORE, PHY, MAC, NET, and IPV6.

All serious implementations of an NCP SHOULD also support the network save feature (See Section 8).

### 7.1. Capabilities

The Thread(R) technology defines the following capabilities:

- "CAP_NET_THREAD_1_0" - Indicates that the NCP implements v1.0 of the Thread(R) standard.

- "CAP_NET_THREAD_1_1" - Indicates that the NCP implements v1.1 of the Thread(R) standard.

## 7.2. Properties

Properties for Thread(R) are allocated out of the "Tech" property section (see Section 5.4).

### 7.2.1. PROP 80: PROP_THREAD_LEADER_ADDR

- Type: Read-Only
- Packed-Encoding: "6"

The IPv6 address of the leader. (Note: May change to long and short address of leader)

### 7.2.2. PROP 81: PROP_THREAD_PARENT

- Type: Read-Only
- Packed-Encoding: "ES"
- LADDR, SADDR

The long address and short address of the parent of this node.

### 7.2.3. PROP 82: PROP_THREAD_CHILD_TABLE

- Type: Read-Only
- Packed-Encoding: "A(t(ES))"

Table containing the long and short addresses of all the children of this node.

### 7.2.4. PROP 83: PROP_THREAD_LEADER_RID

- Type: Read-Only
- Packed-Encoding: "C"

The router-id of the current leader.

### 7.2.5. PROP 84: PROP_THREAD_LEADER_WEIGHT

- Type: Read-Only
- Packed-Encoding: "C"

The leader weight of the current leader.

### 7.2.6. PROP 85: PROP_THREAD_LOCAL_LEADER_WEIGHT

- Type: Read-Write
- Packed-Encoding: "C"

The leader weight for this node.

### 7.2.7. PROP 86: PROP_THREAD_NETWORK_DATA

- Type: Read-Only
- Packed-Encoding: "D"

The local network data.

### 7.2.8. PROP 87: PROP_THREAD_NETWORK_DATA_VERSION

- Type: Read-Only
- Packed-Encoding: "S"

### 7.2.9. PROP 88: PROP_THREAD_STABLE_NETWORK_DATA

- Type: Read-Only
- Packed-Encoding: "D"

The local stable network data.

### 7.2.10. PROP 89: PROP_THREAD_STABLE_NETWORK_DATA_VERSION

- Type: Read-Only
- Packed-Encoding: "S"

### 7.2.11. PROP 90: PROP_THREAD_ON_MESH_NETS

- Type: Read-Write
- Packed-Encoding: "A(t(6CbCb))"

Data per item is:

- "6": IPv6 Prefix
- "C": Prefix length, in bits
- "b": Stable flag
- "C": TLV flags
- "b": "Is defined locally" flag. Set if this network was locally defined. Assumed to be true for set, insert and replace. Clear if the on mesh network was defined by another node.

### 7.2.12. PROP 91: PROP_THREAD_LOCAL_ROUTES

- Type: Read-Write
- Packed-Encoding: "A(t(6CbC))"

Data per item is:

- "6": IPv6 Prefix
- "C": Prefix length, in bits
- "b": Stable flag
- "C": Other flags

### 7.2.13. PROP 92: PROP_THREAD_ASSISTING_PORTS

- Type: Read-Write
- Packed-Encoding: "A(S)"

### 7.2.14. PROP 93: PROP_THREAD_ALLOW_LOCAL_NET_DATA_CHANGE

- Type: Read-Write
- Packed-Encoding: "b"

Set to true before changing local net data. Set to false when finished. This allows changes to be aggregated into single events.

### 7.2.15. PROP 94: PROP_THREAD_MODE

- Type: Read-Write
- Packed-Encoding: "C"

This property contains the value of the mode TLV for this node. The meaning of the bits in this bitfield are defined by section 4.5.2 of the Thread(R) specification.

### 7.2.16. PROP 5376: PROP_THREAD_CHILD_TIMEOUT

- Type: Read-Write
- Packed-Encoding: "L"

Used when operating in the Child role.

### 7.2.17. PROP 5377: PROP_THREAD_RLOC16

- Type: Read-Write
- Packed-Encoding: "S"

### 7.2.18. PROP 5378: PROP_THREAD_ROUTER_UPGRADE_THRESHOLD

- Type: Read-Write
- Packed-Encoding: "C"

### 7.2.19. PROP 5379: PROP_THREAD_CONTEXT_REUSE_DELAY

- Type: Read-Write
- Packed-Encoding: "L"

### 7.2.20. PROP 5380: PROP_THREAD_NETWORK_ID_TIMEOUT

- Type: Read-Write
- Packed-Encoding: "C"

Allows you to get or set the Thread(R) "NETWORK_ID_TIMEOUT" constant, as defined by the Thread(R) specification.

### 7.2.21. PROP 5381: PROP_THREAD_ACTIVE_ROUTER_IDS

- Type: Read-Write/Write-Only
- Packed-Encoding: "A(C)" (List of active thread router ids)

Note that some implementations may not support "CMD_GET_VALUE" router ids, but may support "CMD_REMOVE_VALUE" when the node is a leader.

### 7.2.22. PROP 5382: PROP_THREAD_RLOC16_DEBUG_PASSTHRU

- Type: Read-Write
- Packed-Encoding: "b"

Allow the HOST to directly observe all IPv6 packets received by the NCP, including ones sent to the RLOC16 address.

Default value is "false".

### 7.2.23. PROP 5383: PROP_THREAD_ROUTER_ROLE_ENABLED

- Type: Read-Write
- Packed-Encoding: "b"

Allow the HOST to indicate whether or not the router role is enabled. If current role is a router, setting this property to "false" starts a re-attach process as an end-device.

### 7.2.24. PROP 5384: PROP_THREAD_ROUTER_DOWNGRADE_THRESHOLD

- Type: Read-Write
- Packed-Encoding: "C"

### 7.2.25. PROP 5385: PROP_THREAD_ROUTER_SELECTION_JITTER

- Type: Read-Write
- Packed-Encoding: "C"

Specifies the self imposed random delay in seconds a REED waits before registering to become an Active Router.

### 7.2.26. PROP 5386: PROP_THREAD_PREFERRED_ROUTER_ID

- Type: Write-Only
- Packed-Encoding: "C"

Specifies the preferred Router Id. Upon becoming a router/leader the node attempts to use this Router Id. If the preferred Router Id is not set or if it can not be used, a randomly generated router id is picked. This property can be set only when the device role is either detached or disabled.

### 7.2.27. PROP 5387: PROP_THREAD_NEIGHBOR_TABLE

- Type: Read-Only
- Packed-Encoding: "A(t(ESLCcCbLL))"

Data per item is:

- "E": Extended/long address
- "S": RLOC16
- "L": Age
- "C": Link Quality In
- "c": Average RSS
- "C": Mode (bit-flags)
- "b": "true" if neighbor is a child, "false" otherwise.
- "L": Link Frame Counter
- "L": MLE Frame Counter

### 7.2.28. PROP 5388: PROP_THREAD_CHILD_COUNT_MAX

- Type: Read-Write
- Packed-Encoding: "C"

Specifies the maximum number of children currently allowed. This parameter can only be set when Thread(R) protocol operation has been stopped.

### 7.2.29. PROP 5389: PROP_THREAD_LEADER_NETWORK_DATA

- Type: Read-Only
- Packed-Encoding: "D"

The leader network data.

### 7.2.30. PROP 5390: PROP_THREAD_STABLE_LEADER_NETWORK_DATA

- Type: Read-Only
- Packed-Encoding: "D"

The stable leader network data.

### 7.2.31. PROP 5391: PROP_THREAD_JOINERS

- Type: Insert/Remove Only (optionally Read-Write)
- Packed-Encoding: "A(t(ULE))"
- Required capability: "CAP_THREAD_COMMISSIONER"

Data per item is:

- "U": PSKd
- "L": Timeout in seconds
- "E": Extended/long address (optional)

Passess Pre-Shared Key for the Device to the NCP in the commissioning process. When the Extended address is ommited all Devices which provided a valid PSKd are allowed to join the Thread(R) Network.

### 7.2.32. PROP 5392: PROP_THREAD_COMMISSIONER_ENABLED

- Type: Write only (optionally Read-Write)
- Packed-Encoding: "b"
- Required capability: "CAP_THREAD_COMMISSIONER"

Set to true to enable the native commissioner. It is mandatory before adding the joiner to the network.

### 7.2.33. PROP 5393: PROP_THREAD_BA_PROXY_ENABLED

- Type: Read-Write
- Packed-Encoding: "b"
- Required capability: "CAP_THREAD_BA_PROXY"

Set to true to enable the border agent proxy.

### 7.2.34. PROP 5394: PROP_THREAD_BA_PROXY_STREAM

- Type: Read-Write-Stream
- Packed-Encoding: "dSS"
- Required capability: "CAP_THREAD_BA_PROXY"

Data per item is:

- "d": CoAP frame
- "S": source/destination RLOC/ALOC
- "S": source/destination port

| | | | | |
|---|---|---|---|---|
| Octects: | 2 | n | 2 | 2 |
| Fields: | Length | CoAP | locator | port |

This property allows the host to send and receive border-agent-related CoAP requests/responses from the NCP's RLOC address. This allows the host driver to implement a Thread(R) border agent.

### 7.2.35. PROP 5395: PROP_THREAD_DISOVERY_SCAN_JOINER_FLAG

- Type: Read-Write
- Packed-Encoding:: "b"

This property specifies the value used in Thread(R) MLE Discovery Request TLV during discovery scan operation. Default value is "false".

### 7.2.36. PROP 5396: PROP_THREAD_DISCOVERY_SCAN_ENABLE_FILTERING

- Type: Read-Write
- Packed-Encoding:: "b"

This property is used to enable/disable EUI64 filtering during discovery scan operation. Default value is "false".

### 7.2.37. PROP 5397: PROP_THREAD_DISCOVERY_SCAN_PANID

- Type: Read-write
- Packed-Encoding:: "S"

This property specifies the PANID used for filtering during discovery scan operation. Default value is "0xffff" (broadcast PANID) which disables PANID filtering.

#### 7.2.38. PROP 5398: PROP_THREAD_STEERING_DATA

- Type: Write-Only
- Packed-Encoding: "E"
- Required capability: "CAP_OOB_STEERING_DATA"

This property can be used to set the steering data for MLE Discovery Response messages.

- All zeros to clear the steering data (indicating no steering data).
- All 0xFFs to set the steering data (bloom filter) to accept/allow all.
- A specific EUI64 which is then added to steering data/bloom filter.

## 8. Feature: Network Save

The network save/recall feature is an optional NCP capability that, when present, allows the host to save and recall network credentials and state to and from nonvolatile storage.

The presence of the save/recall feature can be detected by checking for the presence of the "CAP_NET_SAVE" capability in "PROP_CAPS".

Network clear feature allows host to erase all network credentials and state from non-volatile memory.

### 8.1. Commands

#### 8.1.1. CMD 9: (Host->NCP) CMD_NET_SAVE

| | | |
|---|---|---|
| Octets: | 1 | 1 |
| Fields: | HEADER | CMD_NET_SAVE |

Save network state command. Saves any current network credentials and state necessary to reconnect to the current network to non-volatile memory.

This operation affects non-volatile memory only. The current network information stored in volatile memory is unaffected.

The response to this command is always a "CMD_PROP_VALUE_IS" for "PROP_LAST_STATUS", indicating the result of the operation.

This command is only available if the "CAP_NET_SAVE" capability is set.

### 8.1.2. CMD 10: (Host->NCP) CMD_NET_CLEAR

| | | |
|---|---|---|
| Octets: | 1 | 1 |
| Fields: | HEADER | CMD_NET_CLEAR |

Clear saved network settings command. Erases all network credentials and state from non-volatile memory. The erased settings include any data saved automatically by the network stack firmware and/or data saved by "CMD_NET_SAVE" operation.

This operation affects non-volatile memory only. The current network information stored in volatile memory is unaffected.

The response to this command is always a "CMD_PROP_VALUE_IS" for "PROP_LAST_STATUS", indicating the result of the operation.

This command is always available independent of the value of "CAP_NET_SAVE" capability.

### 8.1.3. CMD 11: (Host->NCP) CMD_NET_RECALL

| | | |
|---|---|---|
| Octets: | 1 | 1 |
| Fields: | HEADER | CMD_NET_RECALL |

Recall saved network state command. Recalls any previously saved network credentials and state previously stored by "CMD_NET_SAVE" from non-volatile memory.

This command will typically generated several unsolicited property updates as the network state is loaded. At the conclusion of loading, the authoritative response to this command is always a "CMD_PROP_VALUE_IS" for "PROP_LAST_STATUS", indicating the result of the operation.

This command is only available if the "CAP_NET_SAVE" capability is set.

## 9. Feature: Host Buffer Offload

The memory on an NCP may be much more limited than the memory on the host processor. In such situations, it is sometimes useful for the NCP to offload buffers to the host processor temporarily so that it can perform other operations.

Host buffer offload is an optional NCP capability that, when present, allows the NCP to store data buffers on the host processor that can be recalled at a later time.

The presence of this feature can be detected by the host by checking for the presence of the "CAP_HBO" capability in "PROP_CAPS".

### 9.1. Commands

#### 9.1.1. CMD 12: (NCP->Host) CMD_HBO_OFFLOAD

- Argument-Encoding: "LscD"
    - "OffloadId": 32-bit unique block identifier
    - "Expiration": In seconds-from-now
    - "Priority": Critical, High, Medium, Low
    - "Data": Data to offload

#### 9.1.2. CMD 13: (NCP->Host) CMD_HBO_RECLAIM

- Argument-Encoding: "Lb"
    - "OffloadId": 32-bit unique block identifier
    - "KeepAfterReclaim": If not set to true, the block will be dropped by the host after it is sent to the NCP.

#### 9.1.3. CMD 14: (NCP->Host) CMD_HBO_DROP

- Argument-Encoding: "L"
    - "OffloadId": 32-bit unique block identifier

#### 9.1.4. CMD 15: (Host->NCP) CMD_HBO_OFFLOADED

- Argument-Encoding: "Li"
    - "OffloadId": 32-bit unique block identifier
    - "Status": Status code for the result of the operation.

### 9.1.5. CMD 16: (Host->NCP) CMD_HBO_RECLAIMED

- Argument-Encoding: "LiD"
    - "OffloadId": 32-bit unique block identifier
    - "Status": Status code for the result of the operation.
    - "Data": Data that was previously offloaded (if any)

### 9.1.6. CMD 17: (Host->NCP) CMD_HBO_DROPPED

- Argument-Encoding: "Li"
    - "OffloadId": 32-bit unique block identifier
    - "Status": Status code for the result of the operation.

## 9.2. Properties

### 9.2.1. PROP 10: PROP_HBO_MEM_MAX

- Type: Read-Write
- Packed-Encoding: "L"

| | |
|---|---|
| Octets: | 4 |
| Fields: | "PROP_HBO_MEM_MAX" |

Describes the number of bytes that may be offloaded from the NCP to the host. Default value is zero, so this property must be set by the host to a non-zero value before the NCP will begin offloading blocks.

This value is encoded as an unsigned 32-bit integer.

This property is only available if the "CAP_HBO" capability is present in "PROP_CAPS".

### 9.2.2. PROP 11: PROP_HBO_BLOCK_MAX

- Type: Read-Write
- Packed-Encoding: "S"

| | |
|---|---|
| Octets: | 2 |
| Fields: | "PROP_HBO_BLOCK_MAX" |

Describes the number of blocks that may be offloaded from the NCP to the host. Default value is 32. Setting this value to zero will cause host block offload to be effectively disabled.

This value is encoded as an unsigned 16-bit integer.

This property is only available if the "CAP_HBO" capability is present in "PROP_CAPS".

## 10. Feature: Jam Detection

Jamming detection is a feature that allows the NCP to report when it detects high levels of interference that are characteristic of intentional signal jamming.

The presence of this feature can be detected by checking for the presence of the "CAP_JAM_DETECT" (value 6) capability in "PROP_CAPS".

### 10.1. Properties

#### 10.1.1. PROP 4608: PROP_JAM_DETECT_ENABLE

- Type: Read-Write
- Packed-Encoding: "b"
- Default Value: false
- REQUIRED for "CAP_JAM_DETECT"

| | |
|---|---|
| Octets: | 1 |
| Fields: | "PROP_JAM_DETECT_ENABLE" |

Indicates if jamming detection is enabled or disabled. Set to true to enable jamming detection.

This property is only available if the "CAP_JAM_DETECT" capability is present in "PROP_CAPS".

#### 10.1.2. PROP 4609: PROP_JAM_DETECTED

- Type: Read-Only
- Packed-Encoding: "b"
- REQUIRED for "CAP_JAM_DETECT"

| | |
|---|---|
| Octets: | 1 |
| Fields: | "PROP_JAM_DETECTED" |

Set to true if radio jamming is detected. Set to false otherwise.

When jamming detection is enabled, changes to the value of this property are emitted asynchronously via "CMD_PROP_VALUE_IS".

This property is only available if the "CAP_JAM_DETECT" capability is present in "PROP_CAPS".

### 10.1.3. PROP 4610: PROP_JAM_DETECT_RSSI_THRESHOLD

- Type: Read-Write
- Packed-Encoding: "c"
- Units: dBm
- Default Value: Implementation-specific
- RECOMMENDED for "CAP_JAM_DETECT"

This parameter describes the threshold RSSI level (measured in dBm) above which the jamming detection will consider the channel blocked.

### 10.1.4. PROP 4611: PROP_JAM_DETECT_WINDOW

- Type: Read-Write
- Packed-Encoding: "c"
- Units: Seconds (1-64)
- Default Value: Implementation-specific
- RECOMMENDED for "CAP_JAM_DETECT"

This parameter describes the window period for signal jamming detection.

### 10.1.5. PROP 4612: PROP_JAM_DETECT_BUSY

- Type: Read-Write
- Packed-Encoding: "i"
- Units: Seconds (1-64)
- Default Value: Implementation-specific
- RECOMMENDED for "CAP_JAM_DETECT"

This parameter describes the number of aggregate seconds within the detection window where the RSSI must be above "PROP_JAM_DETECT_RSSI_THRESHOLD" to trigger detection.

The behavior of the jamming detection feature when "PROP_JAM_DETECT_BUSY" is larger than "PROP_JAM_DETECT_WINDOW" is undefined.

#### 10.1.6. PROP 4613: PROP_JAM_DETECT_HISTORY_BITMAP

- Type: Read-Only
- Packed-Encoding: "LL"
- Default Value: Implementation-specific
- RECOMMENDED for "CAP_JAM_DETECT"

This value provides information about current state of jamming detection module for monitoring/debugging purpose. It returns a 64-bit value where each bit corresponds to one second interval starting with bit 0 for the most recent interval and bit 63 for the oldest intervals (63 sec earlier). The bit is set to 1 if the jamming detection module observed/detected high signal level during the corresponding one second interval. The value is read-only and is encoded as two "L" (uint32) values in little-endian format (first "L" (uint32) value gives the lower bits corresponding to more recent history).

## 11. Feature: GPIO Access

This feature allows the host to have control over some or all of the GPIO pins on the NCP. The host can determine which GPIOs are available by examining "PROP_GPIO_CONFIG", described below. This API supports a maximum of 256 individual GPIO pins.

Support for this feature can be determined by the presence of "CAP_GPIO".

### 11.1. Properties

#### 11.1.1. PROP 4096: PROP_GPIO_CONFIG

- Argument-Encoding: "A(t(CCU))"
- Type: Read-write (Writable only using "CMD_PROP_VALUE_INSERT", Section 4.5)

An array of structures which contain the following fields:

- "C": GPIO Number
- "C": GPIO Configuration Flags
- "U": Human-readable GPIO name

GPIOs which do not have a corresponding entry are not supported.

The configuration parameter contains the configuration flags for the GPIO:

```
                    0   1   2   3   4   5   6   7
                  +---+---+---+---+---+---+---+---+
                  |DIR|PUP|PDN|TRIGGER|  RESERVED |
                  +---+---+---+---+---+---+---+---+
                          |O/D|
                          +---+
```

- "DIR": Pin direction.  Clear (0) for input, set (1) for output.
- "PUP": Pull-up enabled flag.
- "PDN"/"O/D": Flag meaning depends on pin direction:

  - Input: Pull-down enabled.
  - Output: Output is an open-drain.
- "TRIGGER": Enumeration describing how pin changes generate asynchronous notification commands (TBD) from the NCP to the host.

  - 0: Feature disabled for this pin
  - 1: Trigger on falling edge
  - 2: Trigger on rising edge
  - 3: Trigger on level change
- "RESERVED": Bits reserved for future use.  Always cleared to zero and ignored when read.

As an optional feature, the configuration of individual pins may be modified using the "CMD_PROP_VALUE_INSERT" command.  Only the GPIO number and flags fields MUST be present, the GPIO name (if present) would be ignored.  This command can only be used to modify the configuration of GPIOs which are already exposed---it cannot be used by the host to add addional GPIOs.

### 11.1.2.  PROP 4098: PROP_GPIO_STATE

- Type: Read-Write

Contains a bit field identifying the state of the GPIOs.  The length of the data associated with these properties depends on the number of GPIOs.  If you have 10 GPIOs, you'd have two bytes.  GPIOs are numbered from most significant bit to least significant bit, so 0x80 is GPIO 0, 0x40 is GPIO 1, etc.

For GPIOs configured as inputs:

- "CMD_PROP_VAUE_GET": The value of the associated bit describes the logic level read from the pin.

- "CMD_PROP_VALUE_SET": The value of the associated bit is ignored for these pins.

For GPIOs configured as outputs:

- "CMD_PROP_VAUE_GET": The value of the associated bit is implementation specific.
- "CMD_PROP_VALUE_SET": The value of the associated bit determines the new logic level of the output. If this pin is configured as an open-drain, setting the associated bit to 1 will cause the pin to enter a Hi-Z state.

For GPIOs which are not specified in "PROP_GPIO_CONFIG":

- "CMD_PROP_VAUE_GET": The value of the associated bit is implementation specific.
- "CMD_PROP_VALUE_SET": The value of the associated bit MUST be ignored by the NCP.

When writing, unspecified bits are assumed to be zero.

#### 11.1.3. PROP 4099: PROP_GPIO_STATE_SET

- Type: Write-only

Allows for the state of various output GPIOs to be set without affecting other GPIO states. Contains a bit field identifying the output GPIOs that should have their state set to 1.

When writing, unspecified bits are assumed to be zero. The value of any bits for GPIOs which are not specified in "PROP_GPIO_CONFIG" MUST be ignored.

#### 11.1.4. PROP 4100: PROP_GPIO_STATE_CLEAR

- Type: Write-only

Allows for the state of various output GPIOs to be cleared without affecting other GPIO states. Contains a bit field identifying the output GPIOs that should have their state cleared to 0.

When writing, unspecified bits are assumed to be zero. The value of any bits for GPIOs which are not specified in "PROP_GPIO_CONFIG" MUST be ignored.

# 12. Feature: True Random Number Generation

This feature allows the host to have access to any strong hardware random number generator that might be present on the NCP, for things like key generation or seeding PRNGs.

Support for this feature can be determined by the presence of "CAP_TRNG".

Note well that implementing a cryptographically-strong software-based true random number generator (that is impervious to things like temperature changes, manufacturing differences across devices, or unexpected output correlations) is non-trivial without a well-designed, dedicated hardware random number generator. Implementors who have little or no experience in this area are encouraged to not advertise this capability.

## 12.1. Properties

### 12.1.1. PROP 4101: PROP_TRNG_32

- Argument-Encoding: "L"
- Type: Read-Only

Fetching this property returns a strong random 32-bit integer that is suitable for use as a PRNG seed or for cryptographic use.

While the exact mechanism behind the calculation of this value is implementation-specific, the implementation must satisfy the following requirements:

- Data representing at least 32 bits of fresh entropy (extracted from the primary entropy source) MUST be consumed by the calculation of each query.
- Each of the 32 bits returned MUST be free of bias and have no statistical correlation to any part of the raw data used for the calculation of any query.

Support for this property is REQUIRED if "CAP_TRNG" is included in the device capabilities.

### 12.1.2. PROP 4102: PROP_TRNG_128

- Argument-Encoding: "D"
- Type: Read-Only

Fetching this property returns 16 bytes of strong random data suitable for direct cryptographic use without further processing(For example, as an AES key).

While the exact mechanism behind the calculation of this value is implementation-specific, the implementation must satisfy the following requirements:

- Data representing at least 128 bits of fresh entropy (extracted from the primary entropy source) MUST be consumed by the calculation of each query.
- Each of the 128 bits returned MUST be free of bias and have no statistical correlation to any part of the raw data used for the calculation of any query.

Support for this property is REQUIRED if "CAP_TRNG" is included in the device capabilities.

### 12.1.3. PROP 4103: PROP_TRNG_RAW_32

- Argument-Encoding: "D"
- Type: Read-Only

This property is primarily used to diagnose and debug the behavior of the entropy source used for strong random number generation.

When queried, returns the raw output from the entropy source used to generate "PROP_TRNG_32", prior to any reduction/whitening and/or mixing with prior state.

The length of the returned buffer is implementation specific and should be expected to be non-deterministic.

Support for this property is RECOMMENDED if "CAP_TRNG" is included in the device capabilities.

## 13. Security Considerations

### 13.1. Raw Application Access

Spinel MAY be used as an API boundary for allowing processes to configure the NCP. However, such a system MUST NOT give unprivileged processess the ability to send or receive arbitrary command frames to the NCP. Only the specific commands and properties that are required should be allowed to be passed, and then only after being checked for proper format.

# 14. References

## 14.1. URIs

[1] https://www.w3.org/TR/exi/#encodingUnsignedInteger

[2] http://reveng.sourceforge.net/crc-catalogue/16.htm#crc.cat.kermit

[3] https://github.com/miekg/mmark

[4] http://xml2rfc.ietf.org/

# Appendix A. Framing Protocol

Since this NCP protocol is defined independently of the physical transport or framing, any number of transports and framing protocols could be used successfully. However, in the interests of compatibility, this document provides some recommendations.

## A.1. UART Recommendations

The recommended default UART settings are:

- Bit rate: 115200
- Start bits: 1
- Data bits: 8
- Stop bits: 1
- Parity: None
- Flow Control: Hardware

These values may be adjusted depending on the individual needs of the application or product, but some sort of flow control MUST be used. Hardware flow control is preferred over software flow control. In the absence of hardware flow control, software flow control (XON/ XOFF) MUST be used instead.

We also *RECOMMEND* an Arduino-style hardware reset, where the DTR signal is coupled to the "R&#773;E&#773;S&#773;" pin through a 0.01[micro]F capacitor. This causes the NCP to automatically reset whenever the serial port is opened. At the very least we *RECOMMEND* dedicating one of your host pins to controlling the "R&#773;E&#773;S&#773;" pin on the NCP, so that you can easily perform a hardware reset if necessary.

### A.1.1. UART Bit Rate Detection

When using a UART, the issue of an appropriate bit rate must be considered. A bitrate of 115200 bits per second has become a defacto standard baud rate for many serial peripherals. This rate, however, is slower than the theoretical maximum bitrate of the 802.15.4 2.4GHz PHY (250kbit). In most circumstances this mismatch is not significant because the overall bitrate will be much lower than either of these rates, but there are circumstances where a faster UART bitrate is desirable. Thus, this document proposes a simple bitrate detection scheme that can be employed by the host to detect when the attached NCP is initially running at a higher bitrate.

The algorithm is to send successive NOOP commands to the NCP at increasing bitrates. When a valid "CMD_LAST_STATUS" response has been received, we have identified the correct bitrate.

In order to limit the time spent hunting for the appropriate bitrate, we RECOMMEND that only the following bitrates be checked:

- 115200
- 230400
- 1000000 (1Mbit)

The bitrate MAY also be changed programmatically by adjusting "PROP_UART_BITRATE", if implemented.

### A.1.2. HDLC-Lite

_HDLC-Lite_ is the recommended framing protocol for transmitting Spinel frames over a UART. HDLC-Lite consists of only the framing, escaping, and CRC parts of the larger HDLC protocol---all other parts of HDLC are omitted. This protocol was chosen because it works well with software flow control and is widely implemented.

To transmit a frame with HDLC-lite, the 16-bit CRC must first be appended to the frame. The CRC function is defined to be CRC-16/CCITT, otherwise known as the KERMIT CRC [2].

Individual frames are terminated with a frame delimiter octet called the 'flag' octet ("0x7E").

The following octets values are considered _special_ and should be escaped when present in data frames:

| Octet Value | Description |
|---|---|
| 0x7E | Frame Delimiter (Flag) |
| 0x7D | Escape Byte |
| 0x11 | XON |
| 0x13 | XOFF |
| 0xF8 | Vendor-Specific |

When present in a data frame, these octet values are escaped by prepending the escape octet ("0x7D") and XORing the value with "0x20".

When receiving a frame, the CRC must be verified after the frame is unescaped. If the CRC value does not match what is calculated for the frame data, the frame MUST be discarded. The implementation MAY indicate the failure to higher levels to handle as they see fit, but MUST NOT attempt to process the deceived frame.

Consecutive flag octets are entirely legal and MUST NOT be treated as a framing error. Consecutive flag octets MAY be used as a way to wake up a sleeping NCP.

When first establishing a connection to the NCP, it is customary to send one or more flag octets to ensure that any previously received data is discarded.

### A.2. SPI Recommendations

We RECOMMEND the use of the following standard SPI signals:

- "C&#773;S&#773;": (Host-to-NCP) Chip Select
- "CLK": (Host-to-NCP) Clock
- "MOSI": Master-Output/Slave-Input
- "MISO": Master-Input/Slave-Output
- "I&#773;N&#773;T&#773;": (NCP-to-Host) Host Interrupt
- "R&#773;E&#773;S&#773;": (Host-to-NCP) NCP Hardware Reset

The "I&#773;N&#773;T&#773;" signal is used by the NCP to indicate to the host that the NCP has frames pending to send to it. When asserted, the host SHOULD initiate a SPI transaction in a timely manner.

We RECOMMEND the following SPI properties:

- "C&#773;S&#773;" is active low.
- "CLK" is active high.

- "CLK" speed is larger than 500 kHz.
- Data is valid on leading edge of "CLK".
- Data is sent in multiples of 8-bits (octets).
- Octets are sent most-significant bit first.

This recommended configuration may be adjusted depending on the individual needs of the application or product.

### A.2.1. SPI Framing Protocol

Each SPI frame starts with a 5-byte frame header:

| Octets: | 1 | 2 | 2 |
|---|---|---|---|
| Fields: | HDR | RECV_LEN | DATA_LEN |

- "HDR": The first byte is the header byte (defined below)
- "RECV_LEN": The second and third bytes indicate the largest frame size that that device is ready to receive. If zero, then the other device must not send any data. (Little endian)
- "DATA_LEN": The fourth and fifth bytes indicate the size of the pending data frame to be sent to the other device. If this value is equal-to or less-than the number of bytes that the other device is willing to receive, then the data of the frame is immediately after the header. (Little Endian)

The "HDR" byte is defined as:

```
  0   1   2   3   4   5   6   7
+---+---+---+---+---+---+---+---+
|RST|CRC|CCF|  RESERVED |PATTERN|
+---+---+---+---+---+---+---+---+
```

- "RST": This bit is set when that device has been reset since the last time "C&#773;S&#773;" was asserted.
- "CRC": This bit is set when that device supports writing a 16-bit CRC at the end of the data. The CRC length is NOT included in DATA_LEN.
- "CCF": "CRC Check Failure". Set if the CRC check on the last received frame failed, cleared to zero otherwise. This bit is only used if both sides support CRC.
- "RESERVED": These bits are all reserved for future used. They MUST be cleared to zero and MUST be ignored if set.
- "PATTERN": These bits are set to a fixed value to help distinguish valid SPI frames from garbage (by explicitly making "0xFF" and "0x00" invalid values). Bit 6 MUST be set to be one and bit 7

MUST be cleared (0). A frame received that has any other values for these bits MUST be dropped.

Prior to a sending or receiving a frame, the master MAY send a 5-octet frame with zeros for both the max receive frame size and the the contained frame length. This will induce the slave device to indicate the length of the frame it wants to send (if any) and indicate the largest frame it is capable of receiving at the moment. This allows the master to calculate the size of the next transaction. Alternatively, if the master has a frame to send it can just go ahead and send a frame of that length and determine if the frame was accepted by checking that the "RECV_LEN" from the slave frame is larger than the frame the master just tried to send. If the "RECV_LEN" is smaller then the frame wasn't accepted and will need to be transmitted again.

This protocol can be used either unidirectionally or bidirectionally, determined by the behavior of the master and the slave.

If the the master notices "PATTERN" is not set correctly, the master should consider the transaction to have failed and try again after 10 milliseconds, retrying up to 200 times. After unsuccessfully trying 200 times in a row, the master MAY take appropriate remedial action (like a NCP hardware reset, or indicating a communication failure to a user interface).

At the end of the data of a frame is an optional 16-bit CRC, support for which is indicated by the "CRC" bit of the "HDR" byte being set. If these bits are set for both the master and slave frames, then CRC checking is enabled on both sides, effectively requiring that frame sizes be two bytes longer than would be otherwise required. The CRC is calculated using the same mechanism used for the CRC calculation in HDLC-Lite (See Appendix A.1.2). When both of the "CRC" bits are set, both sides must verify that the "CRC" is valid before accepting the frame. If not enough bytes were clocked out for the CRC to be read, then the frame must be ignored. If enough bytes were clocked out to perform a CRC check, but the CRC check fails, then the frame must be rejected and the "CRC_FAIL" bit on the next frame (and ONLY the next frame) MUST be set.

## A.3. I^2C Recommendations

TBD

[CREF2]

### A.4. Native USB Recommendations

TBD

[CREF3]

## Appendix B. Test Vectors

### B.1. Test Vector: Packed Unsigned Integer

| Decimal Value | Packet Octet Encoding |
|---:|---|
| 0 | "00" |
| 1 | "01" |
| 127 | "7F" |
| 128 | "80 01" |
| 129 | "81 01" |
| 1,337 | "B9 0A" |
| 16,383 | "FF 7F" |
| 16,384 | "80 80 01" |
| 16,385 | "81 80 01" |
| 2,097,151 | "FF FF 7F" |

[CREF4]

### B.2. Test Vector: Reset Command

- NLI: 0
- TID: 0
- CMD: 1 ("CMD_RESET")

Frame:

```
80 01
```

### B.3. Test Vector: Reset Notification

- NLI: 0
- TID: 0
- CMD: 6 ("CMD_VALUE_IS")
- PROP: 0 ("PROP_LAST_STATUS")
- VALUE: 114 ("STATUS_RESET_SOFTWARE")

Frame:

```
80 06 00 72
```

## B.4. Test Vector: Scan Beacon

- NLI: 0
- TID: 0
- CMD: 7 ("CMD_VALUE_INSERTED")
- PROP: 51 ("PROP_MAC_SCAN_BEACON")
- VALUE: Structure, encoded as "Cct(ESSc)t(iCUd)"

  * CHAN: 15
  * RSSI: -60dBm
  * MAC_DATA: (0D 00 B6 40 D4 8C E9 38 F9 52 FF FF D2 04 00)

    + Long address: B6:40:D4:8C:E9:38:F9:52
    + Short address: 0xFFFF
    + PAN-ID: 0x04D2
    + LQI: 0
  * NET_DATA: (13 00 03 20 73 70 69 6E 65 6C 00 08 00 DE AD 00 BE EF 00 CA FE)

    + Protocol Number: 3
    + Flags: 0x20
    + Network Name: "spinel"
    + XPANID: "DE AD 00 BE EF 00 CA FE"

Frame:

```
80 07 33 0F C4 0D 00 B6 40 D4 8C E9 38 F9 52 FF FF D2 04 00
13 00 03 20 73 70 69 6E 65 6C 00 08 00 DE AD 00 BE EF 00 CA
FE
```

## B.5. Test Vector: Inbound IPv6 Packet

CMD_VALUE_IS(PROP_STREAM_NET)

[CREF5]

## B.6. Test Vector: Outbound IPv6 Packet

CMD_VALUE_SET(PROP_STREAM_NET)

[CREF6]

## B.7. Test Vector: Fetch list of on-mesh networks

- NLI: 0
- TID: 4
- CMD: 2 ("CMD_VALUE_GET")
- PROP: 90 ("PROP_THREAD_ON_MESH_NETS")

Frame:

```
                                 84 02 5A
```

### B.8. Test Vector: Returned list of on-mesh networks

- NLI: 0
- TID: 4
- CMD: 6 ("CMD_VALUE_IS")
- PROP: 90 ("PROP_THREAD_ON_MESH_NETS")
- VALUE: Array of structures, encoded as "A(t(6CbC))"

| IPv6 Prefix | Prefix Length | Stable Flag | Other Flags |
|---|---|---|---|
| 2001:DB8:1:: | 64 | True | ?? |
| 2001:DB8:2:: | 64 | False | ?? |

Frame:

```
84 06 5A 13 00 20 01 0D B8 00 01 00 00 00 00 00 00 00 00 00
00 40 01 ?? 13 00 20 01 0D B8 00 02 00 00 00 00 00 00 00 00
00 00 40 00 ??
```

### B.9. Test Vector: Adding an on-mesh network

- NLI: 0
- TID: 5
- CMD: 4 ("CMD_VALUE_INSERT")
- PROP: 90 ("PROP_THREAD_ON_MESH_NETS")
- VALUE: Structure, encoded as "6CbCb"

| IPv6 Prefix | Prefix Length | Stable Flag | Other Flags |
|---|---|---|---|
| 2001:DB8:3:: | 64 | True | ?? |

Frame:

```
85 03 5A 20 01 0D B8 00 03 00 00 00 00 00 00 00 00 00 00 40
01 ?? 01
```

[CREF7]

## B.10. Test Vector: Insertion notification of an on-mesh network

- NLI: 0
- TID: 5
- CMD: 7 ("CMD_VALUE_INSERTED")
- PROP: 90 ("PROP_THREAD_ON_MESH_NETS")
- VALUE: Structure, encoded as "6CbCb"

| IPv6 Prefix | Prefix Length | Stable Flag | Other Flags |
|---|---|---|---|
| 2001:DB8:3:: | 64 | True | ?? |

Frame:

```
85 07 5A 20 01 0D B8 00 03 00 00 00 00 00 00 00 00 00 00 40
01 ?? 01
```

[CREF8]

## B.11. Test Vector: Removing a local on-mesh network

- NLI: 0
- TID: 6
- CMD: 5 ("CMD_VALUE_REMOVE")
- PROP: 90 ("PROP_THREAD_ON_MESH_NETS")
- VALUE: IPv6 Prefix "2001:DB8:3::"

Frame:

```
86 05 5A 20 01 0D B8 00 03 00 00 00 00 00 00 00 00 00 00
```

## B.12. Test Vector: Removal notification of an on-mesh network

- NLI: 0
- TID: 6
- CMD: 8 ("CMD_VALUE_REMOVED")
- PROP: 90 ("PROP_THREAD_ON_MESH_NETS")
- VALUE: IPv6 Prefix "2001:DB8:3::"

Frame:

```
86 08 5A 20 01 0D B8 00 03 00 00 00 00 00 00 00 00 00 00
```

## Appendix C. Example Sessions

### C.1. NCP Initialization

[CREF9]

Check the protocol version to see if it is supported:

- CMD_VALUE_GET:PROP_PROTOCOL_VERSION
- CMD_VALUE_IS:PROP_PROTOCOL_VERSION

Check the NCP version to see if a firmware update may be necessary:

- CMD_VALUE_GET:PROP_NCP_VERSION
- CMD_VALUE_IS:PROP_NCP_VERSION

Check interface type to make sure that it is what we expect:

- CMD_VALUE_GET:PROP_INTERFACE_TYPE
- CMD_VALUE_IS:PROP_INTERFACE_TYPE

If the host supports using vendor-specific commands, the vendor should be verified before using them:

- CMD_VALUE_GET:PROP_VENDOR_ID
- CMD_VALUE_IS:PROP_VENDOR_ID

Fetch the capability list so that we know what features this NCP supports:

- CMD_VALUE_GET:PROP_CAPS
- CMD_VALUE_IS:PROP_CAPS

If the NCP supports CAP_NET_SAVE, then we go ahead and recall the network:

- CMD_NET_RECALL

### C.2. Attaching to a network

[CREF10]

We make the assumption that the NCP is not currently associated with a network.

Set the network properties, if they were not already set:

- CMD_VALUE_SET:PROP_PHY_CHAN

- CMD_VALUE_IS:PROP_PHY_CHAN
- CMD_VALUE_SET:PROP_NET_XPANID
- CMD_VALUE_IS:PROP_NET_XPANID
- CMD_VALUE_SET:PROP_MAC_15_4_PANID
- CMD_VALUE_IS:PROP_MAC_15_4_PANID
- CMD_VALUE_SET:PROP_NET_NETWORK_NAME
- CMD_VALUE_IS:PROP_NET_NETWORK_NAME
- CMD_VALUE_SET:PROP_NET_MASTER_KEY
- CMD_VALUE_IS:PROP_NET_MASTER_KEY
- CMD_VALUE_SET:PROP_NET_KEY_SEQUENCE_COUNTER
- CMD_VALUE_IS:PROP_NET_KEY_SEQUENCE_COUNTER
- CMD_VALUE_SET:PROP_NET_KEY_SWITCH_GUARDTIME
- CMD_VALUE_IS:PROP_NET_KEY_SWITCH_GUARDTIME

Bring the network interface up:

- CMD_VALUE_SET:PROP_NET_IF_UP:TRUE
- CMD_VALUE_IS:PROP_NET_IF_UP:TRUE

Bring the routing stack up:

- CMD_VALUE_SET:PROP_NET_STACK_UP:TRUE
- CMD_VALUE_IS:PROP_NET_STACK_UP:TRUE

Some asynchronous events from the NCP:

- CMD_VALUE_IS:PROP_NET_ROLE
- CMD_VALUE_IS:PROP_NET_PARTITION_ID
- CMD_VALUE_IS:PROP_THREAD_ON_MESH_NETS

## C.3. Successfully joining a pre-existing network

[CREF11]

This example session is identical to the above session up to the point where we set PROP_NET_IF_UP to true. From there, the behavior changes.

- CMD_VALUE_SET:PROP_NET_REQUIRE_JOIN_EXISTING:TRUE
- CMD_VALUE_IS:PROP_NET_REQUIRE_JOIN_EXISTING:TRUE

Bring the routing stack up:

- CMD_VALUE_SET:PROP_NET_STACK_UP:TRUE
- CMD_VALUE_IS:PROP_NET_STACK_UP:TRUE

Some asynchronous events from the NCP:

- CMD_VALUE_IS:PROP_NET_ROLE
- CMD_VALUE_IS:PROP_NET_PARTITION_ID
- CMD_VALUE_IS:PROP_THREAD_ON_MESH_NETS

Now let's save the network settings to NVRAM:

- CMD_NET_SAVE

### C.4. Unsuccessfully joining a pre-existing network

This example session is identical to the above session up to the point where we set PROP_NET_IF_UP to true. From there, the behavior changes.

- CMD_VALUE_SET:PROP_NET_REQUIRE_JOIN_EXISTING:TRUE
- CMD_VALUE_IS:PROP_NET_REQUIRE_JOIN_EXISTING:TRUE

Bring the routing stack up:

- CMD_VALUE_SET:PROP_NET_STACK_UP:TRUE
- CMD_VALUE_IS:PROP_NET_STACK_UP:TRUE

Some asynchronous events from the NCP:

- CMD_VALUE_IS:PROP_LAST_STATUS:STATUS_JOIN_NO_PEERS
- CMD_VALUE_IS:PROP_NET_STACK_UP:FALSE

### C.5. Detaching from a network

TBD

### C.6. Attaching to a saved network

[CREF12]

Recall the saved network if you haven't already done so:

- CMD_NET_RECALL

Bring the network interface up:

- CMD_VALUE_SET:PROP_NET_IF_UP:TRUE
- CMD_VALUE_IS:PROP_NET_IF_UP:TRUE

Bring the routing stack up:

- CMD_VALUE_SET:PROP_NET_STACK_UP:TRUE
- CMD_VALUE_IS:PROP_NET_STACK_UP:TRUE

Some asynchronous events from the NCP:

- CMD_VALUE_IS:PROP_NET_ROLE
- CMD_VALUE_IS:PROP_NET_PARTITION_ID
- CMD_VALUE_IS:PROP_THREAD_ON_MESH_NETS

## C.7. NCP Software Reset

[CREF13]

- CMD_RESET
- CMD_VALUE_IS:PROP_LAST_STATUS:STATUS_RESET_SOFTWARE

Then jump to Appendix C.1.

## C.8. Adding an on-mesh prefix

TBD

## C.9. Entering low-power modes

TBD

## C.10. Sniffing raw packets

[CREF14]

This assumes that the NCP has been initialized.

Optionally set the channel:

- CMD_VALUE_SET:PROP_PHY_CHAN:x
- CMD_VALUE_IS:PROP_PHY_CHAN

Set the filter mode:

- CMD_VALUE_SET:PROP_MAC_PROMISCUOUS_MODE:MAC_PROMISCUOUS_MODE_MONITOR
- CMD_VALUE_IS:PROP_MAC_PROMISCUOUS_MODE:MAC_PROMISCUOUS_MODE_MONITOR

Enable the raw stream:

- CMD_VALUE_SET:PROP_MAC_RAW_STREAM_ENABLED:TRUE
- CMD_VALUE_IS:PROP_MAC_RAW_STREAM_ENABLED:TRUE

Enable the PHY directly:

- CMD_VALUE_SET:PROP_PHY_ENABLED:TRUE
- CMD_VALUE_IS:PROP_PHY_ENABLED:TRUE

Now we will get raw 802.15.4 packets asynchronously on PROP_STREAM_RAW:

- CMD_VALUE_IS:PROP_STREAM_RAW:...
- CMD_VALUE_IS:PROP_STREAM_RAW:...
- CMD_VALUE_IS:PROP_STREAM_RAW:...

This mode may be entered even when associated with a network. In that case, you should set "PROP_MAC_PROMISCUOUS_MODE" to "MAC_PROMISCUOUS_MODE_PROMISCUOUS" or "MAC_PROMISCUOUS_MODE_NORMAL", so that you can avoid receiving packets from other networks or that are destined for other nodes.

## Appendix D. Glossary

[CREF15]

| | |
|---|---|
| FCS | Final Checksum. Bytes added to the end of a packet to help determine if the packet was received without corruption. |
| NCP | Network Control Processor. |
| NLI | Network Link Identifier. May be a value between zero and three. See Section 2.1.2 for more information. |
| OS | Operating System, i.e. the IPv6 node using Spinel to control and manage one or more of its IPv6 network interfaces. |
| PHY | Physical layer. Refers to characteristics and parameters related to the physical implementation and operation of a networking medium. |
| PUI | Packed Unsigned Integer. A way to serialize an unsigned integer using one, two, or three bytes. Used throughout the Spinel protocol. See Section 3.2 for more information. |
| TID | Transaction Identifier. May be a value between zero and fifteen. See Section 2.1.3 for more information. |

## Appendix E. Acknowledgments

Thread is a registered trademark of The Thread Group, Inc.

Special thanks to Nick Banks, Jonathan Hui, Abtin Keshavarzian, Piotr Szkotak, Arjuna Sivasithambaresan and Martin Turon for their substantial contributions and feedback related to this document.

This document was prepared using mmark [3] by (Miek Gieben) and xml2rfc (version 2) [4].

## Editorial Comments

[CREF1] RQ: Eventually, when https://github.com/miekg/mmark/issues/95 is addressed, the above table should be swapped out with this: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |---|---|---|---|---|---|---|---| | FLG || NLI || TID ||||

[CREF2] RQ: It may make sense to have a look at what Bluetooth HCI is doing for native I^2C framing and go with that.

[CREF3] RQ: It may make sense to have a look at what Bluetooth HCI is doing for native USB framing and go with that.

[CREF4] RQ: The PUI test-vector encodings need to be verified.

[CREF5] RQ: FIXME: This test vector is incomplete.

[CREF6] RQ: FIXME: This test vector is incomplete.

[CREF7] RQ: FIXME: This test vector is incomplete.

[CREF8] RQ: FIXME: This test vector is incomplete.

[CREF9] RQ: FIXME: This example session is incomplete.

[CREF10] RQ: FIXME: This example session is incomplete.

[CREF11] RQ: FIXME: This example session is incomplete.

[CREF12] RQ: FIXME: This example session is incomplete.

[CREF13] RQ: FIXME: This example session is incomplete.

[CREF14] RQ: FIXME: This example session is incomplete.

[CREF15] RQ: Alphabetize before finalization.

## Authors' Addresses

Robert S. Quattlebaum
Nest Labs, Inc.
3400 Hillview Ave.
Palo Alto, California 94304
USA

Email: rquattle@nestlabs.com

James Woodyatt
Nest Labs, Inc.
3400 Hillview Ave.
Palo Alto, California 94304
USA

Email: jhw@nestlabs.com