NXP Semiconductors
User's Guide

Document Number: KTSSHIUG
Rev. 6, 08 April 2019

Kinetis Thread Stack Shell Interface

Contents

Chapter 1
About This Document

This document describes the commands supported by the Kinetis Thread Stack when using the shell command line user interface available on a serial (UART or USB) connection to a Thread application.

Chapter 2
Shell Command Overview

This section describes the shell commands available in all projects.

By typing `help` in a terminal, a list of the available commands is displayed, as shown in the listing below:

```
$ help
help                print command description/usage
version             print version of all the registered modules
history             print history

thr                 Thread Stack commands
ifconfig            IP Stack interfaces configuration
route               IP static routes management
firewall            Manage IP Firewall rules
mcastgroup          Multicast groups management
ping                IP Stack ping tool
dnsrequest          Send DNS request
coap                Send CoAP message
reboot              MCU Reset
factoryreset        FactoryReset Command
macfilter           MAC filtering commands
identify            Identify Device
getnodesip          Gets IPv6 addresses from all Thread nodes
```

Chapter 3
Generic Commands

The source file for the Shell interface is `shell_ip.c`. User can edit this file to change the behavior of the interface by adding code according to their application.

The basic Thread application usually uses one communication interface. Some applications like the border router and the hybrid Bluetooth LE Thread applications can have two or more interfaces. These interfaces have a unique ID associated, that can be found in `network_utils.h`, as the `ipIfUniqueId_t` enumeration.

```
/*! Unique interface ID enumeration*/
typedef enum
{
    gIpIfSlp0_c = 0x00,               /*!<SLWP0 interface */
    gIpIfSlp1_c,                      /*!<SLWP1 interface */
    gIpIfEth0_c,                      /*!<ETH0 interface */
    gIpIfEth1_c,                      /*!<ETH1 interface */
    gIpIfWiFi0_c,                     /*!<WiFi0 interface */
    gIpIfWiFi1_c,                     /*!<WiFi1 interface */
    gIpIfUsbRndis_c,                  /*!<RNDIS interface */
    gIpIfSerialTun_c,                 /*!<Serial TUN interface */
    gIpIfBle0_c,                      /*!<BLE0 interface */
    gIpIfBle1_c,                      /*!<BLE1 interface */
    gIpIfLoopback_c,                  /*!<Loopback interface */
    gIpIfUndef_c = THR_ALL_FFs8,      /*!<Undefined interface */
}ipIfUniqueId_t;
```

These unique IDs can be associated to strings representing the desired names to be shown in the Shell interface. This can be configured in the `shell_ip` source file.

```
/* Interface table. User is advised to add the name of the interface according to the existing interface
*/
  const aShellThrInterface_t mThrInterfaces[] =
  {
    {(uint8_t*)"6LoWPAN",      gIpIfSlp0_c},
    {(uint8_t*)"eth",          gIpIfEth0_c},
    {(uint8_t*)"serialtun",    gIpIfSerialTun_c},
    {(uint8_t*)"usbEnet",      gIpIfUsbRndis_c},
    {(uint8_t*)"6LoBLE",       gIpIfBle0_c},
    {(uint8_t*)"Loopback",     gIpIfLoopback_c},
   };
```

The available commands are explained below with examples.

`help` - Command used for displaying available helping text.

```
$ help help
help - print brief description of all commands
help <command> - print detailed usage of 'command'

$ help ifconfig
ifconfig - IP Stack interfaces configuration
ifconfig - displays all interfaces and addresses configured on the device
ifconfig <interface ID> ip <IP address>
```

`version` – Displays the version for the registered stack components

`history` – Displays the last four commands issued in the shell interface

`ifconfig` - Displays and updates address configuration for IP interfaces.

```
$ help ifconfig
ifconfig - IP Stack interfaces configuration
   ifconfig displays all interfaces and addresses configured on the device
   ifconfig <interface ID> ip <IP address>
```

`ifconfig` - Displays all addresses configured on each interface available on the device. The output for a device with a single interface is shown in the image below.

```
$ ifconfig
Interface 1: 6LoWPAN
Link local address (LL64): fe80::c980:6982:b03f:161b
Mesh local address (ML64): fd05:3d45:6a97:18a4:dcd7:8247:c937:8333
Mesh local address (ML16): fd05:3d45:6a97:18a4::ff:fe00:0
Realm local all Thread Nodes(MCast): ff33:40:fd05:3d45:6a97:18a4::1
Interface 0: Loopback
```

`ifconfig <interface ID> ip <IP address>` - Adds a new IPv6 address on the specified interface.

```
$ ifconfig 0 ip 2001::1
interface configured
```

NOTE

Users can check if the IP address assignment was correctly performed by typing the command `ifconfig` and observing that the address has been added to the list.

`route` – This command is used to configure a static IPv6 route for a destination network.

`route add -p <dest ipv6 addr> -h <next hop IPv6 addr> -I <next hop interface> -len <prefix length> -m <route metric>` - This command lets the user add a static IPv6 route. Argument -p <dest ipv6 addr> is for setting the destination network prefix, while -h <next hop IPv6 addr> can be used to specify the next hop address. The prefix's length is specified by argument -len <prefix length> and the interface for the destination network is specified by the -I <next hop interface> argument. This route can have a specified route metric, by using the argument -m <*route metric*>.

```
$ route add -p fd7c:30de:39ca:ba36:: -h fd7c:30de:39ca:ba36::ff:fe00:4400 -I 6LoWPAN -len 64 -m 5
Success!
```

`route remove -p <dest ipv6 address> -h <next hop>` - This command removes the specified entry from the IPv6 static routes table. Network address can be specified by -p <dest ipv6 address> argument and the -h <next hop> argument lets the user specify the hop address

```
$ route remove -p fd7c:30de:39ca:ba36:: -h fd7c:30de:39ca:ba36::ff:fe00:4400
Success!
```

`route print` – This command prints the available static routes.

```
$ route print
Destination                                Next Hop                                NextHopIf
PrefixLen Metric
```

```
fd7c:30de:39ca:ba36::                         fd7c:30de:39ca:ba36::ff:fe00:4400            6LoWPAN
64        35
fd7c:30de:39ca:ba36::                      ::                                      6LoWPAN    64       34
```

`firewall` – This command is used to manage IP firewall rules. The firewall is useful to manage connections at the IP layer, allowing or denying packets according to specified rules.

```
    $ help firewall
   firewall - Manage IP Firewall rules
   firewall enable
   firewall disable
   firewall add -dst <IPv6Addr> -src <IPv6Addr> -srcport <UDPPort> -dstport <UDPPort> -lendst <bitsLen>
-lensrc <bitsLen> -sec <secLevel> -I <IP Interface>
   firewall remove <ruleNb>
   firewall show
   Valid interfaces: 6LoWPAN, eth, serialtun, usbEnet
```

`firewall enable` – This command enables the IPv6 Firewall.

`firewall disable` – This command disables the IPv6 Firewall.

`firewall add -dst <IPv6Addr> -src <IPv6Addr> -srcport <UDPPort> -dstport <UDPPort> -lendst <bitsLen> -lensrc <bitsLen> -sec <secLevel> -I <IP Interface>` - Adding a rule in the firewall is done by using firewall add command. Through -dst <IPv6Addr> and -src <IPv6Addr>, user can specify the source and destination addresses or network prefixes to be used as a firewall rule. Arguments -lendst <bitsLen> and -lensrc <bitsLen> can be used to set the number of bits in the entered prefixes. UDP source and destination ports can be set by -srcport <UDPPort> and -dstport <UDPPort>. -sec <secLevel> can be used to add a security level based rule for the IP packets. -I <IP Interface> is used to specify the interface on which the firewall rule applies. Security levels for Thread networks are 0 (unsecured) and 5 (secured). User must always input a valid IP address or prefix, even when entering a rule that is not based on IP addresses or prefixes. In this case, the user can use -dst :: -src ::.

```
      $ firewall add -dst fd4d:d79b:650b:d941::ff:fe00:0 -src :: -portsrc 80
      Success!
      $ firewall add -dst :: -src :: -portsrc 8080 -portdst 1234
      Success!
```

firewall remove <ruleNb> - To remove a firewall rule, user can input this command, replacing the <ruleNb> with the appropriate number of the rule shown by issuing command firewall show.

```
    $ firewall remove 4
    Success!
```

`firewall show` – This is the command that the user can use to see the firewall rules that have been previously set.

```
      $ firewall show
      ruleNb src     dst    srcPort dstPort lenSrc lenDst secLevel ifId
      0       ::      ::      0       49153   0       0       0       6LoWPAN
      1       ::      ::      0       1026    0       0       0       Loopback
      2       ::      ::      0       49192   0       0       0       Loopback
      3       ::      ::      0       49192   0       0       0       6LoWPAN
      4       ::      fd4d:d79b:650b:d941::ff:fe00:0  80      0       0       0       0       6LoWPAN
```

```
      5       ::      ::      8080    1234    0       0       0       6LoWPAN
```

`mcastgroup` - Set of commands for IP multicast group management.

```
$ help mcastgroup
mcastgroup - Multicast groups management
   mcastgroup show - displays all joined multicast groups
   mcastgroup add <IP group address> <interface id>- joins to a new multicast group
   mcastgroup leave <IP group address> <interface id>- leaves a multicast group
Warning! Do not remove addresses that may affect stack's behaviour!
```

`mcastgroup show` - Displays all the group addresses that the device is currently part of, on each IP interface.

```
$ mcastgroup show

Interface 0:
Multicast Group: ff02::1
Multicast Group: ff03::1
Multicast Group: ff32:40:fd6e:8865:56d8::1
Multicast Group: ff33:40:fd6e:8865:56d8::1
Multicast Group: ff02::2
Multicast Group: ff03::2
Multicast Group: ff03::1:3
```

`mcastgroup add` - Joins a new multicast group.

```
$ mcastgroup add ff05::3 0
 Success!
```

`mcastgroup leave` - Leaves a multicast group.

NOTE

Do not leave the multicast groups that might affect stack's behavior!

`ping` - Sends an ICMP Echo Request to the specified address.

```
$ help ping
 ping - IP Stack ping IPv4/IPv6 addresses
 ping <ip address> -I <interface> -i <timeout> -c <count> -h <hoplimit> -s <size> -t <continuous
ping> -S <source IP address>
   Valid interfaces: 6LoWPAN, eth, serialtun, usbEnet
```

```
$ ping fe80::166e:a00:0:01
Pinging fe80::166e:a00:0:01 with 32 bytes of data
Reply from fe80::166e:a00:0:01: bytes=32 time=21ms
```

`reboot` - Resets the MCU.

`factoryreset` - Reverts all changes to attributes to factory settings.

`identify` - Blinks an LED on the device in order to correlate the shell terminal with the physical board. The LED blinks until user sends Ctrl+C to stop.

`getnodesip` - This command is used to retrieve IPv6 addresses from all Thread devices in the network. To enable this command, set `GETIPv6ADDRESSES_APP` to **1** in the `config.h` file.

```
      $getnodesip
      $
```

```
Received IP Addresses from node:
fe80::fc1c:899:9d70:4a36
fd08:74e6:b5f7:f91a:191:add9:241:6e3c
fd08:74e6:b5f7:f91a::ff:fe00:400
```

Chapter 4
Thread Network-Specific Commands

`thr` - Commands for configuring a Thread network.

```
$ help thr
thr - Commands for Thread Network
thr create
thr join
thr scan [active|energy|both]
thr detach
thr commissioner [start|stop]
thr joiner add <psk> <eui>
thr joiner remove <eui>
thr joiner removeall
thr joiner view
thr sync steering
thr sync nwkdata
thr get attributes- displays a list of all Thread attributes available for getting/setting
thr get <ATTRNAME/TABLENAME> - displays the value for the specified attribute
thr set <ATTRNAME> <value> - changes the value of the attribute with the specified value
thr nwkdata add slaac -p <<Prefix> - len <prefixLength> -t <lifetime in seconds>
thr nwkdata add dhcpserver -p <Prefix> - len <prefixLength> -t <lifetime in seconds>
thr nwkdata add extroute -p <Prefix> - len <prefixLength> -t <lifetime in seconds>
thr nwkdata add dnsserver <ipAddress>
thr nwkdata remove dnsserver <ipAddress>
thr nwkdata remove -p <Prefix> - len <prefixLength>
thr nwkdata removeall
```

NOTE

thr mgmt commands are available if SHELL_DUT_COMMISSIONER is TRUE

`thr create` - The device automatically becomes the leader of a new Thread Network.

```
$ thr create
Creating network...
$
Node has taken the Leader role
Created a new Thread network on channel 20 and PAN ID:0x7fe5

Interface 1: 6LoWPAN
        Mesh local address (ML64): fd40:d301:ca5b::cdcb:33fb:ebe4:e264
        Mesh local address (ML16): fd40:d301:ca5b::ff:fe00:0
```

`thr join` - The device joins an existing Thread network.

```
$ thr join
Joining network...
Commissioning successful
Attaching to Thread network...
Attached to network with PAN ID: 0x18c
Requesting to become Active Router...
Success

Interface 1: 6LoWPAN
```

```
        Mesh local address (ML64): fd39:8b1:5f1f::b085:d477:7e95:1a2e
        Mesh local address (ML16): fd39:8b1:5f1f::ff:fe00:400
```

`thr detach` - The device detaches from the current Thread network.

```
$ thr detach
Detaching from network...

$  Success!
```

`thr commissioner` - Command set used for starting/stopping a commissioner. If there is already a commissioner started in the network, an error message is displayed.

```
$ thr commissioner start
(Local) Commissioner Started

$ thr commissioner stop
```

`thr joiner` - Command set used for configuring joiners in Thread Commissioning. For adding a joiner, the PSKc and the joiner's extended address are needed. After adding the joiner data, `thr sync steering` is needed to add the joiner to the bloom filter used for steering new devices onto the network. If the sync succeeded, a data response is propagated with the updated bloom filter in Thread network.

```
$ thr joiner add kinetis 0x146E0A0000000002
 Success!
$ thr sync steering
```

`thr get` - Command used for displaying attributes of the Thread network. A list of all attributes available is shown by the command `thr get attributes`.

```
$ thr get channel
channel: 17
$ thr get panid
panid: 0x18c0
$ thr get masterkey
masterkey: 0x00112233445566778899AABBCCDDEEFF
```

`thr get` commands also displays neighbor table, routing table, detailed information about a specific neighbor or parent.

`thr get neighbors` - Displays the neighbors of the current device, along with several information about each neighbor: the extended MAC address, the short address, number of seconds since the last communication with the neighbor, the link reported in the last communication, and whether the neighbor is one of the children.

```
$ thr get neighbors
Index Extended Address     ShortAddr  LastTime LinkMargin   Child
0     0xA3EE39B029C01FA7   0x0000     1        58            no
```

For detailed information about a specific neighbor, the command **`thr get neighborinfo <index>`** is used, where the index for each neighbor is the index from neighbor table.

```
$ thr get neighbor 0
Extended Address: 0x73B72371DF834B11
Short Address:    0x0001
Last communication: 0 seconds ago
InLinkMargin: 84
```

```
Device Timeout value: 13 seconds
Neighbor Synced
```

`thr get routes` - Displays the routing table of the current device, the ID sequence and the routing ID mask at the moment of the interrogation, along with several information about each entry, such as the router ID, the short address, next hop, cost, neighbor out link quality, and neighbor in link quality.

```
$ thr get routes
ID Sequence: 38
Router ID Mask: A000000000000000
RouterID    Short Address    Next Hop    Cost    NOut    NIn
0           0x0000           0x0000      1       3       3
```

`thr get parent` - Displays the short and extended address of the parent node.

```
$ thr get parent
Parent short address: 0x0000
Parent extended address: 0xA3EE39B029C01FA7
```

`thr set` - Command used for modifying the values of the attributes.

NOTE

To check the attributes that can modified from shell, please enter "thr get attributes"

```
$ thr set iscommissioned 0
 Success!
```

`thr nwkdata` - Performs Thread Network Data operations.

To add a DHCPv6 server, SLAAC server or external route, use the parameters, such as the length of the prefix or lifetime that can be set. The default values for prefix length is 64 and for lifetime is THR_NWK_DATA_MIN_STABLE_LIFETIME. The prefix is mandatory for adding network data. For SLAAC and DHCPv6 prefixes, the lifetime parameter must be at least 3600, which is the default minimum prefix lifetime imposed by the Thread specification.

Adding a dhcpserver prefix from the shell interface enables that device to become a DHCPv6 server in the Thread network. The device is able to assign IPv6 addresses to the other devices in the network. This is called a stateful autoconfiguration. Using a SLAAC prefix enables the device to compute an IPv6 address using the prefix and the EUI-64 address. This method is called the stateless autoconfiguration. Devices are also capable of having external routes for destination prefixes that need to get routed outside of the Thread network. Border Routers advertise such routes in Thread Network Data in tuples (destination prefix, RLOC16 address of the Border Router), hence egress traffic to the destination prefix is sent to the border router and it is forwarded to the external interface, via Ethernet/RNDIS/Serial TUN/TAP, depending on the capabilities.

After adding the prefixes, `thr sync nwkdata` propagates network data in the Thread network.

```
$ thr nwkdata add dhcpserver -p 2001::1
 Success!
$ thr nwkdata add slaac -p 2001::2 -t 3600
 Success!
$ thr nwkdata add extroute -p 2001:2002::0 -len 48 -t 120
 Success!
$ thr sync nwkdata
 Success!
```

`thr nwkdata remove` - Removes one prefix.

`thr nwkdata removeall` - Removes all network data from the current device.

`thr sync nwkdata` is needed after removing network data to propagate the updated information.

```
$ thr nwkdata remove -p 2001::1
 Success!
$ thr sync nwkdata
 Success!
```

`thr mgmt` - Performs Thread management commands that allow nodes to read and set network attributes. This set of commands is by default disabled, to enable it set SHELL_DUT_COMMISSIONER on TRUE.

`thr mgmt gettlvs` - displays a list of all network attribute names that can be set/get with thr mgmt commands.

`ch` - Channel TLV expressed as a decimal value (e.g. ch 12)

`pan` - PAN ID TLV expressed as a 2 bytes hexadecimal value (e.g. pan 0xafce)

`xpan` - Extended PAN ID TLV expressed as an 8 bytes hexadecimal value (e.g. xpan 0x1122334455667788)

`nwkname` - Network Name TLV expressed as a string of maximum 32 characters (e.g. nwkname threadtest)

`pskc` - PSKc TLV expressed as a 16 bytes hexadecimal value (e.g. pskc 0x7468726561646a70616b657465737400)

`masterkey` - Network Master Key TLV expressed as a 16 bytes hexadecimal value (e.g. masterkey 0x00112233445566778899aabbccddeeff)

`mlprefix` - Network Mesh-Local Prefix TLV expressed as an 8 bytes hexadecimal value (e.g. mlprefix 0xfd000db800000000)

`steering` - Steering Data TLV expressed as a hexadecimal value of maximum 16 bytes in length (e.g. steering 0x113320440000)

`brloc` - Border Agent Locator TLV expressed as a 2 bytes hexadecimal value (e.g. brloc 0x0400)

`sid` - Commissioner Session ID TLV expressed as a 2 bytes hexadecimal value (e.g. sid 0xffff). This TLV is added by default with a valid session ID. This TLV can be added only when Test Harness requires an invalid value.

`secpol` - Security Policy TLV has two: the rotation time expressed as a decimal value and the security policy bits expressed in hexadecimal (e.g. secpol 3600 0xc7)

`activets` - Active Timestamp TLV expressed in decimal value, in seconds (e.g. activets 101)

`pendts` - Pending Timestamp TLV expressed in decimal value, in seconds (e.g. pendts 101)

`delaytmr` - Delay Timer TLV expressed in decimal value (e.g. delaytmr 3000)

`chmsk` - Channel Mask TLV expressed as a 4 bytes hexadecimal value (e.g. chmsk 0x001fffe0)

`scan` - Scan Duration TLV is used only with get commands. The current implementation does not support setting a value for this TLV.

`energylist` - Energy List TLV is used only with get commands. The current implementation does not support setting a value for this TLV.

`future` – This is an unknown TLV value used for negative testing. No value should follow this TLV name in a set command.

`thr mgmt activeset` – this command sent from Commissioner or other Thread devices may provide a new Active Operational Dataset to the Leader. The arguments of this command are the IPv6 destination address and all attributes that need to be set followed by their value.

NOTE

All attribute names are followed by their value, exception makes “secpol”, Steering Data TLV, which is followed by two parameters: rotation interval (decimal value) and policy flags (hexadecimal value). After sending the command, the recipient will send an acknowledgment message with the status: Accept, Reject or Pending.

```
$ thr mgmt activeset fd00:db8::ff:fe00:0 activets 107 chmsk 0x001fffe0
xpan 0x000db70000000000 nwkname GRL pskc
0x7468726561646a70616b657465737400 secpol 3600 0xc7 steering
```

```
            0x113320440000 future
             Success!
            $ Received state: Reject
```

NOTE

The same behavior applies to thr mgmt pendset and thr mgmt commset commands

`thr mgmt activeget` – this command sent from Commissioner or other Thread devices may retrieve Active Operational Dataset values from the Leader. The arguments of this command are the a list of all attribute names to be retrieved.

```
            $ thr mgmt activeget fd00:db8::ff:fe00:0 ch pan xpan nwkname pskc
            masterkey mlprefix steering brloc sid secpol activets pendts delaytmr chmsk
            Success!
            $ ch: 14
            pan: 0xface
            xpan: 0x000DB70000000000
            nwkname: GRL
            pskc: 0x7468726561646A70616B657465737400
            masterkey: 0x00112233445566778899AABBCCDDEEFF
            mlprefix: 0xFD000DB800000000
            secpol: 3600, 0xc7
            activets: 107 sec,  0 ticks
```

NOTE

The same behavior applies to thr mgmt pendget and thr mgmt commget commands.

`thr mgmt query` – this command is sent from Commissioner to request one or more devices to perform an IEEE 802.15.4 Active Scan and determine if a specific PAN ID value is in use. The arguments of this command are the IPv6 destination address, the pan ID and the channel mask.

```
            $ thr mgmt query fd00:db8::ff:fe00:0 chmsk 0x001fffe0 pan 0xAFCE
            Success!
            $
```

`thr scan active` - Scans all channels for active Thread networks.

`thr scan energy` - Performs energy detection on all channels, from 11 to 26. The output of the commnd contains a byte for each scanned channel.

`thr scan both` - Performs both types of scanning, active and energy.

```
 $ thr scan active
Thread Network: 0
    PAN ID: 0xface
    Channel: 11
    LQI: 135
    Received beacons: 2
Thread Network: 1
    PAN ID: 0xbdb9
    Channel: 22
    LQI: 121
    Received beacons: 1

$ thr scan energy
```

```
Energy on channel(s): 0x1D

$ thr scan both

$
Energy on channel(s): 0x1D
Thread Network: 0
    PAN ID: 0xface
    Channel: 11
    LQI: 121
    Received beacons: 2
```

Chapter 5
CoAP Messages

`coap` - Sends a packet over the air using CoAP. When sending a CoAP message, all command parameters are mandatory. One has to specify the type of the message CON (confirmable message, waits for ACK) or NON (does not wait for any ACK), the request code, the IPv6 destination address, the URI-path options included in the message, and payload.

NOTE

URI-path options must be separated by /. Also the / separator is expected before the first URI-path option.

```
$ help coap
coap - Send CoAP message
   coap <reqtype: CON/NON> <reqcode (GET/POST/PUT/DELETE)> <IP addr dest> <URI path> <payload ASCII>
Example:
         coap CON POST 2001::1 /led on
         coap CON POST 2001::1 /led off
         coap CON POST 2001::1 /led toggle
         coap CON POST 2001::1 /led flash
         coap CON POST 2001::1 /led rgb r255 g255 b255
         coap CON GET 2001::1 /temp
```

If no ACK for the CoAP message is received after 4 retransmissions, an error message is displayed.

```
$ coap CON GET fd20:9237:6247::ff:fe00:400 /temp
$ coap rsp from fd20:9237:6247::ff:fe00:400 ACK Temp:25.23

$ coap CON POST fd20:9237:6247::ff:fe00:400 /led off
$ coap rsp from fd20:9237:6247::ff:fe00:400 ACK

$ coap CON POST fd20:9237:6247::ff:fe00:401 /led r100 r010 b200
$ No response received!
```

Chapter 6
MAC Filtering Commands

`macfilter` - Performs operations on filtering incoming packets at the MAC layer.

```
$ help macfilter
macfilter - MAC filtering commands
   macfilter enable <reject|accept>
   macfilter add <neighbor extended address> reject <0/1> lqi <neighbor link indicator>
   macfilter remove <neighbor extended address>
   macfilter disable
   macfilter show

   Example:
   macfilter enable accept
   macfilter add 0x1122334455667788 reject 1
   Neighbor link indicator values:
      Good Link: 20 - 255
      Medium Link: 11 - 20
      Bad Link: 3 - 10
```

`macfilter enable <reject|accept>`- Enables filtering at MAC layer. Flags `reject` or `accept` refers to the default policy that applies to all the received packets from devices that are not in the filter list.

If the default policy is `accept`, all packets received from nodes that are not in the table will be accepted, unless the entry has the entry `reject` flag set to 1. If the `reject` flag is set to 0, and `lqi` flag is specified, the Link Quality Indicator for the incoming packets from the neighbor are always set to this value.

If the default policy is set to `reject`, all packets are dropped, unless the sending device is in the list, with `reject` flag set to 0. If the `lqi` flag is specified, the Link Quality Indicator for the incoming packets from the neighbor is always set to this value.

```
$ macfilter enable accept
MAC Filtering Status: Enabled - Default policy: Accept
```

`macfilter disable` - Disables filtering at MAC layer.

```
$ macfilter disable
 MAC Filtering Disabled
```

`macfilter show` - Prints MAC filtering table.

```
$ macfilter show
MAC Filtering Status: Enabled, Default policy: Accept

Idx   Extended Address     Short Address   Link Quality  Reject 0     0x00049F02B4610039
0x0400          137           FALSE
1     0x00049F02B4610038   0x0401          137           TRUE
2     0x00049F02B4610038   0x0800          126            FALSE

End of MAC Filtering Table.
```

Chapter 7
Echo UDP Commands

The Echo UDP module is used to send test UDP data to a destination which echoes the payload back to the originator. The module can be enabled/disabled by setting the `UDP_ECHO_PROTOCOL` macro definition to TRUE/FALSE in the configuration file app_thread_config.h.

`echoudp` - Initiates an echo request over UDP to the specified destination. An echo reply is expected.

```
$ help echoudp
echoudp - Echo udp client
   echoudp -s<size> -S<source address> -t<continuous request> -i<timeout> <target ip address>

$ echoudp -s 500 -t fd31:cd3d:a447::ff:fe00:0
Message sent to fd31:cd3d:a447::ff:fe00:0 with 500 bytes of data:
Message received from fd31:cd3d:a447::ff:fe00:400: bytes=500, time=787ms
```

Chapter 8
Socket Commands

Socket commands allow using the Thread IP stack functionality via the shell at the socket level. The module can be enabled/disabled by setting the `SOCK_DEMO` macro definition to 1/0 or TRUE/FALSE in the application configuration file (e.g.: `config.h`).

The shell commands available for the socket demo application are displayed after typing `help socket` in console.

```
$ help socket
socket - IP Stack BSD Sockets commands
      socket open <protocol> <local ip addr> <local port> - Local address can be 0::0 and local port
0 for automatic selection
      socket send <socket id> <payload>
      socket close <socket id>
      socket connect <socket id> <remote ip addr> <remote port>
```

These commands are valid for opening a socket as a UDP client.

`socket open <protocol> <local ip addr> <local port>` - Opens a socket and binds it to local address and local port. Usually local address is the unspecified address 0::0 so that the stack selects a source address according to destination address.

In this example, a UDP socket is opened and connected to the remote address 2001::0, port 1234.

```
$ socket open udp 2001::0 1234
Opening Socket... OK
Socket id is: 0
$ socket connect 0 2001:1 1234
Socket is configured with remote peer information
```

`socket send <socket id> <payload>` - Sends the ASCII payload on a previously open socket. The socket must be connected by calling socket connect command first.

`socket close <socket id>` - Closes the socket using its identifier and all allocated resources are freed.

```
$ socket send 0 rawsocketdata
Socket Data Sent
$ socket close 0
Socket 0 was closed
```

Chapter 9
DNS Client

DNS commands allow testing a simple DNS client in a Thread node. To enable this functionality, set the DNS_ENABLED macro definition to 1. Note that this application is disabled by default.

To initiate a DNS request, a DNS server must be advertised in Thread network. Therefore, a border router that is connected to a DNS server should propagate to Thread nodes the IPv6 address of the DNS server and the user can trigger a DNS request for resolving a domain name.

For example, if a border router is connected through Ethernet or RNDIS to a DNS server, network data may be updated by using the following shell commands:

```
$ thr nwkdata add dnsserver fd01::e211:a00:27ff:fe87:a104
 Success!
$ thr sync nwkdata
 Success!
```

where the IPv6 address is the DNS server's.

All Thread nodes will update their information about the address where the DNS server is found.

Therefore, if a DNS request is initiated from any of the nodes of the Thread network, the message will be forwarded by the border router to the DNS server and back.

In order to trigger a DNS request message, use the shell command dnsrequest and the URL to be resolved:

```
$ dnsrequest kinetisthread.local
kinetisthread.local is at 2003::3ead
```

To remove the information about the DNS server from the Thread network, remove it from network data as follows:

```
$ thr nwkdata remove dnsserver fd01::e211:a00:27ff:fe87:a104
 Success!
$ thr sync nwkdata
 Success!
```

Chapter 10
Revision History

This table summarizes revisions to this document.

Table 1. Revision history

Revision number	Date	Substantive changes
0	10/2015	Initial release
1	03/2016	Added identify command, DNS client functionality, and updated echoudp
2	06/2016	Updates for multicast group commands
3	03/2017	Added detach command and updates to scan command
4	01/2018	Updates for KW41 Maintenance Release
5	05/2018	Updates for K3S Beta Release
6	04/2019	Updates for KW41Z Maintenance Release 3.

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, All other product or service names are the property of their respective owners. Arm, AMBA, Arm Powered, are registered trademarks of Arm Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved.

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 08 April 2019
Document identifier: KTSSHIUG

arm